# Automated transfer in reinforcement learning

# Automated Transfer in Reinforcement Learning

DISSERTATION

to obtain the degree of Doctor at
Maastricht University,
on the authority of the Rector Magnificus,
Prof. dr. L. L. G. Soete,
in accordance with the decision
of the Board of Deans,
to be defended in public
on Thursday June 13, 2013 at 12:00 hours

by

Haitham Bou Ammar

**Supervisor:**
Prof. dr. G. Weiss

**Co-supervisors:**
Dr. K. P. Tuyls
Dr. M. E. Taylor (Washington State University)

**Assessment Committee:**
Prof. dr. ir. R. L. M. Peeters (*chairman*)
Prof. dr. D. Ernst (Liège University)
Prof. dr. ir. J. C. Scholtes
Prof. P. Stone (University of Texas at Austin)
Dr. ir. R. L. Westra

SIKS Dissertation Series No. 2013-24
*The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.*

For my family.

In memory of **Kamal Jumblatt**. Your teachings left fingerprints of grace, knowledge, enlightenment, and higher erudition in all our lives.

# Acknowledgements

I would like to thank my supervisor Karl Tuyls for guiding me throughout this research and for all the interesting and productive discussions we had. When I was lost, Karl was the person re-adjusting the tracks to push my work forward. Furthermore, I would like to thank Gerhard Weiss for giving me the opportunity to conduct research in the interesting field of transfer learning, as well as for the nice, funny, and productive smoking breaks we had between now and then. I would like also to thank Matthew Taylor for all the help, guidance, and discussions we had. Thank you for recommending me to be admitted as a Ph.D. candidate at the university of Maastricht. Matt is the person through whom I got introduced to the field of transfer learning. Moreover, my gratitude to my office mate Kurt Driessens. I would like to thank Kurt for all the funny moments we had, as well for playing the "devil advocate" in all our discussions. Definitely, these improved my dissertation.

Besides my promotors, supervisors, and co-supervisors, my dissertation has greatly benefited from the comments and suggestions of the assessment committee. A special thanks goes to Damien Ernst, Peter Stone, Jan Scholtes, Ronald Westra, and Ralf Peeters, for the extensive comments that definitely pushed this dissertation forward.

A part of being a researcher is to exhibit interest in different domains broader than one specific topic. Having this interest, I was privileged to meet and get in contact with experts that widened my scope of understanding. I would like to thank Sam Harris for all the interesting discussions on philosophy, and Brian Greene and Ronald Westra for opening my eyes on different aspects of our physical world.

Moreover, I would like to thank my colleagues at the swarmlab for making the work in Maastricht more exciting. Thanks to each of, Daan Bloembergen, Sjriek Alers, Siqi Chen, Bijan Ranjbar-Sahraei, Daniel Claes, Josha Fossel, Frans Oliehoek, and Michael Kaisers.

Furthermore, I would like to thank Decebal and Elena Mocanu for all the nice evening, interesting talks, and fruitful collaborations. A special thanks to You Li for making my life more exciting by the day.

My friends and colleagues in Germany also played a prominent role in accomplishing this dissertations. I would like to thank each of, Mohammad Chami, Richard Cubek, Benjamin Staehle, Markus Schneider, Tobias Fromm, and Karl Glatz.

Having started a business, I would like to thank three visionaries. Firstly, I would like to thank Ghassan El Aridi, a visionary that is transforming the tourism world into a better place. I have learned a lot and I am excited for much more to come. Secondly,

I would like to thank Rabih El Aridi for all the help, comments, and suggestions. Giving me a hard-time at first, definitely improved my plans and made me learn a lot. Thirdly, I would like to thank my friend, and brother Jad El Aridi. With Jad the limit is the sky. Thank you for all the nice times, the interesting discussions, and the amazing future plans. I have learned a lot from Jad was it on the personal or the business side.

Maybe the most important part of my life is my family that played above all a vital role in finishing this dissertation. I would like to thank my mother (Rawya Bou Ammar) for all the care and the hard years, my brother (Dany Bou Ammar) for all the fun moments and for always being there for me, and my father (Joseph Bou Ammar) for teaching me endurance. If it was not for you I would have never been here.

While staying in Maastricht I acquired three additional family members to whom I like to express the deepest gratitudes. Thank you Steven de Jong for always being there for me, and for all the fun, fruitful, and productive collaborations we had. Furthermore, I would like to thank Stephanie Van Nispen for being one of the most caring and loving persons I have known. Finally, I would like to thank Daniel Hennes the friend and the colleague for all the interesting talks, drinking evenings, advices, and fruitful discussions we had together. Welcome to our small family!

Haitham Bou Ammar
May 27, 2013

# Contents

# 1 Introduction

Humanity has been craving the assimilation of diverse challenging questions. It is amazing to recognize that a species contrived from fundamental particles is capable of prospering, and adapting to reach a stage of development unanticipated by any of its ancestors. As Kamal Jumblatt says, "We, essentially formed of the same material the universe is made of, evolved to develop consciousness allowing us to probe nature at an empirical level. It seems as if the universe is trying to understand itself!" This form of empirical probing to nature is framed under the context of science. Science generated immense and exciting discoveries. For example, discovering quantum theory one of the major driving forces behind all technological developments, allowing the creation of computers, which facilitated the accomplishment of tasks never possible before. With the aid of computers, an activity that required hours to be executed, can now be performed in a matter of seconds. The other important milestone of scientific exploration dates back to the second of September 1969. At that date the internet was born. This new discovery again shifted our lives, and affected us profoundly. Nowadays, there are civilizations built around the internet. New terms never heard before, such as social networks, emerged, paving the way to more connectivity and ease of communication. Of course, the discoveries in science range widely beyond the scope of computers and the internet. Developments in medicine, biology, chemistry, and engineering have also been observed. Each affected our lives in its own way. However, according to Michio Kaku[1], this century is the century of computers. Therefore, the focus will be on computers as the running example as this will prove to be an interesting introduction to the topic of this thesis.

However amazing these scientific explorations were, the driving thrust behind their discoveries and implementations is human intelligence. It would have never been possible to have such milestones if the human brain had not existed. Understanding intelligence is at the core of all scientific developments and discoveries. If we are to

[1]Michio Kaku, the digital age in an analog world, and Michio Kaku, the future of quantum computing.

prosper far beyond our current state, cultivating such intelligence creating machines (i.e., brains) is of major interest. Furthermore, comprehending human intelligence will allow for the creation of *truly* intelligent computers or agents. At the end, the question is how can any one replicate a certain phenomenon, such as human intelligence, without understanding its underlying operations? Attractive as it might sound, conceiving human intelligence is an extremely hard and far-fetched goal. There are a lot of different factors to be included, various definitions to be adopted, highly complex and unobserved contributors that need to be taken into account, and above all the possibility of un-empirical characteristics being involved.

When it is hard to comprehend a phenomenon directly, the most efficient method to analyze it is indirectly. Dazzling with the question of human intelligence and trying to approach the problem indirectly, the field of Artificial Intelligence (AI) was established. AI tries to create autonomously operating, highly adaptable computer agents that try to mimic human intelligence. This form of indirect reasoning is interesting since: (1) if successful, computers can have abilities to solve highly complex problems not possible before, (2) these successes might resemble similarities to human intelligence, and (3) such resemblance can be described in the most accurate language known to humanity (i.e., mathematics). Although all of the previous three points are important, the most appealing is the third. Describing certain behaviors in mathematics gives the deepest possible form of understanding. Of course intuition is important too, however, no intuition is developed without a deeper and more accurate understanding. To clarify, assume now that we explain in layman terms how computers operate. A computer then is a collection of different hardware units coupled with software, together leading to a programable machine. At this level, no one would be able to design, create, implement or improve on current computers. Even if the discussion dived into more details trying to explain different hardware units and their connections together, the software and its operations et. cetera, it will still be hard for anyone to take such knowledge and create anything tangible. However, if the explanation was performed in a more accurate language, such as mathematics– even rule-based mathematics– all of the above becomes possible and feasible. This is a trend that has been observed over and over again throughout the history of science.

As AI developed, different sub-fields were created. These range from heuristic based approaches to mathematically grounded ones such as these residing in the realm of machine learning (ML). Maybe the closest to human learning is the reinforcement learning (RL) sub-field of ML. In RL, an agent lives in an environment which it can perceive through sensory signals and affect by taking actions. To asses the behavior of a certain action completed by an agent, a mathematical signal (i.e., reward) is defined. This reward will punish the agent in case it has performed a “bad” action, and reward it for a “good” one. The agent then learns to maximize its total positive

signal. To better understand how RL relates to human learning, consider the following example. Assume you want to teach your child, not to touch fire. Every time the child approaches fire, you yell at him, and every time he avoids it you reward him, say by candy. With time, the child's brain will relate his positive and negative rewards to the actions of avoiding or approaching fire, respectively. As the child loves sweets, he would want to maximize his positive reward and thus end-up "converging" to the behavior of avoiding fire. In the scenario discussed previously, the agent is learning similarly. Any time it performs a "bad" action, say to approach fire, it receives a negative reward (analogous to being yelled at in the child's case) and any time it performs a "good" action, say to avoid fire, it receives a positive reward (analogous to receiving candy in the child's case). With time the agent will learn to maximize it's behavior and thus avoid fire. [2]

Although interesting, RL suffers from major computational problems. In RL, agents typically require a huge amount of time and experience to learn meaningful behavior on complex tasks. The main reason behind this short-coming is that these agents typically start their learning from scratch. Thus, the agent has to build its knowledge, based mostly on random interactions with the environment, in order to reason and learn a successful behavior. On the contrary, in humans learning never starts from scratch. Objective thinking is extremely hard in everyday life. It is impossible for anybody to generate ideas and directions that are not similar to previously encountered experiences and events. We are constantly re-using knowledge and behaviors already acquired in similar tasks, to help in obtaining a new skill. For instance, consider Dany, who would like to learn physics. Dany has no idea whatsoever about physics, but has acquired, throughout his study, a thorough understanding of mathematics. Once attempting to study physics, Dany will perform much better by using his mathematical knowledge compared to starting from scratch. Ideas such as derivatives, integrals, linear algebra, and geometry will indeed help him in successfully mastering topics in physics. In RL this form of knowledge re-use is framed under *transfer learning*. Therefore, if any RL agent is to mimic human learning, transfer learning has to be considered a crucial ingredient of the overall framework. By including this learning scheme in RL agents, not only more resemblance to human learning might emerge, but also the overall behavior of agents in new tasks will be improved.

## 1.1 Aim of the Thesis

So far, different transfer algorithms for RL agents, with varying degrees of autonomy, have been proposed. However, there exists no approach that is fully autonomous,

[2] It is worth noting, that this example of course is an oversimplification of human learning, but serves as a perfect illustration of the ideas behind RL.

where it does not require substantial human intervention to successfully perform transfer. As described in [107], to be fully autonomous, an RL transfer agent has to perform all of the following steps successfully:

- Given a target task, select an appropriate source task or set of tasks from which to transfer.
- Learn how the source task(s) and target task are related.
- Effectively transfer knowledge from the source task(s) to the target task.

To clarify, consider the example, shown in Figure 1.1. The target task is the so-called cart-pole problem. In this problem, there is a pole attached to a cart that can translate on a flat surface. The goal of the agent is to execute the correct linear actions so to control the pole in an upright position.

Furthermore, the target agent has access to a collection of source tasks. In Figure 1.1, four different tasks are shown. The first is the inverted pendulum, where the goal is to choose the correct rotational actions such that the pole is controlled in an upright position. The difficulty in the task is that the actions can not upswing the pole in one shot, rather it has to oscillate around its equilibrium position gaining enough momentum to swing upwards. The second task is the mountain car. Starting at the bottom of the valley, the goal of the agent is to drive the car up the hill. However, the thrust of the engine is not enough for the car to reach the top of the hill in one shot. It rather has to oscillate to gain enough momentum to achieve the goal. The third task is the simple mass system. In this task the goal is to position the mass at a certain pre-specified value. Finally, in the fourth (i.e., the double mass system) the goal is to control the first mass at a certain position while only being able to apply actions that affect the second mass.

To automatically transfer between one of these source tasks and the cart pole, three steps need to be successfully executed. Firstly, the transfer agent has to decide on the source task to transfer from. In other words, it has to decide which of the source tasks is most *useful* to the given target task. Such notion of similarity mainly depends on two factors: (1) dynamical similarity, and (2) goal similarity. As detailed in Chapter 5 and Chapter 6, under certain assumptions, focusing on the first type of similarity is to some extent capable of capturing goal similarities and can lead to successful transfer. After deciding on what task to transfer from, the agent has to reason about the relation between the source and target task in order to successfully conduct transfer. For example, assume that in the first step the transfer agent has decided to choose the inverted pendulum as the source task. This task's state and actions are described in different spaces compared to these of the cart pole. Therefore, before any transfer can be conducted the relation between the source and target task

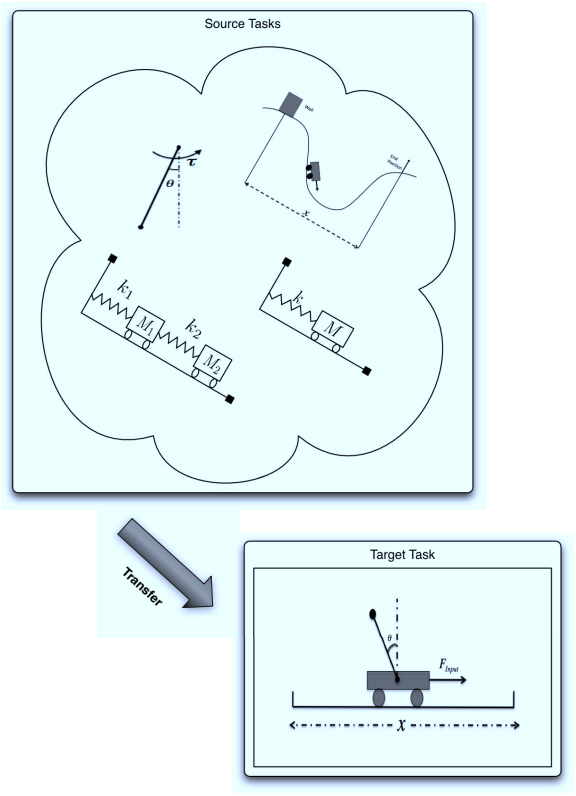



Figure 1.1: An example of a transfer Learning scenario. The upper panel shows a bag of different source tasks, while the lowers shows a specific target task. To autonomously transfer, the agent has to: (1) choose a relevant source task(s), (2) reason about the relations between the tasks, and (3) effectively use the source task knowledge.

state-action space has to be determined. This relation will configure the source task's knowledge such that it is compatible to be received by the target agent. Finally, the third step is then to efficiently and effectively transfer the source's knowledge to the target. It is worth noting, that this transfer will not provide the optimal behavior in the target task, the agent has still to improve using normal reinforcement techniques to attain the optimal behavior. However, the hope is that the transferred knowledge can provide a "good" starting prior that can be used to improve the learning performance.

## 1.2 Problem Definition

Current transfer learning methods require substantial human intervention. Aiming at providing an autonomous transfer learning framework the following problem statement is formulated:

**Problem Statement:** *How to automate transfer between different reinforcement learning tasks?*

Considering this problem statement, it is important to tackle the above three challenging steps that need to be performed by a transfer agent: (1) choose a specific source tasks, (2) reason about the relations between the two tasks, and (3) effectively transfer knowledge between the different tasks. However, the above problems are approached in the following manner: first, the second two steps are considered and then the first challenge is solved. The problem statement is then split into two parts. In the first, the second and third steps are answered, while in the second, the questions of choosing a relevant source task is detailed.

### **Part I**: Effective Learning and Transfer

Before diving into the details of creating autonomous transfer agents, a detailed review of the transfer literature was required. To this end, the following research questions are formulated.

**Research Question 1:** *What categorization of the current transfer algorithms exist?*

Before creating autonomous transfer agents, a survey of current transfer learning algorithms was conducted. The algorithmic devision presented here, generates a mathematically grounded framework for transfer in reinforcement learning. More specifically, it frames all transfer learning methods depending on the assumptions and procedure followed to learn the relation between the tasks.

Having performed the literature review, it was clear that no current transfer algorithm is capable of autonomously transferring between different reinforcement learning tasks. Therefore, the second research question was derived.

**Research Question 2:** *To what extent is it possible to reduce human intervention in reasoning about the relations between different tasks?*

Given that the current transfer learning algorithms are capable, with substantial human intervention, of transferring between different tasks, the question is wether it is possible to reduce this effort made by the user. As described previously the source and target tasks might have different state and action spaces. Therefore, to conduct

transfer, the knowledge in the source has to be correctly mapped to be received by the target. Aiming at reducing the human role in learning such a mapping, the problem is approached by defining a common state-subspace. This sub-space is used to relate transitions in each of the source and target tasks. Having such a correspondence, regression is then applied to learn this relational mapping.

Although successful, this approach still suffers when the common subspace is hard to be manually defined. To address this problem, the next research question is presented.

**Research Question 3:** *Is it possible to automate learning the relation between different reinforcement learning tasks?*

To automate learning the relation between tasks with different constituents, an approach based on automatically discovering a rich feature space is adopted. This space is capable of representing both the source and target task transitions, making it suitable for transfer. Given such a space, the relations between the tasks is then learned using: (1) a similarity measure, and (2) regression technique. These relation can then be successfully used to perform transfer.

Although successful, this approach might suffer from robustness problem depending on the nature of the presented samples. Leveraging this problem, the next question is formulated.

**Research Question 4:** *Are there any possible more robust alternatives to the learning of the relations between the tasks?*

Discovering the common space that is capable of relating source and target tasks can be performed using different techniques. Here an alternative based on deep learning techniques (detailed in Chapter 2) is adopted in order to learn such a relation between the tasks that can then be used to successfully conduct transfer.

Having figured out three different techniques that automatically relate source and target tasks, it is time to derive a method that can effectively make use of these mappings to successfully perform transfer. To this end, the following research question is formulated.

**Research Question 5:** *How to effectively utilize the learned mapping to successfully transfer between different tasks?*

Given that the relation has been already learned, two novel algorithms are proposed that are capable of effectively and efficiently transferring between two different tasks. To insure a broad scope of applicability the focus was on reinforcement learning approaches that operate within a continuous state space setting. These new transfer techniques make use of the advantages possessed by state-of-the-art reinforcement learning methods, thus assuring transfer efficiency.

### **Part II:** Choosing the Relevant Source Task

Having successfully solved the first and second steps towards attaining autonomous transfer agents, the next problem was to determine a technique that allows agent to successfully choose a relevant source task for a given target. It seemed that a similarity measure quantifying the relations between reinforcement learning tasks was essential. Therefore, the next two research questions were evaluated. It is worth noting, that the proposed techniques are data-driven and can operate within the context of highly dissimilar tasks.

**Research Question 6:** *Is it possible to learn a similarity measure between reinforcement learning tasks with same state and action spaces?*

To quantify the similarity between two reinforcement learning tasks, a measure based on deep learning techniques is proposed. The main intuition is that if two tasks are related then a high informative rich space describing the first should also be capable of characterizing the second.

However, this measure is only able of finding the similarities between tasks with the same state and action spaces. Aiming at a more general framework, the second research question is posed.

**Research Question 7:** *Is it possible to learn a similarity measure between reinforcement learning tasks with different state and action spaces?*

Extending the previous approach, this question is answered by constructing a more general method that learns the similarity between the tasks. The main idea is to generalize the previous configuration into a more informative representation while allowing the flexibility for differences in all the tasks' constituents.

## 1.3 Structure of the Thesis

The thesis is structured as follows. Chapter 2 provides the reader with background knowledge needed to understand the remaining chapters. Various ideas from machine learning, including supervised learning, unsupervised and deep learning, and reinforcement learning are discussed.

The research questions are addressed in Chapter 3 to 7. Firstly, (i.e., Chapter 3) research question 1 is answered. Namely, a theoretical framework categorizing different transfer learning algorithms is presented. Chapter 4, answers research question 2, by presenting the first attempts to reducing human intervention for learning a mapping between source and target tasks. Research question 3, 4 and 5 are then answered in Chapters 5, and 6. Two methods for automatically learning the relations between

source and target tasks are detailed. These are then used, by adapting two efficient reinforcement learning algorithms. Chapter 7 then answers research questions 6 and 7. Two similarity measures capable of capture the similarities between different tasks are presented.

Finally, Chapter 8 provides the conclusion by answering the proposed research questions and problem statement. Also interesting future work directions are detailed.

# 2 Preliminaries

This chapter provides background knowledge needed by the reader to comprehend the remainder of the dissertation. The reader is assumed to have background knowledge in probability theory, statistics, and numerical optimization. It draws upon different ideas from machine learning. Various methods from supervised, unsupervised, and deep learning adopted in this work are described.

First, in Section 2.1, supervised learning methods from machine learning are explained. These include both parametric and an instantiation of nonparametric techniques. Supervised learning methods are needed later in Chapters 4 and 6 to learn the relation between two reinforcement learning tasks. Second, in Section 2.2, unsupervised and learning methods are detailed. These include sparse coding and Boltzmann machines that are used in Chapters 5 and 6 to discover a unifying space among two reinforcement learning tasks. Finally in Section 2.3, reinforcement learning the basis behind this dissertation is presented.

## 2.1 Supervised Learning

Supervised Learning (SL) is a widely used machine learning technique. SL algorithms answer the question of finding a latent relation between a set of dependent and independent variables. On a high level, there are two types of SL techniques. Their main differences depend on the nature of the output variable. In regression, the first type, the output is continuous, while in classification, the output is discrete and belongs to a preset number of classes. To clarify, consider the following two examples. Suppose dealing with weather forecasting. If the goal was to predict temperature, then the solution is within the regression realm since this variable is continuous and can take on infinite number of values within certain ranges. On the other hand, consider predicting weather conditions say, rainy, sunny, et. cetera. In this example the output is discrete, whereby it can belong to a set of predefined classes and thus, associates to the classification world.

In this dissertation most of the adopted algorithms are from regression. The motivations behind endorsing regression reside in the nature of predictions made in continuous reinforcement learning problems. An overview of different regression techniques is presented next. For a thorough discussion of other machine learning algorithms, the reader is referred to [13].

The regression problem can be stated as follows: *given a data set* $\mathcal{D} = \{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{n}$*, where* $\boldsymbol{x} \in \mathbb{R}^d$ *is the d-dimensional independent vector,* $y \in \mathbb{R}$ *is the dependent vector, and* $n$ *is the number of available data points, the question is to determine a latent function,* $f$*, such that the model "fits" the data well.*

To asses the performance of the model, different types of estimators need to be introduced. An estimator is a mathematical construction that (probabilistically) quantifies the behavior of the adopted model. For instance, likelihood, is one form of these estimators. Roughly speaking, the likelihood measures how *likely* are the outputs to occur given the model and the input data.

However, before discussing the available estimation techniques, different model types should first be introduced. On a high level, a model is simply a combination of features and parameters that are likely to describe the available data. In regression, two types exist: (1) parametric, and (2) nonparametric models. The differences between the two, mostly depend on the nature of the defined distributions. For instance, in parametric models, the probability distributions are defined over the free parameters or weights of the model. On the other hand, in nonparametric methods, the probability distributions are typically defined directly in the *function space*. Please note, that although the name suggests the absence of any parameters, in nonparametric regression, there are still free parameters to be fitted. The main differences to parametric methods, are that these parameters define probability distributions in the function space rather than the parametric space (i.e., the space spanned by the free parameters).

Next, each of these are detailed. It is worth noting, that since the parametric approach is primitive and incurs additional assumptions on the learnt function, it will not be described in full details. Interested readers are referred to [13, 22] for a more careful discussion.

## Parametric Regression

One of the biggest problems of parametric methods, are the assumptions made on the model. Namely, the overall shape or variation of the latent relation between the $\mathbf{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$, is assumed to be captured by the designer. In parametric regression, a manual and careful analysis of the "style" of the model to be used is typically performed by the designer. For example, he/she might decide that a linear

function[1] is a good fit for a given data set. Although plausible, it is hard in most real-world applications to determine such information since, for instance, the data could be of a high dimensional nature. However, it is worth noting that there are certain parametric regression techniques that relax these assumptions. For instance, Neural Networks (NNs) are one form of these methods aiming at reducing the burden on the designer in choosing the correct model. Albeit, NNs suffer from their own problems related to local minima and complexity expressiveness, see [22, 70].

After deciding on the overall shape of the model, the free parameters are then inferred from the available data set. To asses the behavior of model, an estimate quantifying this performance needs to be defined. In machine learning there are different types of these estimates. Next two such estimates, the maximum likelihood and the maximum a posteriori, are described.

### Maximum Likelihood

The first type of estimators to be defined is the maximum likelihood [13]. Maximum likelihood is widely used in different applications of machine learning [49, 67, 77, 95]. The main idea is to fit the model free parameters or weights such that the likelihood of the output to occur conditioned on both the input data as well as the free parameters is maximized. For that there exists the need to define the so-called likelihood function, which is a function of the parameters and *not* of the input data.

**Definition: Likelihood Function** Let $\Gamma$ be the parameter space and $p_\gamma$ be the density with respect to $\gamma \in \Gamma$, the likelihood function is defined as:

$$
\begin{aligned}
&L_{\mathbf{x}} : \Gamma \to \mathbb{R}_+ \\
&L_{\mathbf{x}}(\gamma) = p_\gamma \quad \forall \gamma \in \Gamma \; \forall \mathbf{x} \quad \in \mathcal{X}, \text{ where } \mathcal{X} \text{ is the sample space.}
\end{aligned}
$$

The estimator $\mathcal{E}$ is called the maximum likelihood estimator if:

$$
\begin{aligned}
&\mathcal{E} : \mathcal{X} \to \Gamma \\
&\text{with,} \\
&L_{\mathbf{x}}\left(\mathcal{E}(\mathbf{x})\right) = \sup_{\gamma \in \Gamma} L_{\mathbf{x}}(\gamma), \quad \forall \mathbf{x} \in \mathcal{X}
\end{aligned}
$$

To maximize the likelihood the derivative of the likelihood function is needed. In a lot of settings, deriving the likelihood function is hard and therefore, the logarithmic

[1] Please note that linear does not imply that the fitted model is a straight line. On the contrary, the function might include nonlinear features. Linearity here refers to the parameter space, in which these features are linearly combined. For example, $\theta_1 x_1 + \theta_2 x_1^2 \cos(x_2^3)$ is a linear functions in the parameter space spanned by $\theta_1$ and $\theta_2$.

likelihood is typically used.

**Example 1** [ML Bernoulli Distribution]
Let $\mathbf{x}^{(i)}$, $i \in \mathbb{N}_n$, be i.i.d. (independently and identically distributed) according to $\text{Bern}_\mu\left(\mathbf{x}^{(i)}\right)$ with $p\left(\mathbf{x}^{(i)}=1\right)=\mu$ and $p\left(\mathbf{x}^{(i)}=0\right)=1-\mu$. The joint probability of the data is given by:

$$\begin{aligned} p(\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(n)}) &= \prod_{i=1}^{n} p\left(\mathbf{x}^{(i)}\right) \\ &= \prod_{i=1}^{n} \mu^{\mathbf{x}^{(i)}}\left(1-\mu\right)^{\left(1-\mathbf{x}^{(i)}\right)} \\ &= \left[\mu^{\mathbf{x}^{(1)}}\left(1-\mu\right)^{\left(1-\mathbf{x}^{(1)}\right)}\right]\ldots\left[\mu^{\mathbf{x}^{(n)}}\left(1-\mu\right)^{\left(1-\mathbf{x}^{(n)}\right)}\right] \\ &= \mu^{\sum_{i=1}^{n}\mathbf{x}^{(i)}}\left(1-\mu\right)^{\left(n-\sum_{i=1}^{n}\mathbf{x}^{(i)}\right)} \end{aligned}$$

To maximize the likelihood, derive the previous equation with respect to the free parameter, $\mu$, to attain:

$$\begin{aligned} \frac{\partial}{\partial\mu}\ln L_{\mathbf{x}}(\mu) &= \frac{\partial}{\partial\mu}\left(\sum_{i=1}^{n}\mathbf{x}^{(i)}\ln(\mu)+\left(n-\sum_{i=1}^{n}\mathbf{x}^{(i)}\right)\ln(1-\mu)\right) \\ &= \frac{1}{\mu}\sum_{i=1}^{n}\mathbf{x}^{(i)}-\frac{1}{1-\mu}\left(n-\sum_{i=1}^{n}\mathbf{x}^{(i)}\right) \\ \implies \mu &= \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}, \text{ which is the sample mean.} \end{aligned}$$

When faced with a "function approximation" problem, most practitioners resort to least squares. In this widely used approximation technique, the parameters are fitted such that the sum of squared errors between the fitted model or function and the "real" data is minimized. Contrary to most belief, such an approximation method is not a heuristic. This model could be derived naturally from the maximum likelihood estimate under certain assumptions. In the next example, the least squares problem is derived starting from the probabilistic treatment of regression. Moreover, it will be shown, that minimizing the sum of squared errors, is actually equivalent to maximizing the likelihood estimate.

**Example 2** [Deriving Least Squares]
First, a data set $\mathcal{D}=\{\mathbf{x}^{(i)},y^{(i)}\}_{i=1}^{n}$, where $\mathbf{x}\in\mathbb{R}^d$ is the input vector, $y\in\mathbb{R}$ is the output variable, and $n$ representing the number of available data points is given. To derive least squares, one assumes that the output

noise term, $\epsilon$, is a Gaussian with a zero mean, and $\sigma^2$ covariance. The justifications behind this assumption is that such distributions are tractable making the mathematical derivations easier.

As in the case of any parametric regression technique, the shape of the function to be chosen is decided upon upfront. In other words, the features describing the sought function are determined by the designer, and further combined together. The combination in this example is linear, where the sought latent function is of the following form: $f(\mathbf{x}) = \boldsymbol{\theta}^T \Phi(\mathbf{x})$, with $\boldsymbol{\theta} \in \Gamma$ being the vector of free parameters that need to be inferred from the data, and $\Phi(\mathbf{x})$ representing the features used to describe the model.[2] Combining all the above assumptions together, the following is derived for one data point:

$$
\begin{aligned}
p(\epsilon^{(i)}; \sigma^2) &= \mathcal{N}(0, \sigma^2) \\
&= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left( -\frac{1}{2\sigma^2} \left( \epsilon^{(i)} \right)^2 \right) \\
&\implies p(y^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left( -\frac{1}{2\sigma^2} \left( y^{(i)} - \boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}) \right)^2 \right)
\end{aligned}
$$

In words, the last equation is saying that the probability for an output point to occur conditioned on the given inputs and parameterized by the free parameters is a Gaussian with a mean centered at the function values and a variance of $\sigma^2$. The perturbations of the real outputs compared to these predicted by the model is varying from the mean (i.e., $\boldsymbol{\theta}^T \Phi(\mathbf{x})$) by the noise variance $\sigma^2$. Given the data set and assuming that the data instances are identically independently distributed (i.i.d.), the joint likelihood function is calculated as:

$$
p\left(\mathbf{y} | \mathbf{X}; \boldsymbol{\theta}, \sigma^2\right) = \prod_{i=1}^{n} \mathcal{N}\left( \boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}), \sigma^2 \right) \tag{2.1}
$$

$$
p(\mathbf{y} | \mathbf{X}; \boldsymbol{\theta}, \sigma^2) = \prod_{i=1}^{n} \left[ \frac{1}{2\sigma^2} \exp\left( -\frac{1}{2\sigma^2} \left( y^{(i)} - \boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}) \right)^2 \right) \right] \tag{2.2}
$$

where $\mathbf{y} = [y^{(1)}, y^{(2)}, \ldots, y^{(n)}]^T \in \mathbb{R}^{n \times 1}$ is the vector collecting all the outputs, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the so-called design matrix collecting all the input vectors. Now the goal is to maximize the above function with respect

[2] Note, choosing the features of the model depends on a lot of factors. One typically defines these features, such that the data in the high-dimensional feature space is more likely to be linearly separable by a hyperplane.

to the free parameters $\boldsymbol{\theta}$. This is done by first taking the natural logarithm of the above and deriving it with respect to the parameters as follows:

$$\begin{aligned}
&\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} \left[ \frac{1}{2\sigma^2} \exp\left( -\frac{1}{2\sigma^2} \left( y^{(i)} - \boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}) \right)^2 \right) \right] \\
\Leftrightarrow &\max_{\boldsymbol{\theta}} \left[ -\frac{1}{2\sigma^2} \sum_{i=1}^{n} \left( y^{(i)} - \boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}) \right)^2 - \frac{n}{2} \ln \sigma^2 - \frac{n}{2} \ln 2\pi \right]
\end{aligned}$$

To maximize the likelihood, the derivative of the previous equation with respect to $\boldsymbol{\theta}$ needs to be calculated to yield:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \ln p(\mathbf{y}|\mathbf{X}; \boldsymbol{\theta}, \sigma^2) &= \nabla_{\boldsymbol{\theta}} \left[ -\frac{1}{2\sigma^2} \sum_{i=1}^{n} \left( y^{(i)} - \boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}) \right)^2 - \frac{n}{2} \ln \sigma^2 - \frac{n}{2} \ln 2\pi \right] \\
&= \frac{1}{\sigma^2} \sum_{i=1}^{n} \left( y^{(i)} - \boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}) \right) \Phi(\mathbf{x}^{(i)})
\end{aligned}$$

This is then set to zero to calculate the maximum point:

$$\begin{aligned}
\sum_{i=1}^{n} y^{(i)} \Phi(\mathbf{x}^{(i)}) - \sum_{i=1}^{n} \left[ \boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}) \right] \Phi(\mathbf{x}^{(i)}) &= 0 \\
\sum_{i=1}^{n} y^{(i)} \Phi(\mathbf{x}^{(i)}) - \sum_{i=1}^{n} \left[ \Phi(\mathbf{x}^{(i)}) \Phi(\mathbf{x}^{(i)})^T \right] \boldsymbol{\theta} &= 0 \\
\boldsymbol{\Phi}^T(\mathbf{X}) \mathbf{y} - \boldsymbol{\Phi}^T(\mathbf{X}) \boldsymbol{\Phi}(\mathbf{X}) \boldsymbol{\theta} &= 0
\end{aligned}$$

with $\boldsymbol{\Phi}(\mathbf{X}) \in \mathbb{R}^{n \times k}$ being the matrix of the $k$-features evaluated at each data point. Solving the so-called normal equations yields:

$$\begin{aligned}
\boldsymbol{\Phi}^T(\mathbf{X}) \mathbf{y} &= \boldsymbol{\Phi}^T(\mathbf{X}) \boldsymbol{\Phi}(\mathbf{X}) \boldsymbol{\theta} \\
\boldsymbol{\theta} &= \left( \boldsymbol{\Phi}^T(\mathbf{X}) \boldsymbol{\Phi}(\mathbf{X}) \right)^{-1} \boldsymbol{\Phi}^T(\mathbf{X}) \mathbf{y}
\end{aligned}$$

On a slight digression, maximizing the natural logarithm of the likelihood is the same as performing the following minimization problem:

$$\min_{\boldsymbol{\theta}} \left[ \frac{1}{2} \sum_{i=1}^{n} \left( y^{(i)} - \boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}) \right)^2 \right] \tag{2.3}$$

Equation 2.3 is the minimization performed by most practitioners in the

field, namely the least squares problem. To minimize Equation 2.3 the derivative of the function with respect to the free parameters need to be calculated. After some algebraic manipulations, one recognizes that this yields the same solution for the parameters as in maximum likelihood (i.e., $\boldsymbol{\theta} = \left(\boldsymbol{\Phi}^T(\mathbf{X})\boldsymbol{\Phi}(\mathbf{X})\right)^{-1}\boldsymbol{\Phi}^T(\mathbf{X})\mathbf{y}$). Therefore, one deduces that minimizing the sum of squared error is yet another special case of maximum likelihood under a Gaussian noise likelihood assumption.

It is worth noting, that reaching the above solution also assumed the presence of a linear combination of features. In case the feature combination was nonlinear then an exact solution is unattainable. In such scenarios, one resorts to gradient based techniques, such as gradient descent to solve the minimization problem. Next, the gradient descent algorithm is briefly explained.

**Gradient Descent** The main steps of gradient decent are shown in Algorithm 2. The main point to note in the algorithm is choice of the line-

**Algorithm 1** Main steps of Gradient Decent (GD) algorithm.

1: **Requires:** Function to be minimized, initial parameters guess (e.g., $\boldsymbol{\theta} = \mathcal{N}(0, \sigma^2\mathbf{I})$)
2: **while** parameters did not converge **do**
3: Update parameters in a line-search direction
4: **end while**
5: **Output:** Final parameter values.

search direction. One typically descends in the direction of the gradient. Again this direction is not a heuristics. On the contrary, there is a rigid mathematical derivation behind this line-search direction, see [75]. For the ease of discussion, consider the problem of minimizing $\min_{\boldsymbol{\theta}} \Psi_{\boldsymbol{\theta}}(\mathbf{x})$, where $\Psi_{\boldsymbol{\theta}}(\mathbf{x})$ could, for instance, be the sum of squared errors function. If one performs a Taylor expansion around a specific point $\mathbf{x}^{(i)}$, the following could be calculated:

$$\Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)} + \zeta\mathbf{p}) = \Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + \zeta\mathbf{p}^T\nabla_{\boldsymbol{\theta}}\Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + \mathcal{O}(||\zeta||)$$

where $\mathbf{p}$ is a direction in which the point is perturbed from $\mathbf{x}^{(i)}$ and $\zeta$ is amount of that perturbation in the $\mathbf{p}$ direction. Therefore, the direction

of change is $\mathbf{p}^T \nabla_{\boldsymbol{\theta}} \Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$. To minimize $\Psi_{\boldsymbol{\theta}}$, one needs to minimize:

$$\min_{\mathbf{p}} \mathbf{p}^T \nabla_{\boldsymbol{\theta}} \Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$$
$$\text{subject to } \; ||\mathbf{p}|| = 1$$

But,

$$\mathbf{p}^T \nabla_{\boldsymbol{\theta}} \Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = ||\mathbf{p}|| \; ||\nabla_{\boldsymbol{\theta}} \Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})|| \cos(\alpha)$$

where $\alpha$ is the angle between $\mathbf{p}$ and $\nabla_{\boldsymbol{\theta}} \Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$. It is clear that the minimum is when $\cos(\alpha) = -1$, leading to:

$$\mathbf{p} = -\frac{\nabla_{\boldsymbol{\theta}} \Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})}{||\nabla_{\boldsymbol{\theta}} \Psi_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})||}$$

It is for this specific reason, that the gradient direction is used in gradient descent. Please note that if the goal was the maximization of a function, the same derivation could be repeated to reach the positive of the gradient as the line search direction.

Although widely used, maximum likelihood might face some problems depending on the chosen features. Namely, if the features used in the model are highly more complex than the original latent function, over-fitting occurs. On the other hand, if the chosen features were less complex than the latent function, maximum likelihood under-fits.

### Maximum a posteriori estimate

To deal with the problem of over-fitting, one idea is to regularize the learning model. For that, a more Bayesian treatment of regression is needed, see [13]. Bayesian inference is a technique to describe evidence about a certain process using beliefs. More specifically, in Bayesian statistics there exists three main ingredients: (1) prior, (2) likelihood, and (3) posterior. The first reflects the beliefs of the designer. The second corresponds to the normal likelihood function which is analogous to the one described in the previous section. These two ingredients are then used to determine the third (i.e., the posterior). Calculating the posterior is done according to:

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

After this calculation, the posterior reflects all the designer beliefs about the approximated model. This posterior is then maximized to attain the so-called maximum a posteriori (MAP) estimate.

**Example 3** [Deriving Regularized Least Squares] As mentioned before, over-fitting can be leveraged through regularization. Starting from certain assumptions and using a more Bayesian treatment of regression, the regularized least squares problem can be derived. For a Bayesian treatment of regression, three necessary ingredients are required. The first is the prior over the model's free parameters. To determine the regularized least squares problem[3], a zero-mean isotropic Gaussian is chosen. Formally, the prior over the free parameters is given by: $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|0, \boldsymbol{\Sigma}_p)$, with $\boldsymbol{\Sigma}_p$ being the covariances of the free parameters of the model. Second, the likelihood is again a Gaussian given by Equation 2.2. Using this information, the posterior (i.e., the third Bayesian ingredient) is calculated as follows:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}; \sigma^2) = \prod_{i=1}^{n} \mathcal{N}\left(y^{(i)}|\boldsymbol{\theta}^T \Phi(\mathbf{x}^{(i)}), \sigma^2\right) \qquad \boxed{\text{Likelihood function}}$$

$$p(\boldsymbol{\theta}; \alpha) = \mathcal{N}\left(\boldsymbol{\theta}|0, \boldsymbol{\Sigma}_p\right) \qquad \boxed{\text{Prior}}$$

According to the Bayes rule, writing only the terms that depend on the free parameters $\boldsymbol{\theta}$, and *completing* the squares, the posterior is:

$$p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) \propto \exp\left( -\frac{1}{2\sigma^2}\left(\mathbf{y} - \boldsymbol{\theta}^T \boldsymbol{\Phi}(\mathbf{X})\right)^T \left(\mathbf{y} - \boldsymbol{\theta}^T \boldsymbol{\Phi}(\mathbf{X})\right)\right) \exp\left( -\frac{1}{2}\boldsymbol{\theta}^T \boldsymbol{\Sigma}_p^{-1} \boldsymbol{\theta}\right)$$

$$\propto \exp\left( -\frac{1}{2}\left(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}\right)^T \left(\frac{1}{\sigma^2}\boldsymbol{\Phi}(\mathbf{X})\boldsymbol{\Phi}(\mathbf{X})^T + \boldsymbol{\Sigma}_p^{-1}\right)\left(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}\right)\right)$$

with $\bar{\boldsymbol{\theta}} = \sigma^{-2}\left(\sigma^{-2}\boldsymbol{\Phi}(\mathbf{X})\boldsymbol{\Phi}^T(\mathbf{X}) + \boldsymbol{\Sigma}_p^{-1}\right)^{-1} \boldsymbol{\Phi}(\mathbf{X})\mathbf{y}$. Furthermore, notice that the posterior of the free parameters is a Gaussian with:

$$p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) \sim \mathcal{N}\left(\frac{1}{\sigma^2} S^{-1} \boldsymbol{\Phi}(\mathbf{X})\mathbf{y}, S^{-1}\right)$$

where $S = \sigma^{-2}\boldsymbol{\Phi}(\mathbf{X})\boldsymbol{\Phi}^T(\mathbf{X}) + \boldsymbol{\Sigma}_p^{-1}$. Regularized least squares could be

[3]Here regularization corresponds to the second norm regularization of the model free parameters.

attained as follows:

$$p(\boldsymbol{\theta}|\mathbf{X},\mathbf{y}) \propto \exp\left(-\frac{1}{2\sigma^2}\left(\mathbf{y}-\boldsymbol{\theta}^T\boldsymbol{\Phi}(\mathbf{X})\right)^T\left(\mathbf{y}-\boldsymbol{\theta}^T\boldsymbol{\Phi}(\mathbf{X})\right)\right)\exp\left(-\frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\theta}\right)$$

$$\ln p(\boldsymbol{\theta}|\mathbf{X},\mathbf{y}) = \ln\left[\exp\left(-\frac{1}{2\sigma^2}\left(\mathbf{y}-\boldsymbol{\theta}^T\boldsymbol{\Phi}(\mathbf{X})\right)^T\left(\mathbf{y}-\boldsymbol{\theta}^T\boldsymbol{\Phi}(\mathbf{X})\right)\right)\exp\left(-\frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\theta}\right)\right]$$

$$= -\frac{1}{2\sigma^2}\left(\mathbf{y}-\boldsymbol{\theta}^T\boldsymbol{\Phi}(\mathbf{X})\right)^T\left(\mathbf{y}-\boldsymbol{\theta}^T\boldsymbol{\Phi}(\mathbf{X})\right) - \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\theta}$$

Please note the slight abuse of notation. The $\propto$ was switched into an equal sign as the normalization term (i.e., $p(\mathbf{y}|\mathbf{X})$) plays no role in fitting the weights. In case of identical parameters for the prior and using the sum notation the above could be re-written as:

$$\ln p(\boldsymbol{\theta}|\mathbf{X},\mathbf{y}) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y^{(i)}-\boldsymbol{\theta}^T\Phi(\mathbf{x}^{(i)})\right)^2 - \frac{1}{2\alpha}\boldsymbol{\theta}^T\boldsymbol{\theta} + \text{constant}$$

Performing the maximum of the posterior (i.e., MAP) with respect to the free parameters, $\boldsymbol{\theta}$, leads to:

$$\boldsymbol{\theta} = \left(\lambda\mathbf{I} + \boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{y}$$

with,

$$\lambda = \frac{\sigma^2}{\alpha}, \quad \text{and} \quad \boldsymbol{\Phi} = \begin{bmatrix} \text{—} & \Phi(\mathbf{x}^{(1)}) & \text{—} \\ \text{—} & \Phi(\mathbf{x}^{(2)}) & \text{—} \\ \vdots & \vdots & \vdots \\ \text{—} & \Phi(\mathbf{x}^{(n)}) & \text{—} \end{bmatrix}.$$

Here again a linear function of the features was assumed. In case of nonlinearity a closed form solution can not be attained and therefore, algorithms such as Gradient Decent are adopted. For a detailed discussion of the topic, the reader is referred to [75].

It is clear from the previous discussion and examples that parametric regression techniques might face a lot of problems especially when it comes to complex applications. In this dissertation the one important focus is to automatically *intertask* mappings between source and target reinforcement learning tasks. A formal treatment of the subject is presented in Chapter 3. In such scenarios, it is a huge burden

on the designer to preset the model to be used. This problem is amplified especially in the high dimensional case. Roughly speaking, an intertask mapping typically relates source and target transitions together. In case the dimensionality of these transitions is in the order of four dimensions and above, plotting the data is impossible and thus, deciding on the type of model to be used is hard. Therefore, the more automated nonparametric techniques are adopted to aid in learning such mappings. Next, an instance of such nonparametric methods, the Gaussian Processes framework and its variants are presented.

## Nonparametric Regression

Nonparametric regression is an extension of parametric techniques, such that the model is inferred from the available data set rather than being predefined by the user. An intuition is to think of having a space that potentially includes all possible latent functions. After observing data, the best function from that space is chosen to maximize a certain statistical estimate.

Consider the parametric regression problem again. The overall process discussed previously can be summarized in the high level schematic of Figure 2.1. In parametric methods, the original dataset distribution is typically mapped into a broad space of different hypotheses (i.e., functions) relating the dependent and independent variables. To choose the best hypothesis describing the sought relation, the weights or free parameters are then fitted. This typically occurs in the weight space describing these parameters. For instance, consider again the least squares problem. In least squares one maximizes the likelihood function with respect to the free parameters. In other words, a search in the parameter space for those maximizing the likelihood is performed. These are then used in the function space for predictions. If only the overall procedure could be performed directly in the function space, the extra loop (i.e., Function space $\rightarrow$ Weight Space $\rightarrow$ Function space) could be eliminated.

To eliminate the "extra" loop, there exists the need to define probability distributions over function spaces. This is where nonparametric techniques such as Gaussian Processes (GPs) come into play. Such distributions allow for inference directly in the function space without the need for the "extra-loop". Next, the details of GPs including all the mathematical derivations needed for regression are explained.

### Gaussian Processes

GPs define probability distributions over functions. To better understand the intuition behind GPs, think of the problem of defining such distributions. Roughly speaking, a function could be defined as an infinite dimensional vector. Therefore, a distribution over an infinite dimensional vector is actually a distribution over a func-

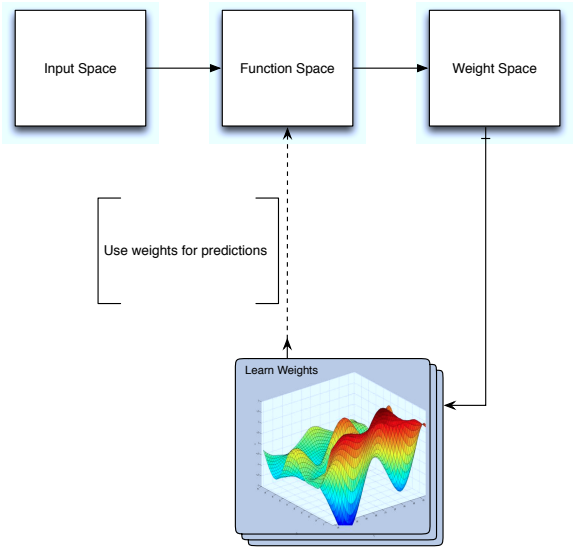



Figure 2.1: Overall high level schematic of parametric regression. The loop has three high level phases: (1) Function space representation, (2) Weight space representation, and (3) learning the weights in the latter space. It is worth noting, that after the weights have been learned, the are then used back again in the function space to perform predictions.

tion. Taking the multivariate Gaussian distribution and extending its mean vector and covariance matrix into infinite dimensions gives rise to new form of distributions. These are not arbitrary and rather have very interesting characteristics. For instance, they define probability distributions over infinite dimensional vectors, which according to the previous intuition are functions. Therefore, with this extension, a new scheme for working directly in the function space, could be developed as detailed next.

Formally, as described in [86], a GP is defined as follows:

**Definition: Gaussian Process** A Gaussian process *is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

A GP is fully specified by its mean and covariance function. The mean function, $m(\mathbf{x})$ and covariance function, $k(\mathbf{x}, \mathbf{x}')$ of a real process $f(\mathbf{x})$ are defined as:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}\left[f(\mathbf{x})\right], \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))\right] \end{aligned}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are two sample points in the sample space $\mathcal{X}$.

If a *function* or a real process is sampled according to a GP, the following notation is used:

$$f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right)$$

According to [86], since GPs are defined as a collection of random variables, this automatically implies a consistency property. This property is also sometimes referred to as the marginalization property. This simply means that if $(y_1, y_2) \sim \mathcal{N}(\mu, \Sigma)$, then it must also specify that $y_1 \sim \mathcal{N}(\mu_1, \Sigma_{1,1})$ with $\Sigma_{1,1}$ being the relevant sub-matrix of $\Sigma$. For the marginalization properties of Gaussian distributions refer to [85]. Please note, that this consistency property is automatically fulfilled if the entries of the covariance matrix are determined through the covariance function.

Usually a constant mean function with $m(\mathbf{x}) = 0$ is used. This is mostly done for simplicity and ease of notation. Furthermore, such an assumption is also reasonable if no prior knowledge about the mean function is known.

The type of functions sampled from a GP depend on the choice of the covariance function. Next, different choices of covariance functions as well as some of their mathematical properties are discussed. For a more thorough and detailed discussion of the topic, the reader is referred to [86].

**Covariance Functions** The choice of the covariance function plays an important role in GP regression and prediction as it encodes the assumptions made by the designer concerning that latent model. The clarifications for this discussion stem back to an implicit assumption made in almost all regression techniques. The notion of similarity plays a crucial role in regression, where it is expected that points lying close to each other are more likely to produce similar outputs and vice versa. In GPs such a similarity is captured through the covariance function. In general, any arbitrary function can not serve the role of a covariance function. One of the reasons is that a GP is a valid probability distribution. This in turn dictates that the covariance matrix induced by the covariance function to be positive semidefinite (PSD). In this section, different covariance functions used in GPs are surveyed.

A function $k$ that maps pair of inputs $\mathbf{x} \in \chi$, $\mathbf{x}' \in \chi$ into $\mathbb{R}$ is called a kernel. For a kernel to be deemed as a valid covariance kernel, it has to be Positive Semi Definite (PSD).

There are different types of covariance kernels that can be used in GPs. The differentiation between these types typically depend on the nature of transformation conducted by that function. For instance, a *stationary* covariance function is a function of $\mathbf{x} - \mathbf{x}'$. Thus this function is invariant to translations in the input space. Furthermore, if the covariance function is a function of $|\mathbf{x} - \mathbf{x}'|$ it is called isotropic. On the other hand, if a covariance function depends on $\mathbf{x}$ and $\mathbf{x}'$ through $\mathbf{x}.\mathbf{x}'$, it is

called a *dot product* covariance function. These functions are invariant to rotations in the input space but not to translations.

To asses the validity of different functions to be covariance functions, some definitions are required. A kernel is said to be *symmetric* if $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$. It is clear from the previous definitions that a covariance kernel should be symmetric.

Given a data set $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{n}$ and a kernel, one can now compute the *Gram matrix* $\mathbf{K} \in \mathbb{R}^{n \times n}$ corresponding to that kernel. The entries of $\mathbf{K}$ are computed using $\mathbf{K}_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$. When $k(\cdot, \cdot)$ is a valid covariance kernel, then $\mathbf{K}$ is called the covariance matrix.

At the end, marginalizing out a GP when data points are available will produce a multidimensional Gaussian distribution. Therefore, the covariance matrix induced by the covariance function should satisfy certain properties. Namely, $\mathbf{K} \in \mathbb{R}^{n \times n}$ should be positive semidefinite (PSD). A real matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ is PSD if $Z(\boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{K} \boldsymbol{\beta} \geq 0$ for all $\boldsymbol{\beta} \in \mathbb{R}^n$. If $Z(\boldsymbol{\beta}) = 0$ only when $\boldsymbol{\beta} = \mathbf{0}$ the matrix is called positive definite. Furthermore, a symmetric matrix is PSD if and only if all of its eigenvalues are non-negative. A general Gram matrix corresponding to any kernel need not to be PSD. However, that corresponding to a covariance kernel should be PSD. This automatically enforces an additional condition of being PSD on a general kernel to become a covariance kernel. A kernel is said to be PDS if:

$$\int k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}) f(\mathbf{x}') d\mu(\mathbf{x}) d\mu(\mathbf{x}') \geq 0,$$

for all $f \in L_2(\chi, \mu)$, with $\mu$ being a measure[4].

**Covariance Function Examples** The squared exponential (SE) covariance kernel is one of the widely used stationary and isotropic covariance functions. It delivers very smooth sample functions, that are infinitely differentiable. It has the following form:

$$k_{SE} = \alpha^2 \exp\left( -\frac{r^2}{2\lambda^2} \right) \tag{2.4}$$

where $r = ||\mathbf{x} - \mathbf{x}'||_2$. In Equation 2.4 there are two *hyperparameters* $\alpha$, and $\lambda$. The first, $\alpha$, controls the amplitude, while the second, $\lambda$, controls the lengthscale of variation. It is important to note that the choice of these two hyperparameters dictate the properties of the functions to be sampled from a GP with an SE covariance kernel.

An SE kernel and a mean function fully define a GP. Samples from that GP are different function. As an intuition, one could think of sampling an infinite dimensional vector from an infinite dimensional distribution. This vector is then plotted

[4]Informally speaking, the reader will usually be able to substitute $d\mathbf{x}$ or $p(\mathbf{x})d\mathbf{x}$ for $d\mu(\mathbf{x})$.

to represent a function. In Figures 2.2, different samples from such GPs are shown.[5] The difference between the two figures is the lengthscale used for the SE covariance function. In the first, a relatively large lengthscale of 0.5 was used, while in the second a smaller length scale of 0.05 was adopted. It is clear from the figures that the properties of the functions described by the GP depend on the covariance function hyperparameters.

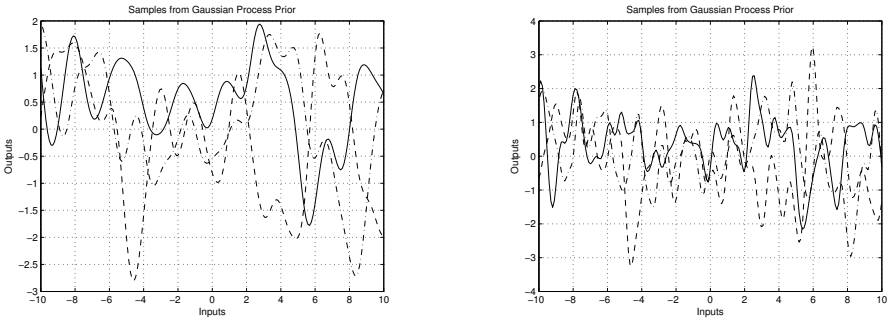



Figure 2.2: Different samples from two different Gaussian processes. Each of the processes makes use of a squared exponential kernel. The difference between the two figure is in the width (i.e., one of the hyperparameters) used for the kernel. In the left figure the width was set to 0.5, while in the second the width was set to 0.02.

The second example of covariance functions that are widely used in GPs are the so-called Matern class of covariance functions. These have the following form:

$$k_{Matern} = \frac{2^{1-\nu}}{\Gamma(\nu)} \left( \frac{\sqrt{2\nu r}}{l} \right)^{\nu} K_{\nu} \left( \frac{\sqrt{2\nu r}}{l} \right)$$

with $\nu$, and $l$ being positive hyperparameters and $K_{\nu}$ being a modified Bessel function [115]. Please note, the scaling is chosen so that for $\nu \to \infty$ the SE covariance function is obtained. The analysis of this function in general cases, is complex and the reader is referred to [86] for a detailed discussion of the topic. Maybe the most

[5]For details on sampling from a Gaussian process, the reader is referred to [85].

interesting cases for machine learning are $\nu = \frac{3}{2}$ and $\nu = \frac{5}{2}$, for which:

$$k_{\nu=\frac{3}{2}} = \left(1 + \frac{\sqrt{3}r}{l}\right) \exp\left(-\frac{\sqrt{3}r}{l}\right)$$
$$k_{\nu=\frac{7}{2}} = \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right)$$

The arguments of why these functions are most suitable for machine learning stems back to practices as described in [86]. It is common for machine learning regressors to discover smooth latent functions. For $\nu = \frac{1}{2}$ the process induced by the Matern covariance kernels, is rough. Moreover, in the absence of explicit prior knowledge (which is the case in most machine learning applications) about the existence of higher order derivatives, it is probably very hard from finite noisy training examples to distinguish between values for $\nu \geq \frac{7}{2}$. Figure 2.3 shows an example of different samples from two different GPs. More examples could be seen in [85]. These include:

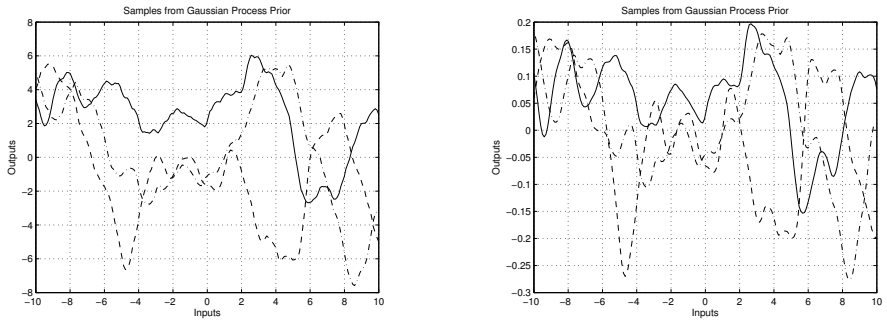



Figure 2.3: Different samples from two different Gaussian processes. The first are samples from a GP using a Matern kernel with $\nu = \frac{3}{2}$, while the second is using a $\nu = \frac{5}{2}$ for the Matern kernel.

1. Anisotropic covariance functions.[6]
2. Non-stationary covariance functions.
3. Dot product covariance functions.

As it is hard to define a valid covariance function, it is generally common to combine two simple functions to generate a more complex one [86]. This should be done using

[6]These are functions of a norm, e.g., $||\mathbf{x} - \mathbf{x}'||_M^2 = (\mathbf{x} - \mathbf{x}')^T \mathbf{M}(\mathbf{x} - \mathbf{x}')$, where $\mathbf{M}$ is some PSD matrix.

one or a combination of the following properties,

$$
\begin{aligned}
k(\mathbf{x},\mathbf{x}') &= ck_1(\mathbf{x},\mathbf{x}') \\
k(\mathbf{x},\mathbf{x}') &= k_1(\mathbf{x},\mathbf{x}') + k_2(\mathbf{x},\mathbf{x}') \\
k(\mathbf{x},\mathbf{x}') &= k_1(\mathbf{x},\mathbf{x}').k_2(\mathbf{x},\mathbf{x}') \\
k(\mathbf{x},\mathbf{x}') &= f(\mathbf{x}).k_1(\mathbf{x},\mathbf{x}')f(\mathbf{x}') \\
k(\mathbf{x},\mathbf{x}') &= q(k_1(\mathbf{x},\mathbf{x}')) \\
k(\mathbf{x},\mathbf{x}') &= \exp(k_1(\mathbf{x},\mathbf{x}')) \\
k(\mathbf{x},\mathbf{x}') &= \mathbf{x}^T\mathbf{A}\mathbf{x} \\
k(\mathbf{x},\mathbf{x}') &= k_3(\Xi(\mathbf{x}),\Xi(\mathbf{x}'))
\end{aligned}
$$

where, $k_1(\mathbf{x},\mathbf{x}')$ and $k_2(\mathbf{x},\mathbf{x}')$ are two valid covariance kernels, $c > 0$ , $f(.)$ can be any function, $q(.)$ is a polynomial with nonnegative coefficients, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\Xi$ maps $\mathbf{x}$ to a space $\mathbb{R}^M$ and $k_3(.,.)$ is a valid covariance kernel in $\mathbb{R}^M$.

Any regression framework includes two phases. In the first, the parameters are inferred from the given data points, while in the second, these parameters are used to perform predictions on unobserved data. This section is dedicated to the explanation of these two steps in a GP setting. However, for the ease of understanding these two steps are described in reverse order.

**Gaussian Process Prediction:** The question is to determine a latent function $f$ that best fits data points of some available real-process. It is also assumed that the observations are noisy and therefore, the noise $\epsilon$, is sampled from a Gaussian distribution with a zero mean and a variance of $\sigma_n^2$ (i.e., $\epsilon \sim \mathcal{N}(0,\sigma_n^2)$).

The training data set comprising $n$ input points with their corresponding observations is denoted by $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^n$, where $\mathbf{x} \in \mathbb{R}^d$ is the $d$-dimensional input vector and $y \in \mathbb{R}$ is the observation value. For the noise-free case, the real function value is known and therefore, $\{\mathbf{x}^{(i)}, f^{(i)}\}_{i=1}^n$ is accessible. For now the noise-free case is considered and the noisy case will be detailed later. To simplify the notation, all training inputs (i.e., $\{\mathbf{x}^{(i)}\}_{i=1}^n$) are collected in the so-called design matrix $\mathbf{X} \in \mathbb{R}^{d\times n}$. Also the test inputs (i.e., $\{\mathbf{x}_\star^{(j)}\}_{j=1}^{n_\star}$) are collected in the test matrix $\mathbf{X}_\star \in \mathbb{R}^{d\times n_\star}$.[7]

As described in [86], the key in GP is to recognize that the *joint* distribution of the training outputs $\mathbf{f} = \{f^{(i)}\}_{i=1}^n$ and the test outputs $\mathbf{f}_\star = \{f_\star^{(j)}\}_{j=1}^{n_\star}$ are sampled

[7]It is interesting to see that the test and training set are assumed to be in the same feature space. This is the case in all regression algorithms. Unfortunately, this is not the case in many real-world applications. This problem was one of the motivations behind transfer learning in supervised learning tasks.

according to a multivariate normal distribution as:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_\star \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}(\mathbf{X},\mathbf{X}) & \mathbf{K}(\mathbf{X},\mathbf{X}_\star) \\ \mathbf{K}(\mathbf{X}_\star,\mathbf{X}) & \mathbf{K}(\mathbf{X}_\star,\mathbf{X}_\star) \end{bmatrix}\right) \tag{2.5}$$

with $\mathbf{K}(\mathbf{X},\mathbf{X}) \in \mathbb{R}^{n\times n}$ being the $n\times n$ Gram matrix formed by applying the covariance function to the training inputs (i.e., $K_{ij} = k(\mathbf{x}^{(i)},\mathbf{x}^{(j)})$), and $\mathbf{K}(\mathbf{X},\mathbf{X}_\star) \in \mathbb{R}^{n\times n_\star}$ is the covariance matrix formed between the training and test points. In details:

$$\mathbf{K}(\mathbf{X},\mathbf{X}) = \begin{bmatrix} \mathbf{k}(\mathbf{x}^{(1)},\mathbf{x}^{(1)}) & \mathbf{k}(\mathbf{x}^{(1)},\mathbf{x}^{(2)}) & \dots & \mathbf{k}(\mathbf{x}^{(1)},\mathbf{x}^{(n)}) \\ \mathbf{k}(\mathbf{x}^{(2)},\mathbf{x}^{(1)}) & \mathbf{k}(\mathbf{x}^{(2)},\mathbf{x}^{(2)}) & \dots & \mathbf{k}(\mathbf{x}^{(2)},\mathbf{x}^{(n)}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{k}(\mathbf{x}^{(n)},\mathbf{x}^{(1)}) & \mathbf{k}(\mathbf{x}^{(n)},\mathbf{x}^{(2)}) & \dots & \mathbf{k}(\mathbf{x}^{(n)},\mathbf{x}^{(n)}) \end{bmatrix} \tag{2.6}$$

To perform predictions on the test data, the joint probability of Equation 2.5 has to be conditioned on the known information. More specifically, the joint probability of $\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_\star \end{bmatrix}$ has to be transformed to $\mathbf{f}_\star|\mathbf{f},\mathbf{X},\mathbf{X}_\star$. The distribution is a Gaussian and therefore conditioning is not hard, since if:

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_\mathbf{x} \\ \boldsymbol{\mu}_\mathbf{y} \end{bmatrix}, \begin{bmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^T & \mathbf{B} \end{bmatrix}\right), \tag{2.7}$$

then conditioning goes as follows:

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}\left(\boldsymbol{\mu}_\mathbf{x} + \mathbf{C}\mathbf{B}^{-1}(\mathbf{y}-\boldsymbol{\mu}_\mathbf{y}), \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^T\right) \tag{2.8}$$

From Equations 2.7 and 2.8, predictions in the *noise-free* GP setting are done according to:

$$\mathbf{f}_\star|X_\star,X,\mathbf{f} \sim \mathcal{N}\Big(\mathbf{K}(X_\star,X)\mathbf{K}(X,X)^{-1}\mathbf{f}, \\ \mathbf{K}(X_\star,X_\star) - \mathbf{K}(X_\star,X)\mathbf{K}(X,X)^{-1}\mathbf{K}(X,X_\star)\Big)$$

The previous result could be easily extended to the noisy case, as discussed in [85]. Assuming additive i.i.d Gaussian noise, $\epsilon$, with a variance $\sigma_n^2$, the prior on the noisy observation becomes:

$$cov(y^{(p)},y^{(q)}) = k(\mathbf{x}^{(p)},\mathbf{x}^{(q)}) + \sigma_n^2\delta_{pq} \text{ or } cov(\mathbf{y}) = \mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I}$$

where $\langle y^{(p)}, y^{(q)} \rangle \in \mathbb{R}^2$ are two output noisy observations, and $\langle \mathbf{x}^{(p)}, \mathbf{x}^{(q)} \rangle \in \mathbb{R}^{d \times d}$ are two input vectors. Finally, $\delta_{pq}$ represents the Kronicker delta function:

$$\delta_{pq} = \begin{cases} 1 & \text{if p=q} \\ 0 & \text{otherwise} \end{cases}$$

With this in mind, the joint distribution of the *noisy* outputs and the test outputs is calculated as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_\star \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I} & \mathbf{K}(\mathbf{X},\mathbf{X}_\star) \\ \mathbf{K}(\mathbf{X}_\star,\mathbf{X}) & \mathbf{K}(\mathbf{X}_\star,\mathbf{X}_\star) \end{bmatrix}\right) \tag{2.9}$$

The predictive distribution can now be derived in a similar manner to the noise-free case by using Equation 2.7 and 2.8, to reach:

$$\begin{aligned} \mathbf{f}_\star|\mathbf{X}_\star,\mathbf{X},\mathbf{y} &\sim \mathcal{N}(\bar{\mathbf{f}}_\star, cov[\mathbf{f}_*]) \text{ where} \\ \bar{f}_\star &= \mathbf{K}(\mathbf{X}_\star,\mathbf{X})(\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_n^2)^{-1}\mathbf{y} \\ cov[\mathbf{f}_*] &= \mathbf{K}(\mathbf{X}_\star,\mathbf{X}_\star) - \mathbf{K}(\mathbf{X}_\star,\mathbf{X})(\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I})^{-1}\mathbf{K}(\mathbf{X},\mathbf{X}_\star)) \end{aligned}$$

It is clear that the covariance does not depend on the outputs. The first term (i.e., $\mathbf{K}(\mathbf{X}_\star,\mathbf{X}_\star)$) is called the prior variance, while the second term (i.e., $\mathbf{K}(\mathbf{X}_\star,\mathbf{X})(\mathbf{K}(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I})^{-1}\mathbf{K}(\mathbf{X},\mathbf{X}_\star)$) includes all information that $\mathbf{X}$ and the noise term induces. On the contrary, the mean function is linearly dependent on the outputs and is called a linear predictor.

**Model Selection:** The details of this section are adopted from [86]. Although GPs[8] are considered nonparametric techniques, they still have certain parameters that need to be inferred from the available data. Please note, that the main distinction between parametric and nonparametric techniques reside in the type of distributions defined. Namely, in the former the probability distributions are defined over the parameter space, while in the latter these are defined directly in the function space. As mentioned before, there are two phases in GPs: (1) model selection, and (2) prediction. In the previous section the prediction phase was detailed. Given a certain mean and covariance function, predictions on unobserved data can now be calculated. In this section, the first phase, in which inference about the hyperparameters of the model is performed, will be detailed including all necessary mathematical formalizations.

To clarify, consider a GP making use of the following covariance function, which

[8]Parts of these sections are taken from Advanced Mathematics for Engineers lecture at University of Applied Sciences Ravensburg-Weingarten, with the script written by Wolfgang Ertel, Markus Schneider, and Haitham Bou Ammar.

is a combination of a squared exponential together with a polynomial, a noise, and a constant term:

$$k(\mathbf{x}^{(p)}, \mathbf{x}^{(q)}) = \sigma_f^2 \exp\left(-\frac{1}{2}||\mathbf{x}^{(p)} - \mathbf{x}^{(q)}||^2_{\mathbf{M}}\right) + \sigma_p^2 \mathbf{x}^{(p)T} \mathbf{\Sigma} \mathbf{x}^{(q)} + \sigma_n^2 \delta_{pq} + \sigma_c^2 \quad (2.10)$$

The hyperparameters for this model are concatenated in $\boldsymbol{\theta} = \left[\{\mathbf{M}\}, \{\mathbf{\Sigma}\}, \sigma_f^2, \sigma_p^2, \sigma_n^2, \sigma_c^2\right]^T$. Fitting the hyperparameters as well as the choice of the covariance function to be used is called the model-selection problem. The ordinary Bayesian inference works in two stages: (1) compute the posterior distribution, and (2) evaluate all statements under the posterior. The posterior can be calculated using the Bayes rule as follows:

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}$$

The marginal likelihood plays a very important role in model selection. This could be interpreted as a priory predictive distribution for the model without having seen any data. It is very similar to likelihood, but the parameters have been integrated out. After attaining a data set, the marginal likelihood can measure the probability of generating that data set under the model with parameters randomly sampled from the prior. It follows, that if a model can explain a lot of different datasets it will have only a small marginal likelihood on each individual set. In contrast, a very simple model that can only describe a very small range of data sets will have a high marginal likelihood if one of these is actually observed. This effect is often referred to as *Occam's razor* principle.

To determine the marginal likelihood the following needs to be computed:

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X}) d\mathbf{f} \quad (2.11)$$

Solving this integral in general is hard and might even be intractable. Under the Gaussian process model the prior is Gaussian, $\mathbf{f}|\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}))$, and the likelihood is a factorized Gaussian with $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 \mathbf{I})$. Making use of the following property for the product of two Gaussians:

$$\mathcal{N}(\mathbf{x}|\mathbf{a}, \mathbf{A})\mathcal{N}(\mathbf{x}|\mathbf{b}|\mathbf{B}) = \mathcal{N}(\mathbf{x}|\mathbf{c}, \mathbf{C}), \text{ with}$$
$$\mathbf{c} = \mathbf{C}\left(\mathbf{A}^{-1}\mathbf{a} + \mathbf{B}^{-1}\mathbf{b}\right) \text{ and } \mathbf{C} = \left(\mathbf{A}^{-1} + \mathbf{B}^{-1}\right)^{-1},$$

and the normalizing term, $Z^{-1}$, is itself a Gaussian with the following form:

$$Z^{-1} = (2\pi)^{-\frac{d}{2}} |\mathbf{A} + \mathbf{B}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{a}-\mathbf{b})^T(\mathbf{A}+\mathbf{B})^{-1}(\mathbf{a}-\mathbf{b})\right),$$

the integral of Equation 2.11 can now be evaluated according to:

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}) &= \int p(\mathbf{y}|\mathbf{f},\mathbf{X})p(\mathbf{f}|\mathbf{X})d\mathbf{f} \\ &= \int \mathcal{N}(\mathbf{f},\sigma_n^2\mathbf{I})\mathcal{N}(\mathbf{0},\mathbf{K}(\mathbf{X},\mathbf{X}))d\mathbf{f} \\ &= (2\pi)^{-\frac{n}{2}} |\mathbf{K}(\mathbf{X},\mathbf{X})) + \sigma_n^2|^{-\frac{1}{2}} \exp\left(\mathbf{y}^T(\mathbf{K}(\mathbf{X},\mathbf{X})) + \sigma_n^2)^{-1}\mathbf{y}\right) \end{aligned}$$

By taking the natural logarithm of the previous equation, the marginal likelihood reads as:

$$\log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T\left(\mathbf{K}(X,X) + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}(X,X) + \sigma_n^2\mathbf{I}| - \frac{n}{2}\log 2\pi \tag{2.12}$$

As detailed in [86], the first term in Equation 2.12 is called the data fit, while the second is called the complexity penalty. It is interesting to see that the latter automatically avoids over-fitting since it works as a regularization factor in the equation. Maximizing Equation 2.12 is often referred to as empirical Bayes, evidence optimization or type II maximum likelihood estimation. In order to maximize the marginal likelihood, the partial derivatives with respect to the hyperparameters should be derived. For that, two interesting properties from matrix derivatives are needed, see [85] for further details:

$$\begin{aligned} \frac{\partial}{\partial\theta}\mathbf{K}^{-1} &= -\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta} \\ \frac{\partial}{\partial\theta}\log|\mathbf{K}| &= \mathrm{tr}\left(\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta}\right) \end{aligned}$$

The derivative of the marginal likelihood can be calculated as:

$$\begin{aligned}\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) &= -\frac{1}{2}\frac{\partial}{\partial\theta_j}\left[\mathbf{y}^T\left(\mathbf{K}(X,X)+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right] - \frac{1}{2}\frac{\partial}{\partial\theta_j}\left[\log|\mathbf{K}(X,X)+\sigma_n^2\mathbf{I}|\right] \\ &\quad - \frac{n}{2}\frac{\partial}{\partial\theta_j}[\log 2\pi] \\ &= -\frac{1}{2}\frac{\partial|\mathbf{K}(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I}|}{\partial\theta_j} - \frac{1}{2}\mathbf{y}^T\frac{\partial\left(\mathbf{K}(X,X)+\sigma_n^2\mathbf{I}\right)^{-1}}{\partial\theta_j}\mathbf{y} \\ &= -\frac{1}{2}\mathrm{tr}\left(\mathbf{K}_\mathbf{y}^{-1}\frac{\partial\mathbf{K}_\mathbf{y}}{\partial\theta_j}\right) + \frac{1}{2}\mathbf{y}^T\mathbf{K}_\mathbf{y}^{-1}\frac{\partial\mathbf{K}_\mathbf{y}}{\partial\theta_j}\mathbf{K}_\mathbf{y}^{-1} \\ &= \frac{1}{2}\mathrm{tr}\left(\mathbf{y}^T\mathbf{K}_\mathbf{y}^{-1}\frac{\partial\mathbf{K}_\mathbf{y}^{-1}}{\partial\theta_j}\mathbf{K}_\mathbf{y}^{-1}\mathbf{y} - \mathbf{K}_\mathbf{y}^{-1}\frac{\partial\mathbf{K}_\mathbf{y}^{-1}}{\partial\theta_j}\right) \\ &= \frac{1}{2}\mathrm{tr}\left(\left[\boldsymbol{\alpha}^T\boldsymbol{\alpha} - \mathbf{K}_\mathbf{y}^{-1}\right]\frac{\partial\mathbf{K}_\mathbf{y}^{-1}}{\partial\theta_j}\right)\end{aligned}$$

where $\boldsymbol{\alpha} = \mathbf{K}_\mathbf{y}^{-1}\mathbf{y}$, $\mathbf{K}_\mathbf{y} = \mathbf{y}^T\left(\mathbf{K}(X,X)+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}$, and tr is the trace operator. These derivative can be used in gradient ascent-like algorithms to maximize the log marginal likelihood function. In GPs conjugate gradient decent (CGD) is used to solve for the optimization of the hyperparameters[9]. For details on CGD, the reader is referred to [75].

**Computational Complexity of Gaussian Processes:** Although appealing, GPs are not a free-lunch in machine learning. The main problems in GPs arise with the increase in the number of available data points. It is clear from the previous discussion that learning the hyperparameters involve inverting the covariance matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$, where $n$ is the number of data points, at each iteration of the gradient-based algorithm. Unfortunately, inverting this covariance matrix incurs a cost of $\mathcal{O}(n^3)$, making them unsuitable for real-world applications. Furthermore, after learning the hyperparameters and given $n_\star$ test points, predictions incur an extra total cost of $\mathcal{O}(n^2 n_\star)$.

As mentioned before, GPs will be adopted to learn a specific form of intertask mappings relating source and target reinforcement learning tasks, see Chapter 5. With these computational problems, GPs are not well suited for such a scenario. Therefore, a more more efficient variant of GPs, the *Sparse Pseudo-Inputs Gaussian Processes*, are used. These are detailed next.

[9]Other methods such as, Expectation Maximization, Laplace Approximation or Markov Chain Monte Carlo exist. One drawback is that these methods are slower than the aforementioned maximization approach.

### Sparse Pseudo-Input Gaussian Processes

There are different approaches aiming at making GPs more efficient. For a detailed discussion, the reader is referred to [81, 86]. On a high level, all these methods try to approximate the covariance matrix using low rank matrices as:

$$\mathbf{K}(\mathbf{X}, \mathbf{X}) = \mathbf{V}\mathbf{V}^T$$

where $\mathbf{V} \in \mathbb{R}^{m \times n}$ is a low rank matrix, with $m << n$. As an intuition, the above could be thought of as the eigenvalue decomposition of the covariance matrix with taking only the first $m$ eigenvectors.[10] Such an approximation reduces the learning cost to $\mathcal{O}(nm^2)$, and that of prediction to $\mathcal{O}(m^2 n_\star)$. According to [92, 93], to create computationally efficient GP models, a detailed analysis of the GP predictive distribution for one data point has to be conducted.

The mean and the covariance of the predictive distribution of a GP can be written as:

$$\mu_\star = \mathbf{K}(\mathbf{X}_\star, \mathbf{X}) \left[\mathbf{K}(\mathbf{X}, \mathbf{X} + \sigma_n^2 \mathbf{I})\right]^{-1}$$
$$\sigma_\star^2 = \mathbf{K}(\mathbf{X}_\star, \mathbf{X}_\star) - \mathbf{K}(\mathbf{X}_\star, \mathbf{X}) \left[\mathbf{K}(\mathbf{X}, \mathbf{X} + \sigma_n^2 \mathbf{I})\right]^{-1} \mathbf{K}(\mathbf{X}, \mathbf{X}_\star) + \sigma_n^2$$

If the hyperparameters are determined and learned, these functions could be thought of as being parameterized by the locations of the $n$ training points (i.e., $\mathbf{X}$ and $\mathbf{y}$). As mentioned in [93], the intuition behind Sparse Pseudo Input Gaussian Processes (SPGPs) is to replace the original data set by a pseudo data set, and then use the GP predictive distribution from this pseudo data set as the parameterized model likelihood. Therefore, the output for one data point is given by:

$$p(y|\mathbf{x}, \bar{\mathbf{X}}, \bar{\mathbf{f}}) = \mathcal{N}\Big(\mathbf{K}(\mathbf{x}, \bar{\mathbf{X}})\mathbf{K}^{-1}(\bar{\mathbf{X}}, \bar{\mathbf{X}})\bar{\mathbf{f}}, \\ \mathbf{K}(\mathbf{x}, \mathbf{x}) - \mathbf{K}(\mathbf{x}, \bar{\mathbf{X}})\mathbf{K}^{-1}(\bar{\mathbf{X}}, \bar{\mathbf{X}})\mathbf{K}(\bar{\mathbf{X}}, \mathbf{x}) + \sigma_n^2\Big)$$

with, $\bar{\mathbf{X}} = \{\bar{\mathbf{x}}^{(j)}\}_{j=1}^m$ representing the pseudo-inputs, $\bar{\mathbf{f}} = \{\bar{\mathbf{f}}^{(j)}\}_{j=1}^m$ describing the pseudo-outputs and $m$ is the number of these pseudo-"variables".

Under the assumption that the data is i.i.d., the total likelihood can now be written

[10] Of course eigenvalue decomposition can not be applied to the problem, as it also incurs a cost $\mathcal{O}(n^3)$. The idea is presented for clarification purposes.

as:

$$p(\mathbf{y}|\bar{\mathbf{f}}, \bar{\mathbf{X}}, \mathbf{X}) = \prod_{i=1}^{n} p(y^{(i)}|\bar{\mathbf{f}}, \bar{\mathbf{X}}, \mathbf{X})$$
$$= \mathcal{N}\left(\mathbf{K}(\mathbf{X}, \bar{\mathbf{X}})\mathbf{K}^{-1}(\bar{\mathbf{X}}, \bar{\mathbf{X}})\bar{\mathbf{f}}, \operatorname{diag}\left(\mathbf{K}(\mathbf{X}, \mathbf{X}) - \mathbf{F}(\mathbf{X}, \mathbf{X}) + \sigma_n^2\mathbf{I}\right)\right)$$

with $\mathbf{F}(\mathbf{X}, \mathbf{X}) = \mathbf{K}(\mathbf{X}, \bar{\mathbf{X}})\mathbf{K}^{-1}(\bar{\mathbf{X}}, \bar{\mathbf{X}})\mathbf{K}(\bar{\mathbf{X}}, \mathbf{X})$, and diag representing the diagonal operator.

Furthermore, by assuming a Gaussian prior on the pseudo inputs, one can compute the marginal likelihood by marginalizing out the pseudo outputs to attain:

$$p(\mathbf{y}|\bar{\mathbf{X}}, \mathbf{X}) = \int p(\mathbf{y}|\bar{\mathbf{f}}, \bar{\mathbf{X}}, \mathbf{X})p(\bar{\mathbf{f}})d\bar{\mathbf{f}} \tag{2.13}$$
$$= \mathcal{N}\left(\mathbf{0}, \mathbf{F}(\mathbf{X}, \mathbf{X}) + \operatorname{diag}\left(\mathbf{K}(\mathbf{X}, \mathbf{X}) - \mathbf{F}(\mathbf{X}, \mathbf{X})\right) + \sigma_n^2\mathbf{I}\right) \tag{2.14}$$

According to [93] the SPGP marginal likelihood can be obtained by replacing the full GP likelihood matrix by the low rank covariance matrix $\mathbf{F}(\mathbf{X}, \mathbf{X})$ everywhere except on the diagonals.

Next to determine the predictive distribution two steps are required: (1) determine the joint distribution, and (2) condition the previous on the observed data. The joint is exactly the same as in Equation 2.14, and once conditioned it will deliver:

$$p(y_\star|\mathbf{y}, \mathbf{X}, \bar{\mathbf{X}}) = \mathcal{N}(\mu_\star, \sigma_\star^2)$$
$$\mu_\star = \mathbf{F}(\mathbf{X}_\star, \mathbf{X})\left[\mathbf{F}(\mathbf{X}, \mathbf{X}) + \operatorname{diag}\left(\mathbf{K}(\mathbf{X}, \mathbf{X}) - \mathbf{F}(\mathbf{X}, \mathbf{X})\right) + \sigma_n^2\mathbf{I}\right]^{-1}\mathbf{y}$$
$$\sigma_\star^2 = \mathbf{K}(\mathbf{X}_\star, \mathbf{X}_\star) - \mathbf{F}(\mathbf{X}_\star, \mathbf{X})\left[\mathbf{F}(\mathbf{X}, \mathbf{X}) + \operatorname{diag}\Big(\mathbf{K}(\mathbf{X}, \mathbf{X}) - \mathbf{F}(\mathbf{X}, \mathbf{X})\Big) + \sigma_n^2\mathbf{I}\right]^{-1}$$
$$\mathbf{F}(\mathbf{X}, \mathbf{X}_\star) + \sigma_n^2$$

This could still be written in a more efficient manner using the matrix inversion lemma, see [86, 93]:

$$\left[\mathbf{K}(\mathbf{X}, \bar{\mathbf{X}})\mathbf{K}^{-1}(\bar{\mathbf{X}}, \bar{\mathbf{X}})\mathbf{K}(\bar{\mathbf{X}}, \mathbf{X}) + (\mathbf{\Lambda} + \sigma_n^2\mathbf{I})\right]^{-1} =$$
$$(\mathbf{\Lambda} + \sigma_n^2)^{-1} - (\mathbf{\Lambda} + \sigma_n^2\mathbf{I})^{-1}\mathbf{K}(\mathbf{X}, \bar{\mathbf{X}})\mathbf{B}^{-1}\mathbf{K}(\bar{\mathbf{X}}, \mathbf{X})(\mathbf{\Lambda} + \sigma_n^2\mathbf{I})^{-1}$$

with $\mathbf{\Lambda} = \operatorname{diag}\left(\mathbf{K}(\mathbf{X}, \mathbf{X}) - \mathbf{Q}(\mathbf{X}, \mathbf{X})\right)$, and $\mathbf{B} = \mathbf{K}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) + \mathbf{K}(\bar{\mathbf{X}}, \mathbf{X})(\mathbf{\Lambda} + \sigma_n^2\mathbf{I})^{-1}\mathbf{K}(\mathbf{X}, \bar{\mathbf{X}})$.

Once applied to the predictive distribution the following is attained:

$$\mu_\star = \mathbf{K}(\mathbf{X}_\star, \bar{\mathbf{X}})\mathbf{B}^{-1}\mathbf{K}(\bar{\mathbf{X}}, \mathbf{X})(\mathbf{\Lambda} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}$$
$$\sigma_\star^2 = \mathbf{K}(\mathbf{X}_\star, \mathbf{X}_\star) - \mathbf{K}(\mathbf{X}_\star, \bar{\mathbf{X}})(\mathbf{K}^{-1}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) - \mathbf{B}^{-1})\mathbf{K}(\bar{\mathbf{X}}, \mathbf{X}_\star) + \sigma_n^2$$

**Model Selection:** After having the predictive distribution of SPGP, now model learning is explained. Model learning here means the determination of both the hyperparameters as well as the locations of the pseudo-inputs. As in the normal GP case, learning these parameters requires the derivatives of the marginal likelihood.

The negative logarithm of the marginal likelihood is given by:

$$-\log p(\mathbf{y}|\mathbf{X}, \bar{\mathbf{X}}) = \frac{1}{2}\log|\mathbf{F}(\mathbf{X}, \mathbf{X}) + \operatorname{diag}(\mathbf{K}(\mathbf{X}, \mathbf{X}) - \mathbf{F}(\mathbf{X}, \mathbf{X}))\,\sigma_n^2\mathbf{I}| + \frac{1}{2}\mathbf{y}^T(\mathbf{Q}(\mathbf{X}, \mathbf{X}) + \operatorname{diag}(\mathbf{K}(\mathbf{X}, \mathbf{X}) - \mathbf{F}(\mathbf{X}, \mathbf{X}))\,\sigma_n^2\mathbf{I}) + \frac{n}{2}\log 2\pi$$

For computational efficient, this could be rewritten as [93]:

$$-\log p(\mathbf{y}|\mathbf{X}, \bar{\mathbf{X}}) = \mathbb{L}_1 + \mathbb{L}_2 + \frac{n}{2}\log 2\pi$$

The first term is:

$$2\mathbb{L}_1 = \log|\mathbf{A}| - \log|\mathbf{K}(\bar{\mathbf{X}}, \bar{\mathbf{X}})| + \log|\mathbf{\Lambda}| + (n-m)\log\sigma_n^2$$

where $\mathbf{A} = \sigma_n^2\mathbf{K}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) + \mathbf{K}(\bar{\mathbf{X}}, \mathbf{X})\mathbf{\Lambda}^{-1}\mathbf{K}(\mathbf{X}, \bar{\mathbf{X}})$, $\sigma_n^2\mathbf{\Lambda} = \operatorname{diag}(\mathbf{K}(\mathbf{X}, \mathbf{X}) - \mathbf{F}(\mathbf{X}, \mathbf{X})) + \sigma_n^2\mathbf{I}$. While the second is given by:

$$2\mathbb{L}_2 = \sigma_n^{-2}\left(||\boldsymbol{\beta}||_2^2 - ||\mathbf{A}^{-\frac{1}{2}}\bar{\mathbf{K}}(\bar{\mathbf{X}}, \mathbf{X})\boldsymbol{\beta}||_2^2\right)$$

where, $\boldsymbol{\beta} = \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{y}$, and $\bar{\mathbf{K}}(\bar{\mathbf{X}}, \mathbf{X}) = \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{K}(\bar{\mathbf{X}}, \mathbf{X})$.

First the derivative with respect to the free parameters including the pseudo-inputs can be derived as:

$$2\dot{\mathbb{L}}_1 = \operatorname{tr}\left(\mathbf{A}^{-\frac{1}{2}}\dot{\mathbf{A}}\mathbf{A}^{-\frac{T}{2}}\right) - \operatorname{tr}\left(\mathbf{K}_M^{-\frac{1}{2}}\dot{\mathbf{K}}_M\mathbf{K}_M^{-\frac{T}{2}}\right) + \operatorname{tr}\left(\mathbf{\Lambda}^{-\frac{1}{2}}\dot{\mathbf{\Lambda}}\mathbf{\Lambda}^{-\frac{1}{2}}\right)$$

and,

$$2\dot{\mathbb{L}}_2 = \sigma_n^2\Bigg[-\frac{1}{2}\boldsymbol{\beta}^T\dot{\bar{\mathbf{K}}}\boldsymbol{\beta} + \left(\mathbf{A}^{-\frac{1}{2}}\mathbf{K}_{mn}^{-}\boldsymbol{\beta}\right)^T\Bigg(-\frac{1}{2}\mathbf{A}^{-\frac{1}{2}}\dot{\mathbf{A}}\mathbf{A}^{-\frac{T}{2}}(\mathbf{A}^{-\frac{1}{2}}\mathbf{K}_{mn}^{-}\boldsymbol{\beta}) - \mathbf{A}^{-\frac{1}{2}}\dot{\mathbf{K}}_{mn}^{-}\boldsymbol{\beta} + \mathbf{A}^{-\frac{1}{2}}\dot{\mathbf{K}}_{mn}^{-}\mathbf{\Lambda}^{-\frac{1}{2}}\dot{\mathbf{\Lambda}}\mathbf{\Lambda}^{-\frac{1}{2}}\boldsymbol{\beta}\Bigg)\Bigg]$$

with $\mathbf{K}_{mn} = \mathbf{K}(\bar{\mathbf{X}}, \mathbf{X})$, and $\dot{\mathbf{K}}_{mn} = \mathbf{\Lambda}^{-\frac{1}{2}}\dot{\mathbf{K}}_{mn}$. According to [93], to complete the derivatives the following are needed:

$$\dot{\mathbf{A}} = \sigma_n^2\dot{\mathbf{K}}_M + 2\text{sym}\left(\dot{\mathbf{K}}_{mn}\mathbf{\Lambda}^{-1}\mathbf{K}_{nm}\right) - \mathbf{K}_{mn}\mathbf{\Lambda}\dot{\mathbf{\Lambda}}\mathbf{\Lambda}^{-1}\mathbf{K}_{nm}$$
$$\dot{\mathbf{\Lambda}} = \sigma_n^{-2}\text{diag}\left(\dot{\mathbf{K}}_{nn} - 2\dot{\mathbf{K}}_{nm}\mathbf{K}_{nm}^{-1}\mathbf{K}_{mn} + \mathbf{K}_{nm}\mathbf{K}_m^{-1}\dot{\mathbf{K}}_m\mathbf{K}_m^{-1}\mathbf{K}_{mn}\right)$$

For details on the derivatives of the covariance functions with respect to the free derivatives see [93]. The derivatives with respect to the noise parameter could be done in the same manner. The derivations are left as an exercise for the reader.

## 2.2 Unsupervised and Deep Learning:

So far learning of a latent function describing a set of inputs and outputs was detailed. Another interesting domain in machine learning, is that when only a set of inputs are given without their labels. The questions in this case is to determine certain patterns, or groups in the data. Opposing supervised learning these algorithms belong to the unsupervised world as no labels are given.[11] Examples of these methods are clustering[12], feature extraction and density estimation algorithms.

### Clustering

Given an unlabeled data set, clustering tries to find patterns or groupings in the data. There are different methods that can successfully perform clustering. In this dissertation, Gaussian Mixture Models (GMMs) are adopted, see [13, 84]. These are detailed next.

**Gaussian Mixture Models:** Given an unlabeled data set $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$, the idea is to model the data according to the following joint model $p\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}\right) = p\left(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}\right)p\left(\mathbf{z}^{(i)}\right)$. In this case, $\mathbf{z}^{(i)}$ is a latent or unobserved random variable that is sampled according to $\mathbf{z}^{(i)} \sim \text{Multinomial}(\boldsymbol{\beta})$, where $\boldsymbol{\beta}^{(j)} \geq 0$, $\sum_{j=1}^{k}\boldsymbol{\beta}^{(j)} = 1$, and $\boldsymbol{\beta}^{(j)}$ gives the probability of $p\left(\mathbf{z}^{(i)} = j\right)$. Furthermore, $\mathbf{x}^{(i)}|\mathbf{z}^{(i)} = j \sim \mathcal{N}\left(\boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right)$, where $\boldsymbol{\mu}^{(j)}$ and $\boldsymbol{\Sigma}^{(j)}$ are the mean vector and the covariance matrix of the Gaussian distribution, respectively. Let $k$ denote the number of values that $\mathbf{z}^{(i)}$ can take on. The model, therefore, posits that each $\mathbf{x}^{(i)}$ was generating by randomly drawing a

[11] Please note, that the transition from supervised to unsupervised learning is not discrete in machine learning. When one makes such a transition, he/she typically encounters the field of semi-supervised learning, which could be thought of as a mixture between supervised and unsupervised learning.

[12] Clustering could also be seen as classification but with the absence of any class labels

$\mathbf{z}^{(i)}$ from $\{1, 2, \ldots, k\}$, and then $\mathbf{x}^{(i)}$ was drawn from one of the $k$ Gaussians. This is called the mixture of Gaussians model. The parameters that of the model are then $\boldsymbol{\beta}$, $\boldsymbol{\mu}$, and $\boldsymbol{\Sigma}$. To estimate these, the logarithmic data likelihood is written as:

$$\begin{aligned}\mathcal{L}\left(\boldsymbol{\beta}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) &= \sum_{i=1}^{n} \log \left[ p\left(\mathbf{x}^{(i)}; \boldsymbol{\beta}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \right] \\ &= \sum_{i=1}^{n} \log \left[ \sum_{\mathbf{z}^{(i)}=1}^{k} p\left(\mathbf{x}^{(i)} | \mathbf{z}^{(i)}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) p\left(\mathbf{z}^{(i)}; \boldsymbol{\beta}\right) \right]\end{aligned}$$

Deriving this equation and trying to find the parameters will not yield a closed form solution. Note, however, that if the value of $\mathbf{z}^{(i)}$ was known, then the maximization becomes easy. The data likelihood can be written as:

$$\begin{aligned}\mathcal{L}\left(\boldsymbol{\beta}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) &= \sum_{i=1}^{n} \log \left[ p\left(\mathbf{x}^{(i)}; \boldsymbol{\beta}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \right] \\ &= \sum_{i=1}^{n} \log \left[ p\left(\mathbf{x}^{(i)} | \mathbf{z}^{(i)}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) p\left(\mathbf{z}^{(i)}; \boldsymbol{\beta}\right) \right] \\ &= \sum_{i=1}^{n} \left[ \log p\left(\mathbf{x}^{(i)} | \mathbf{z}^{(i)}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) + \log p\left(\mathbf{z}^{(i)}; \boldsymbol{\beta}\right) \right]\end{aligned}$$

Maximizing the above equation yields:

$$\begin{aligned}\boldsymbol{\beta}^{(j)} &= \frac{1}{n} \sum_{i=1}^{n} \mathcal{I}\{\mathbf{z}^{(i)} = j\}, \\ \boldsymbol{\mu}^{(j)} &= \frac{\sum_{i=1}^{n} \mathcal{I}\{\mathbf{z}^{(i)} = j\} \mathbf{x}^{(i)}}{\mathcal{I}\{\mathbf{z}^{(i)} = j\}}, \\ \boldsymbol{\Sigma}^{(j)} &= \frac{\sum_{i=1}^{n} \mathcal{I}\{\mathbf{z}^{(i)} = j\} (\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(j)})(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(j)})^T}{\sum_{i=1}^{n} \mathcal{I}\{\mathbf{z}^{(i)} = j\}}\end{aligned}$$

Since the latent variable is not known, the expectation maximization (EM) algorithm is used instead. The main steps of EM are shown in Algorithm 2. In the E-step, the posterior of the parameters, $\mathbf{z}^{(i)}$, given the data and the current setting of the model parameters is calculated as:

$$p\left(\mathbf{z}^{(i)} = j | \mathbf{x}^{(i)}; \boldsymbol{\beta}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{p\left(\mathbf{x}^{(i)} | \mathbf{z}^{(i)} = j; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) p\left(\mathbf{z}^{(i)}; \boldsymbol{\beta}\right)}{\sum_{i=1}^{k} p\left(\mathbf{x}^{(i)} | \mathbf{z}^{(i)} = l; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) p\left(\mathbf{z}^{(i)} = l; \boldsymbol{\beta}\right)}$$

In the M-step maximization to improve the guesses is performed. It is worth noting,

**Algorithm 2** Expectation Maximization

1: **Repeat:**

2: **(E-step):** for each $i$ and $j$ set:

$$w_j^{(i)} = p\left(\mathbf{z}^{(i)} = j | \mathbf{x}^{(i)}; \boldsymbol{\beta}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$$

3: **(M-step):**

$$\boldsymbol{\beta}^{(j)} = \frac{1}{n}\sum_{i=1}^{n} \mathcal{I}\{\mathbf{z}^{(i)} = j\},$$
$$\boldsymbol{\mu}^{(j)} = \frac{\sum_{i=1}^{n} \mathcal{I}\{\mathbf{z}^{(i)} = j\}\mathbf{x}^{(i)}}{\mathcal{I}\{\mathbf{z}^{(i)} = j\}},$$
$$\boldsymbol{\Sigma}^{(j)} = \frac{\sum_{i=1}^{n} \mathcal{I}\{\mathbf{z}^{(i)} = j\}(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(j)})(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(j)})^T}{\sum_{i=1}^{n} \mathcal{I}\{\mathbf{z}^{(i)} = j\}}$$

that EM is a reminiscent of the K-means clustering algorithm, with "soft" assignments [13, 50].

## Deep Learning and Feature Extraction Techniques:

In this section different deep learning and feature extraction methods are detailed. Firstly, sparse coding, an unsupervised learning technique is explained. Secondly, restricted Boltzmann machines and deep belief networks are described.

**Sparse Coding:** Sparse coding is an unsupervised algorithm for finding succinct representation of an unlabeled data set. Given only unlabeled input data, it learns basis functions that captures high-level features in the inputs [10, 57]. Formally, given a data set $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}$, where $\mathbf{x}^{(i)} \in \mathbb{R}^d$, the question is to represent $\mathbf{x}^{(i)}$ as a combination of basis vectors $\vec{\mathbf{b}}_1, \vec{\mathbf{b}}_2, \ldots, \vec{\mathbf{b}}_l$, where $\vec{\mathbf{b}}_j \in \mathbb{R}^d$, and sparse activations $\vec{\mathbf{a}}^{(i)} \in \mathbb{R}^l$ such that $\mathbf{x}^{(i)} \approx \sum_{j=1}^{l} \vec{\mathbf{b}}_j a_j^{(i)}$. The basis set can be overcomplete, where $l > d$, and therefore can capture a large number of patterns in the input data. Sparse coding aims at solving the following optimization problem:

$$\min_{\{\vec{\mathbf{b}}_j\}, \{\vec{\mathbf{a}}_j^{(i)}\}} \sum_{i=1}^{n} \frac{1}{2\sigma^2} \left|\left|\mathbf{x}^{(i)} - \sum_{j=1}^{l} \vec{\mathbf{b}}_j a_j^{(i)}\right|\right|_2^2 + \beta \sum_{i=1}^{n}\sum_{j=1}^{l} \Lambda\left(a_j^{(i)}\right)$$
$$\text{subject to} \quad ||\vec{\mathbf{b}}_j||_2^2 \leq c, \ \forall j \in \{1, 2, \ldots, l\}$$

where, the first term $\left(\text{i.e., } \left|\left|\mathbf{x}^{(i)} - \sum_{j=1}^{l} \vec{\mathbf{b}}_j a_j^{(i)}\right|\right|_2^2\right)$ is the reconstruction error, and the second $\left(\text{i.e., } \beta \sum_{i=1}^{n} \sum_{j=1}^{l} \Lambda\left(a_j^{(i)}\right)\right)$ is the sparsity regularization, with $\Lambda(\cdot)$, being the penalty function, and $\beta \in \mathbb{R}$ is a constant. Different, $\Lambda$ can be used. These are detailed in Chapter 5. The one used in this dissertation is the $L_1$ norm, which has been shown to induce sparse activations and is robust to irrelevant features [72].

Assuming the usage of $L_1$ penalty as the sparsity function, the optimization problem is convex in the bases, while holding the activations fixed, and convex in the activations when holding the bases fixed. However, the problem is not convex in both the activations and bases simultaneously [57, 82]. Different methods for solving the optimization problem have been proposed. However, in this dissertation the algorithms proposed in [57] are adopted. The solution of the above problem is twofold. In the first, the bases are fixed, and the activations are determined. Given the activation, the next step commences to solve for the bases. According to [57], the first step (i.e., solving for the activations with the bases being fixed) can be solved by optimizing each $a_j^{(i)}$ individually:

$$\min_{\vec{\mathbf{a}}^{(i)}} \left|\left|\mathbf{x}^{(i)} - \sum_{j=1} \vec{\mathbf{b}}_j a_j^{(i)}\right|\right|_2^2 + (2\sigma^2\beta) \sum_j |a_j^{(i)}|$$

It is worth noting, that if the sign of the activations at the optimal value was known, it could be simply substituted by either $a_j^{(i)}$, $-a_j^{(i)}$, or 0. Given non zero coefficients, this reduces to a standard unconstrained quadratic optimization algorithm. Essentially, the feature-sign algorithm, proposed to solve for the activations in [57], searches for the sign of the activations. Given this sign, the resulting quadratic optimization problem can then be efficiently solved.

Given the above activations, solving for the bases boils down to the following optimization problem [57]:

$$\begin{aligned} &\min \left|\left|\mathbf{X} - \mathbf{BA}\right|\right|_F^2 \\ &\text{subject to } \sum_{i=1}^{d} B_{i,j} \leq c, \ \forall j \in \{1, 2, \ldots, l\} \end{aligned}$$

where $\mathbf{X}$ is the design matrix collecting all the inputs, $\mathbf{B}$ is the matrix of all basis vectors, and $\mathbf{A}$ is the matrix of all the activations.

As described in [57], this is a least squares problem with quadratic constraints that

can be efficiently solved using Lagrange duals. First the Lagrangian can be written as:

$$\mathcal{L}_{lagrange} = \text{trace}\Bigg(\Big(\mathbf{X} - \mathbf{BA}\Big)^T\Big(\mathbf{X} - \mathbf{BA}\Big)\Bigg) + \sum_{j=1}^{l}\lambda_j\Big(\sum_{i=1}^{d}B_{i,j}^2 - c\Big)$$

where, $\lambda_j \geq 0$ is a dual variable. Minimizing with respect to $\mathbf{B}$ yields:

$$\mathbb{D}(\vec{\lambda}) = \min_{\mathbf{B}} = \text{trace}\Bigg(\mathbf{X}^T\mathbf{X} - \mathbf{X}\mathbf{A}^T(\mathbf{A}\mathbf{A}^T + Z)^{-1}(\mathbf{X}\mathbf{A}^T)^T - cZ\Bigg)$$

with $Z = \text{diag}(\vec{\lambda})$. At this step the gradients of the Hessian of $\mathbb{D}$ can be computed and optimization using Newton's method or conjugate gradients can commence. Please note, the details of these algorithms are beyond scope of this section. The reader is referred to [57] for a detailed discussion of the topic.

**Restricted Boltzmann Machines:** Boltzmann machines (BMs) [2, 11, 56, 91] are a form of fully connected bidirectional neural networks with stochastic nodes. In BMs all the nodes are connected to each other. Unfortunately, learning with such models is computationally expensive. But this problem is remedied with the inclusion of some restrictions on the network's topology leading to the so-called *restricted Boltzmann machines* (RBMs) that are discussed next.

The machine shown in Figure 2.4 represents an illustration of an RBM with two layers. The first is the visible layer, $\mathcal{V}$, consisting of $m$ *visible* nodes $v^{(1)} \dots v^{(m)}$, while the second is the hidden layer, $\mathcal{H}$ with $n$ *hidden* nodes, $h^{(1)} \dots h^{(n)}$ hidden nodes. Each of the nodes in the visible and the hidden layer is also associated with its own bias term $a^{(i)}$, with $i$ denoting the index of the $i^{th}$ visible unit and $b^{(j)}$, with $j$ being the index of the $j^{th}$ hidden unit, respectively.

RBMs are generative models, in the sense that they model a valid probability distribution of a given dataset. In other words, learning in a RBM means adjusting the weights (i.e., the connections in Figure 2.4) such that the data is represented well in a richer feature space [35, 91, 102]. The hidden units model dependencies among the observations [35, 37]. Therefore RBMs could also be viewed as powerful nonlinear feature extractors. Moreover, due to the stochastisity exhibited by the neurons, RBMs also have the advantage of escaping local minima.

RBMs are formalized in terms of their *energy* function which in turn defines the joint probability distribution over both the visible and hidden units. Formally this is define as:

$$E(\mathcal{V}, \mathcal{H}) = -\sum_{i=1}^{m}a^{(i)}v^{(i)} - \sum_{j=1}^{n}b^{(j)}h^{(i)} - \sum_{i=1}^{m}\sum_{j=1}^{n}w_{i,j}v^{(i)}h^{(j)} \tag{2.15}$$

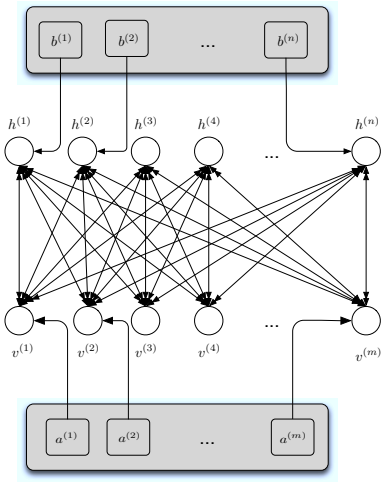



Figure 2.4: High level schematic of a Boltzmann Machine showing three two layers accompanied with the biases for each of the nodes. Please note, that this machine is a bidirectional fully connected one.

with $\mathcal{V} = [v^{(1)}, \ldots, v^{(m)}]^T$ is the vector of all visible nodes, $\mathcal{H} = [h^{(1)}, \ldots, h^{(n)}]^T$ is the vector of all hidden nodes, and $w_{i,j}$ representing all weight connections. From Equation 2.15, the joint probability of both the visible and hidden units is defined as:

$$p(\mathcal{V}, \mathcal{H}) = \frac{1}{\Psi} \exp\left(-E(\mathcal{V}, \mathcal{H})\right) = \frac{1}{\Psi} \exp\left(\sum_{i=1}^{m} a^{(i)} v^{(i)} + \sum_{j=1}^{n} b^{(j)} h^{(i)} + \sum_{i=1}^{m} \sum_{j=1}^{n} w_{i,j} v^{(i)} h^{(j)}\right)$$

with $\Psi$, the partition function being:

$$\Psi = \sum_{i=1}^{m} \left[ \sum_{j=1}^{n} \exp\left(\sum_{i=1}^{m} a^{(i)} v^{(i)} - \sum_{j=1}^{n} b^{(j)} h^{(i)} - \sum_{i=1}^{m} \sum_{j=1}^{n} w_{i,j} v^{(i)} h^{(j)}\right)\right]$$

When some data is given this is delivered to the visible nodes of the network. In other words, initially no information about the hidden units is available. Therefore,

the joint probability has to be marginalized as follows:

$$
\begin{aligned}
p(\mathcal{V}) &= \frac{1}{\Psi}\sum_{\mathcal{H}} p(\mathcal{V},\mathcal{H}) = \frac{1}{\Psi}\exp\Bigg(-E(\mathcal{V},\mathcal{H})\Bigg) \\
&= \frac{1}{\Psi}\sum_{\mathcal{H}}\exp\left(\sum_{i=1}^{m} a^{(i)}v^{(i)} + \sum_{j=1}^{n} b^{(j)}h^{(j)} + \sum_{i=1}^{n}\left[\sum_{j=1}^{m} w_{i,j}v^{(i)}h^{(j)}\right]\right) \\
&= \frac{1}{\Psi}\sum_{h^{(1)}}\sum_{h^{(2)}}\cdots\sum_{h^{(n)}}\exp\left(\sum_{i=1}^{m} a^{(i)}v^{(i)}\right)\exp\left(\sum_{j=1}^{n} b^{(j)}h^{(j)}\right)\exp\left(\sum_{i=1}^{n}\left[\sum_{j=1}^{m} w_{i,j}v^{(i)}h^{(j)}\right]\right) \\
&= \frac{1}{\Psi}\exp\left(\sum_{i=1}^{m} a^{(i)}v^{(i)}\right)\sum_{h^{(1)}}\sum_{h^{(2)}}\cdots\sum_{h^{(n)}}\left[\prod_{j=1}^{n}\exp\left(h^{(j)}\left(b^{(j)} + \sum_{i=1}^{m} w_{i,j}v^{(i)}\right)\right)\right] \\
&= \frac{1}{\Psi}\exp\left(\sum_{i=1}^{m} a^{(i)}v^{(i)}\right)\sum_{h^{(1)}}\exp\left(h^{(1)}\left(b^{(1)} + \sum_{i=1}^{m} w_{i,1}v^{(i)}\right)\right) \\
&\qquad \sum_{h^{(2)}}\exp\left(h^{(2)}\left(b^{(2)} + \sum_{i=1}^{m} w_{i,2}v^{(i)}\right)\right)\cdots\sum_{h^{(n)}}\exp\left(h^{(n)}\left(b^{(n)} + \sum_{i=1}^{m} w_{i,n}v^{(i)}\right)\right)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{\Psi}\exp\left(\sum_{i=1}^{n} a^{(i)}v^{(i)}\right)\prod_{j=1}^{n}\left[\sum_{h^{(j)}}\exp\left(h^{(j)}\left(b^{(j)} + \sum_{i=1}^{m} w_{i,j}v^{(i)}\right)\right)\right] \\
&= \frac{1}{\Psi}\prod_{i=1}^{m}\exp\left(a^{(i)}v^{(i)}\right)\prod_{j=1}^{n}\left[1 + \exp\left(b^{(j)} + \sum_{i=1}^{m} w_{i,j}v^{(i)}\right)\right]
\end{aligned}
$$

It is for this reason, that RBMs are referred to sometimes as *product of experts*, for more details see [89].

**Inference in restricted Boltzmann Machines** Performing inference in the model is simply done by conditioning on the observed data. It also depends on the type of distributions used for the nodes. In the original [91] both the visible and hidden units included sigmoidal functions of the form $\text{sigmoid}(x) = 1/1 + \exp(-x)$. Moreover, based on Figure 2.4 there are no connections neither between the units of the hidden layer nor between those of the visible layer. Therefore, inference is done

in parallel for each of the nodes according to:

$$\begin{aligned} p(\mathcal{H}|\mathcal{V}) &= \prod_{j=1}^{n} p(h^{(j)} = 1|\mathcal{V}) \\ &= \prod_{j=1}^{n} \text{sigmoid}\left( \sum_{i=1}^{n} w_{i,j} v^{(i)} + b^{(j)} \right) \\ p(\mathcal{V}|\mathcal{H}) &= \prod_{i=1}^{m} p(v^{(i)} = 1|\mathcal{H}) \\ &= \prod_{i=1}^{m} \text{sigmoid}\left( \sum_{j=1}^{n} w_{i,j} h^{(j)} + a^{(i)} \right) \end{aligned}$$

To see why the inputs in the previous equation are what they are, let $\mathcal{V}_{-,l}$ denote the state of all the visible units expect the $l^{th}$ one. Further define the following:

$$\alpha_l\Big(\mathcal{H}\Big) = -\sum_{j=1}^{n} w_{j,l} h^{(j)}$$

and

$$\beta\Big(\mathcal{V}_{-,l}, \mathcal{H}\Big) = -\sum_{j=1}^{n} \sum_{i=1, i\neq l}^{m} w_{i,j} h^{(j)} v^{(i)} - \sum_{i=1, i\neq l}^{m} a^{(i)} v^{(i)} - \sum_{j=1}^{n} b^{(j)} h^{(j)}$$

Then $E\Big(\mathcal{V}, \mathcal{H}\Big) = \beta\Big(\mathcal{V}_{-,l}, \mathcal{H}\Big) - v^{(l)} \alpha_l\Big(\mathcal{H}\Big)$, for details see [35, 91]. Using these definitions the following can be derived:

$$\begin{aligned} p\Big(v^{(l)} = 1|\mathcal{H}\Big) = p\Big(v^{(l)} = 1|\mathcal{V}_{-,l}, \mathcal{H}\Big) &= \frac{p\Big(v^{(l)} = 1, \mathcal{V}_{-,l}, \mathcal{H}\Big)}{p\Big(\mathcal{V}_{-,l}, \mathcal{H}\Big)} \\ &= \frac{\exp\left( -\beta\Big(\mathcal{V}_{-,l}, \mathcal{H}\Big) - 1.\alpha_l\Big(\mathcal{H}\Big) \right)}{\exp\left( -\beta\Big(\mathcal{V}_{-,l}, \mathcal{H}\Big) - 1.\alpha_l\Big(\mathcal{H}\Big) \right) + \exp\left( -\beta\Big(\mathcal{V}_{-,l}, \mathcal{H}\Big) - 0.\alpha_l\Big(\mathcal{H}\Big) \right)} \\ &= \frac{\exp\left( -\beta\Big(\mathcal{V}_{-,l}, \mathcal{H}\Big) \right) \exp\left( -\alpha_l\Big(\mathcal{H}\Big) \right) \Big)}{\exp\left( -\beta\Big(\mathcal{V}_{-,l}, \mathcal{H}\Big) \right) \exp\left( -\alpha_l\Big(\mathcal{H}\Big) \right) + \exp\left( -\beta\Big(\mathcal{V}_{-,l}, \mathcal{H}\Big) \right)} \end{aligned}$$

$$
\begin{aligned}
&= \frac{\exp\Big(-\beta\Big(\mathcal{V}_{-,l},\mathcal{H}\Big)\Big)\exp\Big(-\alpha_l\Big(\mathcal{H}\Big)\Big)}{\exp\Big(-\beta\Big(\mathcal{V}_{-,l},\mathcal{H}\Big)\Big)\Big[\exp\Big(-\alpha_l\Big(\mathcal{V}_{-,l},\mathcal{H}\Big)\Big)+1\Big]} \\
&= \frac{\exp\Big(-\alpha_l\Big(\mathcal{V}_{-,l},\mathcal{H}\Big)\Big)}{\exp\Big(-\alpha_l\Big(\mathcal{V}_{-,l},\mathcal{H}\Big)\Big)+1} = \frac{1}{1+\exp\Big(\alpha_l\Big(\mathcal{V}_{-,l}\mathcal{H}\Big)\Big)} \\
&= \text{sigmoid}\Big(-\alpha_l\Big(\mathcal{H}\Big)\Big) = \text{sigmoid}\Big(\sum_{j=1}^{n} w_{i,j}h^{(j)}+a^{(i)}\Big)
\end{aligned}
$$

The same can be performed to determine that $p(h^{(j)} = 1|\mathcal{V}) = \text{sigmoid}\Big(\sum_{i=1}^{m} w_{i,j}v^{(i)}+b^{(j)}\Big)$.

**Learning in restricted Boltzmann machines** In RBMs learning means the determinations of the weights and biases such that the likelihood of the data is maximized. The natural logarithm of the likelihood of the data is given by:

$$
\begin{aligned}
\log p\Big(\mathcal{V}\Big) &= \log\left[\frac{1}{\Psi}\sum_{\mathcal{H}}\exp\Big(-E(\mathcal{V},\mathcal{H})\Big)\right] \\
&= -\log\Psi + \log\left[\sum_{\mathcal{H}}\exp\Big(-E(\mathcal{V},\mathcal{H})\Big)\right] \\
&= \log\left[\sum_{\mathcal{H}}\exp\Big(-E(\mathcal{V},\mathcal{H})\Big)\right] - \log\left[\sum_{\mathcal{H},\mathcal{V}}\exp\Big(-E(\mathcal{V},\mathcal{H})\Big)\right]
\end{aligned}
\tag{2.16}
$$

To maximize the likelihood of the model the derivatives need to be calculated with respect to the free parameters (i.e., weights and biases). Next, these derivations are

shown[13] leading to the following calculations:

$$\begin{aligned}\frac{\partial}{\partial w_{i,j}}\log p\Big(\mathcal{V}\Big) &= \frac{\partial}{\partial w_{i,j}}\left[\log\left[\sum_{\mathcal{H}}\exp\Big(-E(\mathcal{V},\mathcal{H})\Big)\right]\right] - \frac{\partial}{\partial w_{i,j}}\left[\log\left[\sum_{\mathcal{V},\mathcal{H}}\exp\Big(-E(\mathcal{V},\mathcal{H})\Big)\right]\right] \\ &= -\frac{1}{\sum_{\mathcal{H}}\exp\left(-E(\mathcal{V},\mathcal{H})\right)}\sum_{\mathcal{H}}\exp\left(-E(\mathcal{V},\mathcal{H})\right)\frac{\partial}{\partial w_{i,j}}E(\mathcal{V},\mathcal{H}) \\ &\qquad + \frac{1}{\sum_{\mathcal{V},\mathcal{H}}\exp\left(-E(\mathcal{V},\mathcal{H})\right)}\sum_{\mathcal{V},\mathcal{H}}\exp\left(-E\left(\mathcal{V},\mathcal{H}\right)\right)\frac{\partial}{\partial w_{i,j}}E\left(\mathcal{V},\mathcal{H}\right),\end{aligned}$$

using:

$$p(\mathcal{H}|\mathcal{V}) = \frac{p(\mathcal{V},\mathcal{H})}{p(\mathcal{V})} = \frac{\frac{1}{\Psi}\exp\left(-E\left(\mathcal{V},\mathcal{H}\right)\right)}{\frac{1}{\Psi}\sum_{\mathcal{H}}\exp\left(-E\left(\mathcal{V},\mathcal{H}\right)\right)} = \frac{\exp\left(-E\left(\mathcal{V},\mathcal{H}\right)\right)}{\sum_{\mathcal{H}}\exp\left(-E\left(\mathcal{V},\mathcal{H}\right)\right)}$$

the following is derived:

$$= \underbrace{-\sum_{\mathcal{H}}p(\mathcal{H}|\mathcal{V})\frac{\partial}{\partial w_{i,j}}E\left(\mathcal{V},\mathcal{H}\right)}_{\text{expectation under the data}} + \underbrace{\sum_{\mathcal{V},\mathcal{H}}p(\mathcal{V},\mathcal{H})\frac{\partial}{\partial w_{i,j}}E\left(\mathcal{V},\mathcal{H}\right)}_{\text{expectation under the model}}$$

Given a training set $\mathcal{D} = \{\mathcal{V}^{(o)}\}_{o=1}^{L}$, the mean of the log-likelihood derivative can be calculated now as:

$$\begin{aligned}\frac{1}{L}\sum_{\mathcal{V}\in\mathcal{D}}\frac{\partial}{\partial w_{i,j}}\log p(\mathcal{V}) &= \frac{1}{L}\sum_{\mathcal{V}\in\mathcal{D}}\left[-\mathbb{E}_{p(\mathcal{H}|\mathcal{V})}\left[\frac{\partial}{\partial w_{i,j}}E\left(\mathcal{V},\mathcal{H}\right)\right] + \mathbb{E}_{p(\mathcal{V},\mathcal{H})}\left[\frac{\partial}{\partial w_{i,j}}E\left(\mathcal{V},\mathcal{H}\right)\right]\right] \\ &= \frac{1}{L}\sum_{\mathcal{V}\in\mathcal{D}}\left[\mathbb{E}_{p(\mathcal{H}|\mathcal{V})}\left[v^{(i)}h^{(j)}\right] - \mathbb{E}_{p(\mathcal{V},\mathcal{H})}\left[v^{(i)}h^{(j)}\right]\right] \\ &= \left\langle v^{(i)}h^{(j)}\right\rangle_{p(\mathcal{H}|\mathcal{V})q(\mathcal{V})} - \left\langle v^{(i)}h^{(j)}\right\rangle_{p(\mathcal{V},\mathcal{H})}\end{aligned}$$

with $q(\mathcal{V})$ being the empirical distribution. This leads to the often stated rule:

$$\frac{1}{L}\sum_{\mathcal{V}\in\mathcal{D}}\frac{\partial}{\partial w_{i,j}}\log p(\mathcal{V}) \propto \left\langle v^{(i)}h^{(j)}\right\rangle_{\text{data}} - \left\langle v^{(i)}h^{(j)}\right\rangle_{\text{model}}$$

The derivatives with respect to biases can be calculated in a similar manner. These are left as an exercise for the reader.

Calculating the above summations exactly is computationally expensive [35, 36] and intractable. In order to reduce this computational cost, different RBM learning

[13]The derivations with respect to the biases could be performed similarly and is left as an exercise for the reader.

algorithms were suggested. Most rely on an approximation of the maximum likelihood [19, 36, 37]. One of the widely used techniques to deal with these computational problems is the so-called contrastive divergence algorithm.

**Contrastive Divergence** Maximum likelihood works on minimizing the Kullback-Lieber divergence between the data and the equilibrium distribution [36, 60]. Typically, the equilibrium distribution is produced by prolonged Gibbs sampling from the generative model (i.e., the RBM).

The proof of this claim is simple. A sketch of proving the claim is presented next. Other variants could be seen in

*Proof.* Consider $p^{emp}(\mathbf{x})$ and $\hat{p}(\mathbf{x}|\boldsymbol{\theta})$ be the empirical and model distribution, respectively. Given a data set $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}$ drawn from the *real* distribution $p(\mathbf{x})$, then $p^{emp}(\mathbf{x})$ is define as the empirical distribution which puts $\frac{1}{n}$ probability oe each data point as:

$$p^{emp}(\mathbf{x}) = \frac{1}{n}\sum_{t=1}^{n}\delta(\mathbf{x} - \mathbf{x}^{(t)})$$

with $\delta(\cdot)$ being the Kroncker delta function. The Kullback-Lieber divergence from the empirical to the model distribution is defined as:

$$\begin{aligned} p^{emp}(\mathbf{x})||\hat{p}(\mathbf{x}|\boldsymbol{\theta}) &= \int p^{emp}(\mathbf{x}) \log\left[\frac{p^{emp}(\mathbf{x})}{\hat{p}(\mathbf{x}|\boldsymbol{\theta})}\right] dx \\ &= -H(p^{emp}) - \int p^{emp} \log(\hat{p}(\mathbf{x}|\boldsymbol{\theta}))dx \end{aligned}$$

with $H(p^{emp}) = -\int p^{emp}(\mathbf{x}) \log p^{emp}(\mathbf{x})dx$ representing the entropy of $p^{emp}$. It follows that:

$$\arg\min_{\boldsymbol{\theta}} [p^{emp}(\mathbf{x})||\hat{p}(\mathbf{x}|\boldsymbol{\theta})] \equiv \arg\max_{\boldsymbol{\theta}} \left\langle \log \hat{p}(\mathbf{x}|\boldsymbol{\theta}) \right\rangle_{p^{emp}}$$

with $\left\langle \cdot \right\rangle_{-}$ representing the expectation under the $-$ distribution. Furthermore:

$$\begin{aligned} \left\langle \log \hat{p}(\mathbf{x}|\boldsymbol{\theta}) \right\rangle_{p^{emp}} &= \frac{1}{n}\int \sum_{t=1}^{n} \delta(\mathbf{x} - \mathbf{x}^{(t)}) \log \hat{p}(\mathbf{x}|\boldsymbol{\theta})dx \\ &= \frac{1}{n}\sum_{t=1}^{n} \log \hat{p}(\mathbf{x}^{(t)}|\boldsymbol{\theta}) \end{aligned}$$

which is apart from the scaling factor identical to the log-likelihood function. □

Depending on the generative model, computing the gradients in maximum likelihood is intractable. It involves a summation over an exponential number of terms. Therefore, Hinton [36] suggested contrastive divergence (CD). CD approximately follows the gradient of a different objective function. In CD the gradient follows the direction of the difference between two divergence [18, 36] as:

$$CD_\lambda = p_0||p_\infty - p_\lambda||p_\infty \tag{2.17}$$

where $p_0$ is the original empirical data distribution, $p_\infty$ is that of the model, and $p_\lambda$ is the distribution after $\lambda$ steps of a Markov chain starting at the original probability distribution $p_0$. According to [18, 30, 36] this reduces both the computational cost per gradient step and the variance of the estimated gradient. Since $p_\lambda$ is $\lambda$ steps closer to the equilibrium distribution $p_\infty$, then Equation 2.17 is guaranteed to be always positive [36]. Furthermore, CD could only be zero if the model is perfect, whereby the difference in Equation 2.17 becomes zero.

The mathematical motivation behind CD is clear. Namely, the equilibrium distribution cancels out making the computational of the derivatives tractable and efficient. After computing the derivatives the following update rules for the weights are derived:

$$w_{i,j}^{\tau+1} = w_{i,j}^{\tau+1} + \alpha \left( \left\langle \langle v^{(i)} h^{(j)} \rangle_{p(\mathcal{H}|\mathcal{V};\mathbf{W})} \right\rangle_0 - \langle v^{(i)} h^{(j)} \rangle \right) \tag{2.18}$$

with $\mathbf{W}$ being the matrix of all the weights between the connections.

Algorithm 3 represents the main steps performed in CD as described in [36]. As an input Algorithm 3 requires the setting for the visible and hidden layers of the RBM as well as a batch of samples to train on. After the initializations shown in line 2 of the algorithm, it proceeds to perform the Markov chain run for $\lambda$ steps. Throughout these steps Algorithm 3 fixes the visible layer configuration and samples the assuming these given the hidden configuration as shown in line 7. This is then repeated but fixing the hidden layer this time in line 10. The weights are then updated such that the reconstruction error is minimized, line 14.

**Restricted Boltzmann Machine Variants** So far RBMs with only binary layers have been described. There exists different variants that allow these to work with continuous inputs. This could be achieved by redefining the energy function to:

$$E(\mathcal{V}, \mathcal{H}) = \sum_{i,j} \phi_{i,j}(v^{(i)}, h^{(j)}) + \sum_i \alpha_i(v^{(i)}) + \sum_j \nu_j(h^{(j)})$$

with real valued functions $\phi_{i,j}$, $\alpha_i$, and $\nu_j$, $\forall i \in \{1, 2, \ldots, m\}$ and $\forall j \in \{1, 2, \ldots, n\}$, fulfilling the constraint that the partition function $\Psi$ is finite. The details on the

**Algorithm 3** $\lambda$ steps contrastive divergence

**Input:** $\mathcal{V} = [v^{(1)}, v^{(2)}, \ldots, v^{(m)}]^T$, $\mathcal{H} = [h^{(1)}, h^{(2)}, \ldots, h^{(n)}]^T$, training batch $\mathcal{D}$
**Initializations:** $\delta w_{i,j}$, $\delta a^{(i)}$, $\delta b^{(j)}$ to zero $\forall i \in \{1, 2 \ldots, m\}$ and $\forall j \in \{1, 2, \ldots, n\}$.
**forall** $\mathcal{V} \in \mathcal{D}$ **do**
  $\mathcal{V}^{(0)} = \mathcal{V}$
  **for** $\tau = 0, \ldots, \lambda - 1$ **do**
    **for** $j = 1, \ldots, n$ **do**
      Sample $h_\tau^{(j)} \sim p(h^{(i)}|\mathcal{V}_{(\tau)})$
    **end for**
    **for** $i = 1, \ldots, m$ **do**
      Sample $v_{\tau+1}^{(i)} \sim p(v^{(i)}|\mathcal{H}_\tau)$
    **end for**
  **end for**
  **for** $i = 1, \ldots, m$ and $j = 1, \ldots, n$ **do**
    Update weights according to Equation 2.18.
  **end for**
**endfor**
**Return:** $\delta w_{i,j}$, $\delta a^{(i)}$, $\delta b^{(j)}$ $\forall i \in \{1, 2 \ldots, m\}$ and $\forall j \in \{1, 2, \ldots, n\}$

selection of these functions are clarified in Chapter 6 as needed.

Finally, RBMs could be extended to have more than two layers. These are called high-order RBMs as described in [68, 102]. Details of these will be described in Chapter 6, where two new RBMs suitable for transfer learning are constructed.

## 2.3 Reinforcement Learning

Reinforcement learning (RL) is an algorithmic technique for solving sequential decision making problems [17]. These problems appear in a wide variety of fields, such as artificial intelligence, automatic control, economics, operational research, as well as medicine. In artificial intelligence, RL is of major importance to the design of intelligent agents that monitor an environment through perception and influence it by applying actions. A high level schematic of an intelligent agent construction is shown in Figure 2.5. Opposing to SL, RL agents are not given the correct actions to execute in a state. They rather have to learn to choose these actions by maximizing a total return signal. This signal quantifies the agent's behavior for executing an action in a given state.

An agent interacts with the environment by applying actions from an allowed action set. Being at a certain state, the agent decides to execute an action. Due to this execution it then transitions into a successor state, to which it receives a performance quantifying signal called the reward. This reward assesses the behavior of the agent in the environment. The goal is then to learn a behavior, whereby the

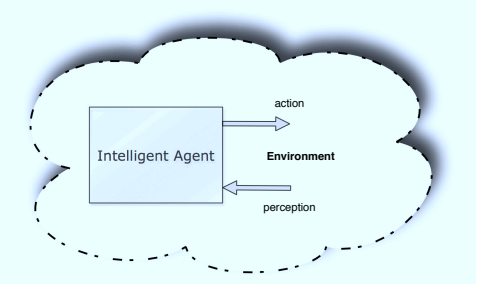



Figure 2.5: Intelligent agents live in an environment that they percept and affect by applying actions.

learner chooses "correct" actions that maximize the total discounted return from any initial state. This is called an optimal behavior. RL is typically formalized in terms of Markov Decision Processes which are a mathematical constructions allowing the formalization of sequential decision making problems. These essential ingredients are discussed next.

## Markov Decision Processes

Reinforcement Learning (RL) is formalized in terms of Markov Decision Processes (MDPs) which are a tuple of $\mathcal{M} = \langle \mathcal{S}, \mathcal{U}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{U}$ represents the actions space, $\mathcal{T}$ is the transition probability, $\mathcal{R}$ is the reward function, and $\gamma$ is the discount factor. Furthermore, RL algorithms abide by the *Markov property*[14] where the current state and the current action are enough statistics to determine the successor state and reward. In other words, to determine the future the agent only requires knowledge of the current state and current action with no regard to the prior history.

On a high level, MDPs can be split into two settings. The first is the deterministic setting, while the second is the stochastic one. Each of these are detailed next as described in [17].

### Deterministic Setting

Deterministic MDPs [17, 33] are defined by the state space $\mathcal{S}$ in which the agent can exist in, the action space $\mathcal{U}$ which includes the actions allowed the by agent to execute, the transition probability function $\mathcal{T}$ determining the transition to a successor state starting from an initial and applying an action, and the reward function, $\mathcal{R}$, evaluating

[14] Write why it is not the case.

the performance of the agent. Starting from a state $s_t \in \mathcal{S}$ and applying an action $u_t \in \mathcal{U}$ with $t$ representing the time index, the agent transitions to $s_{t+1}$ according to $\mathcal{T} : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$:

$$s_{t+1} = \mathcal{T}(s_t, a_t)$$

Due to this transition the agent receives a reward $r_{t+1}$, according to the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{U} \to \mathbb{R}$:

$$r_{t+1} = \mathcal{R}(s_t, a_t)$$

where it is assumed that $||\mathcal{R}||_\infty = \sup_{s \in \mathcal{S}, u \in \mathcal{U}} |\mathcal{R}(s, u)|$ is finite. The agent chooses actions according to a policy $\pi : \mathcal{S} \to \mathcal{U}$, according to:

$$u_t = \pi(s_t)$$

**Optimality in the Deterministic Setting** Deterministic MDPs are defined by the state space $\mathcal{S}$ the agent can exist in, the action space $\mathcal{U}$ which includes the actions allowed by the agent to execute, the transition probability $\mathcal{T}$ determining the transition to a successor state starting from an initial and applying an action, and the reward function, $\mathcal{R}$, evaluating the performance of the agent. Starting from a state $s_t \in \mathcal{S}$ and applying an action $u_t \in \mathcal{U}$ with $t$ representing the time index, the agent transitions to $s_{t+1}$ according to $\mathcal{T} : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$:

$$s_{t+1} = \mathcal{T}(s_t, a_t)$$

It then receives a reward $r_{t+1}$, calculated through the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{U} \to \mathbb{R}$:

$$r_{t+1} = \mathcal{R}(s_t, a_t)$$

where it is assumed that $||\mathcal{R}||_\infty = \sup_{s \in \mathcal{S}, u \in \mathcal{U}} |\mathcal{R}(s, u)|$ is finite. The agent chooses actions according to a policy $\pi : \mathcal{S} \to \mathcal{U}$, as:

$$u_t = \pi(s_t)$$

**Optimality in the Deterministic Setting** The goal in RL is for the agent to find an *optimal policy* that maximize the return from *any* starting state $s_0 \in \mathcal{S}$, see [17, 80, 101]. The total return is the accumulation of the rewards along a certain trajectory starting from $s_0$ and applying action according to a policy $\pi$. Different types of the total return exist, mainly differing by the scheme in which the return are collected. For instance, in the *infinite horizon* case, the return or total-payoff is

defined as:

$$\begin{aligned}\rho^{\pi}(s_0) &= \sum_{t=0}^{\infty} \gamma^t r_{t+1} \\ &= \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, \pi(s_t))\end{aligned} \tag{2.19}$$

with $\gamma \in [0, 1[$ being the discount factor. The role of $\gamma$ is to avoid the series from diverging.

Thus, the goal of an RL agent is to find a policy that maximizes the total return, while using feedback about the immediate one step rewards. This leads to the problem of delayed-rewards in which actions taken in the present affect the potential to achieve good rewards in the far future, however the immediate rewards provide no information about the long-term effects [17].

Other forms of returns can be also defined. As an example, if the discount factor in Equation 2.19 is set to 1, the return will simply be the sum of rewards with no discounting. Unfortunately, in most cases such a sum is unbounded, and therefore, an alternative to use is:

$$\begin{aligned}\rho^{\pi}(s_0) &= \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} \mathcal{R}(s_t, u_t) \\ &= \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} \mathcal{R}(s_t, \pi(s_t))\end{aligned}$$

which is bounded in most cases.

Finite horizon returns can be obtained by restricting the length of the trajectories. Namely, rather than using infinite length trajectories, the *finite horizon* setting makes use of finite length trajectories. The return is then defined as:

$$\begin{aligned}\rho^{\pi}(s_0) &= \sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, u_t) \\ &= \sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, \pi(s_t))\end{aligned}$$

The undiscounted return is easier to use in the finite horizon case since the return is bounded if the rewards are.

In this thesis the main focus is on the *infinite horizon* return as it exhibits interesting theoretical results. For instance, in such type of return there always exists at least one *stationary* optimal policy $\pi^{\star} : \mathcal{S} \to \mathcal{U}$.

**Bellman Equations for the Deterministic Setting** To characterize the behavior of an agent following a certain policy, typically value functions are used. There exists two types of these functions: (1) value functions (V-functions), and (2) State-action value functions (Q-functions). This section is adopted from [17] and [101].

The state-action value function, Q, of a policy, $\pi$, is defined as a mapping from states and actions to real values (i.e., $Q^\pi : \mathcal{S} \times \mathcal{U} :\to \mathbb{R}$). This function gives the return of an agent starting at a certain state $s \in \mathcal{S}$ applying an action $u \in \mathcal{U}$ and following the policy $\pi$ thereafter:

$$\begin{aligned} Q^\pi(s,u) &= \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, u_t) \\ &= \mathcal{R}(s,u) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} \mathcal{R}(s_t, \pi(s_t)) \\ &= \mathcal{R}(s,u) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} \mathcal{R}(\mathcal{T}(s_t, u_t)) \\ &= \mathcal{R}(s,u) + \gamma \rho^\pi(\mathcal{T}(s_t, u_t)) \end{aligned}$$

with $\langle s_0, u_0 \rangle = \langle s, u \rangle$, $s_{t+1} = \mathcal{T}(s_{t+1}, u_{t+1})$, and $u_t = \pi(s_t)$.

The optimal state-action value function, $Q^\star$, is defined as:

$$\begin{aligned} Q^\star(s,u) &= \max_{\pi \in \Pi} Q^\pi(s,u) \\ &= \max_{\pi \in \Pi} [\mathcal{R}(s,u) + \gamma \rho^\pi(\mathcal{T}(s_t, u_t))] \end{aligned}$$

with $\Pi$ being the space of all allowed stationary policies. An optimal policy is defined as the policy that greedily chooses actions in the optimal state-action value function:

$$\pi^\star(s) = \arg\max_{u \in \mathcal{U}} Q^\star(s,u)$$

Typically, the Q-function are recursively characterized by the so-called *Bellman*

*equations*. These can be derived as follows:

$$\begin{aligned} Q^{\pi}(s,u) &= \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, u_t) \\ &= \mathcal{R}(s,u) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} \mathcal{R}(s_t, \pi(s_t)) \\ &= \mathcal{R}(s,u) + \gamma \left[ \mathcal{R}\Big(\mathcal{T}(s,u), \pi\left(\mathcal{T}(s,u)\right)\Big) + \gamma \sum_{t=2}^{\infty} \gamma^{t-2} \mathcal{R}\Big(s_t, \pi(s_t)\Big) \right] \\ &= \mathcal{R}(s,u) + \gamma Q^{\pi}\left(\mathcal{T}(s,u), \pi\Big(\mathcal{T}(s,u)\Big)\right) \end{aligned}$$

with $\langle s_0, u_0 \rangle = \langle s, u \rangle$, $s_{t+1} = \mathcal{T}(s_{t+1}, u_{t+1})$, and $u_t = \pi(s_t)$. The Bellman optimality equation is defined as:

$$Q^{\star}(s,u) = \mathcal{R}(s,a) + \gamma \max_{u' \in \mathcal{U}} Q^{\star}\left(\mathcal{T}(s,u), u'\right)$$

After discussing the Q-functions, now the attention is reverted back to the value functions (V-functions) case. The V-function is defined as a mapping from states to real number as: $V^{\pi}(s) : \mathcal{S} \rightarrow \mathbb{R}$, with $\pi \in \Pi$ representing the policy followed by the agent. When representing the Q-function to be only a function of the state, the V-function could be computed as:

$$V^{\pi}(s) = \rho^{\pi}(s) = Q^{\pi}(s, \pi(s))$$

The optimal V-function can then be defined as:

$$\begin{aligned} V^{\star}(s) &= \max_{\pi \in \Pi} V^{\pi}(s) \\ &= \max_{u \in \mathcal{U}} Q^{\star}(s,u) \end{aligned}$$

An optimal policy can be computed by taking greedy actions in the optimal V-function as follows[15]:

$$\pi^{\star}(s) = \arg\max_{u \in \mathcal{U}} \left[ \mathcal{R}(s,a) + \gamma V^{\star}\Big(\mathcal{T}(s,u)\Big) \right] \tag{2.20}$$

[15]Dealing with this equation is harder than the case of the Q-functions, since here a model in the form of transitions and rewards are required. In the Q-function case, this is not required as information about the transitions are included into the definition of the function.

Furthermore the V-functions satisfy the following *Bellman equations*:

$$V^{\pi}(s) = \mathcal{R}(s, \pi(s)) + \gamma V^{\pi}\left(\mathcal{T}\Big(s, \pi(s)\Big)\right)$$
$$V^{\star}(s) = \max_{u \in \mathcal{U}} \left[\mathcal{R}(s, u) + \gamma V^{\star}\left(\mathcal{T}\Big(s, u\Big)\right)\right]$$

**Stochastic Setting**

In the stochastic case the next state can not be deterministically calculated from the current state and action [17]. Formally, the deterministic transition function is not replaced by the *probability density* function $\tilde{\mathcal{T}} : \mathcal{S} \times \mathcal{U} \times \mathcal{S} \to [0, \infty[$. When an action $u_t$ is applied at a certain state $s_t$, the probability that the next state $\mathcal{S}_{t+1} \subseteq \mathcal{S}$ is computed according to:

$$p(s_{t+1} \in \mathcal{S} | s_t, u_t) = \int_{\mathcal{S}_{t+1}} \tilde{\mathcal{T}}(s_t, u_t, s')ds'$$

For any $s$ and $u$, $\tilde{\mathcal{T}}(s, u, \cdot)$ must define a valid probability density function of the argument $\cdot$ as detailed in [17]. Furthermore, the rewards are also associated with the transitions and therefore, these can not be deterministic anymore. The stochastic reward function $\tilde{\mathcal{R}} : \mathcal{S} \times \mathcal{U} \times \mathcal{S} \to \mathbb{R}$ is used instead to determine the instantaneous reward of a transition as:

$$r_{t+1} = \tilde{\mathcal{R}}(s_t, u_t, s_{t+1})$$

where the reward is bounded (i.e., $||\tilde{\mathcal{R}}||_{\infty} = \sup_{\langle s,u,s' \rangle \in \mathcal{S} \times \mathcal{U} \times \mathcal{S}} \tilde{\mathcal{R}}(s, u, s')$) is finite.

**Optimality in the Stochastic Setting** Following the description in [17], the infinite discounted total return in the stochastic setting is defined as:

$$\begin{aligned}\tilde{\rho}^{\pi}(s_0) &= \lim_{T \to \infty} \mathbb{E}_{\left\{s_{t+1} \sim \tilde{\mathcal{T}}(s_t, \pi(s_t), .)\right\}} \left[\sum_{t=0}^{T} \gamma^t r_{t+1}\right] \\ &= \lim_{T \to \infty} \mathbb{E}_{\left\{s_{t+1} \sim \tilde{\mathcal{T}}(s_t, \pi(s_t), .)\right\}} \left[\sum_{t=0}^{T} \gamma^t \tilde{\mathcal{R}}\Big(s_t, \pi(s_t), s_{t+1}\Big)\right]\end{aligned}$$

**Bellman Equations for the Stochastic Setting** The previous equations for the Q and value functions need to extended to fit the stochastic case. The Q-function is

now defined as:

$$Q^{\pi}(s,a) = \mathbb{E}_{\left\{s' \sim \tilde{\mathcal{T}}(s,\pi(s),.)\right\}}\left[\tilde{\mathcal{R}}(s,u,s') + \gamma\tilde{\rho}^{\pi}(s')\right]$$

The optimal state-action value function, as well as the optimal policy remain unchanged.

On the other hand, the *Bellman equations* for both the Q and the optimal Q-functions need to be redefined as:

$$Q^{\pi}(s,u) = \mathbb{E}_{\left\{s' \sim \tilde{\mathcal{T}}(s,\pi(s),.)\right\}}\left[\tilde{\mathcal{R}}(s,u,s') + \gamma Q^{\pi}(s',\pi(s'))\right]$$

$$Q^{\star}(s,u) = \mathbb{E}_{\left\{s' \sim \tilde{\mathcal{T}}(s,\pi(s),.)\right\}}\left[\tilde{\mathcal{R}}(s,u,s') + \gamma \max_{u' \in \mathcal{U}} Q^{\star}(s',u')\right]$$

The definition of the V-function remains unchanged as in the deterministic case. However, the computation of the optimal policies become:

$$\pi^{\star}(s) = \arg\min_{u \in \mathcal{U}} \mathbb{E}_{s' \sim \tilde{\mathcal{T}}(s,u,.)}\left[\mathcal{R}(s,u,s') + \gamma V^{\star}(s')\right]$$

The Bellman equations of the V-functions is changed by only adding the expectation over one step transition. This yields:

$$V^{\pi}(s) = \mathbb{E}_{s' \sim \tilde{\mathcal{T}}(s,u,.)}\left[\tilde{\mathcal{R}}(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

$$V^{\star}(s) = \max_{u \in \mathcal{U}} \mathbb{E}_{s' \sim \tilde{\mathcal{T}}(s,u,.)}\left[\tilde{\mathcal{R}}(s,\pi(s),s') + \gamma V^{\star}(s')\right]$$

## Reinforcement Learning in Continuous Spaces

Classical RL algorithms require an exact representation of the value functions and/or policies [17]. As described in the previous sections, RL makes use of: (1) Q-functions, (2) V-functions, and (3) policies. For exact representations, distinct estimates of the Q-function values for the state-action transitions is required, while in the V-functions case distinct estimation for each state is needed. Furthermore, when exactly representing a policy, exact actions for each state need to be stored. Unfortunately, such a scheme can not be possibly adopted when the state and/or action spaces take on infinite values. Moreover, continuous state and/or action spaces are of major interest in many real-world applications. Apart from this *representational problem*,

most RL algorithms have to repeatedly solve non-concave optimization problems when solving for the optimal behavior. When the spaces are continuous looping over all the possible state-action transitions is impossible. Therefore, *sample-based* approximation is a possible solution. In details, the algorithm fetches the optimal behavior on a given (offline case) or collected (online case) samples. In the offline case of course the quality of the solution depends on the relative "goodness" of the collected samples.

To deal with such scenarios, function approximation techniques from supervised learning are used. As in any regression algorithm there are two directions the designer can take when deciding on the model to be used: (1) parametric, and (2) nonparametric. The reader is referred to Section 2.1 for the details of these approaches.

**Approximate Reinforcement Learning Framework**

The details of this section are adopted from [17, 101]. To formalize approximate RL techniques, define $\mathcal{Q}$ to be the set of all possible state-action value functions. Further, let $\mathbf{\Omega} : \mathbf{\Gamma} :\rightarrow \mathcal{Q}$, be the approximation mapping. Therefore, the approximation of the Q-function can be written as:

$$\hat{Q}(s,u) = [\mathbf{\Omega}(\boldsymbol{\gamma})](s,u), \text{ with } s \in \mathcal{S} \text{ and } u \in \mathcal{U} \tag{2.21}$$

where $\boldsymbol{\gamma} \in \mathbb{R}^k$ is the vector of parameters used to parametrize the Q-function, and $[\mathbf{\Omega}(\boldsymbol{\gamma})](s,u)$ denotes the parameterized Q-function evaluated at a state $s \in \mathcal{S}$ and action $u \in \mathcal{U}$. It is worth noting that this representation is capable of only describing a certain subspace of $\mathcal{Q}$, and therefore, arbitrary Q-function in $\mathcal{Q}$ can only be represented to a certain degree of accuracy.

Given $k$-basis functions $\phi^{(1)}, \ldots, \phi^{(k)} : \mathcal{S} \times \mathcal{U} :\rightarrow \mathbb{R}$ , Equation 2.21 can be rewritten as:

$$\begin{aligned}\hat{Q}(s,u) &= [\mathbf{\Omega}(\boldsymbol{\gamma})](s,u) \\ &= \sum_{i=1}^{k} \phi^{(i)}(s,u)\gamma^{(i)} = \boldsymbol{\gamma}^T \mathbf{\Phi}(s,u)\end{aligned}$$

with $\mathbf{\Phi} = [\phi^{(1)}(s,u), \ldots, \phi^{(k)}(s,u)]^T$ is the vector of all basis functions [17]. The values of these parameters are fitted such that certain criterion is satisfied.

Other versions of this approximation can be distinguished as in the normal supervised learning case. For instance, a nonparametric approximation of the Q-function could also be performed as described in [17].

## Reinforcement Learning Algorithms

One categorization of RL algorithms is shown in Figure 2.6. The algorithms are split into three types. In value iteration, a search for the optimal value function is first computed. The optimal policy is then derived from the attained function according to Equation 2.20.

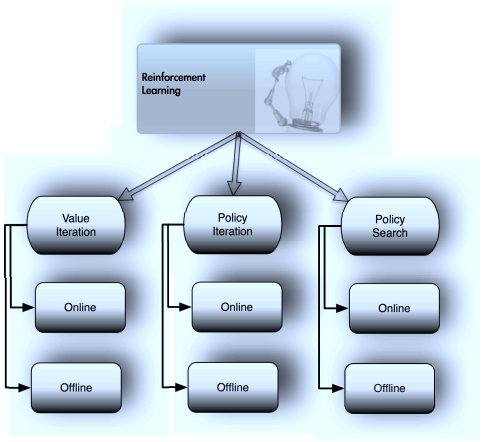



Figure 2.6: A high level pictorial categorization of reinforcement learning algorithms. These could be split into three types of algorithms: (1) value iteration, (2) policy iteration, and (3) policy search. Further, each has an online and offline version.

On the other hand, policy iteration algorithms evaluate policies by computing their value functions that are then used to produce better policies. Finally, in policy search, the agent makes use of direct optimization in the policy space to determine the optimal behavior. Furthermore, in each of these three different classes two additional variants of the algorithms can be distinguished. In the offline case, the agent reasons about the process using data[16] collected in advance, while in the offline scenario the agent learns by interacting with the process. Online RL algorithms have to further balance the trade-off between exploration and exploitation. In other words, the agent has to decide whether to choose random actions to gain more knowledge about the system, or whether to exploit its current knowledge by for instance, taking greedy actions in the learnt Q-function. There is has been a lot of work in this domain, to balance this problem.

Next, each of these different types are surveyed, whereby the online, and the offline variants of the algorithms are described. Moreover, the counter-part extensions to continuous state spaces are detailed.

[16]This data is typically collected in the form of transitions.

**Value Iteration Algorithms: Finite State Space Case**

To survey the value iteration algorithms extra definitions need to be introduced. Define $T : \mathcal{Q} \to \mathcal{Q}$ to be the so-called *Bellman operator* mapping from the space of Q-functions to that same space [17, 101].

In the deterministic case $T$ is defined as:

$$[T^{\pi}(Q)](s,u) = \mathcal{R}(s,u) + \gamma Q(\mathcal{T}(s,\pi(s)),\pi(s')),$$

while in the stochastic case this operator is given by:

$$[T^{\pi}(Q)](s,u) = \mathbb{E}_{\{s' \sim \bar{\mathcal{T}}(s,a,.)\}}\left[\tilde{\mathcal{R}}(s,a,s') + \gamma Q(s',\pi(s'))\right] \tag{2.22}$$

Of course for the finite state space case Equation 2.22 can be written in terms of the transition probabilities as:

$$[T^{\pi}(Q)](s,u) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{T}}(s,\pi(s),s')\left[\tilde{\mathcal{R}}(s,a,s') + \gamma Q(s',\pi(s'))\right]$$

Further define the *optimal Bellman Operator*, $T^{\pi^\star} : \mathcal{Q} \to \mathcal{Q}$, which in the deterministic case is:

$$[T^{\pi^\star}(Q)](s,u) = \sup_{u \in \mathcal{U}}\left[\mathcal{R}(s,u) + \gamma Q(\mathcal{T}(s,u),u')\right],$$

and in the stochastic case this operator is:

$$[T^{\pi^\star}(Q)](s,u) = \sup_{u \in \mathcal{U}}\left[\sum_{s' \in \mathcal{S}} \bar{\mathcal{T}}(s,\pi(s),s')\left[\tilde{\mathcal{R}}(s,a,s') + \gamma Q(s',\pi(s'))\right]\right]$$

All value iteration based algorithms (e.g., Q-learning) start at an arbitrary initialization of the Q-function. This is done updated according to:

$$Q_{\tau+1} = T^{\pi}(Q_{\tau}) \tag{2.23}$$

where $\tau$ is the iteration number and $T^{\pi}$ is the Bellman operator defined previously. It is clear that Q is a fixed point in Equation 2.23. Moreover, it is also easy to see

that $T^\pi$ is a contraction mapping in infinite norm:

$$\begin{aligned}||T^\pi(Q) - T^\pi(Q')||_\infty &= \gamma \sup_{s\in\mathcal{S}} \left| \sum_{s'} \bar{\mathcal{T}}(s,u,s')(Q(s',u') - Q'(s',u')) \right| \\ &\leq \sup_{s\in\mathcal{S}} \sum_{s'} \bar{\mathcal{T}}(s,u,s')|Q(s',u') - Q'(s',u')| \\ &\leq \gamma \sup_{s\in\mathcal{S}} \sum_{s'} \bar{\mathcal{T}}||Q - Q'||_\infty \\ &= \gamma||Q - Q'||_\infty\end{aligned}$$

Having this property is of major importance since now an algorithm could start from an arbitrary initialization of the Q-function and then apply the Bellman operator in sequence and it could be proven using *Banach's fixed-point* theorem that the function will converge to the operator's fixed point (i.e., Q), see [69, 101].

Now from the optimal Bellman operation, the following can be derived:

$$\begin{aligned}||T^{\pi^\star}(Q) - T^{\pi^\star}(Q')||_\infty &\leq \sup_{\langle s,u\rangle\in\mathcal{S}\times\mathcal{U}} \sum_{s'} \bar{\mathcal{T}}(s,u,s') \left| Q(s') - Q'(s') \right| \\ &\leq \gamma \sup_{\langle s,u\rangle\in\mathcal{S}\times\mathcal{U}} \sum_{s'} \bar{\mathcal{T}}(s,u,s')||Q - Q'||_\infty \\ &= \gamma||Q - Q'||_\infty\end{aligned}$$

Therefore, $T^{\pi^\star}$ is also a contraction and has a fixed-point. From this the following theorem can be presented according to [69]:

**Theorem** Let Q be the fixed point of $T^{\pi^\star}$, if there is a policy greedy in Q: $T^\pi(Q) = T^{\pi^\star}(Q)$. Then $Q = Q^\star$ and $\pi$ is an optimal policy.

The proof can be seen in [101] and only a sketch is provided here:

*Proof.* For any stationary policy $\pi$, $T^\pi(Q) \leq T^{\pi^\star}(Q)$, therefore $Q^\pi \leq T^{\pi^\star}(Q^\pi)$. For any $\tau \geq 0$, the following is attained:

$$Q^\pi \leq (T^{\pi^\star})^\tau Q^\pi \tag{2.24}$$

Pick a policy $\pi^\star$ such that $T^\pi(Q) = T^{\pi^\star}Q$. Since $Q$ is the fixed point of Equation 2.24, then $T^\pi Q = Q$. Since $T^\pi$ is a contraction and has a fixed point $Q^\pi = Q$, then $Q^\star = Q$ and $\pi^\star$ is optimal. □

Next three of the widely used value iteration algorithms are detailed. Essentially,

these are variants of Q-learning for each the deterministic, and stochastic cases. Moreover, their online variants are explained.

**Q-Iteration** First Q-iteration, one of the widely used value iteration based RL algorithm is described. The idea behind Q-iteration is to update the Q-function at each iteration, using the optimal Bellman operator. Firstly, two variants of the Q-iteration algorithm are shown in Algorithm 4 and 5. Both of the presented algorithms work in a finite state space MDP setting. The main difference between them is that Algorithm 4 operates within a deterministic setting, while Algorithm 5 works in a stochastic setting. Both are model based, in the sense that they require a transition and reward model. They start by initializing the Q-function as shown in line 2 of the algorithms. According to the previous discussion, updating an arbitrary Q-function using the Bellman operator, will end up converging to $Q^\star$ after $\tau$ iterations. These updates are shown in line 5 of both algorithms. The main differences is that in Algorithm 4 the update is performed using the deterministic Bellman operator, while in Algorithm 5 the Q-function update is conducted using the stochastic operator.

**Algorithm 4** Q-Iteration for Deterministic MDPs with Finite State Space

1: **Input:** Deterministic dynamics $\mathcal{T}$, reward function $\mathcal{R}$, discount factor $\gamma$, action space $\mathcal{U}$, and state space $\mathcal{S}$
2: Initialize the Q-function
3: **for** $\tau = 0, 1, \dots$ **do**
4: **for** all $\langle s, u \rangle$ **do**
5: $Q_{\tau+1}(s,u) = \mathcal{R}(s,u) + \gamma \max_{u' \in \mathcal{U}} \left[ Q_\tau(\mathcal{T}(s,u), u') \right]$
6: **end for**
7: **end for**
8: **Return:** $Q^\star = Q_\tau$

**Algorithm 5** Q-Iteration for Stochastic MDPs with Finite State Space

1: **Input:** Stochastic dynamics $\bar{\mathcal{T}}$, stochastic reward function $\tilde{\mathcal{R}}$, discount factor $\gamma$, action space $\mathcal{U}$, and state space $\mathcal{S}$
2: Initialize the Q-function
3: **for** $\tau = 0, 1, \dots$ **do**
4: **for** all $\langle s, u \rangle$ **do**
5: $Q_{\tau+1}(s,u) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{T}}(s,u,s') \left[ \tilde{\mathcal{R}}(s,u,s') + \gamma \max_{u' \in \mathcal{U}} Q_\tau(s',u') \right]$
6: **end for**
7: **end for**
8: **Return:** $Q^\star = Q_\tau$

In Algorithm 6, a variant of Q-learning for continuous state spaces is presented.

Again the algorithm is model based in the sense that it requires a transition and reward model. Furthermore, the variants shown in Algorithm 6 is deterministic. The idea here is similar to the previous Q-learning algorithms. However, the Q-function can not be represented fully and therefore a parametrization as shown in line 5 is used. The values of these parameters are the updated by solving the least-squares minimization problem in line 7 of Algorithm 6. This is repeated until the optimal parameterization, $\boldsymbol{\gamma}^\star$ of the Q-function is attained, which is then returned.

**Algorithm 6** Least Squares Approximate Q-Iteration for Deterministic MDPs

1: **Input:** Deterministic dynamics $\mathcal{T}$, deterministic reward function $\mathcal{R}$, discount factor $\gamma$, action space $\mathcal{U}$, and state space $\mathcal{S}$, approximation mapping $\boldsymbol{\Omega}$, samples $\{s^{(i)}, u^{(i)}\}_{i=1}^m$
2: Initialize the parameter vector $\boldsymbol{\gamma}$
3: **for** $\tau = 0, 1, \ldots$ **do**
4: **for** $i = 1, \ldots, m$ **do**
5: $Q_{\tau+1}^{i(r)}(s^{(i)}, u^{(i)}) = \mathcal{R}(s^{(i)}, u^{(i)}) + \gamma \max_{u' \in \mathcal{U}} \left[ [\boldsymbol{\Omega}(\boldsymbol{\gamma}_\tau)] \left( \mathcal{T}\left(s^{(i)}, u^{(i)}\right), u' \right) \right]$
6: **end for**
7: Find $\boldsymbol{\gamma}_{\tau+1}$ by solving:

$$\boldsymbol{\gamma}_{\tau+1} \in \arg\min_{\boldsymbol{\gamma} \in \boldsymbol{\Gamma}} \sum_{i=1}^{m} \left( Q_{\tau+1}^{i(r)}\left(s^{(i)}, u^{(i)}\right) - [\boldsymbol{\Omega}(\boldsymbol{\gamma})]\left(s^{(i)}, u^{(i)}\right) \right)^2$$

8: **end for**
9: **Return:** $\hat{\boldsymbol{\gamma}}^\star = \boldsymbol{\gamma}_{\tau+1}$

In the model free case, another type of approximation is needed. Namely, since the model is not present the algorithm needs to be provided by samples representing the model. These in turn are used to approximate the optimal Q-function that represents the optimal policy. Fitted Q-Iteration is one form of these algorithms [17, 88]. Further, there are two types of approximation of the Q-function: (1) parametric, and (2) nonparametric approximation. Both variants for the offline case are presented next.

Fitted Q-Iteration with parametric approximation is presented in Algorithm 7. First the algorithm requires: (1) discount factor, (2) parametric approximation function, and (3) samples in form of transitions, as shown in line 1. The samples are used in order to determine the next iteration Q-values, line 5. These values are then considered as outputs for minimizing the sum of squared errors as shown in line 7. The result of this minimization is the update of the approximation parameters. This process is repeated until the parameters have converged, which are then returned as the optimal parametric vector.

The other variant of Fitted Q-iteration is with the usage of nonparametric regression techniques. Algorithm 8 describes this variant. The main difference to Algorithm 7 is the usage of nonparametric approximation to fit the Q-function. It is

**Algorithm 7** Least Squares Fitted Q-Iteration with Parametric Approximation

**Input:** Discount factor $\gamma$, approximate mapping $\boldsymbol{\Omega}$, samples $\{\langle s^{(i)}, u^{(i)}, s'^{(i)}, r^{(i)}\rangle\}_{i=1}^{m}$
Initialize the parameter vector $\boldsymbol{\gamma}$
**for** $\tau = 0, 1, \ldots$ **do**
  **for** $i = 1, \ldots, m$ **do**

$$Q_{\tau+1}^{i(r)}(s^{(i)}, u^{(i)}) = r^{(i)} + \gamma \max_{u' \in \mathcal{U}} [\boldsymbol{\Omega}(\boldsymbol{\gamma}_\tau)](s'^{(i)}, u') \text{ (deterministic setting)}$$

$$Q_{\tau+1}^{(r)}(s^{(i)}, u^{(i)}) = \mathbb{E}_{\{s' \sim \bar{\mathcal{T}}(s,u,.)\}} \left[ \tilde{\mathcal{R}}(s^{(i)}, u^{(i)}, s'^{(i)}) + \gamma \max_{u' \in \mathcal{U}} [\boldsymbol{\Omega}(\boldsymbol{\gamma}_\tau)](s', u') \right] \text{ (stochastic setting)}$$

  **end for**
  Find $\boldsymbol{\gamma}_{\tau+1}$ by solving:

$$\boldsymbol{\gamma}_{\tau+1} \in \arg\min_{\boldsymbol{\gamma} \in \boldsymbol{\Gamma}} \sum_{i=1}^{m} \left( Q_{\tau+1}^{i(r)} \left( s^{(i)}, u^{(i)} \right) - [\boldsymbol{\Omega}(\boldsymbol{\gamma})] \left( s^{(i)}, u^{(i)} \right) \right)^2$$

**end for**
**Return:** $\hat{\boldsymbol{\gamma}}^\star = \boldsymbol{\gamma}_{\tau+1}$

important to mention that essentially using nonparametric approximation for fitted Q-iteration is preferred. The main reason is that the approximation of the Q-function is performed on a number of samples. These approximation should be representative enough if this approximated optimal function is to perform well on the real system in areas not visited by the samples.

**Algorithm 8** Least Squares Fitted Q-Iteration with Nonparametric Approximation

**Input:** Discount factor $\gamma$, samples $\{\langle s^{(i)}, u^{(i)}, s'^{(i)}, r^{(i)}\rangle\}_{i=1}^{m}$
Initialize the function approximator
**for** $\tau = 0, 1, \ldots$ **do**
  **for** $i = 1, \ldots, m$ **do**
    $Q_{\tau+1}^{i(r)}(s^{(i)}, u^{(i)}) = r^{(i)} + \gamma \max_{u' \in \mathcal{U}} \hat{Q}_\tau(s'^{(i)}, u')$
  **end for**
  Solve for $\hat{Q}_{\tau+1}$ by performing nonparametric regression on $\mathcal{D} = \{\langle s^{(i)}, u^{(i)} \rangle, Q_{\tau+1}^{i(r)}\}_{i=1}^{m}$
**end for**
**Return:** $\hat{Q}^\star = \hat{Q}_{\tau+1}$

### Policy Iteration

In policy iteration, the other subcategory of RL algorithms, learners first evaluate a policy, and then use this evaluation to generate new improved policies. In other words, starting from an initial policy $\pi_0$, the learner first evaluates this policy by constructing its value function, and then it takes greedy actions in that value function, in order

to generate better policies. First policy iteration algorithms for the finite state space MDPs are described. These are then extended to the continuous state space setting.

**Policy Iteration: Finite State Space Case** A general description of policy iteration is shown in Algorithm 9. There are two main steps: (1) policy evaluation, and (2) policy improvement. The important step to note here is that of policy evaluation (i.e., line 3 of Algorithm 9). For details on the deterministic case the reader is referred to [17]. Here only the stochastic evaluation will be detailed. Algorithm 10 shows a pseudo-code of the policy evaluation in the finite state-space stochastic MDP setting. Given a policy $\pi$, the main idea is simply to compute the Q-function of $\pi$. This procedure is computed as shown in line 5 of the algorithm.

**Algorithm 9** Policy Iteration with Q-functions

1: Initialize the policy, $\pi_0$
2: **for** $\tau = 0, 1, \ldots$ **do**
3: Evaluate policy by finding $Q^{\pi_\tau}$
4: Improve policy using: $\pi_{\tau+1} \in \arg\max_{u \in \mathcal{U}} Q^{\pi_\tau}(s, u)$
5: **end for**
6: **Return:** $\pi^\star = \pi_\tau$, and $Q^\star = Q^{\pi_\tau}$

**Algorithm 10** Policy Evaluation in Stochastic MDPs with Finite State Space

1: Policy $\pi$ to be evaluated, transition dynamics $\bar{\mathcal{T}}$, reward function $\tilde{\mathcal{R}}$, discount factor
2: Initialize the Q-function
3: **for** $\tau_1 = 0, 1, \ldots$ **do**
4: **for** all $\langle s, u \rangle$ **do**
5: $Q^\pi_{\tau_1+1} = \sum_{s' \in \mathcal{S}} \bar{\mathcal{T}}(s, u, s') \left[ \tilde{\mathcal{R}}(s, u, s') + \gamma Q^\pi_{\tau_1}(s', \pi(s')) \right]$
6: **end for**
7: **Return:** $Q^\pi = Q^\pi_{\tau_1}$

The policy iteration algorithms presented above are model-based, where a transition and a reward model were needed. In the class of model-free policy iteration algorithms, SARSA [98] is considered the most famous. SARSA starts with an arbitrary Q-function and updates it at each step using:

$$Q_{t+1}(s_t, u_t) = Q_t(s_t, u_t) + \alpha_t \underbrace{\left[ r_{t+1} + \gamma Q_t(s_{t+1}, u_{t+1}) - Q_t(s_t, u_t) \right]}_{\text{temporal difference error}} \tag{2.25}$$

with $\alpha_t \in ]0, 1]$ is the learning rate.

The overall SARSA pseudo-code is shown in Algorithm 11. After initializing the Q-function, line 2, an action is greedily chosen according to:

$$\begin{cases} u \in \arg\max_u Q(s,u) \text{ with probability } 1-\epsilon \\ \text{random action from } \mathcal{U} \text{ with probability } \epsilon \end{cases} \tag{2.26}$$

This action is then applied to the system, where the successor state as well as the reward are measured, line 6. These are then used to perform the update of the Q-values as shown in line 8 of the algorithm.

**Algorithm 11** SARSA with $\epsilon$-greedy exploration

1: **Input:** Discount factor $\gamma$, exploration schedule $\{\epsilon_t\}_{t=0}^{\infty}$, learning rate schedule $\{\alpha_t\}_{t=0}^{\infty}$
2: Initialize the Q-function
3: Measure initial state, $s_0$
4: Choose action, $u_0$ according to $\epsilon$-greedy exploration, with $\epsilon_0$
5: **for** $t = 0, 1, \ldots$ **do**
6: Apply $u_t$, measure $s_{t+1}$, and $r_{t+1}$
7: Choose action $u_{t+1}$ according to $\epsilon$-greedy exploration, with $\epsilon_{t+1}$
8: Perform update according to Equation 2.25
9: **end for**

**Policy Iteration: Continuous State Space Case** In continuous state spaces, the previous algorithms can not be directly applied. The main problem arises in the policy evaluation step, which can not be solved exactly [17, 51]. Policy evaluation in the continuous state space case, is a hard problem as it also requires the solution for a Bellman equation. On the other hand, policy improvement is an easier problem[17]. Often an explicit representation of the policy can be avoided. This can be performed, by computing improved actions on demand from the current value function [17]. For an overview of different approximate policy iteration algorithms the reader is referred to [17]. In this thesis, Least Squares Policy Iteration (LSPI), one of the widely adopted approximate policy iteration algorithms is adopted. Therefore, the discussion is restricted to LSPI. This discussion is adopted from [17, 51].

To start the discussion the projected Bellman equations need to be introduced. For the ease of discussion assume that the state and action spaces are finite (i.e., $\mathcal{S} = \{s^{(i)}\}_{i=1}^{|\mathcal{S}|}$, and $\mathcal{U} = \{u^{(i)}\}_{i=1}^{|\mathcal{U}|}$). Now the policy evaluation mapping $T^\pi$ can be written as:

$$[T^\pi(Q)](s,u) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{T}}(s,u,s') \left[ \tilde{\mathcal{R}}(s,u,s') + \gamma Q(s', \pi(s')) \right] \tag{2.27}$$

[17] Policy improvement can also be a very hard problem, in case the action space is large.

In the linearly parameterized case the approximated Q-function can be written as:

$$\hat{Q}^{\pi}(s,u) = \boldsymbol{\gamma}_{\boldsymbol{\pi}}^{\boldsymbol{T}} \boldsymbol{\Phi}(s,u)$$

with $\boldsymbol{\gamma}_{\pi}$ being the parametrization vector of the policy $\pi$, and $\boldsymbol{\Phi} = [\phi^{(1)}(s,u), \ldots, \phi^{(k)}(s,u)]^T$ being the vector of the k-basis function. This approximate function satisfies the following version of the projected Bellman equation:

$$\hat{Q}^{\pi} = (\mathcal{P}^{\mathbf{w}} \mathrm{o}\ T^{\pi})(\hat{Q}^{\pi})$$

where $\mathcal{P}^{\mathbf{w}}$ is a weighted least-squares projection on the space of representable Q-functions (i.e., $\{\boldsymbol{\Phi}^T(s,u)\boldsymbol{\gamma} | \boldsymbol{\gamma} \in \mathbb{R}^k\}$). This projection is defined as:

$$[\mathcal{P}^{\mathbf{w}}(Q)](s,u) = \hat{\boldsymbol{\gamma}^T} \boldsymbol{\Phi}(s,u) \text{ with}$$
$$\hat{\boldsymbol{\gamma}} \in \arg\min_{\boldsymbol{\gamma}} \sum_{\langle s,u \rangle \in \mathcal{S} \times \mathcal{U}} w(s,u) \Big( \boldsymbol{\gamma}^T \boldsymbol{\Phi}(s,u) - Q(s,u) \Big)^2$$

with $w : \mathcal{S} \times \mathcal{U} \to [0,1]$.

To facilitate the representation of the topic, next the projected Bellman equations are written in a matrix form. First, define the matrix Bellman operator as follows:

$$\mathbf{T}^{\pi}(\mathbf{Q}) = \tilde{\mathbf{R}} + \gamma \bar{\mathbf{T}} \boldsymbol{\pi} \mathbf{Q}$$

with:

1. $\mathbf{Q} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{U}|}$ with $\mathbf{Q}_{[i,j]} = Q(s^{(i)}, u^{(j)})$
2. $\tilde{\mathbf{R}} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{U}|}$ with $\tilde{\mathbf{R}}_{[i,j]} = \sum_{i'} \bar{\mathcal{T}}(s^{(i)}, u^{(j)}, s^{(i')}) \tilde{\mathcal{R}}(s^{(i)}, u^{(j)}, s^{(i')})$
3. $\bar{\mathbf{T}} \in \mathbb{R}^{|\mathcal{S}||\mathcal{U}| \times \mathcal{U}}$ with $\bar{\mathbf{T}}_{[i,j],i'} = \bar{\mathcal{T}}(s^{(i)}, u^{(j)}, s^{(i')})$
4. $\boldsymbol{\pi} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}||\mathcal{U}|}$ is a matrix representation of the policy with:

$$\boldsymbol{\pi}_{i',[i,j]} = \begin{cases} 1 \text{ and } \pi(s^{(i)}) = u^{(j)} & \text{if } i' = i \\ 0 & \text{otherwise} \end{cases}$$

In the approximate setting define the basis function matrix $\boldsymbol{\Phi}^{\dagger} \in \mathbb{R}^{|\mathcal{S}||\mathcal{U}| \times k}$ and the weight diagonal matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{S}||\mathcal{U}| \times |\mathcal{S}||\mathcal{U}|}$ with:

$$\boldsymbol{\Phi}^{\dagger}_{[i,j],l} = \phi^{(l)}(s^{(i)}, u^{(j)})$$
$$\mathbf{W}_{[i,j],[i,j]} = w(s^{(i)}, u^{(j)})$$

The approximation of the Q-function for all states and actions can now be written as:

$$\hat{\mathbf{Q}} = \mathbf{\Phi}^{\dagger}\boldsymbol{\gamma}$$

and therefore, the projected Bellman equation can be rewritten as:

$$\boldsymbol{\mathcal{P}}^{\mathbf{W}}\mathbf{T}^{\pi}(\hat{\mathbf{Q}}^{\pi}) = \hat{\mathbf{Q}}^{\pi}$$

here $\boldsymbol{\mathcal{P}}^{\mathbf{W}} = \mathbf{\Phi}^{\dagger}(\mathbf{\Phi}^{T\dagger}\mathbf{W}\mathbf{\Phi}^{\dagger})^{-1}\mathbf{\Phi}^{T\dagger}\mathbf{W}$, see [17, 51].

After some algebraic manipulations the following can be reached:

$$\mathbf{\Phi}^{T\dagger}\mathbf{W}\mathbf{\Phi}^{\dagger}\boldsymbol{\gamma}_{\pi} = \gamma\mathbf{\Phi}^{T\dagger}\mathbf{W}\bar{\mathbf{T}}\boldsymbol{\pi}\mathbf{\Phi}^{\dagger}\boldsymbol{\gamma}_{\pi} + \mathbf{\Phi}^{\dagger}\mathbf{W}\tilde{\mathbf{R}} \tag{2.28}$$

Following [17], defining the following matrices and vectors:

$$\begin{aligned}\varpi &= \mathbf{\Phi}^{T\dagger}\mathbf{W}\mathbf{\Phi}\\ \beth &= \mathbf{\Phi}^{T\dagger}\mathbf{W}\bar{\mathbf{T}}\boldsymbol{\pi}\mathbf{\Phi}^{\dagger}\\ \Upsilon &= \mathbf{\Phi}^{T\dagger}\mathbf{W}\tilde{\mathbf{R}}\end{aligned}$$

Equation 2.28 could be written in the following form:

$$\varpi\boldsymbol{\gamma}_{\pi} = \gamma\beth\boldsymbol{\gamma}_{\pi} + \Upsilon \tag{2.29}$$

The defined matrices and vectors can also be rewritten as a sum of simpler matrices and vectors. For details on these the reader is referred to [17, 51].

Having written the projected Bellman equations in a matrix format LSPI can now be introduced. In Algorithm 12, the main steps followed by LSPI are shown.

**Algorithm 12** Least Squares Policy Iteration

1: **Input:** Discount factor $\gamma$, Basis functions $\phi^{(1)}, \ldots, \phi^{(k)} : \mathcal{S} \times \mathcal{U} \rightarrow \mathbb{R}$, samples $\{s^{(i)}, u^{(i)}, s'^{(i)}, r^{(i)}\}_{i=1}^{m}$
2: Initialize the policy $\pi_0$
3: **for** $\tau = 0, 1, \ldots,$ **do**
4: Evaluate policy $\pi_{\tau}$ generating $\boldsymbol{\gamma}_{\tau}$
5: Improve policy using $\pi_{\tau+1}(s) = u$, where $u \in \arg\min_{u' \in \mathcal{U}} \mathbf{\Phi}(s, u')\boldsymbol{\gamma}_{\tau} \forall\ s\ \in \mathcal{S}$
6: **end for**
7: **Return:** $\hat{\pi}^{\star} = \pi_{\tau+1}$

The algorithm requires a discount factor, a set of basis functions defined over the state-actions space, and samples in form of transition quadruples. LSPI starts by initializing a policy $\pi_0$. As in any other policy iteration algorithm, LSPI first evaluates the policy as shown in line 4 of Algorithm 12. There are different forms for evaluating

the policy, such as least squares temporal difference for Q-function (LSTD-Q) [51]– detailed in Algorithm 13, and least squares policy evaluation for Q-function (LSPE-Q) [52]. These evaluation algorithm return a vector of parameters, that is then used by LSPI to improve the policy, line 5 of Algorithm 12. After the convergence criterion has been reached, the approximation of the optimal policy is then returned.

To evaluate a policy, this thesis adopts LSTD-Q that is described in Algorithm 13. LSTD-Q is a one shot algorithm and the parameter vector computed does not depend on the order in which the samples are processed.

**Algorithm 13** Least Squares Temporal Difference for Q-functions

1: **Input:** Discount factor $\gamma$, Basis functions $\phi^{(1)}, \ldots, \phi^{(k)} : \mathcal{S} \times \mathcal{U} \rightarrow \mathbb{R}$, samples $\{s^{(i)}, u^{(i)}, s'^{(i)}, r^{(i)}\}_{i=1}^{m}$, and a policy $\pi$
2: Initialize the matrices $\varpi^{(0)}$, $\beth^{(0)}$, and the vector $\Upsilon^{(0)}$
3: **for** $i = 1, 2, \ldots, m$ **do**
4: Update the following:

$$\begin{aligned}\varpi^{(i)} &= \varpi^{(i-1)} + \mathbf{\Phi}(s^{(i)}, u^{(i)})\mathbf{\Phi}^T(s^{(i)}, u^{(i)}) \\ \beth^{(i)} &= \beth^{(i-1)}\mathbf{\Phi}(s^{(i)}, u^{(i)})\mathbf{\Phi}^T(s'^{(i)}, \pi(s'^{(i)})) \\ \Upsilon^{(i)} &= \Upsilon^{(i-1)} + \mathbf{\Phi}(s^{(i)}, u^{(i)})r^{(i)}\end{aligned}$$

5: **end for**
6: Solve $\frac{1}{m}\varpi^{(m)}\hat{\boldsymbol{\gamma}}_\pi = \gamma\frac{1}{m}\beth^{(m)}\hat{\boldsymbol{\gamma}}_\pi + \frac{1}{m}\Upsilon^{(m)}$
7: **Return:** $\hat{\boldsymbol{\gamma}}_\pi$

The last subcategory of RL algorithm is that of policy search. Algorithms, such as natural actor-critic, and neural fitted natural actor critic are variants of this realm. Although, interesting and have shown various successes in a lot of real-world robotic applications [38, 78], they are out of the scope of this thesis. Interested readers are referred to [17, 74] for a thorough discussion of this topic.

# Part I

## Effective Learning and Transfer

# 3

# Transfer for Reinforcement Learning

**This chapter is based on:** *H. B. Ammar, S. Chen, K. Tuyls, and G. Weiss, "Reinforcement Learning Transfer a Survey and Formal Framework," German AI, KI Knstliche Intelligenz, 2013.*

A survey of different transfer learning algorithms is provided in this chapter. First of all, motivations behind using transfer learning methods to speed up reinforcement learning algorithms are presented. Mainly, these argue that RL was optimistically defined as being the solution for optimal control problems with tabula rasa learners. Second of all, different proposed domains dealing with RL challenges, such as imitation learning and reward shaping, are briefly reviewed. Third, a formalization of the transfer in reinforcement learning problem is provided, paving the way for a theoretical classification of the transfer learning algorithms.

## 3.1 Motivations and Related Paradigms

When humans attempt to acquire new behaviors, providing negative or positive reinforcement has shown successful results [23]. Reinforcement learning was initially inspired by this trait [98], where it was believed that if computer agents are provided with such reinforcement (i.e., reward) they potentially can learn an optimal behavior in an unknown environment. This section argues that such a definition of RL is overoptimistic.

First, when attempting to gain a new behavior, humans never start from scratch. Objective thinking is impossible in everyday life. Almost always, new decisions made by humans are biased by already learnt knowledge in some similar tasks. Kamal Jumblat argues that: "Complete abstract and objective thinking at such low levels is almost impossible". Even infants never start from scratch when learning a new task. There is rather a huge history of evolution encoded in their genetics aiding them

in improving their learning abilities.Therefore, if RL agents had to mimic human learning, *tabula rasa* learners seem to be unrealistic and far-fetched.

Second, tabula rasa RL algorithms seems to be extremely hard to be successfully applied in real-world applications with high dimensional state and/or action spaces. To the best of the author's knowledge, the most impressive results in reality are these reported in [17], where mostly the state and/or action spaces were limited in some sense. On the other hand, providing tabula rasa RL agents with additional knowledge has shown impressive improvements and outstanding results. For instance, Ng et. al [1], have shown successful autonomous control of a helicopter, a highly nonlinear and chaotic system. However, the authors approached this control problem from a slightly different perspective. Their proposed method, apprenticeship learning, required a set of successful demonstrations from an expert. These were used to infer a reward function for the underlying MDP that is then solved. Others, such as [46, 79], have also shown interesting results in real-world robotic applications using RL. These algorithms also needed additional information in form of demonstrations to start from and improve on in a sequel of attaining an optimal behavior.

Therefore, it seems that the definition of RL as being a potential solution for optimal control problems with no information is slightly far-fetched. Moreover, it is apparent that providing the agents with additional knowledge from the same domain has elevated these RL challenges. Extending on providing such additional information different paradigms have been proposed. Next, these are briefly surveyed and their relations to transfer learning is described.

## Related Paradigms

To tackle reinforcement learning speed and quality challenges different approaches have been proposed. These include: (1) life-long learning, (2) imitation learning, (3) reward shaping, (4) human advice, and (5) transfer learning.

### Lifelong Learning

Lifelong learning [110] was initially proposed in a supervised learning setting. This approach assumes that the learner faces a whole collection of problems over its life time. The idea is that such an agent can generalize better by re-using the knowledge from its $n-1$ tasks when faced with then new $n^{\text{th}}$. Others, such as [100, 107], extended lifelong learning to the context of RL, where an agent learning in an environment for an extended period of time will necessarily have to solve a sequence of tasks. Furthermore, this work focused on the importance of tracking a solution, rather than only looking for one convergent optimal behavior. Although interesting,

these approaches mostly assume that the newly encountered tasks, are sub-domains of the original. Exploiting such locality has shown interesting results.

Solving sub-problems, in a bigger MDP can be regarded as a transfer learning. If sub-problems were determined and for each an optimal behavior can be attained, then transferring such *options* might aid learning in a new task. This has been already proposed in transfer literature as described later.

### Imitation Learning

In imitation learning there exists roughly two fields. The first is learning from demonstration (LfD), where the learner tries to reproduce already provided "good" demonstration. While in the second, apprenticeship learning, the learner tries to infer a reward model from the expert demonstrations. This model is then used to solve the underlying MDP. It is worth noting that in apprenticeship learning, the expert is assumed to encode the optimal behavior that the learner tries to achieve. On the other hand, this form of learning is more flexible than LfD since as the reward function is attained, the agent is allowed to explore in the environment to reach optimal behaviors not conveyed in the original demonstrations. In other words, in LfD the learned policy is necessarily defined only in those states encountered, and for those corresponding actions taken, during the example executions [6, 71], while in apprenticeship learning such a condition does not necessarily hold.

Apprenticeship learning can be framed within the transfer learning for RL domain, whereby additional information for the RL agent in form of demonstrations are provided. This prior knowledge can be used to provide a good starting behavior for the agent to learn from. Starting from this knowledge and improving upon it, the agent is more likely to reach the optimal behavior faster.

On the other hand, LfD is essentially a supervised learning problem [6] which of course belongs to the transfer in supervised learning world. However, these relations to SL are out of the scope of this thesis. For a detailed discussion the reader is referred to [76].

### Reward Shaping

In reward shaping, the reward function is altered or shaped locally such that faster behavior is attained. The idea is to make local behavior apparent through localized advice which has shown to improve the learning efficacy [47, 53, 73]. Most of the proposed approaches in shaping actually require substantial human engineering to forbid the emergence of unwanted behaviors. These problem have been elevated in [47]. The agent can automatically learn a valid shaping function by relying on a sequence of tasks with increasing complexity. However, in [47] the agent required

an additional representation, the *agent-sapce*, to successfully learn such a function. Furthermore, not any function can serve as a valid shaping function. These have to satisfy certain conditions as discussed in [73], to ensure successful optimal behavior.

Typical, reward-shaping algorithms operate within the same task domain, where the state space representations of the different tasks encountered by the agent are the same.

Reward shaping can be framed within the context of transfer in reinforcement learning. More specifically, reward shaping can be regarded as a specific case of inter-task transfer, where the knowledge either arrives from a human or a sequence of same domain tasks. The intertask mapping then learns how to improve the description of the reward function such that the overall performance in a target task is improved.

#### Human Advice

Human advice aims at integrating humans to an RL agent. For instance, the human can provide action suggestions to the agent [61, 111, 112] or guide the agent through online feedback [43, 45]. In human advice, two major problems standout. First, there exists the need for optimally integrating the human in the loop of an RL agent, which is a hard problem. The second, and maybe the most prominent problem, is that in all these approaches the human is considered as being the expert. This is highly plausible, however, the human might be an expert in the instantaneous sense, but not in the total discounted pay-off one. To clarify, consider the problem of controlling a helicopter to hover. An action taken by an agent at a time step $t$ might cause the aircraft to crash after a substantial delay and not instantaneously. Only experienced pilots, might posses the ability to detect such a behavior. Moreover, the question of how to combine the instantaneous environmental reward and that of the human is also still an interesting direction of future research in this domain.

Human advice, can also be considered as a transfer learning problem, where the knowledge is transferred from a human expert rather than other agents. In fact, such an approach has been already proposed by [109].

## 3.2 Transfer in Reinforcement Learning

Transfer learning (TL) is another paradigm created to improve learning behaviors of RL agents. The overall framework of transfer learning is shown in Figure 3.1. In essence, the idea is to use extra knowledge available from an external source[1] to help

[1]Please note that some transfer learning algorithm make use of knowledge learned by the *same* agent in different tasks. In this case, this knowledge can also be considered as external to the intended target task.

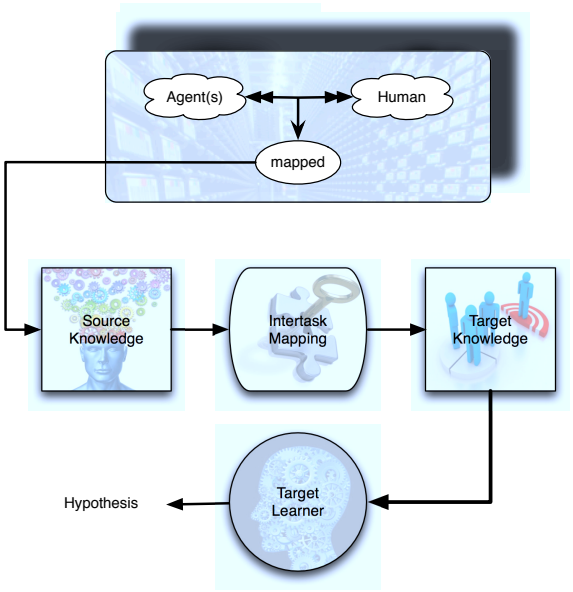



Figure 3.1: The overall transfer framework. Source knowledge is available from either other agents or humans. Generating the source knowledge involves mapping the human and/or agent information in some suitable manner. This is then passed through an inter-task mapping that prepares the source knowledge to be used by the target learner.

learning in target task(s). This source knowledge arrives from either other agents or humans. For instance, in imitation learning, a form of transfer, the source knowledge is available from expert human demonstrations in that same domain. Depending on the source knowledge a mapping is needed to configure it in a way suitable for the target agent. In the imitation learning example, this information is available in the same domain, for instance in form of optimal trajectories, therefore, the *inter-task* mapping is unity. When *transferred*, this additional *target* knowledge is used to bias the target learner in improving its behavior.

Different TL in RL algorithms have been proposed in literature, most of which have been surveyed in [107] and [54]. This survey is not meant to reproduce the others. On the contrary, it is meant to: (1) define the transfer in RL problem, (2) provide a mathematical framework and a taxonomy for the different TL in RL methods, and (3) relate TL to the other related paradigms, explained in Section 3.1.

## Mathematical Framework

A standard learner, $\mathfrak{A}_{learner}$, is simply a mapping from knowledge, $\mathfrak{K}$, to a hypothesis space, $\mathfrak{H}$ (i.e., $\mathfrak{A}_{learner} : \mathfrak{K} \rightarrow \mathfrak{H}$). To clarify, consider the following example, where Fitted-Q Iteration (FQI) is detailed using this description. FQI is an RL algorithm that operates in continuous state space MDPs. Given a set of samples, and a set of basis functions used to approximate the state-action value function, FQI commences by learning the parameters describing the Q-function iteratively. For more details, the reader is referred to Chapter 2.

**Example (FQI):** Consider sample based continuous RL algorithms such as FQI. In the parametric case, the Q-function belongs to the space spanned by the set of $k$ basis functions $\{\phi^{(i)} : \mathcal{S} \times \mathcal{U} \rightarrow \mathbb{R}\}_{i=1}^{k}$. Therefore, a hypothesis in the space $\mathfrak{H}$, is a linear combination of these basis functions, $h(\cdot,\cdot) = \sum_{i=1}^{k} \gamma^{(i)} \phi^{(i)}(\cdot,\cdot) \in \mathfrak{H}$. In FQI, the knowledge acquired by the learner is collected by interacting with the environment to acquire $m$-sample quadruples in the form of $\{\langle s^{(j)}, u^{(j)}, s'^{(j)}, r^{(j)} \rangle\}_{j=1}^{m}$. Thus, the overall knowledge available for FQI is $\mathfrak{K} = \left\{ (\mathcal{S} \times \mathcal{U} \times \mathcal{S} \times \mathcal{R})^m, \mathfrak{B}^k \right\}$, where $\mathfrak{B}$ is the space spanned by the set of basis functions (i.e., $\phi^{(i)} \in \mathfrak{B}$). Using these definitions the FQI learner is a mapping that operates on its available knowledge to map it to the hypothesis space.

As described before, in TL there exists extra knowledge from some external source. This source can be a human or another agent(s). Therefore, this knowledge might sometimes need to be mapped and configured together to provide the source knowledge pool, $\mathfrak{K}_{source} = \chi_{source}\left(\mathfrak{K}_{agent(s)} \times \mathfrak{K}_{human}\right)$, where $\mathfrak{K}_{agent}$ is that knowledge available from other agent(s), $\mathfrak{K}_{human}$ is the human knowledge, and $\chi_{source} : \mathfrak{K}_{agent(s)} \times \mathfrak{K}_{human} \rightarrow \mathfrak{K}_{source}$ is the mapping configuring all this knowledge in a suitable manner for an RL agent. Typically, either one of the above choices is available in almost all transfer learning algorithms provided in literature. When this knowledge has been configured, an inter-task mapping $\chi_{inter} : \mathfrak{K}_{source} \rightarrow \mathfrak{K}_{target}$ acts on the source knowledge to transform it into a starting target knowledge. This is then used by the target learner $\mathfrak{A}_{target}$ to help in improving its performance. Therefore the target learner is defined as:

$$\mathfrak{A}_{target} : \chi_{inter}\left(\mathfrak{K}_{source}\right) \times \mathfrak{K}_{target} \rightarrow \mathfrak{H}_{target}$$

where, $\mathfrak{H}_{target}$ is the hypothesis space of the target task. To clarify, consider the following example where human advice (a form of transfer) is explained.

**Example (Human Advice):** Training an agent manually via evaluative re-

inforcement (TAMER) [42], is a well known human advice framework for RL. In TAMER+RL [41, 44, 45], the authors propose a heuristic mechanism in which human provided reward $\mathcal{R}_H : \mathcal{S} \times \mathcal{U} \rightarrow \mathbb{R}$ is combined with environmental rewards to improve the learning behavior of an RL agent. Of course, the agent has to approximate the human reward and thus $\hat{\mathcal{R}}_H$ is available instead of $\mathcal{R}_H$. In this case, the additional knowledge is a set of human rewards at each state action transition, as well as a set of basis functions to approximate the human reward. The approximation of the human reward needs additional basis functions[2]. Therefore, the available source knowledge is $\mathfrak{K}_{source} = \{\mathcal{R}_H^m, \mathfrak{B}^k\}$, with $\phi^{(i)} \in \mathfrak{B}$. The inter-task mapping, then combines these with the environmental reward to provide a new reward signal for the RL agent. The hope is that if humans were experts at solving the intended task, then combining this reward with the environmental reward will aid the agent in its action selection scheme such as to improve learning. In [45], this combination is done heuristically and the authors propose four different techniques each with its own merits and demerits.

Before diving into the details describing various transfer in RL algorithms, a survey of benchmarks used to evaluate such algorithms are briefly detailed. More specifically, systems used throughout this dissertation, such as, the mountain car, and the inverted pendulum, among others are explained.

## 3.3 Benchmarks

When evaluating transfer in reinforcement learning different benchmarks can be differentiated. These vary from relatively simple tasks, such as grid-worlds, to more complex tasks, such as cart-poles.

In this section the benchmarks used throughout this dissertation are detailed. Firstly, the Mountain Car (MC) task is explained. Secondly, a tougher task, the Inverted Pendulum (IP), is detailed. Third, the Cart-Pole (CP) system is then described. Finally, the simple and double mass tasks are detailed.

### Mountain Car

The mountain car is shown [99] in Figure 3.2. The goal of the agent is to drive the car to an upright position marked by "goal" in the figure. It can choose between two linear forces. Depending on the chosen force the car will move either to the left or to the right. The challenge is that the thrust of the motor is not enough to drive the car to the top-right position in one-shot. Rather it has to oscillate so to gain enough

[2]In case of TAMER, the state space is actually discritized. However, the discussion aims at a general framework.

momentum to drive up the hill. The state space of the environment is described via two continuous state variables, $\langle x, \dot{x} \rangle$, describing the position and the velocity of the car, respectively.

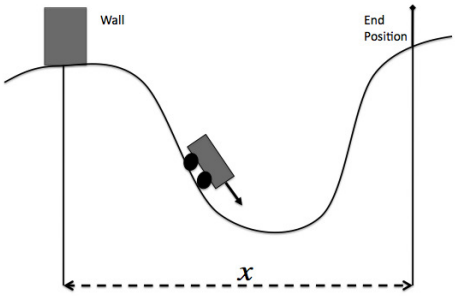



Figure 3.2: Schematic of the mountain car task.

## Inverted Pendulum

The inverted pendulum [17] is shown in Figure 3.3. In this system the pole starts at the bottom and the goal of the agent is to balance the pole at the up top position. The agent is allowed to choose from a set of rotational forces (i.e., torques). Since the available torques are not enough to swing the pole up in one shot, the pole might need to oscillate in order to reach the top position. The states of the system are described via two continuous state space variables, $s = \langle \theta, \dot{\theta} \rangle$, representing the angle and angular velocity of the pole, respectively.

## Cart-pole System

The cart-pole [17, 99] is the system shown in Figure 3.4. The systems consists of a pole mounted on a cart. The agent can choose between a set of linear forces causing the cart to move horizontally on the flat surface. The pole is typically located in a downright position and the goal of the agent is to control the pole in upright position.


Figure 3.3: A high level schematic of a inverted pendulum system.

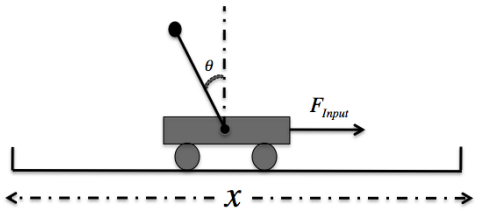
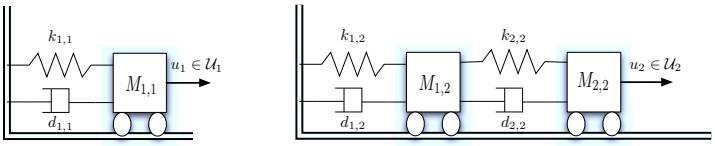



Figure 3.4: A high level schematic of a cart-pole system.




(a) Simple Mass System (b) Double Mass System

Figure 3.5: The single mass spring damper system in (a) and the double mass spring damper system in (b).

The forces allowed to be executed by the agent are restricted in such a way that the cart has to swing back and forth to raise the pole to an upright position. The state space is a four dimensional continuous space consisting of the position and the velocity of the cart, and the angle and angular velocity of the pole.

## Single and Double Mass

The single and double mass systems are shown in Figure 4.2. In the first a mass is attached to a wall with a spring and a damper. It oscillates horizontally, according to the following:

$$M_{1,1}\ddot{x}_{1,1} = u_1 - k_{1,1}x_{1,1} - d_{1,1}\dot{x}_{1,1}$$

where $u_1 \in \mathcal{U}_1$ is the action chosen by the agent, $\ddot{x}_{1,1}$ is the mass acceleration, and $\dot{x}_{1,1}$ is its velocity.

For the double mass system, the state space is described via $\mathcal{S}_2 = \{x_{1,2}, \dot{x}_{1,2}, x_{2,2}, \dot{x}_{2,2}\}$, where $x_{1,2}$ and $\dot{x}_{1,2}$ represent the horizontal position of the first mass $M_{1,2}$, and $x_{2,2}$, and $\dot{x}_{2,2}$ described the position and the velocity of $M_{2,2}$, respectively. The agent is allowed to apply a force from the action set $\mathcal{U}_2$. The system

then oscillates horizontally, according to the following:

$$M_{1,2}\ddot{x}_{1,2} = -k_{1,2}x_{1,2} - d_{1,2}\dot{x}_{1,2} + k_{2,2}(x_{2,2} - x_{1,2}) + d_{2,2}(\dot{x}_{2,2} - \dot{x}_{1,2})$$
$$M_{2,2}\ddot{x}_{2,2} = u_2 + k_{2,2}(x_{1,2} - x_{2,2}) + d_{2,2}(\dot{x}_{1,2} - \dot{x}_{2,2})$$

where $u_2 \in \mathcal{U}_2$, $k_{1,2}$ is the spring constant of the first mass, $d_{1,2}$ is the damping constant of the first mass, $k_{2,2}$ is the spring constant for the second mass, and $d_{2,2}$ is the damper of the second mass.

An important part of the transfer learning literature has been devoted to proposing metrics capable of evaluating any transfer algorithm. These are detailed next.

## 3.4 Metrics

To evaluate the performance of transfer algorithms different metrics have been proposed. In this section, these are briefly surveyed. For a detailed explanation the reader is referred to [107].

### Jumpstart

Jumpstart evaluates the initial performance in a target task. The quantification of this measure depends on the problem at hand. For instance, the initial reward averaged over certain number of episodes can be used as a measure of the jumpstart. To clarify consider transfer from the inverted pendulum (i.e., Figure 3.3) to the cart-pole (i.e., Figure 3.4). Having an algorithm that can can learn the intertask mapping between the IP and CP, transfer can then be conducted. A quantification of the jumpstart metric on the cart-pole system can, for instance, be the number of steps the pole is in an upright position when using the transferred policy. This can be compared to a normal reinforcement learner, or other transfer algorithms to asses the quality of the proposed method.

### Asymptotic Performance

Asymptotic performance measures the final behavior of the learner in a task. Quantifying this measure also depends on the problem at hand. For example, the final attained reward averaged over number of episodes can be used to quantify this measure. Although, in theory, reinforcement learning aims at attaining an optimal behavior, in most complex tasks this is not the case. Generally speaking, if exploration is degraded rapidly, or the value function is not represented in a table format, an optimal behavior is not guaranteed. Moreover, when representing the value function using an

apporximator that might entail local minima, reinforcement learning agents arrive at sub-optimal behaviors. In such scenarios, transfer might provide initial policies that remedy the above problems and improve the final performance. This can be measured and compared among different learners to asses the behavior of a transfer agent.

### Total Reward

The total reward, can be measured as the area under the learning curve. This curve typically has time or sample complexity as its x-axis, and the reward acquired by the agent as its y-axis. Using transfer the overall reward can be improved when compared to a non transfer agent.

### Transfer Ratio

As an extension to the total reward metric, the transfer ratio measures the ratio of the total reward attained by a transfer agent compared to a non-transfer one. If the total reward acquired by a transfer agent is improved, then also the ratio of transfer to non transfer should improve.

### Time to Threshold

This metric requires human intervention to determine a performance threshold for the target agent. The quantification of this metric also depends on the problem at hand. For instance, a certain threshold can be set on the accumulated reward, say $r_{threshold}$. Transfer is then conducted and the agent commences with learning in the target task. The reward in this case is then monitored and time to reach $r_{threshold}$ is recorded. This time can be compared with a normal reinforcement learner in the target task and other transfer algorithms to have a comparison to the proposed method.

Having detailed the benchmarks used as well as the metrics proposed in transfer learning, a taxonomy of various algorithms is presented. Precisely, algorithms are separated based on the depth at which transfer is conducted. Being the basis of such a taxonomy, shallow as well as deep transfer methods are first defined.

### Transfer Learning Taxonomy

TL in RL algorithms can be divided into two categories: (1) shallow, and (2) deep transfer algorithms. In the first, the domains of the source and target task(s) are the same. The differences are in the reward or the transition model.

To clarify, consider the example shown in Figure 3.6. A robot has to reach a certain goal manifested by “G” in the figure. At the middle of the room there is an obstacle

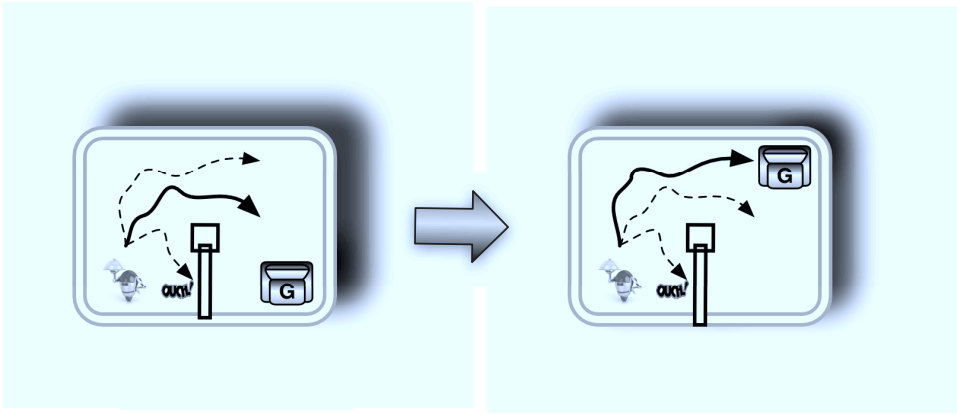



Figure 3.6: Shallow transfer example. The main differences between the tasks, is the location of the goal, and therefore, the reward function. The arrows show different behaviors followed by the agents. The boldfaced line arrows represent the optimal behavior.

that the robot has to avoid. In this example, the left figure is the source task where the goal position is at the bottom right of the environment. The target task, shown on the left side of the figure, differs from the source by the reward function. In other words, the goal position has been changed and the robot has to reach the upper right corner. Of course the robot might have a different transition model too. However, even if both the reward and the transition model have changed, the representation of the state and/or action spaces suffices to perform transfer.

On the other hand, deep transfer is a tougher and more general form of transfer. In deep transfer all the constituents of the tasks are different[3]. An example of deep transfer is shown in Figure 3.7. In this example, the source task agent lives in a one dimensional world, in which it can apply two actions. On the other hand, the target task is a two dimensional problem, and the agent is allowed to apply different types of actions. Furthermore, the reward function and transition probability of both agents can also be different. It is worth noting, that for a problem to be a deep transfer learning one, the difference between the state and action spaces are mostly dimensional. For instance, if the target tasks' state spaces, were simply larger state spaces in number, such problems are still within the shallow transfer realm. The

[3]Automatic transfer has to be able to deal with this general case. In other words, an automatic transfer learning agent should be capable of learning an inter-task mapping at this general level. This dissertation contributes this problem and proposes algorithms that can automatically learn such a mapping at this level.

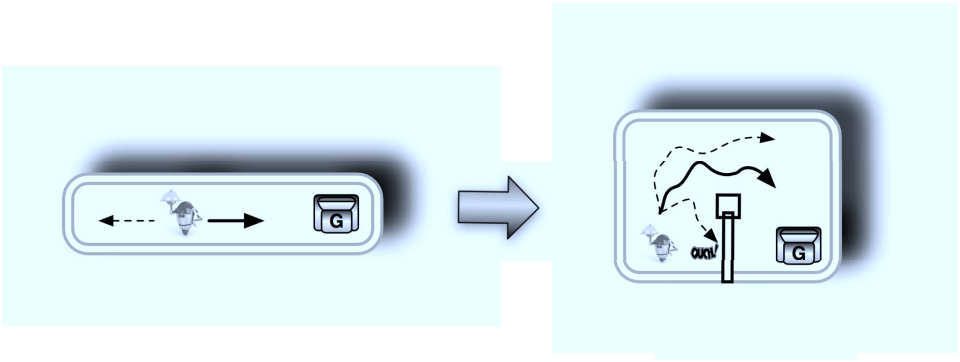



Figure 3.7: Deep transfer example. The source task is a one dimensional problem in which the robot is allowed to apply left of right actions. On the other hand, the target task has a different description of state and action spaces. Furthermore, the reward as well as the transition might also be different.

definitions of shallow and deep transfer are detailed next.

In defining deep and shallow transfer, the source task is assumed to be in a lower dimensional manifolds compared to the target[4].

Spaces might differ in different ways. In shallow transfer, as seen in the previous examples, the state and/or action spaces stay the same, while the reward and/or transition probabilities differ. In some algorithms the state and/or action spaces may differ. However, for a transfer algorithm to stay in the shallow world, the state and/or action spaces vary in a specific way. For instance, if the source task state space was a subspace of that of the target, then transfer will still be shallow. Shallow transfer in such a setting can be also seen as a generalization problem.

**Definition: Shallow Transfer** *Given two MPDs,* $M_1 = \langle \mathcal{S}_1, \mathcal{U}_1, \tilde{\mathcal{T}}_1, \tilde{\mathcal{R}}_1, \gamma_1 \rangle$*, and* $M_2 = \langle \mathcal{S}_2, \mathcal{U}_2, \tilde{\mathcal{T}}_2, \tilde{\mathcal{R}}_2, \gamma_2 \rangle$*, with* $\mathcal{S}_2$ *and* $\mathcal{U}_2$ *are some combination of* $\mathcal{S}_1$ *and* $\mathcal{U}_1$*, shallow transfer occurs if* $\tilde{\mathcal{T}}_1 \neq \tilde{\mathcal{T}}_2$*, or* $\tilde{\mathcal{R}}_1 \neq \tilde{\mathcal{R}}_2$*, or both.*

During its life time an agent might face various types of tasks. Restricting transfer to the shallow case will not cover all possibilities an agent might encounter. Therefore, to create autonomous transfer agents, a more flexible definition is required. This definition need not to restrict in any sense the types of tasks that an agent might encounter. This framework can be covered under deep transfer. In deep transfer all the constituents of the MDPs are allowed to vary. Deep transfer can be defined as follows:

[4]Please note, that this assumption is by no means restrictive.

**Definition: Deep Transfer** *Given two MPDs,* $M_1 = \langle \mathcal{S}_1, \mathcal{U}_1, \tilde{\mathcal{T}}_1, \tilde{\mathcal{R}}_1, \gamma_1 \rangle$*, and* $M_2 = \langle \mathcal{S}_2, \mathcal{U}_2, \tilde{\mathcal{T}}_2, \tilde{\mathcal{R}}_2, \gamma_2 \rangle$*, deep transfer occurs if* $\mathcal{S}_1 \neq \mathcal{S}_2$*,* $\mathcal{U}_1 \neq \mathcal{U}_2$*,* $\tilde{\mathcal{T}}_1 \neq \tilde{\mathcal{T}}_2$*, and* $\tilde{\mathcal{R}}_1 \neq \tilde{\mathcal{R}}_2$*.*

Of course the definition above covers the most general case of transfer in reinforcement learning tasks. In some cases the state and/or action spaces (i.e., domains) might differ, however, the transition probabilities and reward functions are the same in each of the source and target task. This scenario is still considered deep transfer. Transferring correctly between such tasks, however, is a relatively easier problem to solve compared to the deep transfer of the previous definition.

Based on the above definitions a categorization of different transfer algorithms can be represented.

## 3.5 Transfer Learning Categorization

In this section different transfer algorithms will be surveyed. These algorithms can be grouped depending on the depth of transfer. Moreover, in each of the previous classes, further categories can be differentiated depending on the inter-task mapping used.

### Shallow Transfer

As defined previously, shallow transfer occurs when the source and target task have differences in the reward and/or transition probabilities. Typically, in this category the state and/or action spaces remain unchanged. However, in some shallow transfer algorithms, differences between the state and/or action spaces vary. These variation are restricted. The state and/or action spaces grow in number and not in dimensions making the problem substantially easier to solve compared to deep transfer.

In the transfer learning literature most of the proposed methods deal with shallow transfer. Mostly, these differ depending on the assumptions made on learning the intertask mapping and the additional knowledge available.

#### Fixed Domain Shallow Transfer

In this section the domains (i.e., state and action spaces) of the source and target MDPs are fixed. The variations can occur in either the transition probabilities or reward functions. To clarify consider the example shown in Figure 3.8. In the source task (i.e., left image of Figure 3.8), the robot has to reach a goal state at the top-right position of the environment to which it attains a positive reward. In the target task (i.e., right image of Figure 3.8), however, the robot has to reach the goal position at the

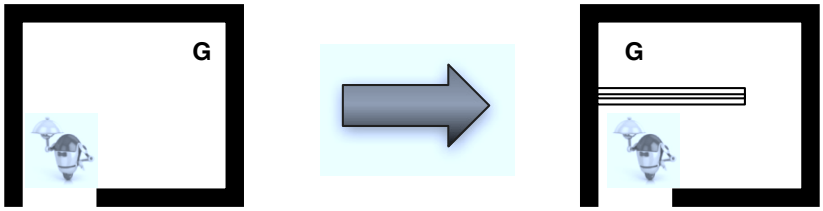



Figure 3.8: An example of shallow transfer. A source (left image) and target task (right image) are shown. The difference between the two tasks are in the goal location (i.e., rewards) and transition probabilities.

top left of the environment, while avoiding an obstacle. Therefore, the main differences between the tasks are the: (1) reward functions, and (2) transition probabilities.

In such scenarios, a mapping between the tasks' transition and reward functions is essential for successful transfer. Various algorithms have been suggested to deal with these problems. In this survey, the aim is not to list all the alternative algorithms within this setting. A broader categorization is rather represented. For a detailed listing the reader is referred to [107].

**Action Space Transfer** Action transfer [90] is an approach that aims at reducing the size of source tasks' action spaces such that learning in a target task is faster. From one source task, random perturbations to generate a set of source tasks is performed. A new action set is generated by removing all non-optimal actions from the original action space. With this new action set, the learner faces a new (target) task. Learning speeds in the target are shown to significantly improve. However, such an improvement arrives at the cost of a loss of optimality of the learned policy. If the source and target tasks are highly similar, this approach is expected to preserve most of the actions required to solve the target task. In action transfer, no knowledge in the target is available, thus $\mathfrak{K}_{target}$ is an empty set. Source task knowledge, $\mathfrak{K}_{source}$, consists of a set of random perturbations of the source task, as well as optimal policies in each of these perturbed sets. The inter-task mapping, $\chi_{inter}$, operates on $\mathfrak{K}_{source}$ to generate $\mathcal{U}_{transfer} \subset \mathcal{U}$, where $\mathcal{U}_{transfer}$ is the action set with non-optimal actions removed, and $\mathcal{U}$ is the original action set of the original source task.

**Option Transfer** Two subcategories can be differentiated. The first is when one source and one target task are available, while the second is when multiple source tasks are given before hand. Firstly, these methods dealing with the former setting are described. Secondly, these operating in a multiple source task setting are detailed.

- **Single Source Task:** Option transfer is another widely used method for transferring between MDPs with the same domains. Three additional assumptions are made in such scenarios. These can be summarized as: (1) the MDPs are assumed to be discrete, (2) the Q-function is represented in a tabular format, and (3) the transition probabilities of the source and target MDPs are the same. Therefore, only the reward functions are allowed to vary between the source and target MDPs.

  The main idea is to exploit the shared dynamics between the MDPs to construct primitive actions that can be used to help in speeding up the learning in the target task. Precisely, a set of state-action-successor-state tuples are collected in the source task. A set of subgoals accompanied with a set of options to reach each of these subgoals are then learned. A learner in the target task can now use these augmented primitives to attain an optimal behavior using the algorithm proposed in [97].

  Although option transfer is widely used, one of the most challenging questions is how to determine "relevant" subgoals. Various approaches have been proposed. In [65], the concept of *bottleneck* state has been introduced. A bottleneck state is defined as that state that is often encountered by an optimal policy. Others [21, 66], define metrics for graph partitioning techniques to identify states that connect different regions of the state-space. These are then considered as subgoals to which options are calculated. A psychology-inspired method aimed at identifying easily-explorable states is detailed in [15]. Finally, a hierarchal Q-learning algorithm based on the notion of access state is described in [34].

- **Multiple Source Tasks:** Based on a set of given source tasks accompanied with their optimal policies, Bernstein [12] introduced the reuse option to speed-up learning on the target task. Given a sufficient number of source tasks, experiments reported in [54] show a dramatic improvement in learning speeds. In [40], a method capable of finding a set of options that reduces the number of iterations required by value-iteration to converge is proposed. This framework is guaranteed to return the optimal set of action primitives to solve the task. Another method that incrementally discovers skills in a hierarchical architecture is proposed in [7].

**Feature Transfer** Feature transfer can also be split into methods that assume the existence of one source MDP, and these that require multiple sources. Next, these are briefly surveyed.

- **Single Source Task:** Another approach for transfer between MDPs with the same domains is proposed in [26, 29, 62, 63]. The method is titled *proto-value functions*. This technique is similar in spirit to that of option transfer. The idea is again to exploit the source task knowledge to attain a representation suitable to the target. Rather than abstracting action primitives, proto-value functions aim at attaining a set of features defining a hypothesis space.

  Proto-value function transfer, relaxes the assumptions made in option transfer. The dynamics of the source and the target tasks are still assumed to be the same, however the value function representation is not assumed to be tabular. On the contrary, the main idea in proto-value function transfer is to discover a feature space that is capable of representing the action-value function as a linear combination of these discovered basis functions (i.e., features). Furthermore, the objective of transferring differs between option and proto-value function transfer. In the former, the goal was to improve the learning speeds in the target, while in the latter, the goal is to achieve a better approximation of the state-action value function of the target task.

  The knowledge available in the source task, $\mathfrak{K}_{source}$, is a set of state-action-reward-successor state tuples. The intertask mapping $\chi_{inter}$, operates on $\mathfrak{K}_{source}$ to discover a set of basis functions. These can then be used to describe the solution space in the target. This space is described as a linear combination of the attained bases. The intuition is that if "general-enough" features will be discovered, these would be able to describe the value function both of the source as well as the target tasks.

  Of course, the toughest challenge is to determine the set of proto-value functions or features. In [63] a method that uses spectral analysis of the Laplacian of the source MDP is introduced. This technique discovers the structure of the manifold that underlies the dynamics of the tasks. In [26] an extension relaxing the similarities between the dynamics in [62] is introduced. Another extension to tasks with different transition and reward functions is proposed in [29].

- **Multiple Source Tasks:** In [31] an unsupervised method to decompose the state space is introduced. Each decomposed task is then solved independently and its value function is used as an additional feature when learning in the target task. A similar method that identifies sub-tasks and features based on exploiting the source task dynamics is introduced in [24]. Walsh [114] deals

with the more general setting in which the MDP is not discrete. The objective is to determine an aggregation of states that are capable of approximating all the optimal value functions of the tasks at hand. In [55] a similar approach to determine a set of features from the source tasks that can then be reused in the target task is explained.

**Instance Transfer** Previously, three methods for options, action space, or features transfer have been described. Transferring instances is another type of knowledge that can be transferred between source and target task(s). The main idea is that transferring source instances may help the target tasks when both tasks are similar enough. If these differed then negative transfer is likely to occur. Lazaric [55] proposed a framework for instance transfer from a set of source tasks to a target task. In this setting knowledge in both the source and target task(s) are available. Precisely, $\mathfrak{K}_{source}$ consists of state-action-reward-successor state tuples and $\mathfrak{K}_{target}$ also consists of state-action-reward-successor state in the target. The main assumption, however, is that the amount of tuples available in the target are much less compared to these available from the set of source tasks. The method makes use of the so-called compliance measure to select these source tasks that are more likely to be similar to the target. A minor modification of this approach using an alternative utility measure is introduce in [3]. Therefore, the intertask mapping, $\chi_{inter}$, can be seen as the measure determining the relevant source task to the specific target. Other instance transfer techniques are also introduced. These deal with a less restrictive setting and belong to the deep transfer domain, and therefore, are explained accordingly.

## Deep Transfer

The methods describe previously have shown significant improvements in transfer between MDPs that share the same domain. Although successful, these proposed techniques operate within the shallow transfer world. In this section, algorithms that deal with deep transfer are surveyed. These aim at solving a substantially harder problem, in which the domains between the source and target tasks differ. Here, an intertask mapping relating the differences between the constituents of the MDPs is essential. Different methods have been proposed. Some ignore the need for this mapping by assuming that the source and target task can both be represented in a higher abstraction space, others assume that this mapping is already delivered to the transfer agent by a designer, and lastly algorithms aiming at directly targeting the learning of this mapping are briefly surveyed.

**Feature Transfer** The main problem when dealing with option transfer in case of different domain tasks, is that options and abstractions discovered in the source can not be directly used in the target. Therefore, the toughest question to answer is how to map these abstractions, such that they are suitable for a target task. In [87] and [94], the authors make use of the homomorphism framework to map tasks to a common abstraction level. The main idea is to construct an abstract MDP to which relativized options are defined (see [87]). Another approach is to frame different tasks as having a shared *agent space* as done in Konidaris et al. [48], so that no mapping is explicitly needed. Unfortunately, this requires the agent acting in both tasks to share the same actions and the human must map new sensors back into the agent space.

**Parameter Transfer** Most of the algorithms reviewed in this section assume the presence of a hand-coded mapping. In [105] the notion of using a hand coded intertask mapping to transfer value functions from source to target tasks has been introduced. Its applications in [96] have shown significant improvements in complex domains. Other applications, such as [8] consider the transfer of value functions in the general game playing. Torrey et al. [112] transfers advice and transfers Q-tables through a hand coded mapping. Taylor and Stone [108] proposes an algorithm that is capable of transferring when the value functions between the source and target task differ in representation or the source and target reinforcement learning algorithms vary.

Other methods in parameter transfer aim at automatically learning a suitable mapping between source and target tasks. In [103, 104] MASTER is proposed. MASTER identifies the best state mapping that guarantees the best prediction of the dynamics of the target environment.

**Instance Transfer** The first attempts to perform transfer between MDPs with differences in their domains are these described in [104, 106]. Given difference between the source and target MDPs state and action spaces, these methods make use of a hand coded intertask mapping to perform transfer. This mapping is split into two sub-mappings. The first, called interstate mapping, $\chi_{\mathcal{S}}$, relates the source and target task state spaces, while the second, called interaction mapping, $\chi_{\mathcal{U}}$, relate the action spaces between the tasks. Given $\chi_{\mathcal{S}}$ and $\chi_{\mathcal{U}}$, samples in the source are then transformed into target samples. Taylor et al. [108] makes use of model-based reinforcement learning algorithms to benefit from the transformed samples.

In this dissertation three methods that automatically learn intertask mappings between source and target tasks are contributed. In the first [5], the assumptions made in [108] are relaxed. More specifically, rather than assuming that both the interstate and interaction mappings are provided by the designer, only a common state-subspace is assumed available. Using this subspace (see Chapter 4) both an

interstate and an interaction mapping are learned. To provide an automated approach to learn the intertask mapping, a method based on sparse coding is contributed in [16]. The main idea is to automatically discover a relevant feature space that is capable of representing similarities between the dynamics of the source and target tasks (see Chapter 5). This space can then be used to automatically learn an intertask mapping and thus perform successful transfer. Finally, a more robust alternative, based on restricted Boltzmann machines, is detailed in [4].

## 3.6 Conclusions

It is clear from the survey in this chapter, prior to the contributions of this dissertation, there are no transfer learning algorithms capable of automatically performing deep transfer between two RL tasks. The above algorithms either assumed, that the mapping between the two tasks was given or such a mapping is not needed. Even these that tried to learn such a mapping, which is essential for enabling transfer, mostly assumed same domains (i.e., state and action spaces) between the tasks.

Aiming at automated transfer algorithms, the next Chapter (i.e., Chapter 4), representing the one of the contributions of this this dissertation, describes the first attempts to create a fully automated algorithm capable of learning an intertask mapping between two RL tasks. The method described next is not fully automated though, since it requires a manually defined common subspace. This can then be used to relate triplets of the source and the target, to learn an intertask mapping.

# 4

# Towards Automated Intertask Mappings

**This chapter is based on:** *H. B. Ammar and M. E. Taylor, "Common Subspace Transfer for Reinforcement Learning Tasks," in Lecture Notes on Artificial Intelligence (LNAI), 2011.*

As mentioned in the previous chapter, transfer can be categorized according to the intertask mappings used. Most of the current transfer algorithms either hand code such mappings, or assume that they are not needed by imposing restrictions on the problem. In this chapter an approach aiming at reducing human intervention in designing intertask mappings is explained. However, the proposed method is not completely autonomous. The designer has to first design a common state subspace that describes the similarities shared between the tasks. Transitions are then project onto such a subspace and a simple similarity measure is adopted to relate them together. Having these relations, an intertask mapping can be learned [1]. This work makes use of locally weighted regression, to learn the intertask mapping.

This chapter, also provides a proof-of-concept for the proposed method in different experiments. The first experiment shows successful transfer from a single mass system to a double mass system. The second experiment uses a policy learned on the simple inverted pendulum task to improve learning on the cart-pole swing-up problem.

Although this approach works in a deterministic model based setting, requires a human-specified subspace and is demonstrated using one reinforcement learning algorithm, results successfully show:

1. an inter-state mapping can be learned from data collected in the source and target tasks with an acceptable number of samples;

2. this inter-state mapping can effectively transfer information from a source task to a target task, even if the state representations and actions differ;

[1] Further automation for learning the intertask mapping is described in the coming chapters.

3. an agent that uses transferred information can learn a higher quality policy in the target task, relative to not using this information, when keeping the number of samples in the target task fixed and without using an explicit action mapping; and

4. an agent using information transferred from a source task can learn an optimal policy faster in the target task, relative to not using this information, when it has access to an unlimited number of target task samples thus reducing the number of samples in the target task.

## 4.1 Problem Definition

In this section the problem definition according to the survey of the previous chapter is detailed. Each of the source and target tasks are MDPs. These vary in each of their constituents. More specifically, the approach described here makes no restrictive assumptions on the type of variations between the tasks. In other words, not only the probability distributions or reward functions may vary between the tasks, but these variations can also happen between the state and/or action spaces.

Therefore, the tackled problem is a *deep transfer* one, which is substantially harder than other forms of shallow transfer, such as these tackled in reward shaping, imitation learning, or lifelong learning. Furthermore, to increase the scope of applicability of the proposed method, the state spaces of both the source and target tasks are continuous in nature.

Moreover, the source agent has already learned a near-optimal behavior and access to samples from the source and target task is available. In this work, learning the intertask mapping is split into two phases. In the first an inter-state mapping relating the state spaces of the source and target task together is learnt through a manually designed state subspace. Having the interstate mapping, $\chi_{inter_{\mathcal{S}}}$, the second phase commences to learn an inter-action mapping $\chi_{inter_{\mathcal{U}}}$, which depends on $\chi_{inter_{\mathcal{S}}}$ to generate an initial policy in the target task. The intuition is that if the two tasks are related, then this starting policy in the target will help the target agent to learn a successful behavior faster compared to starting from scratch.

**Problem Definition:** Given two MDPs $\mathcal{M}_1 = \langle \mathcal{S}_1, \mathcal{U}_1, \mathcal{T}_1, \mathcal{R}_1, \gamma_1 \rangle$ and $\mathcal{M}_2 = \langle \mathcal{S}_2, \mathcal{U}_2, \mathcal{T}_2, \mathcal{R}_2, \gamma_2 \rangle$, where $\mathcal{S}_i$, $\mathcal{U}_i$, $\mathcal{T}_i$, $\mathcal{R}_i$, and $\gamma_i$ correspond to the state space, actions space, transition model, reward function, and discount factor for all $i \in \{1, 2\}$ representing the source and target tasks' MDP constituents, respectively,

and some additional knowledge in both the source and the target, learn two mappings $\chi_{inter_{\mathcal{S}}}$ and $\chi_{inter_{\mathcal{U}}}$.

As in any other transfer learning framework additional knowledge is assumed to be available. In this work, the knowledge available in the source is twofold: (1) random state successor state transitions, and (2) optimal source policy $\pi_1^\star$. Such a policy can be used to generate "optimal" transitions that can then be transferred to aid the target agent. Furthermore, additional knowledge in the form of random state successor state transitions are also available in the target. In the first phase, these and the random samples of the source help in learning an inter-state mapping. It is worth noting, that randomly generating these samples is essential. The reason is that the mapping needs to operate within a broad range of the state and action spaces of both tasks. Learning mappings based only on optimal transitions, can not generalize well to broader unobserved ranges of the state and/or action spaces. However, after learning such a broad range mapping, transferring "bad" information might "hurt" the target agent. Therefore, in the second phase, only *optimal transitions* in the source are used to construct a starting policy in the target.

## 4.2 Overall Framework Description

The overall framework is shown in Figure 4.1. Firstly, starting from both state spaces $\mathcal{S}_2$ and $\mathcal{S}_1$ of both MDPs an inter-state mapping, $\chi_{inter_{\mathcal{S}}}$ is learnt. This is shown by the dotted line at the top of the figure. Such a mapping is attained by projecting the state successor state transitions from both the source and target task onto a common subspace.

Having $\chi_{inter_{\mathcal{S}}}$, an inter-action mapping is then indirectly attained. Learning the inter-action mapping between the tasks is shown at the bottom side of Figure 4.1. Starting from some initial state in the target task's state space, all $q_2$ actions are executed leading the system to transition into $q_2$ successor states, denoted by $s_2^{(1)\prime}, s_2^{(2)\prime}, \ldots, s_2^{(q_2)\prime}$. These successor states are then projected into the common subspace to produce $\mathcal{S}_\mathcal{C} = \{s_{1,c}^{\prime(1)}, \ldots, s_{1,c}^{\prime(q_2)}\}$. To determine which of the executed actions is best, knowledge from the source task is now used. Since "good" information is useful for transfer, then the optimal source task policy is used. This is done in three phases. First, the target state, is mapped to the source using the learned inter-state mapping $\chi_{inter_{\mathcal{S}}}$. Second, starting from this projected state, which is now described in the source, the optimal policy $\pi_1^\star$, is used to transition to the optimal successor state $s_1^{\star\prime}$ as shown in Figure 4.1. Third, the successor state is projected again to the

common subspace, where it is to be compared with the other transitions of the target. This comparison is conducted using a simple similarity measure, and the target action causing the most similar transition to the optimal is chosen as a "good" starting action.

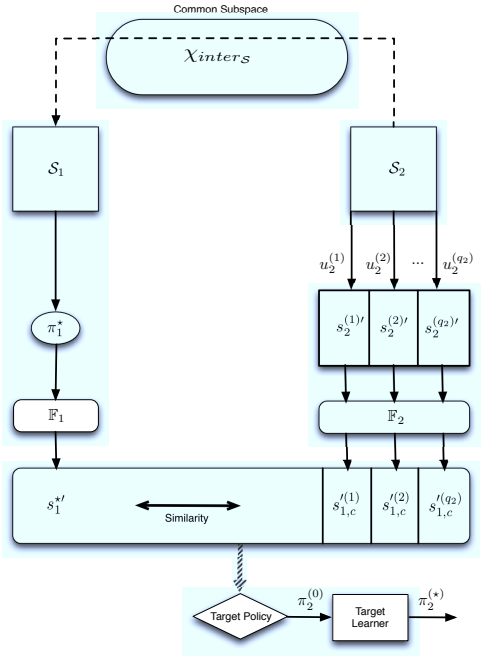



Figure 4.1: High-level schematic of the overall framework.

The overall process is repeated from different initial states to have a data set of the form $\mathcal{D}_T = \{s_2^{(i)}, a_2^{(i)}\}_{i=1}^m$ that can be used to approximate a starting initial policy $\pi_2^{(0)}$ in the target. Although, $\pi_2^{(0)}$ might be a good start for the target agent, it is not optimal. $\pi_2^{(0)}$ still has to be improved in the target in order to reach the optimal policy $\pi_2^\star$.

The details of each of the above phases are explained in the remainder of this chapter. Section 4.3 describes how an inter-state mapping can be learned between two tasks by leveraging a distance-minimization algorithm. Section 4.4 shows how the learned mapping can be used to transfer information between a source and target tasks. Experiments in Section 4.5 evaluate the entire system on two pairs of tasks. Finally, Section 4.6 concludes with a discussion and future work directions.

## 4.3 Learning an Inter-State Mapping

At a high-level, the transfer framework can be decomposed into three major phases. In the first phase, the function $\chi_{inter_{\mathcal{S}}}$, mapping the states from $\mathcal{M}_2$ into $\mathcal{M}_2$ is learned. $\chi_{inter_{\mathcal{S}}}$ is learned by collecting transitions from the source task and target task and identifying correspondences. The second phase finds an initial policy for task two, $\pi_2^{(0)}$ in $\mathcal{M}_2$, by identifying actions in the target task that are most similar to actions selected in the source task by $\pi_1^*$ (see Section 4.4). The third phase uses samples gathered by $\pi_2^{(0)}$ as an initialization for fitted value iteration, rather than using randomly selected samples, finding an optimal policy $\pi_2{}^*$ of $\mathcal{M}_2$ (see Section 4.4).

To learn the interstate mapping a common subspace describing the similarity between the two tasks is essential. A *common task subspace*, $\mathcal{S}_c$, is defined as a subspace that describes shared characteristics between the tasks $\mathcal{M}_1$ and $\mathcal{M}_2$. Generally, $\mathcal{S}_c$ has a lower dimensionality than $\mathcal{S}_1$ or $\mathcal{S}_2$ and is determined by common state semantics shared between the two tasks. This subspace is described via the control problem's definition or is user defined. In many cases, manually defining such a common task subspace is relatively easy. In the case of control problems, the subspace construction can be influenced by the particular goal or goals an agent must achieve in a task. As an illustration, consider the problem of transfer between agents with two different robotic arms, each of which has acts in a different dimensionality space (i.e., has a different description of state because of different sensors and or degrees of freedom). In this case, $\mathcal{S}_c$ can be defined as the position and orientation of the end effector in both robots. Thus, even with such nonlinear continuous MDPs setting, attaining a common task space requires less effort than trying to manually encode the action and state variables mappings as used to be done in other transfer learning approaches.

$\mathcal{S}_c$ is used to determine the correspondence between state successor state pairs of $\mathcal{M}_1$ and $\mathcal{M}_2$, which in turn will generate data used to approximate $\chi_{inter_{\mathcal{S}}}$. Given that the two tasks are related through some common task subspace $\mathcal{S}_c \in \mathbb{R}^{d_c}$, a function $\chi_{inter_{\mathcal{S}}} : \mathcal{S}_2 \rightarrow \mathcal{S}_1$, mapping the two state spaces of $\mathcal{M}_1$ and $\mathcal{M}_2$ together is learned. As discussed in Section 4.4, $\chi_{inter_{\mathcal{S}}}$ alone is capable of transferring policies from $\mathcal{M}_1$ to $\mathcal{M}_2$ by effectively finding a good prior for the agent in $\mathcal{M}_2$.

### Details on Learning $\chi_{inter_{\mathcal{S}}}$

To learn $\chi_{inter_{\mathcal{S}}}$ three ingredients are needed as inputs: (1) $n_1$ state successor state patterns of the $d_1$ dimensional state space $\mathcal{S}_1$, $\langle s_1, s_1' \rangle$ (gathered from interactions with the source task), (2) $n_2$ state successor state patterns of the $d_2$ dimensional state space $\mathcal{S}_2$, $\langle s_2, s_2' \rangle$ (gathered from interactions with the target task), and (3) a common task subspace $\mathcal{S}_c$ with dimensions $d_c \leq \min\{d_1, d_2\}$. The reasons behind this restriction

are twofold: (1) shared information between tasks is highly probable to occur in a lower dimensional manifold [113], and (2) in typical practices it is easier to define a lower dimensional space compared to a high dimensional one [9]. Algorithm 14 proceeds by projecting each of the above patterns into $\mathcal{S}_c$, attaining $n_1$ patterns of the form $\langle s_{c,1}^{(i)}, s_{c,1}^{\prime(i)} \rangle$, were the subscript $\{c, 1\}$ denotes mapping states from $\mathcal{S}_1$ into states in $\mathcal{S}_c$, for $i = \{1, 2, \ldots, n_1\}$, corresponding to the projected $\mathcal{S}_1$ states (line 2 of Algorithm 14). This can be written in a matrix format as follows. Define:

$$\mathbf{S}_1 = \begin{bmatrix} \text{———} & \mathbf{s}_1^{(1)T} & \text{———} \\ \text{———} & \mathbf{s}_1^{(2)T} & \text{———} \\ \vdots & \vdots & \vdots \\ \text{———} & \mathbf{s}_1^{(n_1)T} & \text{———} \end{bmatrix},$$

to be the matrix collecting all the source task states as row vectors. Therefore $\mathbf{S}_1 \in \mathbb{R}^{n_1 \times d_1}$. Next, define the projection matrix $\mathbb{F}_1 \in \mathbb{R}^{d_c \times d_1}$ to be the matrix collecting all the projection functions operating on all the samples. The projected $n_1$ samples described in the common subspace, can now be attained using:

$$\begin{bmatrix} \mathbf{s}_{c,1}^{(1)} \\ \mathbf{s}_{c,1}^{(2)} \\ \vdots \\ \mathbf{s}_{c,1}^{(n_1)} \end{bmatrix} = \mathbb{F}_1 \begin{bmatrix} \mathbf{S}_1^T \end{bmatrix}$$

Additionally, $n_2$ patterns of $\langle s_{c,2}^{(j)}, s_{c,2}^{\prime(j)} \rangle$ are found on line 4 of Algorithm 2, where the subscript $\{c, 2\}$ represents the notion of state space $\mathcal{S}_2$ states in $\mathcal{S}_c$ and $j = \{1, 2, \ldots, n_2\}$, corresponding to the projected $\mathcal{S}_2$ states. This can be done in the same manner as in the previous case as follows:

$$\begin{bmatrix} \mathbf{s}_{c,2}^{(1)} \\ \mathbf{s}_{c,2}^{(2)} \\ \vdots \\ \mathbf{s}_{c,2}^{(n_2)} \end{bmatrix} = \mathbb{F}_2 \begin{bmatrix} \mathbf{S}_2^T \end{bmatrix}$$

The algorithm next calculates a minimum Euclidian distance on the $n_1$ and $n_2$ patterns (lines 6–8). Once a correspondence between the projected states in $\mathcal{S}_c$ has been found, full states rather than subspace states are required to train $\chi_{inter_{\mathcal{S}}}$. These are found by projecting all the state combinations in $\mathcal{S}_1$ and $\mathcal{S}_2$, lines 10–12, generating the recommended $s_{c,1}$ and $s_{c,2}$ (further discussed in Section 4.3) and assigning labels

**Algorithm 14** Learn an Inter-State Mapping

**Require:** $n_1$ random samples of $\langle s_1^{(i)}, s_1'^{(i)} \rangle_{i=1}^{n_1}$; $n_2$ random samples of $\langle s_2^{(k)}, s_2'^{(k)} \rangle_{j=1}^{n_2}$; $f_1$ and $f_2$ representing the functions projecting $\mathcal{S}_1$ and $\mathcal{S}_2$ into $\mathcal{S}_c$, respectively; and threshold $\beta_1$

1: **for** $i = 1 \rightarrow n_1$ **do**
2: $\quad \langle s_{c,1}^{(i)}, s_{c,1}'^{(i)} \rangle \leftarrow f_1\Big(\langle s_1^{(i)}, s_1'^{(i)} \rangle\Big)$
3: **end for**
4: **for** $j = 1 \rightarrow n_2$ **do**
5: $\quad \langle s_{c,2}^{(j)}, s_{c,2}'^{(j)} \rangle \leftarrow f_2\Big(\langle s_2^{(j)}, s_2'^{(j)} \rangle\Big)$
6: **end for**
7: **for** $k = 1 \rightarrow n_2$ **do**
8: $\quad$ **for** $l = 1 \rightarrow n_1$ **do**
9: $\quad\quad d^{(l)} \leftarrow ||\langle s_{c,1}^{(l)}, s_{c,1}'^{(l)} \rangle - \langle s_{c,2}^{(k)}, s_{c,2}'^{(k)} \rangle||_2$
10: $\quad$ **end for**
11: $\quad$ Calculate $l^* \leftarrow \arg\min_l d^{(l)}$
12: $\quad$ **if** $d_{best}^{(l^*)} \leq \beta_1$ **then**
13: $\quad\quad s_{c,1}^{(k)} \leftarrow$ all combinations of $s_1$
14: $\quad\quad s_{c,2}^{(l^*)} \leftarrow$ the combinations of $s_2$
15: $\quad\quad$ Collect all combinations of the latter $s_2$ and $s_1$ as inputs and outputs, respectively, to approximate $\chi_{inter_{\mathcal{S}}}$
16: $\quad$ **else**
17: $\quad\quad$ Do Nothing: *ignore current sample*
18: $\quad$ **end if**
19: **end for**
20: Approximate $\chi_{inter_{\mathcal{S}}}$

and inputs respectively. The algorithm collects these combinations (line 12) so that $\chi_{inter_{\mathcal{S}}}$ represents a best fit mapping between $\mathcal{S}_2$ and $\mathcal{S}_1$ via $\mathcal{S}_c$.

## Problem: Mapping Unrelated States

At this stage two potential problems arise. First it is possible that states in $\mathcal{S}_2$ are mapped into states in $\mathcal{S}_1$, even when they are not related. This is a common problem in transfer learning (related to the problem of *negative transfer* [107]) and cannot be fully solved here, but can be remedied by considering the distance between the successor states.

Consider patterns in the target task, $\langle s_2, s_2' \rangle$, and a pattern in the source task, $\langle s_1, s_1' \rangle$. Using Algorithm 14, lines 2 and 4, $f_2$ and $f_1$ map each of the successor states into the common sub-space as $\langle s_{c,2}, s_{c,2}' \rangle$ and $\langle s_{c,1}, s_{c,1}' \rangle$ respectively. If the distance d, as measured by $||\langle s_{c,1}, s_{c,1}' \rangle, \langle s_{c,2}, s_{c,2}' \rangle||_2$, is greater than some threshold parameter

(line 9), it suggests this mapping is suspect because the initial state successor state pair, $\langle s_2, s_2' \rangle$, has a poor correspondence with the source task pattern, potentially harming the agent's performance in $\mathcal{M}_2$.[2] This state may not be the best choice for a prior in the target task — only states with small distances are used as inputs and outputs for the supervised learning algorithm.

### Problem: Non-injective Mapping

The second potential problem is that the function $\chi_{inter_{\mathcal{S}}}$ must map all state variables from the target task into the source task. However, the correspondence between the inputs, states in $\mathcal{S}_2$, and the outputs, states in $\mathcal{S}_1$, was found in the common state subspace $\mathcal{S}_c$. The projection functions, $f_1$ and $f_2$, from $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively, are not-injective. Thus, there may be a problem when attempting to fully recover the initial data points in $\mathcal{S}_1$ and $\mathcal{S}_2$, corresponding to $s_{c,1}$ and $s_{c,2}$, which is critical when approximating $\chi_{inter_{\mathcal{S}}}$.

This problem is approached by verifying all possible states in $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$ corresponding to the intended $s_{c,1}$ and $s_{c,2}$ respectively. Then consider all combinations of the initial states, on line 12, that were mapped together using Algorithm 14, as inputs and outputs are considered. By that, the need for an inverse mapping $f_1^{-1}$ and $f_2^{-1}$ to recover the original states in $S_1$ and $S_2$ is avoided. Once the correspondence between the patterns of $\mathcal{S}_1$ and $\mathcal{S}_2$ has been determined, a supervised learning scheme attains $\chi_{inter_{\mathcal{S}}}$. Locally weighted regression (LWR), was used (line 15 of Algorithm 14) to approximate $\chi_{inter_{\mathcal{S}}}$, which is used in turn to determine the transferred policy, $\pi_2^{(0)}$, as described in the following section.

## 4.4 Policy Transfer and RL Improvement

To transfer between agents with differences in the action spaces some type of mapping representing the relations between the allowed actions of the source and target agent must be learned. In finding a mapping of the action spaces between the tasks, there exists the problem of the difference in dimensions between the two action spaces. Solving this problem could not be approached as done for the state space case, since it is not trivial to determine some common action space shared between the tasks to be projected to to find the inputs and labels which in turn would be used to map the action spaces together.[3]

[2] Even if the two tasks are closely related this could occur due to a large difference in the action spaces of the two tasks.

[3] This is in addition to the problem of determining an inverse mapping for $\chi_{inter_{\mathcal{S}}}$, since the to approximate a starting policy in the target task prevails.

Rather than approaching this problem explicitly and constructing a mapping between the action spaces of the tasks, an implicit mapping using the inter-state mapping learned is rather developed.

The inter-state mapping, $\chi_{inter_{\mathcal{S}}}$, will enable transfer from $\mathcal{M}_1$ to $\mathcal{M}_2$. This transfer is based on a similarity measure between state and successor states in both MDPs, in the sense that only state transitions that relatively have acceptable distance measures are taken into account. First, some states in $\mathcal{M}_2$ are reflected, using $\chi_{inter_{\mathcal{S}}}$, into $\mathcal{M}_1$. Then, the optimal action will be chosen for that state and the system transitions into some successor state in $\mathcal{M}_1$, according to the source task transition probability. From the corresponding state in $\mathcal{M}_2$, all the actions in $\mathcal{U}_2$ are executed to transition to new successor states. These are reflected through $\chi_{inter_{\mathcal{S}}}$ to their corresponding in $\mathcal{M}_1$. Finally, the action that produces a successor state in $\mathcal{M}_2$ with a minimum distance to the optimal transition in the source is taken as the best action. This section will further detail the above scheme and explain how $\chi_{inter_{\mathcal{S}}}$ is used to conduct policy transfer between the two MDPs.

## Policy Transfer Scheme

The inter-state mapping, as learned in the previous section, is capable of providing the agent in the target task with an informative prior. Finding the transferred policy, $\pi_2^{(0)}$, is done in two phases. First, state-action pairs are collected in the source task, according to $\pi_1^*$. Second, $\pi_2^{(0)}$ is constructed from the collected samples, and the learned inter-state mapping.

Algorithm 15 needs to be able to generate successor states for both MDPs, lines 5–7. Thus, it is necessary for Algorithm 15 to have access to a transition model or simulator, where agents in both tasks can generate next states by taking actions.

Algorithm 15 finds an action, $u_2 \in \mathcal{U}_2$, for a state $s_2 \in \mathcal{S}_2$, by using the inter-state mapping, $\chi_{inter_{\mathcal{S}}}$, and a user-defined threshold, $\beta_2$. Using $\chi_{inter_{\mathcal{S}}}$, the algorithm maps each of the $m$ random states, $s_2^{(1)}$–$s_2^{(m)}$, to corresponding states, $s_1^{(1)}$–$s_1^{(m)}$. It then selects on action, $u_1$, for a state in $S_1$, according to the optimal policy of $\mathcal{M}_1$, and transitions into the optimal $s_1'$ state according to the probability transition function $\mathcal{T}_1(s_1, a_1)$.

The algorithm examines all possible actions in $\mathcal{U}_2$ from the given initial state $s_2^{(i)}$ to transient to $q_2$ different subsequent states $s_2'$ (see line 6 – 7 of Algorithm 15). Then for each $s_2'$, $\chi_{inter_{\mathcal{S}}}$ is used again to find the corresponding $s_1'$ denoted by $s_{1,c}'$ in the algorithm, line 8. At this stage, a minimum distance search between the attained $s_{1,c}'$ and the action recommended by $\pi_1^*$ is executed. If the distance is below the user defined threshold $\beta_2$ then the action $a_2$ corresponding to the minimum distance is chosen to be the best action for that random initial state $s_2$. This sequence is

**Algorithm 15** Collect State-action Pairs

**Require:** $m$ random initial $s_2$ states, optimal policy of the first system $\pi_1^*$, probability transition functions of the two systems $\mathcal{T}_1(s_1, u_1)$ and $\mathcal{T}_2(s_2, u_2)$, the action space of system two $\mathcal{U}_2$, and distance threshold $\beta_2$

1: Set $q_2$ to be the size of $\mathcal{U}_2$
2: **for** $i = 1 \rightarrow m$ **do**
3: $s_1{}^{(i)} \leftarrow \chi_{inter_{\mathcal{S}}}(s_2{}^{(i)})$
4: $u_1^{(i)} \leftarrow \pi_1^*(s_1^{(i)})$
5: Attain $s_1'^{(i)} \sim \mathcal{T}_1(s_1{}^{(i)}, u_1{}^{(i)})$ sampled according to the state transition probability $\mathcal{T}_1$
6: **for** $k = 1 \rightarrow q_2$ **do**
7: Attain $s_2'^{(k)} \sim \mathcal{T}_2(s_2{}^{(i)}, u_2{}^{(k)})$ sampled according to the state transition probability $\mathcal{T}_2$
8: Attain the corresponding $s_{1,c}'^{(k)} \leftarrow \chi_{inter_{\mathcal{S}}}(s_2'^{(k)})$ using the inter-state mapping $\chi_{inter_{\mathcal{S}}}$
9: $d^{(k)} \leftarrow ||s_1'^{(i)} - s_{1,c}'^{(k)}||_2$
10: **end for**
11: $d_{best}^{(i)} \leftarrow \min_k(d^{(k)})$
12: $j \leftarrow \arg\min_k d^{(k)}$
13: **if** $d_{best}^{(i)} \leq \beta_2$ **then**
14: Collect the pattern $(s_2^{(i)}, u_2^{(j)})$ as one training pattern to approximate $\pi_2^{(0)}$
15: **else**
16: Do nothing: *Ignore this sample*
17: **end if**
18: **end for**
19: Using collected patterns, approximate $\pi_2^{(0)}$

repeated for the $m$ different random initial states of $S_2$, resulting in a *data set of state-action pairs in the target task*, guided by $\pi_1^*$.

This data set is used to approximate $\pi_2^{(0)}$, done via LWR in our experiments, and this policy will be used as a starting policy by the target task agent.

## Improving the Transferred Policy

The policy $\pi_2^{(0)}$ serves as an initial policy for the $\mathcal{M}_2$ agent — this section describes how the policy is improved via Fitted Value Iteration (FVI), using an initial trajectory produced by $\pi_2^{(0)}$.

To attain an optimal policy in the target, we used a minor variant of FVI, where the value function is repeatedly approximated after fitting the $\Psi$ values. Starting from a small number of initial states, $f$, sampled through $\pi_2^{(0)}$, we attempt to find an optimal policy $\pi_2^*$, by iteratively re-sampling using the fitted $\Psi$ values as needed.

**Algorithm 16** Fitted Value Iteration Algorithm + Transfer

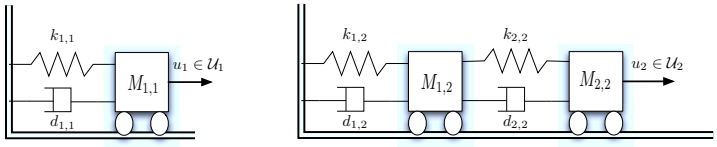
Starting from random initial states, sample $f$ states according to $\pi_2^{(0)}$
$\Psi \leftarrow 0$
$n_2 \leftarrow$ the size of the action space $\mathcal{U}_2$
**repeat**
  **for** $i = 1 \rightarrow f$ **do**
    **for all** $u_2 \in \mathcal{U}_2$ **do**
      $q(u_2) \leftarrow R(s^{(i)}) + \gamma V(s^{(j)\prime})$
    **end for**
    $y^{(i)} \leftarrow \max_{u_2 \in \mathcal{U}_2} q(u_2)$
  **end for**
  $\Psi \leftarrow \arg\min_\Psi \sum_{i=1}^{f} \left(y^{(i)} - \Psi^T \Phi(s^{(i)})\right)^2$
  Greedily sample new $f$ states according to the fitted $\Psi$ values representing $\pi_{fit} = \arg\max_a E_{s' \sim \mathcal{T}_{sa}}[\Psi^T \Phi(s')]$
**until** $\Psi$ converges
Represent $\pi_2^* = \arg\max_a E_{s' \sim \mathcal{T}_{sa}}[\Psi^T \Phi(s')]$

Algorithm 16 works to find optimal values for the parameters to fit the value function on a set number of samples, which were sampled using $\pi_{tr}$. Then, after each iteration of the repeat loop, Algorithm 16 samples a new set of states according to current policy represented by $\pi_{fit}$. The sampling / value fitting process is repeated until convergence, attaining an optimal policy. The difference between Algorithm 16 to the normal fitted-value iteration algorithm, is that the initial samples are not gathered according to a random policy, but by following $\pi_2^{(0)}$. Assuming that $\pi_2^{(0)}$ is a "good" prior, this procedure will focus the exploration of the policy space.

## 4.5 Experiments

As a proof of concept, our algorithms were tested on two different systems. The first was the transfer from a single mass spring damper system to a double mass spring damper system, as shown in Figure 4.2 (repeated from Chapter 3). The second experiment transferred between the inverted pendulum task to the cartpole swing-up task. The following two sub-sections will discuss the details of the experiments and their results.

The values of the discount factor $\gamma$, used in the fitted value iteration algorithms were fixed to 0.8 while those of $\beta_1$ and $\beta_2$, used in Algorithms 14 and 15, were fixed at 0.9 and 1.5, respectively. In fact, we found that varying the values of $\beta_1$ and $\beta_2$ did not significantly affect the performance of the algorithms, suggesting that our




(a) Simple Mass System (b) Double Mass System

Figure 4.2: The single mass spring damper system in (a) and the double mass spring damper system in (b).

algorithms are robust to changes in these parameters.[4]

## Single to Double Mass

For the first experiment, a policy between the systems shown in Figure 4.2 was transferred. $\mathcal{S}_1$ is described by the $\{x_{1,1}, \dot{x}_{1,1}\}$ variables, representing the position and the velocity of the mass $M_{1,1}$. The agent is allowed to apply a force from the action set $\mathcal{U}_1$.

## Single and Double Mass

The single mass system, shown in Figure 4.2(a), oscillates horizontally, according to the following:

$$M_{1,1}\ddot{x}_{1,1} = u_1 - k_{1,1}x_{1,1} - d_{1,1}\dot{x}_{1,1}$$

where $u_1 \in \mathcal{U}_1$ is the action chosen by the agent, $\ddot{x}_{1,1}$ is the mass acceleration, and $\dot{x}_{1,1}$ is its velocity.

For the double mass system, shown in Figure 4.2(b), $\mathcal{S}_2 = \{x_{1,2}, \dot{x}_{1,2}, x_{2,2}, \dot{x}_{2,2}\}$, representing the position and the velocity of $M_{1,2}$ and $M_{2,2}$. The agent is allowed to apply a force from the action set $\mathcal{U}_2$. The system the oscillates horizontally, according to the following:

$$M_{1,2}\ddot{x}_{1,2} = -k_{1,2}x_{1,2} - d_{1,2}\dot{x}_{1,2} + k_{2,2}(x_{2,2} - x_{1,2}) + d_{2,2}(\dot{x}_{2,2} - \dot{x}_{1,2})$$
$$M_{2,2}\ddot{x}_{2,2} = u_2 + k_{2,2}(x_{1,2} - x_{2,2}) + d_{2,2}(\dot{x}_{1,2} - \dot{x}_{2,2})$$

[4]When having similar tasks, it is likely that the distances attained are reflective. Therefore, We believe that carefully setting $\beta_1$ and $\beta_2$ may only be necessary when the source and target tasks are very dissimilar but we leave such explorations to future work.

where $u_2 \in \mathcal{U}_2$, $k_{1,2}$ is the spring constant of the first mass, $d_{1,2}$ is the damping constant of the first mass, $k_{2,2}$ is the spring constant for the second mass, and $d_{2,2}$ is the damper of the second mass.

A reward of $+1$ is given to the agent of system one if the position of the mass $M_{1,1}$ is 1 and $-1$ otherwise. On the other hand, a reward of $+10$ is given to the agent of system two if the position and the velocity of the mass $M_{1,2}$ is 1 and 0 respectively, and otherwise a reward of $-10$ is given. The action spaces of the two systems are $\mathcal{U}_1 = \{-15, 0, 15\}$ and $\mathcal{U}_2 = \{-15, -10, 0, +10, -15\}$, describing the force of the controller in Newtons. The agent's goal is to bring the mass of system two, $M_{1,2}$, to the state $s_2 = \{1, 0\}$, which corresponds to a position of 1 ($x_{1,2} = 1$) and a velocity of zero ($\dot{x}_{1,2} = 0$). In our transfer learning setting, the agent relies on an initial policy delivered from the controller of the system $\mathcal{M}_1$ and improves on it. In the source task, Fitted Value Iteration (FVI) found a policy to bring the mass $M_{1,1}$ to the $s_1 = \{1, 0\}$ goal state.

### Common Task Subspace

In both mass systems the control goal is to settle the first mass so that it reaches location $x = 1$ with zero velocity. Thus, the common task subspace $S_c$ is described via the variables $x$ and $\dot{x}$ for mass #1 in both systems.

### Source Task: Single Mass System

The FVI algorithm was used to learn an optimal policy, $\pi_1^*$, for the first mass system. A parametric representation of the value function was used: $V(s) = \Psi^T \Phi(s)$ and $V(s) = (\psi_1\ \psi_2\ \psi_3\ \psi_4\ \psi_5)(x_{1,1}^2\ x_{1,1}\ \dot{x}_{1,1}^2\ \dot{x}_{1,1}\ 1)^\top$.

Algorithm 16 but starting from random samples (source task), was able to converge to the optimal parametric values approximating the value function on a single core of a dual core 2.2 GHz processor in about 18 minutes, after starting with 5000 random initial samples. The resulting controller, represented as values in $\Psi$, was able, in 0.3 simulated seconds, to control the first mass system in its intended final state: $s_1 = \{1, 0\}$.

### Target Task: Double Mass System

To test the efficacy of our learned $\chi_{inters}$ by Algorithm 14 and transfer method using Algorithms 15 and 16, the values for $n_1$ and $n_2$ were varied from 1000–8000, which correspond to the number of samples used in the target task.[5] Fitted value iteration was run with these different sets of samples, which were in turn used to generate

[5]This corresponds to roughly 10–175 states ignored in Algorithm 14, line 14.

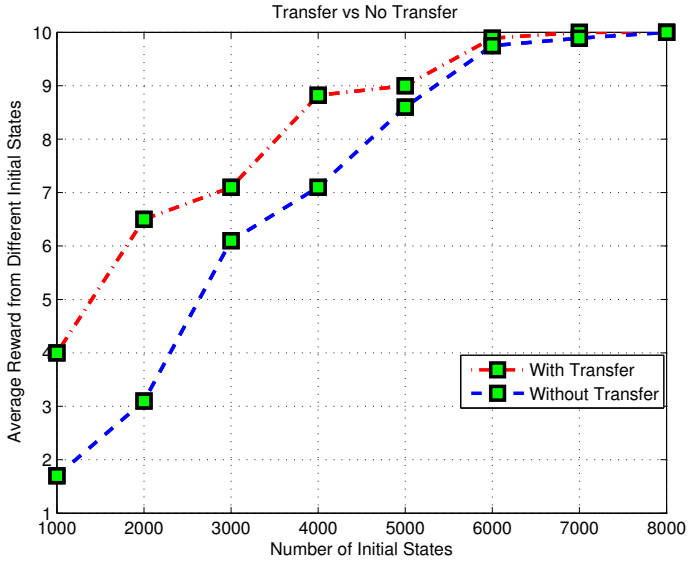



Figure 4.3: This graph compares the performance of converged policies on the double mass system, as measured over 1000 independent samples of random start states in the target task measured over independent 500 trials. The $x$-axis shows the number of target task states used by FVI and the $y$-axis shows the average reward after FVI has converged (without resampling the states).

policies for the target task. The performance of these policies in the target task, after convergence, are shown in Figure 4.3, and are compared to using random initial samples (i.e., no transfer).

The results in Figure 4.3 show that FVI performs better when initialized with a small number of states sampled from $\pi_2^{(0)}$ than when the states are generated by a random policy. Further, results confirm that as the number of samples increase, both transfer and non-transfer learning methods converge to the (same) optimal policy.

**Conclusion I:** $\pi_2^{(0)}$, which uses the learned $\chi_{inters}$, allows an agent to achieve a higher performance with a fixed number of sampled target task states compared to a random scheme.

Algorithm 16 was also used to attain the optimal policy $\pi_2^*$ when supplied with 7000 initial points, where the points were sampled randomly and from $\pi_2^{(0)}$. The convergence time to attain an optimal policy starting from the initial states generated through $\pi_2^{(0)}$ was approximately 4.5 times less than that starting from randomly sampled initial states.

**Conclusion II:** $\pi_2^{(0)}$ allows an agent to converge to an optimal policy faster by

"correctly" sampling the initial states for FVI that are improved on.

## Inverted Pendulum to the Cartpole Swing-up

For the second experiment, we transferred between the IP and CP systems shown in Chapter 3. A detailed description of the task's dynamics can be found in Chapter 3. $\mathcal{S}_1$ is described by the $\theta_1$ and $\dot{\theta}_1$ variables representing the angle and angular speed of the inverted pendulum respectively. $\mathcal{S}_2$ is described by $\theta_2$, $\dot{\theta_2}$, $x$, and $\dot{x}$ representing the angle, angular speed, position, and velocity of the cartpole, respectively.

The reward of system one (inverted pendulum) was defined as $\mathcal{R}_1 = \cos(\theta_1)$ while that of system two (cartpole swing up) was $\mathcal{R}_2 = 10\cos(\theta_2)$. The action spaces of the two systems are $\mathcal{U}_1 = \{-15, -1, 0, 1, 15\}$ and $\mathcal{U}_2 = \{-10, 10\}$, describing the allowed torques, in Newton meters, and forces, in Newtons, respectively. The cart is able to move between $-2.5 \leq x \leq 2.5$. The agent's goal in the source task is to bring the pendulum of system two to state $s_2 = \{0, 0\}$, which corresponds to an angle of 0 ($\theta_2 = 0$) and an angular velocity ($\dot{\theta}_2 = 0$). In our transfer learning setting, the agent relies on an initial policy delivered from the controller of the first system and improves on it. In the source task, FVI found a policy to bring the pendulum to the state $s_1 = \{0, 0\}$.

### Common Task Subspace

In both systems the control goal is to settle the pendulums in the $\{0, 0\}$ upright state corresponding to an angle of zero and an angular velocity of zero. Thus, the common task subspace $S_c$ is described via the variables $\theta$ and $\dot{\theta}$ of both systems.

### Source Task: Simple Pendulum

The FVI algorithm was used to learn an optimal policy, $\pi_1^*$. A parametric representation of the value function was used: $V(s) = \Psi^T \Phi(s)$ and:

$$V(s) = (\psi_1\ \psi_2\ \psi_3\ \psi_4\ \psi_5)(\theta_1^2\ \theta_1\ \dot{\theta}_1^2\ \dot{\theta}_1\ 1)^\top$$

Algorithm 16, was able to converge to the optimal parametric values approximating the value function when on a single core of a dual core 2.2 GHz processor in about 23 minutes after starting from 5000 random initial samples. Then the controller was able, in 0.2 simulated seconds, to control the inverted pendulum in its intended final state $s_1 = \{0, 0\}$.

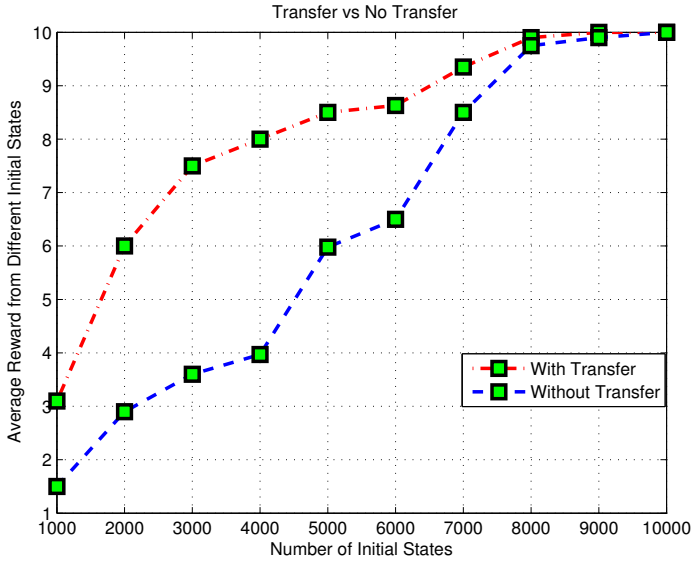



Figure 4.4: This figure compares the performance of the cartpole swing-up task, measured by the averaged reward, vs. different numbers of initial starting states. Starting states can be sampled via the transfer policy (from the inverted pendulum task) or randomly.

### Target Task: Cartpole Swing-up

To test the efficacy of the learned $\chi_{inter_S}$ using Algorithm 14 and the transfer method using Algorithms 15 and 16, the values for $n_1$ and $n_2$ were varied from 1000 – 10000, which corresponded to the number of samples in the target task.[6] Fitted value iteration was run with these different sets of samples, which were in turn used to generate policies for the target task. The performance of these policies in the target task, after convergence, are shown in Figure 4.4, and are compared to the random scheme (i.e., no transfer).

The results in Figure 4.4 show that FVI performs better when initialized with a small number of states sampled from $\pi_2^{(0)}$ than when the states are generated by a random policy. Further, the results confirm that as the number of samples increase, both transfer and non-transfer learning methods converge to the (same) optimal policy.

Finally, Algorithm 16 was used to learn the optimal policy $\pi_2^*$ when supplied with 7000 initial points, where the points were sampled randomly and from $\pi_2^{(0)}$. The convergence time, starting from the initial states generated through $\pi_2^{(0)}$, was

[6]This corresponds to roughly 18 – 250 states ignored in Algorithm 14, line 14.

approximately a factor of 6.3 less than that starting from randomly sampled initial states.

These results, summarized in Table 4.1, confirm the conclusions made in Section 4.5. The performance, as measured by the final average reward, was higher when using TL than when using randomly selected states. Furthermore, FVI was able to find an optimal policy in fewer minutes, denoted by $T$ in the table, when using TL than when using randomly selected initial states.

Table 4.1: Experiment Results Summary

| | Double Mass System | | | |
|---|---|---|---|---|
| | No TL | | With TL | |
| Transitions | Reward | Time | Reward | Time |
| 1000 | 1.7 | 6.5 | **3.9** | **4.5** |
| 5000 | 8.7 | 27 | **9.1** | **9.5** |
| 10000 | 9.9 | 43 | 9.9 | **11.8** |
| | Cartpole Swing-up | | | |
| | No TL | | With TL | |
| Transitions | Reward | Time | Reward | Time |
| 1000 | 1.4 | 10 | **3.1** | **7** |
| 5000 | 6.09 | 32 | **8.4** | **15** |
| 10000 | 9.9 | 160 | 9.9 | **27** |

## 4.6 Conclusions & Future Work

A novel transfer learning approach based on the presence of a common subspace relating two tasks together was presented. The approach is composed of three major phases:

The first is the determination of the inter-state mapping $\chi_{inter_S}$, relating the state spaces of the tasks, using a common task subspace, $S_c$, as described in Section 4.3. It relies on distance measures among state successor state pairs in both tasks to achieve the goal of finding a correspondence between the state spaces of the two tasks and then conducts a function approximation technique to attain $\chi_{inter_S}$.

The second is the determination of starting policy in the target task, $\pi_2^{(0)}$, based on similarity transition measures between the two related tasks as presented in Section 4.4. This is achieved by mapping state successor states pairs in the target task back to corresponding pairs in the source task and then conducting a search to the most similar transition recommended by the optimal policy of the source task. The action in the target task with the closest similarity to that in the source task ac-

companied with the intended initial state is used to approximate a good prior in the target task.

The third is using $\pi_2^{(0)}$ as a starting prior for the agent in the target task, detailed in Section 4.4. Here, the states recommended by $\pi_2^{(0)}$ are used as an initial trajectory to start from and improve on.

Results show that the algorithm was able to surpass ordinary fitted value iteration algorithms by attaining higher reward with fewer initial states. Additionally, results showed significant time reductions when attempting to find optimal policies in the target task, relative to the normal FVI algorithms.

In this approach, the common subspace required human intervention to be defined. This allowed the inter-task mapping to be autonomously learned. In some cases, such a space might be hard to manually define by a human. The next chapter improves on the proposed method and suggests a framework that is capable of autonomously learning an intertask mapping between different MDPs.

# 5

# Sparse Coded Inter-Task Mappings

**This chapter is based on:** *H. B. Ammar, M. E.Taylor, K. Tuyls, K. Driessens, and G. Weiss, "Reinforcement Learning Transfer via Sparse Coding ," in Proceedings of the eleventh conference on Autonomous Agents and Multiagent Systems (AAMAS), Valencia, Spain, 2012.*

In common subspace transfer as defined in the previous chapter, learning an intertask mapping was not fully automated but rather required a manually defined common subspace. That subspace was then used to relate source and target triplets needed to learn an intertask mapping. Although successful, it is sometimes hard to define such a subspace manually. Aiming at solving this problem, this chapter describes a framework that is capable of automatically learning an intertask mapping between MDPs with different constituents.

The primary contribution is a novel method to automatically learn an inter-task mapping based on sparse coding, sparse projection learning, and sparse Gaussian processes. In the first phase, two sparse coding steps are performed. In the first step an automatic dimensional mapping of both the source and target task state-action spaces is performed. While in the second, an automatic discovery of a high dimensional informative space in the source task state-action space is executed. This is also achieved through sparse coding, as described in Section 5.3, ensuring that transfer is conducted in a high representational space of the source task. In order to use a similarity measure among different patterns, the data should be present in the same space. Therefore, the target task triplets are projected to the high representational space of the source using sparse projection learning, as described in Section 5.3. The third and final phase is to approximate the inter-task mapping via a non-parametric regression technique, explained in Section 5.3.

Another contribution is to introduce two new algorithms for transfer between tasks of continuous state spaces: Transfer Least Squares Policy Iteration (TrLSPI) and

Transfer Fitted-Q-Iteration (TrFQI). Experiments illustrate the feasibility and suitability of the presented approach, and furthermore show that this introduced method can successfully transfer between two different RL benchmarks: transfer is successful between the inverted pendulum and the cart pole, as well as between mountain car and the cart pole.

## 5.1 Problem Definition

In this section the problem definition is detailed. Each of the source and target tasks are MDPs. These vary in each of their constituents. More specifically, the approach makes no restrictive assumptions on the type of variations between the tasks. In other words, not only the probability distributions or reward functions may vary between the tasks, but these variations can also happen between the state and/or action spaces. Therefore, the tackled problem is a *deep transfer* one, which is substantially harder than other forms of shallow transfer, such as these tackled in reward shaping, imitation learning, or lifelong learning. Furthermore, to increase the scope of applicability of the proposed methods, the state spaces of both the source and target tasks are continuous in nature and the operation is in a model free setting.

**Problem Definition:** Given two MDPs $\mathcal{M}_1 = \langle \mathcal{S}_1, \mathcal{U}_1, \mathcal{T}_1, \mathcal{R}_1, \gamma_1 \rangle$ and $\mathcal{M}_2 = \langle \mathcal{S}_2, \mathcal{U}_2, \mathcal{T}_2, \mathcal{R}_2, \gamma_2 \rangle$, where $\mathcal{S}_i$, $\mathcal{U}_i$, $\mathcal{T}_i$, $\mathcal{R}_i$, and $\gamma_i$ correspond to the state space, actions space, transition model, reward function, and discount factor for all $i \in \{1, 2\}$ representing the source and target tasks' MDP constituents, respectively, and some additional knowledge in both the source and the target, learn an intertask mapping $\chi$.

As in any other transfer learning framework, additional knowledge is assumed to be available. In this work, the knowledge available in the source is twofold: (1) random state-action-successor state transitions, and (2) (near-) optimal source policy $\pi_1^\star$. Such a policy can be used to generate (near-) optimal transitions that can then be transferred to aid the target agent. Furthermore, additional knowledge in the form of random state-action-successor state transitions is also available in the target. These and the random samples of the source help in learning an intertask mapping. It is worth noting, that randomly generating these samples is essential. The reason is that the mapping needs to operate within a broad range of the state and action spaces of both tasks. Learning mappings based only on optimal transitions, can not generalize well to broader unobserved ranges of the state and/or action spaces. However, after

learning such a broad range mapping, transferring "bad" information might "hurt" the target agent. Therefore, while transferring, only *optimal transitions* in the source are used to construct a starting policy in the target.

## 5.2 Overall Framework

Contrary to the previous method, described in Chapter 4, the intertask mapping is not split into two mappings. The main reason for combining these two together, is that if an algorithm has to automatically learn an intertask mapping, it has to simultaneously have knowledge about the state-action transitions. Moreover, combining the two gives more flexibility for discovering novel combinations of the state and/or action variables not available when the two mappings are split.

The overall procedure is shown in Figure 5.1. The easiest way to approach the problem of learning the intertask mapping is using supervised learning. However, according to the definitions of Section 5.1, $\chi$ maps state-action-successor state triplets in the source to these in the target. Therefore, any supervised learning algorithm will require a data set that has source transitions as inputs and target transitions as outputs. Unfortunately, it is nearly impossible for a human to manually determine which transitions in the source correspond to which in the target. Thus, this problem has also to be approached automatically. The simplest way to determine correspondence is to use a distance measure and search for the transition in the target that is closest to that in the source. However, these transitions might potentially belong to different dimensions. Consequently, before seeking any correspondence, the dimensions of these transitions should be unified.

To unify the dimensions a sparse coding step, denoted by "Sparse Coding Phase One" in Figure 5.1, is followed. Namely, the transitions of the source are sparse coded to attain the same number of dimensions as these of the target. Mainly, the idea is to discover new features in the state-action spaces not anticipated by original variables. However, the number of these new dimensions is limited to be the same as these of the target task.

Essentially, at this stage any transfer learning algorithm can be adopted. However, this new dimensional space might not be informative enough to conduct transfer. To ensure that transfer is performed in a highly informative space, another step of sparse coding is performed to discover "richer" features in the source ("Sparse Coding Phase Two" in Figure 5.1). Here the number of new dimensions can be set to a high value. Interestingly, due to the sparsity condition "unneeded" bases will end up attaining close to zero activations (see [57]) and therefore will not contribute in the transfer procedure.

This new space represents informative features in the source. However, it does

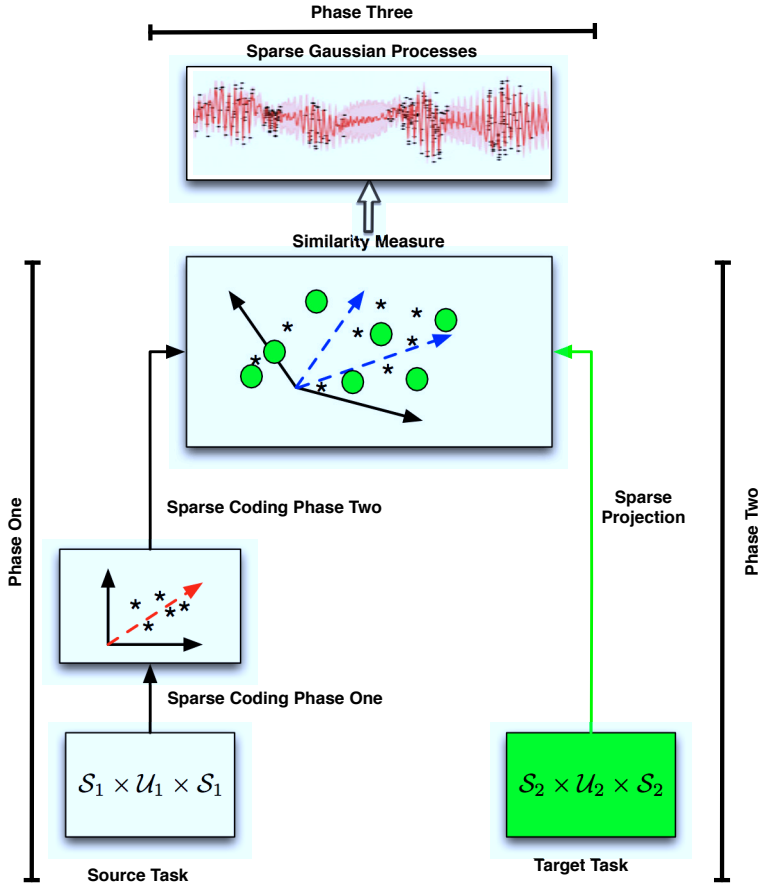



Figure 5.1: A high level schematic of the overall approach. It consists of three major phases. In the first high level features are detected in the source task MDP. In the second, samples from the target task are projected to that space. Finally, in the last phase (i.e., Phase Three), sparse Gaussian processes are used in order to learn the intertask mapping.

not relate yet to the target state and action spaces. Therefore, to use the similarity measure to determine the data set needed to approximate the intertask mapping, target triplets need to be projected to this new space. Projection means finding activations in the new bases that represent the original target transitions. This is performed using sparse projection as shown by "Phase Two" of Figure 5.1.

Now, the similarity measure to correspond the source and target triplets can be used. A minimum distance search between the source and the projected target triplets

is conducted to attain the data set needed by the regressor to learn the intertask mapping. Any supervised learning algorithm can be used. However, the intertask mapping might be a highly complex relation and therefore, the nonparametric sparse Gaussian processes framework (denoted by Phase Three in Figure 5.1) is adopted.

This intertask mapping can now be used to transfer samples from the source to the target task. More specifically, starting from different initial states and using the source task's optimal policy $\pi_1^\star$, samples in form of state-action-successor states can be attained. These can be passed through the inter-task mapping to have an initial batch of samples in the target. Starting from these batches, a sample-based RL algorithm can be used to attain the target's optimal behavior.

Next, each of the above phases are detailed. Section 5.3 describes the three phases, shown in Figure 5.1, needed to learn the inter task mapping. Section 5.4 describes the novel proposed transfer algorithms, TrFQI and TrLSPI. Experimental details are presented in Section 5.5. Namely, four transfer experiments from Inverted Pendulum (IP) to Cart Pole (CP) and from Mountain Car (MC) to CP using TrFQI and TrLSPI are shown. Section 5.6 presents an analysis and discussion of the proposed framework. Finally, Section 5.7 concludes and draws upon interesting directions for future research.

## 5.3 Learning an Inter-Task Mapping

In this section, the problem of learning an inter-task mapping, $\chi$, is casted as a supervised learning problem. As discussed in the previous chapters $\chi$ is typically split into an interstate mapping, $\chi_{inter_{\mathcal{S}}}$, and an interaction mapping $\chi_{inter_{\mathcal{U}}}$. Since the aim of this work is to automatically learn such mappings, a more flexible definition of $\chi$ is followed. Here, $\chi$ is defined to be a mapping between state-action-state triplets from the source task to the target task. This allows the agent to construct mappings that can potentially encode nonlinear combinations of state and/or actions as valid relations between the two tasks. Learning such a mapping requires related triplets from both tasks as data points for training. Unfortunately, obtaining such corresponding triplets is itself a hard problem — it is not trivial for a user to describe which triplets in the source task correspond to which triplets in the target task.

The problem is approached by automatically transforming the source task space (i.e., state-action-state space) into a higher representational space through SC, followed by a sparse projection of the target task triplets onto the attained bases. An Euclidean distance measure[1] is then used to gauge similarity. At this stage, the data

[1] An Euclidian distance might not be optimal, but optimizing this measure is planned as future research. Experiments show that more than reasonable results can be attained using this simple approximation.

set is ready to be provided to the learning algorithm so that it may construct the inter-task mapping. Please note, that multiple regression techniques could be leveraged at this stage but this work focuses on a non-parametric approximation scheme for two reasons. First, it is very hard for the user to presume a certain shape for such a mapping, and second generalization from only a small amount of target task transitions is desirable.

The following sections further clarify each of the above steps and explain the technicalities involved.

## **Phase One:** Sparse Coding Transfer for Reinforcement Learning

In phase one, two sparse coding (SC) steps are performed. The first insures that the state-action spaces of both MDPs are described in the same dimensions. Having this description already allows for transfer. However, this discovered space might not be informative enough. To ascertain that transfer is conducted in a highly rich and informative space, another step of sparse coding is preformed. Here SC discovers new feature in the data unanticipated in the original dimensions. Each of the previous steps, including their mathematical derivations are detailed next.

### Mapping the Source and Target Dimensions

To generate triplet matchings through SC, the first step is to match the dimensions of the state-action-state spaces of the source and target MDPs, which are likely to be different. In principle, after this step any existing TL in RL technique can be used. However, this paper goes further and proposes a new transfer framework based on the discovered bases and activations, described in Section 5.4.

SC is an unsupervised feature extraction algorithm. Given two data sets $\mathcal{D}_1 = \{\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)\prime}\rangle\}_{i=1}^{n_1}$ and $\mathcal{D}_2 = \{\langle s_2^{(j)}, u_2^{(j)}, s_2^{(j)\prime}\rangle\}_{j=1}^{n_2}$, where $n_1$ and $n_2$ represent the number of transitions in the source and the target[2], the first step makes use of sparse coding to unify the dimensions of the source to the target. The mathematical derivations involved in determining the relevant optimization problem to be solved are detailed.

**Mathematical Derivations:** Assuming that $\mathcal{S}_1 \times \mathcal{U}_1 \times \mathcal{S}_1 \in \mathbb{R}^{d_1}$, and $\mathcal{S}_2 \times \mathcal{U}_2 \times \mathcal{S}_2 \in \mathbb{R}^{d_2}$, with $d_1 < d_2$, SC tries to discover a set of over-complete basis by assuming a Gaussian distribution on the reconstruction error and a Laplace distribution on the activations. Given a source data $\mathcal{D}_1 = \{\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)\prime}\rangle\}_{i=1}^{n_1}$, the goal is to describe each vector $\vec{\boldsymbol{\alpha}}^{(i)} = \langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)\prime}\rangle \in \mathbb{R}^{d_1}$ as $\vec{\boldsymbol{\alpha}}^{(i)} \approx \sum_{k=1}^{d_2} \mathbf{b}_k a_k^{(i)}$, where $\mathbf{b}_k \in \mathbb{R}^{d_1}$. In the

[2]Typically, $n_2 << n_1$ where only few transitions are available from the target task.

standard generative model, the reconstruction error (i.e., $e^{(i)} = \vec{\boldsymbol{\alpha}}^{(i)} - \sum_{k=1}^{d_2} \mathbf{b}_k a_k^{(i)}$) is assumed to be distributed according to a zero mean Gaussian distribution with a covariance of $\sigma^2\mathbf{I}$, where $\mathbf{I}$ is the identity matrix[3]. To favor sparsity, the prior distribution for each coefficient $a_k$ is defined as: $p(a) = \frac{\exp(-\zeta\Lambda(a))}{\int \exp(-\zeta\Lambda(s))ds}$, where $\zeta \in \mathbb{R}$ is a constant , and $\Lambda(.)$ is a sparsity function, such as: (1) the $\mathrm{L}_1$ norm $||a_k||_1$, (2) the epsilon-$\mathrm{L}_1$ penalty $(a_k + \epsilon)^{\frac{1}{2}}$, or (3) the log-penalty function $\log(1 + a_k^2)$. For a more comprehensive survey of these different functions, the reader is referred to [57]. This leads to the following joint likelihood:

$$p\left(\vec{\boldsymbol{\alpha}}, \mathbf{A}, \mathbf{B}\right) \propto \prod_{i=1}^{n_1} \mathcal{N}\left(\sum_{k=1}^{d_2} a_k^{(i)} \mathbf{b}_k, \sigma^2\right) \prod_{i=1}^{n_1} \left[\prod_{k=1}^{d_2} \exp\left(-\zeta\Lambda(a_k^{(i)})\right)\right]$$

where, $\vec{\boldsymbol{\alpha}}$ is the collection of all inputs, $\mathbf{A}$ is the matrix collecting all the activations, and $\mathbf{B}$ is the matrix concatenating all the vectors $\mathbf{b}$.

Taking the logarithm yields the following:

$$\log\left(p\left(\vec{\boldsymbol{\alpha}}, \mathbf{A}, \mathbf{B}\right)\right) \propto \log\left(\prod_{i=1}^{n_1} \mathcal{N}\left(\sum_{k=1}^{d_2} a_k^{(i)} \mathbf{b}_k, \sigma^2\right) \prod_{i=1}^{n_1} \left[\prod_{k=1}^{d_2} \exp\left(-\zeta\Lambda(a_k^{(i)})\right)\right]\right)$$

which gives:

$$\begin{aligned}\log\left(p\left(\vec{\boldsymbol{\alpha}}, \mathbf{A}, \mathbf{B}\right)\right) &\propto \sum_{i=1}^{n_1} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left|\left|\vec{\boldsymbol{\alpha}}^{(i)} - \sum_{k=1}^{d_2} \mathbf{b}_k a_k^{(i)}\right|\right|_2^2\right)\right) \\ &\quad + \sum_{i=1}^{n_1}\left[\sum_{k=1}^{d_2}\left[\log\left(\exp\left(-\zeta\Lambda\left(a_k^{(i)}\right)\right)\right)\right]\right] \\ &\propto \log(\text{const.}) - \sum_{i=1}^{n_1} \frac{1}{2\sigma^2}\left|\left|\vec{\boldsymbol{\alpha}}^{(i)} - \sum_{k=1}^{d_2} \mathbf{b}_k a_k^{(i)}\right|\right|_2^2 - \sum_{i=1}^{n_1}\left[\sum_{j=1}^{d_2} \zeta\Lambda\left(a_k^{(i)}\right)\right]\end{aligned}$$

Maximizing the above equation leads to the following minimization problem:

$$\begin{aligned}\min_{\mathbf{a}^{(i)}, \mathbf{b}_k} \frac{1}{2\sigma^2} \sum_{i=1}^{n_1} ||\vec{\boldsymbol{\alpha}}^{(i)} - \sum_{k=1}^{d_2} \mathbf{b}_k a_k^{(i)}||_2^2 + \zeta \sum_{i=1}^{n_1}\left[\sum_{k=1}^{d_2}\left[\Lambda\left(a_k^{(i)}\right)\right]\right] \\ \text{s.t. } ||\mathbf{b}_k||_2^2 \leq c,\ \forall k = \{1, 2, \ldots, d_2\}\end{aligned} \tag{5.1}$$

with, $c \in \mathbb{R}$ is a constant, and $\mathbf{b}_k = [b_{k,1}, b_{k,2}, \ldots, b_{k,d_1}]^T$. The constraint on the bases

[3]Please note, that the vectors are not indexed by the input data points since the question is to determine a set of bases capable of describing all the given data points

was added to assure numerical stability, see [57, 82]. The first thing to recognize in Equation 5.1, is that the bases and the activations are not known in advance. The optimization problem has to be solved to determine these. The condition on a "good" solution is to: (1) minimize the reconstruction error, $\frac{1}{2\sigma^2}\sum_{i=1}^{n_1}||\vec{\boldsymbol{\alpha}}^{(i)} - \sum_{k=1}^{d_2}\mathbf{b}_k a_k^{(i)}||_2^2$, and (2) the activations should be induced sparsely.

Solving the optimization problem of Equation 5.1 is considered to be a tough challenge since the problem is not convex in both the activations and bases. Lee et al. [57], proposed to solve the above problem in two steps. In the first the bases are fixed and a feature sign algorithm is used to determine the activations. Given these activations, the next step commences to solve for the bases using Lagrange duals. It is out of the scope of this chapter to dive into the details of solving the problem of Equation 5.1. Interested readers are referred to [32, 57, 59, 83] for a comprehensive discussion of the topic.

**Unifying Source and Target Dimensions:** The optimization problem above, can now be applied to unify the source and target task state-action spaces. The "dimensional mapping" process is summarized in Algorithm 17. In short, it sparse codes random triplets $\langle s_1, a_1, s_1^{'}\rangle$ from the task with the lowest dimensionality, constrained to learn a number of bases ($d_2$) equal to the higher dimension of the unmodified task (regardless of which is the source and target task). Existing algorithms [57] are used to solve the Equation from step 6.1 of Algorithm 17.

**Algorithm 17** Sparse Coding TL for RL

**Require:** Triplets $\{\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)'}\rangle\}_{i=1}^{n_1}$ and $\{\langle s_2^{(j)}, u_2^{(j)}, s_2^{(j)'}\rangle\}_{j=1}^{n_2}$ from both MDPs.

1: Determine $d_1$ and $d_2$, the dimensions of the state-action-state spaces for both MDPs, where $d_1 \leq d_2$

2: Sparse code the lower dimensional triplets by solving:

$$\min_{\{\mathbf{b}_j\},\{a_j^{(i)}\}} \sum_{i=1}^{n_1}\frac{1}{2\sigma^2}||\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)'}\rangle - \sum_{k=1}^{d_2}\mathbf{b}_k a_k^{(i)}||_2^2 + \zeta\sum_{i=1}^{n_1}\sum_{k=1}^{d_2}\Lambda\left(a_k^{(i)}\right)$$

$$\text{s.t. } ||\mathbf{b}_k||_2^2 \leq c, \forall k = \{1, 2, \ldots, d_2\}$$

3: Solve the above equation by using an existing algorithm [57]

4: Return the Activation matrix ($\mathbf{A} \in \mathbb{R}^{n_1 \times d_2}$) and the Bases ($\mathbf{B} \in \mathbb{R}^{d_2 \times d_1}$)

At this level, any algorithm that is capable of preforming transfer between tasks of same state and/or actions spaces can be used. However, the discovered bases

and activations might not be informative enough to ensure good transfer. Therefore, the second step is to discover more informative features using another step of sparse coding as detailed next.

### High Information Representation

After mapping the source and target dimensions as described in the previous section, SC is again used to discover a succinct higher feature bases of the activations than the unified dimensional spaces discovered in Section 5.3. If successful this step will discover new features in the source task that could better represent relations with the target task than the bases discovered in Section 5.3.

**Mathematical Derivations:** Given the activation of the previous step, SC is conducted again to discover new features in the data. To derive the optimization problem to be solved, similar steps as in the previous SC step are applied. More specifically, a Gaussian distribution is assumed on the reconstruction error, and a sparsity penalty on the activations is added. This leads to the following joint likelihood:

$$p\left(\mathbf{A}, \mathbf{C}, \mathbf{Z}\right) \propto \prod_{i=1}^{n_1} \mathcal{N}\left(\sum_{l=1}^{d_n} c_l^{(i)} \mathbf{z}_l, \sigma^2\right) \prod_{i=1}^{n_1} \left[\prod_{l=1}^{d_n} \exp\left(-\zeta_1 \Lambda_1(c_l^{(i)})\right)\right]$$

where, $\mathbf{A}$ is the matrix collecting all of the activation vectors, $\mathbf{C}$ is the matrix of all activations, $\mathbf{Z}$ is the matrix of all basis vectors, $\zeta_1$ is a constant, $\Lambda_1$ is the sparsity penalty function, and $d_n$ represents the dimensions of the number of bases to be discovered.

Taking the logarithm yields the following:

$$\log\left(p\left(\mathbf{A}, \mathbf{C}, \mathbf{Z}\right)\right) \propto \log\left(\prod_{i=1}^{n_1} \mathcal{N}\left(\sum_{l=1}^{d_n} c_l^{(i)} \mathbf{z}_l, \sigma^2\right) \prod_{i=1}^{n_1} \left[\prod_{l=1}^{d_n} \exp\left(-\zeta_1 \Lambda_1(c_k^{(i)})\right)\right]\right)$$

which gives:

$$\log\left(p\left(\mathbf{A},\mathbf{C},\mathbf{Z}\right)\right) \propto \sum_{i=1}^{n_1} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left|\left|\mathbf{a}^{(i)} - \sum_{l=1}^{d_n} \mathbf{z}_l c_l^{(i)}\right|\right|_2^2\right)\right) + \sum_{i=1}^{n_1}\left[\sum_{l=1}^{d_n}\left[\log\left(\exp\left(-\zeta\Lambda_1\left(c_l^{(i)}\right)\right)\right)\right]\right]$$

$$\propto \log(\text{const.}) - \sum_{i=1}^{n_1}\frac{1}{2\sigma^2}\left|\left|\mathbf{a}^{(i)} - \sum_{l=1}^{d_n}\mathbf{z}_l c_l^{(i)}\right|\right|_2^2 - \sum_{i=1}^{n_1}\left[\sum_{j=1}^{d_n}\zeta\Lambda_1\left(c_l^{(i)}\right)\right]$$

where, $\mathbf{a}^{(i)} = [a_1^{(i)}, a_2^{(i)}, \ldots, a_{d_2}^{(i)}]^T$.

Maximizing the above equation leads to the following minimization problem:

$$\min_{\mathbf{c}^{(i)},\mathbf{z}_l} \frac{1}{2\sigma^2}\sum_{i=1}^{n_1}||\mathbf{a}^{(i)} - \sum_{l=1}^{d_n}\mathbf{z}_l c_l^{(i)}||_2^2 + \zeta_1\sum_{i=1}^{n_1}\left[\sum_{l=1}^{d_n}\left[\Lambda_1\left(c_l^{(i)}\right)\right]\right]$$
$$\text{s.t. } ||\mathbf{z}_l||_2^2 \leq o,\ \forall l = \{1,2,\ldots,d_n\}$$

where, $o \in \mathbb{R}$ is a constant, $\mathbf{c}^{(i)} = [c_1^{(i)}, c_2^{(i)}, \ldots, c_{d_n}^{(i)}]^T$, and $\mathbf{z}_l = [z_{l,1}, z_{l,2}, \ldots, z_{l,d_2}]^T$. The constraint on the bases is again added to assure numerical stability. Solving this minimization problem is also conducted using [57].

**Discovering High Features:** Algorithm 18, similar in spirit to Algorithm 17, describes the process, in which the previous optimization problem is used to discover new features. Algorithm 18 sparse codes the activations, which represent the original

**Algorithm 18** Succinct High Information Representation of MDPs

**Require:** Activations $\mathbf{A}$ from Algorithm 17, Target number of new high dimensional bases $d_n$.

1: Represent the activations in the $d_n$ bases by solving the following problem (again using the algorithm in [58]):

2:

$$\min_{\{\mathbf{z}_j\},\{c_j^{(i)}\}} \sum_{i=1}^{n_1}\frac{1}{2\sigma^2}||\langle\mathbf{a}_{1:d_2}\rangle^{(i)} - \sum_{j=1}^{d_n}\mathbf{z}_j c_j^{(i)}||_2^2 + \zeta_1\sum_{i=1}^{n_1}\sum_{l=1}^{d_n}\Lambda_1(c_l^{(i)})$$
$$\text{s.t. } ||\mathbf{z}_l||_2^2 \leq o, \forall l = \{1,2,\ldots,d_n\}$$

3: **return** Return new activations $\mathbf{C} \in \mathbb{R}^{n_1 \times d_n}$ and bases $\mathbf{Z} \in \mathbb{R}^{d_n \times d_1}$

source task triplets of the MDPs, to a higher representational space, $d_n$[4]. This stage should guarantee that the triplets of the source task MDP are projected into a high feature space where a similarity measure can be used to find a relation between the source and target task triplets. Note that there are no restrictions on the number of bases since unneeded bases will end up with an activation of zero.

Algorithm 18 discovers new features in the source or target state-action-state spaces. As TL typically transfers from a low dimensional source task to a high dimensional target task, SC determines new bases that are of a higher dimensionality than the original representation used for states and actions in the source task. These newly discovered bases can describe features not anticipated in the original design of the MDP's representation. These new features can highlight similarities between the source and target task thus helping and guiding the transfer learning scheme. The re-encoded triplets—described as a linear combinations of the bases and activations—do not yet relate to the triplets of the other task. The target task triplets still need to be projected towards these new sparse coded source task features. This is done as described in Section 5.3.

### **Phase Two:** $L_1$ Sparse Projection Learning

Once the above stages have finished, the source task triplets are described via the activations $\mathbf{C}$ (generated in Algorithm 18). However, target task triplets still have no relationship to the learned activations. Since a similarity correspondence between the source and target task triplets is being sought, the target task triplets should be represented in the same high informational space of the source task.

Therefore, the next step is to learn how to project the target task triplets onto the $\mathbf{Z}$ basis representation. In other words, the goal now is to learn a sparse projection that is capable of representing target task samples as a combination of some activations over the $\mathbf{Z}$ bases generated by Algorithm 18.

**Mathematical Derivations:** The mathematical derivations for determining these activations are closely related to these presented in the previous sections. The main difference here, is that the bases are already known and the optimization is performed only with respect to the activations. Since the bases have no prior, the reconstruction error will still be sampled according to a Gaussian distribution. Moreover, a sparse penalty on the *unknown* activations is also adopted to preserve sparsity. Therefore,

[4]In the experiments $d_n$ was set to be 100, a relatively high number.

the conditional likelihood can now be written as:

$$p(\vec{\boldsymbol{\alpha}}_{\mathbf{2}}{}^{(i)}, \boldsymbol{\phi}^{(i)}|\mathbf{Z}) \propto \mathcal{N}(\sum_{l=1}^{d_n} \phi_l^{(i)} \mathbf{z}_l, \sigma^2) \exp\left( -\beta \Lambda_1 \left( \boldsymbol{\phi}^{(i)} \right) \right)$$

where, $\beta \in \mathbb{R}$ is a constant, $\vec{\boldsymbol{\alpha}}_{\mathbf{2}}{}^{(i)} = \langle s_2^{(i)}, u_2^{(i)}, s_2^{(i)\prime} \rangle$ is an input vector of the target task, $\boldsymbol{\phi}^{(i)} = [\phi_1^{(i)}, \phi_2^{(i)}, \ldots, \phi_{d_n}^{(i)}]^T$ is the activation vector, $\mathbf{z}_l \in \mathbb{R}^{d_2}$ is the $l^{th}$ basis vector discovered in the previous step. Taking the natural logarithm and maximizing the likelihood leads to the following optimization problem:

$$\hat{\boldsymbol{\phi}^{(i)}}(\vec{\boldsymbol{\alpha}}_{\mathbf{2}}^{(\boldsymbol{i})}) = \arg\min_{\boldsymbol{\phi}^{(i)}} \left|\left| \vec{\boldsymbol{\alpha}}_{\mathbf{2}}^{(\boldsymbol{i})} - \sum_{l=1}^{d_n} \phi_l^{(i)} \mathbf{z}_l \right|\right|_2^2 + \beta \Lambda_1 \left( \boldsymbol{\phi}^{(i)} \right)$$

**Mapping Target Triplets:** Having derived the optimization problem, it can now be used to map the target triplets to the high dimensional space. The overall scheme is described in Algorithm 19, where the activations are learned by solving the $L_1$ regularized least squares optimization problem in step 5.2. This optimization problem guarantees that the activations are as sparse as possible and is solved using the interior point method [39]. The next step will be to order the data points from both the source and the target tasks, which are then used to approximate the inter-task mapping.

**Algorithm 19** Mapping Target Task Triplets

**Require:** Sparse Coded Bases $\mathbf{Z}$ generated by Algorithm 18, Target MDP triplets $\{\langle s_2^{(i)}, u_2^{(i)}, s_2^{(i)'} \rangle\}_{i=1}^{n_2}$

1: **for** $i = 1 \rightarrow n_2$ **do**

2: Represent the target data patterns in the sparse coded bases, $\mathbf{Z}$, by solving:

$$\hat{\boldsymbol{\phi}}^{(i)}\left( \left\langle s_2^{(i)}, u_2^{(i)}, s_2^{(i)\prime} \right\rangle \right) = \arg\min_{\boldsymbol{\phi}^{(i)}} \left|\left| \left\langle s_2^{(i)}, u_2^{(i)}, s_2^{(i)'} \right\rangle - \sum_{j=1}^{d_n} \phi_j^{(i)} \mathbf{z}_j \right|\right|_2^2 + \beta \Lambda_1 \left( \boldsymbol{\phi}^{(i)} \right)$$

3: **end for**

4: **return** Activations $\boldsymbol{\Phi}$

## **Phase Three:** Approximating an Inter-Task Mapping

To finalize the problem of approximating $\chi$, corresponding triplets from the source and target task should be provided to a regressor. This problem is approached by using a similarity measure in the high feature space, $\mathbf{Z}$, to identify similar triplets from the two tasks. This similarity measure identifies triplets from the source task that

are most similar to those of the target task, and then maps them together as being inputs and outputs for the regression algorithm, respectively, as shown on line 2 of Algorithm 20. Since the similarity measure is used in the sparse coded spaces, the distance is calculated using the attained activation ($\mathbf{C}$ and $\mathbf{\Phi}$) rather than the triplets themselves. Therefore, the scheme has to trace the data back to the original dimensions of the state-action pairs of the MDPs.

**Algorithm 20** Similarity Measure & Inter-Task mapping approximation

**Require:** Sparse Coded Basis $\mathbf{Z}$, Sparse Coded Activations of the source task $\mathbf{C} \in \mathbb{R}^{n_1 \times d_n}$, Projected Target Task activations $\mathbf{\Phi} \in \mathbb{R}^{n_1 \times d_n}$

1: **for all** $\phi$ **do**
2: Calculate the closest activation in $\mathbf{C}$ minimizing the Euclidean/similarity distance measure.
3: **end for**
4: Create a data set $\mathcal{D}$ from minimum-distance triplets
5: Approximate the Inter-task mapping, $\chi$, from $\mathcal{D}$ with an appropriate learning algorithm
6: **return** The approximated Inter-task mapping, $\chi$

## 5.4 Transfer Scheme

This section proposes two novel transfer algorithms for pairs of tasks with continuous state spaces and discrete action spaces, titled Transfer Least Squares Policy Iteration (TrLSPI) and Transfer Fitted-Q-Iteration (TrFQI), which makes use of a learned source task policy. The novel algorithms build on LSPI and FQI, both of which were explained in Section 2.3. Furthermore, the intertask mapping might be a highly complex function relating the state and action spaces of both MDPs. Therefore, a nonparametric technique is followed to approximate such a relation. More specifically, the Sparse Gaussian Processes framework (SPGP) is adopted for efficiency, accuracy and generalization capabilities. The details are introduced and discussed next.

### Transfer Least Squares Policy Iteration

TrLSPI is described in Algorithm 21 and can be applied to any TL in RL problem having continuous states and discrete action spaces. It is also sample efficient as it preserves the advantages of the normal LSPI algorithm. TrLSPI can be split into two sections. The first (lines 1–4 of Algorithm 21) determine $\chi$ (see Section 5.3), using source and target task triplets.[5] The second section (lines 5–9) provides triplets (using

[5]The source task policy may be optimal or near-optimal, depending on the RL algorithm used in the source task.

$\pi_1^\star$) as a start for the evaluation phase of the LSPI algorithm (LSTDQ), allowing it to improve the target task policy. If the tasks are similar, and if the inter-task mapping is "good enough," then those triplets will 1) bias the target task controller towards choosing good actions and 2) restrict its area of exploration, both of which help to reduce learning times and increase performance.

**Algorithm 21** TrLSPI

**Require:** Source MDP triplets $\{\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)'}\rangle\}_{i=1}^{m}$, Target MDP triplets $\{\langle s_2^{(j)}, u_2^{(j)}, s_2^{(j)'}\rangle\}_{j=1}^{n_2}$, Number for re-samples $n_s$, close to optimal policy for the source system $\pi_1^\star$, State action basis functions for the target task $\psi_1, \ldots, \psi_k$

1: *Map the Dimensions* using Algorithm 17
2: Discover *High Informational Representation* using Algorithm 18
3: *Sparse Project* the target task triplets using Algorithm 19
4: Use a *similarity* measure to attain the data set and *approximate* $\chi$ using Algorithm 20
5: Randomly sample $n_s$ source task triplets $\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)'}\rangle_{i=1}^{n_s}$ greedily in the optimal policy $\pi_1^\star$, set of state-dependent basis function $\psi_1, \ldots, \psi_k : \mathcal{S}_2 \times \mathcal{U}_2 \to \mathbb{R}$
6: **for** $i = 1 \to n_s$ **do**
7: Find the corresponding target task triplets as $\langle s_2^{(i)}, u_2^{(i)}, s_2^{(i)\prime}\rangle = \chi(\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)\prime}\rangle)$
8: **end for**
9: Use the black box generative model of the environment to produce the rewards on the transferred triplets
10: Use LSTDQ to evaluate transformed triplets
11: Improve policy until convergence using LSPI
12: **return** Learned policy $\pi_2^\star$

Algorithm 21 leverages source task triplets to attain a good starting behavior for the target task. The performance of the policy necessarily depends on the state space region where those triplets were provided. In other words, it is not possible to achieve near-optimal performance with a small number of triplets that are in regions far from the goal state.[6] Therefore, to learn a near-optimal policy, it must collect triplets from the target with the current policy, or a large number of source task triplets should be provided. A full model of the system is not required but the algorithm does require a black box generative model for sampling.

[6]This is a problem that is inherit to LSPI and not a property of the TrLSPI algorithm.

## Transfer Fitted-Q-Iteration

The second novel algorithm, Transfer Fitted-Q Iteration (TrFQI), is also capable of transferring between MDPs with continuous state space and countable actions and preserves the advantages of the standard FQI algorithm. The key idea is to provide a good starting sample distribution on which the FQI algorithm can learn. This distribution is provided to the target task agent via $\chi$, which in turn maps the source task triplets (sampled according to $\pi_1^\star$) into the target task. Algorithm 22 presents the pseudocode and can also be split into two parts lines 1-8: (1) Use the inter-task mapping to project the source task triplets to the target task (same steps followed in the first part of TrLSPI) and lines 9-10 (2) provide those triplets to the FQI Algorithm to learn a policy.[7]

**Algorithm 22** TrFQI

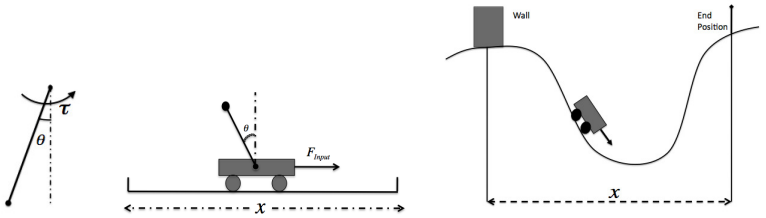
**Require:** Source MDP triplets $\{\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)'}\rangle\}_{i=1}^{n_1}$, Target MDP triplets $\{\langle s_2^{(j)}, u_2^{(j)}, s_2^{(j)'}\}_{j=1}^{n_2}$, Number for re-samples $n_s$, close to optimal policy for the source system $\pi_1^\star$, State action basis functions for the target task $\psi_1, \ldots, \psi_k$

1: *Map the Dimensions* using Algorithm 17
2: Discover *High Informational Representation* using Algorithm 18
3: *Sparse Project* the target task triplets using Algorithm 19
4: Use a *similarity* measure to attain the data set and *approximate* $\chi$ using Algorithm 20
5: Randomly Sample $n_s$ Source task triplets $\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)'}\rangle_{i=1}^{n_s}$ greedily in the optimal policy $\pi_1^\star$, set of state-dependent basis function $\psi_1, \ldots, \psi_k : \mathcal{S}_2 \times \mathcal{U}_2 \to \mathbb{R}$
6: **for** $i = 1 \to n_s$ **do**
7: Find the corresponding target task triplets as $\langle s_2^{(i)}, u_2^{(i)}, s_2^{(i)\prime}\rangle = \chi(\langle s_1^{(i)}, u_1^{(i)}, s_1^{(i)\prime}\rangle)$
8: **end for**
9: Use the black box generative model of the environment to produce the rewards on the transferred triplets
10: Apply FQI
11: **return** Learned policy $\pi_2^\star$

[7] It is also worth noting that the generalization of this algorithm depends on the type of function approximators used to approximate the Q-function. This is a property of FQI and not of TrFQI. Therefore, if the algorithm has to attain near-optimal behavior, either a large amount of triplets should be provided, or it must again have access to a black box generative model of the MDP for re-sampling.

## 5.5 Experiments & Results

Two experiments to evaluate the framework were conducted. The first was the transfer from the Inverted Pendulum (IP), Figure 5.2(a), to the Cart Pole (CP), Figure 5.2(b). The second experiment transfers between Mountain Car (MC), Figure 5.2(c) and CP. This section describes the experiments conducted.




(a) Inverted Pendulum (b) Cart Pole (c) Mountain Car

Figure 5.2: Experimental domains

### Inverted Pendulum to Cart Pole Transfer

The source task was the inverted pendulum problem. The state variables describing the systems are the angle and angular velocity $\{\theta,\ \dot{\theta}\}$. The control objective of the IP is to balance the pendulum in an upright position with an angle, $\theta = 0$ and angular velocity $\dot{\theta} = 0$. The allowed torques are $+50$, 0 and $-50\ Nm$. The reward function is $cos(\theta)$ which yields its maximum value of $+1$ at the up-right position.

In cart pole, the goal is to swing up the pole and keep it balanced in the upright position (i.e., $\theta = \dot{\theta} = 0$). The allowed actions are $(+1)$ for full throttle right and (-1) for full throttle left. The reward function of the system consisted of two parts: (1) $cos(\theta)$, which yields its maximum value of $+1$ at the upright position of the pole, and (2) $-1$ if the cart hits the boundaries of the track. The angle was restricted to be within $|\theta| < \frac{\pi}{9}$ while the position was restricted to $|x| < 3$.

In order to transfer between IP and CP, we first learn an optimal policy in the source task, $\pi^{\star}_{IP}$, with LSPI. $\pi^{\star}_{IP}$ was then used to randomly sample different numbers of initial states of task, to be used by $\chi$. We started with 5000 and 2500 randomly sampled states (using a random policy) for the IP and the CP, respectively. These triplets were used by the algorithm described in Section 5.3 to learn the inter-task mapping $\chi$[8]. After $\chi$ had been learned, different numbers of samples were collected

[8]We believe but have not confirmed that the samples to learn $\chi$ should be provided using random

from the source task using $\pi^{\star}_{IP}$. Specifically, we have sampled 500, 1000,... 20000 states as input to the TrLSPI and the TrFQI algorithms to measure performance and convergence times.

### TrLSPI Results

The results show both an increase in the performance on a fixed number of samples and a decrease in the convergence times in both a predefined number of samples and to attain an optimal policy. The performance was measured as the number of steps during an episode to control the pole in an upright position on a given fixed amount of samples. Figure 5.3 summarizes the results attained on a different number of transferred samples and compares them with those attained through normal LSPI learning scheme. It clearly shows an increase in the number of control steps in the case of the transferred samples compared to a random sampling scheme. It can be seen that at a small number of samples, e.g. 2000, TrLSPI was able to attain an average of 520 control steps with about 400 for the random case. This performance increases with the number of samples to reach 1200 steps at 10000 transferred samples. It is also worth noting that the agent had an additional 300 jumpstart steps once converging on the transferred samples compared to a random sampling scheme. Another performance measure was the time required to learn a near-optimal target task policy using only a fixed number of samples. There was a decrease in the convergence times, represented by the number of iterations in LSPI, provided a fixed amount of transferred samples. LSPI was able to converge faster once provided the transferred samples compared to a random sample data set. For example, it took LSPI 5 iteration to converge provided 5000 transferred samples with 7 iterations for the random case. Further the algorithm converged within 12 iteration provided 20000 transferred samples while it took it about 19 for the random case. Finally, LSPI was able to converge to an acceptable policy within 22.5 minutes after being provided a random data set, compared to 16 minutes with the transferred data set[9]. Calculating $\chi$ took an addition 3.7 minutes.

### TrFQI Results

Similar experiments using the other proposed algorithm TrFQI were performed. Similar results were observed as could be seen from Figure 5.6. It is clear that the transferred samples produce more control steps on the target task compared to learning on random samples. As an illustration, the algorithm was able to produce the threshold of 800 control steps on an amount of 5000 transferred states while it needed

policies in both the source and the target task as we need to cover most large areas of the state-action spaces in both tasks.

[9]The experiments were performed on a 2.8 Ghz Intel Core i7.

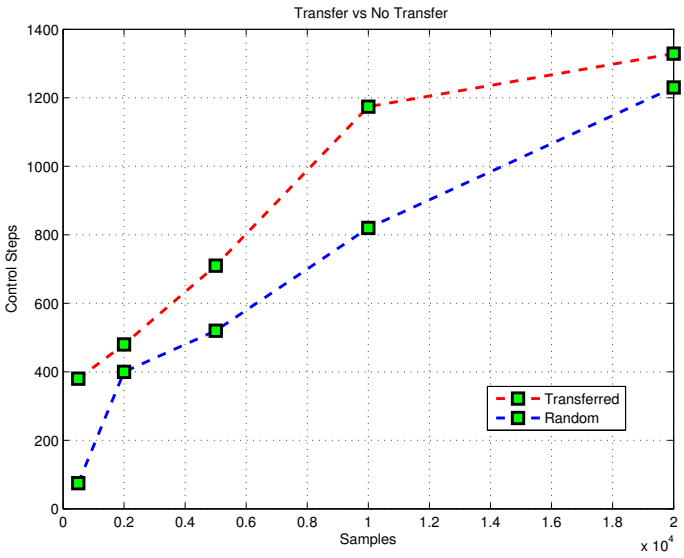



Figure 5.3: Cart Pole results from LSPI and TrLSPI after learning on Inverted Pendulum: the performance is measured after collecting 500 different initial states in the target tasks.

to be provided with about 10000 random samples in order to attain that threshold. A decrease in the number of training iterations in the TrFQI compared to FQI at a fixed number of samples and to attain an optimal policy is also reported. Good performance at 50 iterations of training on transferred samples compared to 70 iterations for the random case was observed. Moreover, TrFQI was able to reach a suboptimal acceptable policy with about 85 iteration once using transferred samples compared to a 150 iterations for the random case.

## Mountain Car to Cart Pole Transfer

To test to what extend the proposed framework is robust another set of experiments were performed. In these experiments, the tasks were highly dissimilar. The source task was the MC problem, a benchmark RL task. The car has to drive up the hill (Figure 5.2(c)). The difficulty is that gravity is stronger than the car's motor—even at maximum throttle the car can not directly reach the top of the hill. The dynamics of the car are described via two continuous state variables ($x$, $\dot{x}$) representing the position and velocity of the center of gravity of the car, respectively. The input action takes on three distinct values: maximum throttle forward (+1), zero throttle

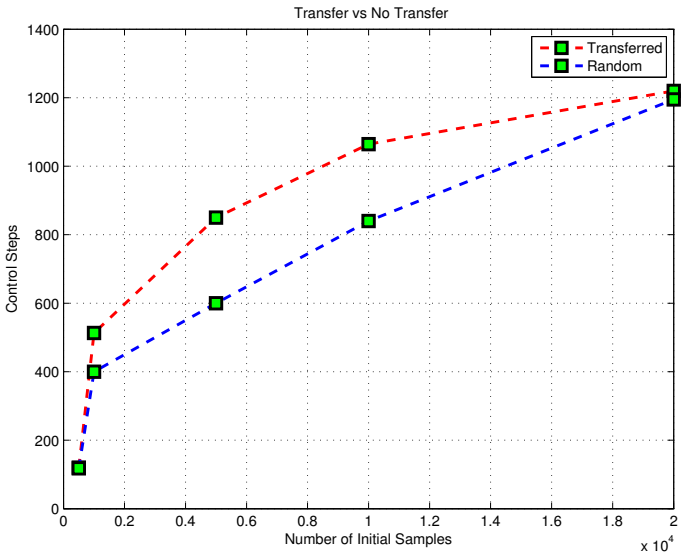



Figure 5.4: Cart Pole results with FQI and TrFQI after learning on Inverted Pendulum: the performance is measured after collecting 500 different initial states in the target tasks.

(0), and maximum throttle reverse (-1). The car is rewarded by +1 once it reaches the top of the hill, $-1$ if it hits the wall, and zero elsewhere.

The target task is the same Cart Pole problem described in the previous experiment.

SARSA($\lambda$) [98] was used to learn $\pi^{\star}_{MC}$ in the source task. The policy was then used to randomly sample different numbers of source task states, to be used by $\chi$. First, 5000 and 2500 randomly sampled states for the Mountain Car and the Cart Pole, respectively. These were used by the algorithm described in Section 5.3 to learn the inter-task mapping $\chi$. After $\chi$ has been learned, different numbers of samples were collected from the source task using $\pi^{\star}_{MC}$. Specifically, 500, 1000,... 20000 states were sampled. These were then provided as input to the TrLSPI and the TrFQI algorithms to measure performance and convergence times.

### TrLSPI Results

Figure 5.5 clearly shows an increase in the number of control steps in the case of the transferred samples compared to a random sampling scheme. It can be seen that at a small number of samples, e.g. 2000, TrLSPI was able to attain an average of

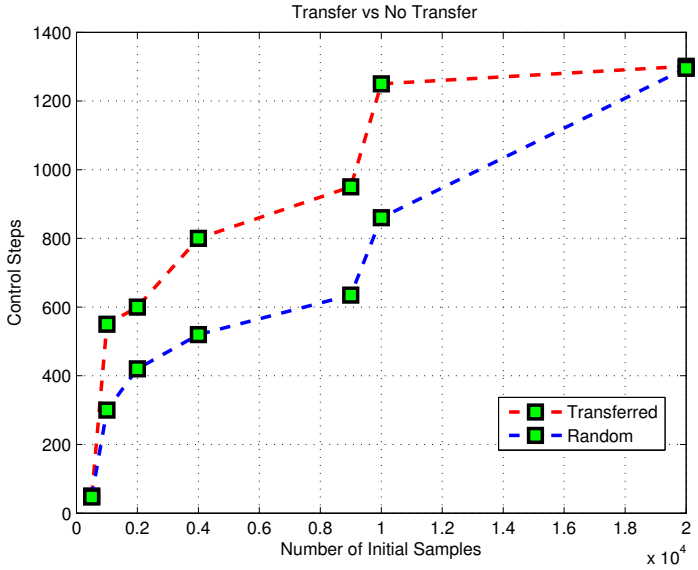



Figure 5.5: Cart Pole results using LSPI and TrLSPI after learning on Mountain Car: the performance is measured after collecting 500 different initial states in the target tasks.

600 control steps with about 4000 samples for the random case. This performance increases with the number of samples to finally reach about 1300 control step on 20000 samples for both cases. A decrease in the convergence times, represented by the number of iterations in LSPI, provided a fixed amount of transferred samples is also reported. LSPI was able to converge faster once provided the transferred samples compared to a random sample data set. For example, it took LSPI 7 iteration to converge provided 5000 transferred samples with 12 iterations for the random case. Further, the algorithm converged within 14 iteration provided 20000 transferred samples while it took it about 19 for the random case. Finally, LSPI was able to converge to an acceptable policy within a 22.5 minutes after being provided a random data set, compared to 17 minutes with the transferred data set[10]. Calculating $\chi$ took an addition 3.7 minutes.

### TrFQI Results

Similar experiments using TrFQI were performed. Close results were observed as could be seen from Figure 5.6. It is clear that the transferred samples where able

[10]Our experiments were performed on a 2.8 Ghz Intel Core i7

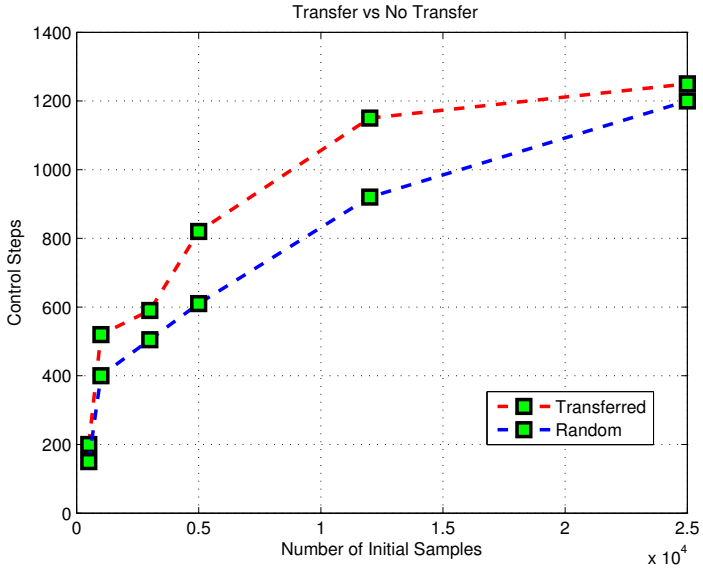



Figure 5.6: Transfer Results on the Cart Pole task using TrFQI after learning on Mountain Car: the performance is measured after collecting 500 different initial states in the target tasks.

to produce a higher amount of control steps on the target task compared to learning on random samples in the target tasks. As an illustration, the algorithm was able to produce the threshold of 800 control steps on an amount of 5000 transferred states while it needed to be provided with about 9000 random samples in order to attain the threshold. It is also clear that both TrFQI and the normal FQI both converged to about the same number of control steps (1200) once provided a large amount of samples. A decrease in the number of training iterations at a fixed number of samples and to attain an optimal policy is also reported. Good performance at 50 iteration of training on transferred samples compared to 80 iterations for the random case is reported. Moreover, TrFQI was able to reach a suboptimal policy with about 91 iteration once using transferred samples compared to a 150 iterations for the random case.

## 5.6 Analysis & Discussion

It is clear from the results presented that the learner's performance increased using the proposed framework, relative to a random selection scheme. Policy performance improved, as measured by the number of control steps achieved by the agent on the

target task. The number of learning iterations required also decreased, as measured by the number of iterations required by the algorithm to converge to a policy on a fixed number of transferred samples. This leads to conclude that TrFQI and TrLSPI both:

1. provided a better distribution of samples compared to random policy in the target task,

2. required less iterations to converge to a fixed policy when provided a fixed number of transferred samples, and

3. reached a near-optimal performance policy faster than when using random selection scheme.

Furthermore, these results show that the proposed framework

4. successfully learned an inter-task mapping between two sets of different RL tasks.

The framework is applicable to any model-free TL in RL problem with continuous state spaces and discrete action spaces, covering many real world RL problems. It has the advantage of automatically finding the inter-task functional mapping using SC and any "good" regression technique. One potential weakness is that the framework should work correctly when the two tasks at hand are *semantically* similar, as the rewards of the two systems were not taken into account in the explained scheme. For instance, consider the transfer example between the cart pole and "cart fall" tasks. The control goals of these two tasks are opposite whereby in the cart pole the pole has to be balanced in the up right position while in the cart fall the pole has to be dropped as fast as possible. In other words, the agents have the same transitions in the two tasks but have to reach two opposite goal states of the pole. The mapping scheme of Section 5.3, once applied, will produce a one-to-one mapping from the source to the target task relating the same transitions from both of the tasks together. Clearly the optimal policies of the two tasks are opposite. Therefore, in this case the target task has been provided with a poor bias, potentially hurting the learner (i.e., negative transfer). The approach will be able to avoid this scheme once the rewards are added to the similarity measure generating the training set to approximate the inter-task mapping $\chi$, but such investigation is left to future work. Furthermore, it might be the case, that if transitions in and out of the goal states were sampled frequently enough, as is the case in episodic tasks, then these relations might end-up being learnt by the proposed algorithm and therefore, avoid negative transfer.

## 5.7 Conclusions & Future Work

This chapter has presented a novel technique for transfer learning in reinforcement learning tasks. The framework may be applied to pairs of reinforcement learning problems with continuous state spaces and discrete action spaces. The main contributions of this chapter are (1) the novel method of automatically attaining the inter-task mapping, $\chi$ and (2) the new TrLSPI and TrFQI algorithms for tasks with continuous state spaces and discrete actions. The problem was approached by framing the approximation of the inter-task mapping as a supervised learning problem that was solved using sparse pseudo input Gaussian processes. Sparse coding, accompanied with a similarity measure, was used to determine the data set required by the regressor for approximating $\chi$. Results demonstrate successful transfer between two similar tasks, inverted pendulum to cart pole, and two very different tasks, mountain car to cart pole task. Success was measured both in an increase in learning performance as well as a reduction in convergence time.

Although successful, the proposed method might suffer from a problem that is inherent to sparse coding. More specifically, SC will not learn informative bases if the delivered data set was highly sparse and non-informative [82]. In RL this might be a potential problem due to the large spaces that need to be explored. Seeking a more robust technique, the next chapter provides an alternative to the learning of such intertask mappings based on Restricted Boltzmann Machines.

# 6

# Transfer Restricted Boltzmann Machines

**This chapter is based on:** *H. B. Ammar, D. C. Mocanu, M. Taylor, K. Driessens, G. Weiss, and K. Tuyls, "Automatically Mapped Transfer Between Reinforcement Learning Tasks via Three-Way Restricted Boltzmann Machines," in Proceedings of the European Conference on Machine Learning (ECML)-in review, Prague, Czech Republic, 2013.*

In the previous chapter, a framework to learn intertask mappings between different MDPs was presented. The previous approach was based on sparse coding, sparse projection, and sparse Gaussian processes to approximate such mappings. Although successful, it might suffer if the samples available in the source and target task are not informative enough. In RL such problems might easily occur due to either the sampling procedure, or due to spaces that need to be explored. This chapter provides a more robust alternative to the learning of such mapping.

It introduces an autonomous framework for learning intertask mappings based on *restricted Boltzmann machines* (RBMs) [91]. RBMs provide a powerful but general framework that can be used to describe an abstract common space for different tasks. This common space is then used to represent the inter-task mapping between two tasks and can successfully transfer knowledge about transition dynamics between the two tasks.

The contributions of this chapter are summarized as follows. First, a novel high-order RBM is proposed that uses a three-way weight tensor (i.e., TrRBM). Since this machine has a computational complexity of $\mathcal{O}(N^3)$, a factored version (i.e., FTrRBM) is then derived that reduces the complexity to $\mathcal{O}(N^2)$. Experiments then transfer samples between pairs of tasks, showing that the proposed method is capable of successfully learning a useful inter-task mapping. Specifically, the results demonstrate that FTrRBM is capable of:

- Automatically learning an inter-task mapping between different MDPs,

- Transferring informative samples that reduce the computational complexity of a sample-based RL algorithm, and
- Transferring informative instances which reduce the time needed for a sample-based RL algorithm to converge to a near-optimal behavior.

## 6.1 Problem Definition

In this section the problem definition is detailed. Each of the source and target tasks are MDPs. These vary in each of their constituents. More specifically, the approach makes no restrictive assumptions on the type of variations between the tasks. In other words, not only the probability distributions or reward functions may vary between the tasks, but these variations can also happen between the state and/or action spaces. Therefore, the tackled problem is a *deep transfer* one, which is substantially harder than other forms of shallow transfer, such as these tackled in reward shaping, imitation learning, or lifelong learning. Furthermore, to increase the scope of applicability of the proposed methods, the state spaces of both the source and target tasks are continuous in nature and the operation is in a model free setting.

Namely, given two MDPs $\mathcal{M}_1 = \langle \mathcal{S}_1, \mathcal{U}_1, \mathcal{T}_1, \mathcal{R}_1, \gamma_1 \rangle$ and $\mathcal{M}_2 = \langle \mathcal{S}_2, \mathcal{U}_2, \mathcal{T}_2, \mathcal{R}_2, \gamma_2 \rangle$, where $\mathcal{S}_i$, $\mathcal{U}_i$, $\mathcal{T}_i$, $\mathcal{R}_i$, and $\gamma_i$ correspond to the state space, actions space, transition model, reward function, and discount factor for all $i \in \{1, 2\}$ representing the source and target tasks' MDP constituents, respectively, and some additional knowledge in both the source and the target, learn and intertask mapping $\chi$. Again $\chi$ is not split into two mappings (i.e., interstate and interaction mappings). The main reason for combining these two together, is that if an algorithm has to automatically learn an intertask mapping, it has to simultaneously have knowledge about the state-action transitions. Moreover, combining the two gives more flexibility for discovering novel combinations of the state and/or action variables not available when the two mappings are split.

As in any other transfer learning framework, additional knowledge is assumed to be available. In this work, the knowledge available in the source is twofold: (1) random state-action-successor state transitions, and (2) (near-) optimal source policy $\pi_1^\star$. Such a policy can be used to generate (near-) optimal transitions that can then be transferred to aid the target agent. Furthermore, additional knowledge in the form of random state-action-successor state transitions is also available in the target.

These and the random samples of the source help in learning an intertask mapping. It is worth noting, that randomly generating these samples is essential. The reason is that the mapping needs to operate within a broad range of the state and action spaces of both tasks. Learning mappings based only on optimal transitions, can

not generalize well to broader unobserved ranges of the state and/or action spaces. However, after learning such a broad range mapping, transferring "bad" information might "hurt" the target agent. Therefore, while transfer, only *optimal transitions* in the source are used to construct a starting policy in the target.

## 6.2 Overall Framework

The idea in this chapter is to robustly and automatically discover a new space that is capable of representing the transitions in both MDPs. The overall framework is shown in Figure 6.1. The common space is encoded by a high-order Restricted Boltzmann Machine. The machine consists of three layers: (1) source task visible layer, (2) target task visible layer, and (3) hidden layer. These layers are connected using a three-way weight tensor. From the random samples of both the source and target task the variables (i.e., free parameters) of the machine are trained. After converging, the hidden layer is capable of relating source and target triplets together. In other words, the attained space is capable of representing both tasks in one unified space.

This representation can now be used in order to perform transfer. However, not to hurt the target agent, only "useful" knowledge is used to transfer. This is manifested by the "Transfer Phase" in Figure 6.1. The idea is then to sample the source task using an optimal policy to have a bag of "good" transitions. These are then passed through the common space that was learned by the RBM. Since this machine is bi-directional, then an initial bag of samples in the target task is attained. These can then be used by a sample-based RL algorithm.

Next each of the above are detailed. The next section presents a derivation of the full model. However, this model is computationally expensive, and a factored version of the model is developed in Section 6.3. The usage of the proposed machines is then detailed in Section 6.4 (learning phase) and Section 6.4 (transfer phase). Section 6.5 details the experiments that have been conducted. Finally, Section 6.6 concludes with a discussion and further directions of future work.

## 6.3 RBMs for Transfer Learning

The core hypothesis of this chapter is that RBMs can automatically build an inter-task mapping using source task and target task samples, because an RBM can discover the latent similarities between the tasks, implicitly encoding an inter-task mapping. To construct an algorithm to test this hypothesis, the TrRBM framework consists of three layers as shown in Figure 6.2. The first is the source task visible layer (describing source samples), the second is the target task visible layer (describing target samples),

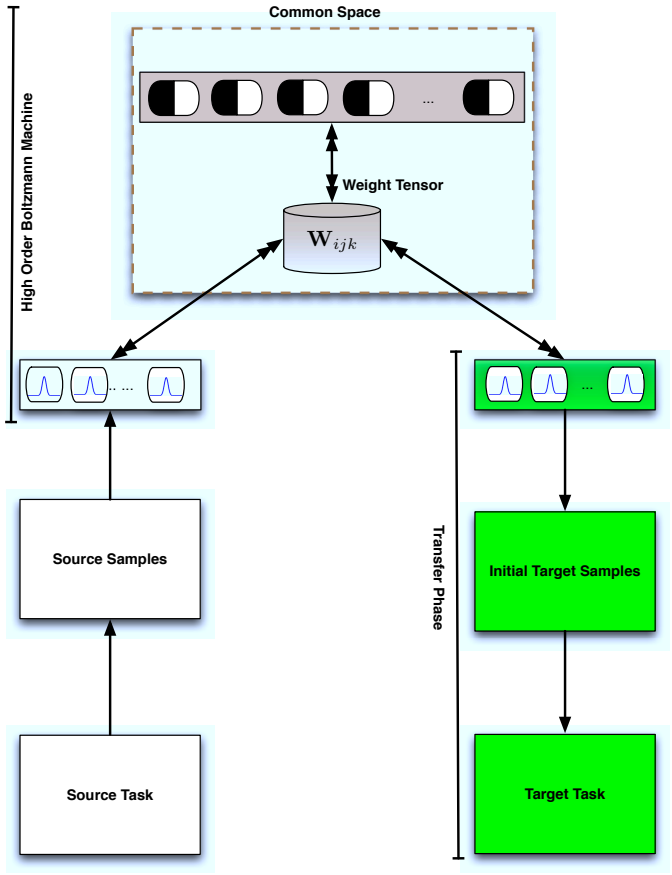



Figure 6.1: The overall framework .

and the third is the hidden layer that encodes similarities between the two tasks. This hidden layer therefore encodes a type of inter-task mapping, which will be used to transfer samples from the source to the target.

## Transfer Restricted Boltzmann Machine

TrRBM is used to automatically learn the inter-task mapping between source and target tasks. TrRBM consists of three layers: (1) a visible source task layer, (2) a visible target task layer, and (3) a hidden layer. The number of units in the visible layers is equal to dimensionality of each of the source and target task transitions, respectively. Since the inputs might be of continuous nature, the units in the visible

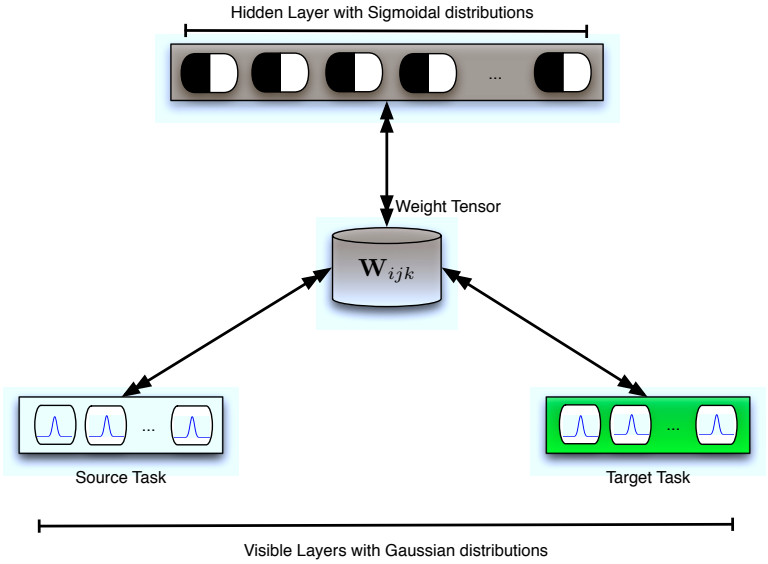



Figure 6.2: The overall structure of the proposed full model. The three-way weight tensor is shown in the middle with its connections to each of the visible source layer, visible target layer, and the hidden layer shown in red, green, and orange respectively.

units are set to be Gaussians with means, which are learned as described later in this section. These three layers are connected with a three-way weight tensor. Formally, the visible source layer is $\mathcal{V}_1 = [v_1^{(1)}, \ldots, v_1^{(d_1)}]$, where $v_1^{(i)}$ is a Gaussian $\mathcal{N}\left(\mu_1^{(i)}, \sigma_1^{(i)2}\right)$, with a mean $\mu_1^{(i)}$, a variance $\sigma_1^{(i)2}$, and $d_1$ representing the number of dimensions of the a source transition, $\langle s_1, u_1, s_1' \rangle$. The visible target layer $\mathcal{V}_2 = [v_2^{(1)}, \ldots, v_2^{(d_2)}]$, where $v_2^{(k)}$ is a Gaussian $\mathcal{N}\left(\mu_2^{(j)}, \sigma_2^{(j)2}\right)$, with a mean $\mu_2^{(j)}$, a variance $\sigma_2^{(j)2}$, and $d_2$ representing the dimensions of a target task transition $\langle s_2, u_2, s_2' \rangle$. The hidden layer $\mathcal{H} = [h^{(1)}, \ldots, h^{(n_h)}]$ consists of sigmoidal activations, where $n_h$ denotes the number of hidden units.

Next, the mathematical formalizations of TrRBM are detailed.

### Energy of the Full Model

The energy function of the full model is defined as:

$$E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = -\sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2} \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^{k}} \right)^2 - \sum_{ijk} W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \tag{6.1}$$

where $W_{ijk}$ is the three-way weight tensor connecting each of the visible source, the visible target, and the hidden layers. The above equation contains four terms. The first term $\left( \text{i.e., } \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right)$ represents the contribution of the source task samples in the energy function of the RBM. The biases of this layer are represented by $a$. The second term $\left( \text{i.e, } \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right)$ presents the contribution of the hidden units to the energy function. The biases of this layer are denoted by $b$. The third denotes the contribution of the target instances to the energy function with its biases denoted by $c$. The last term describes the connection between each of the above layers using the three-way weight tensor $\left( \text{i.e., } W_{ijk} \right)$. It is worth noting that the free parameters that need to be fit are the biases and the three-way weight tensor (i.e., $a$, $b$, $c$, and $W_{ijk}$).

### Inference in the Full Model

Inference corresponds to calculating the conditional probabilities of each of the units in the different layers, depending on the configuration of the other layers. Because there are three layers, three conditional probabilities must to be computed. These are: (1) the probability of the visible source layer given the other two layers and parametrized by the weights $\left( \text{i.e., } p(\mathcal{V}_1 | \mathcal{V}_2, \mathcal{H}; W) \right)$, (2) the probability of the hidden layer given the other two and the weight parameterizations $\left( \text{i.e., } p(\mathcal{H} | \mathcal{V}_1, \mathcal{V}_2; W) \right)$, and (3) the probability of the visible target conditioned on the other two layers and parameterized by the weights $\left( \text{i.e., } p(\mathcal{V}_2 | \mathcal{V}_1, \mathcal{H}; W) \right)$. As mentioned above, the distributions in the visible source and visible target are Gaussians, while the hidden are sigmoids.

Since there are no connections between the units in each of the layers, inference is conducted in parallel. The probability of each of the units in a given layer are

identically and independently distributed from the others in the same layer. Formally,

$$p(\mathcal{V}_1|\mathcal{V}_2,\mathcal{H}) = \prod_{i=1}^{d_1} p(v_1^{(i)}|\mathcal{V}_2,\mathcal{H}) = \prod_{i=1}^{d_1} \mathcal{N}\Big(\mu_1^{(i)}, \sigma_1^{(i)2}\Big)$$
$$p(\mathcal{H}|\mathcal{V}_1,\mathcal{V}_2) = \prod_{j=1}^{n_h} p(h^{(j)}=1|\mathcal{V}_1,\mathcal{V}_2) = \prod_{j=1}^{n_h} \text{sigmoid}\Big(s^{(j)}\Big)$$
$$p(\mathcal{V}_2|\mathcal{V}_1,\mathcal{H}) = \prod_{k=1}^{d_2} p\Big(v_2^{(k)}|\mathcal{V}_1,\mathcal{H}\Big) = \prod_{k=1}^{d_2} \mathcal{N}\Big(\mu_2^{(k)}, \sigma_2^{(k)2}\Big)$$

where, $\mu_1^{(i)}$ represents the $i^{th}$ mean for the $i^{th}$ source task visible unit, $s^{(j)}$ denotes the activation of the $j^{th}$ hidden unit, and $\mu_2^{(k)}$ is the mean of the $k^{th}$ visible target unit. These inputs depend on the configurations of the RBM. Mathematically, such inputs are determined according to:

$$\mu_1^{(i)} = \sum_{j,k} W_{ijk} h^{(j)} v_2^{(k)} + a^{(i)} = \sum_{j=1}^{n_h} \left( \sum_{k=1}^{d_2} \Big[ W_{ijk} h^{(j)} v_2^{(k)} \Big] \right) + a^{(i)}$$
$$\forall i \in \{1, 2, \ldots, d_1\}$$
$$s^{(j)} = \sum_{i,k} W_{ijk} v_1^{(i)} v_2^{(k)} + b^{(j)} = \sum_{i=1}^{d_1} \left( \sum_{k=1}^{d_2} \Big[ W_{ijk} v_1^{(i)} v_2^{(k)} \Big] \right) + b^{(j)}$$
$$\forall j \in \{1, 2, \ldots, n_h\}$$
$$\mu_2^{(k)} = \sum_{i,j} W_{ijk} v_1^{(i)} h^{(j)} + c^{(k)} = \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \Big[ W_{ijk} v_1^{(i)} h^{(j)} \Big] \right) + c^{(k)}$$
$$\forall k \in \{1, 2, \ldots, d_2\}$$

with $a^{(i)}$, $b^{(j)}$, and $c^{(k)}$ being the biases for each of the visible source, visible target, and hidden layers, respectively.

### Update Rules of the Full Model

In this section, the update rules to learn the weights and the biases of TrRBM are described. These rules are attained by deriving the energy function of Equation 2.15

with respect to the weight tensor. The following derivatives need to be computed:

$$\frac{\partial}{\partial W_{ijk}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial W_{ijk}} \Bigg( - \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2} \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 - \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \Bigg)$$

Therefore,

$$\frac{\partial}{\partial W_{ijk}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = - \frac{\partial}{\partial W_{ijk}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial W_{ijk}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\}$$
$$- \frac{\partial}{\partial W_{ijk}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 \right\} - \frac{\partial}{\partial W_{ijk}} \left\{ \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \right\}$$

After some algebraic manipulations the derivative of the previous equation yields:

$$\implies \Delta W_{ijk} \propto \left\langle v_1^{(i)}, h^{(j)}, v_2^{(k)} \right\rangle_0 - \left\langle v_1^{(i)}, h^{(j)}, v_2^{(k)} \right\rangle_\lambda$$

where $\langle \cdot \rangle_0$ is the average attained from the original data distribution and $\langle \cdot \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting from the original data distribution.

The previous derivation is next repeated for the remaining free parameters of the visible source layer (i.e., the biases) as follows:

$$\frac{\partial}{\partial a^{(i)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial a^{(i)}} \Bigg( - \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2} \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 - \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \Bigg)$$

Since the sum of the derivatives is the derivatives of the sum, the above equations

could be re-written as:

$$\frac{\partial}{\partial a^{(i)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = -\frac{\partial}{\partial a^{(i)}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial a^{(i)}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\}$$
$$- \frac{\partial}{\partial a^{(i)}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 \right\} - \frac{\partial}{\partial a^{(i)}} \left\{ \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \right\}$$

This yields the following updates for the biases of the visible source layer:

$$\implies \Delta a^{(i)} \propto \left\langle v_1^{(i)} \right\rangle_0 - \left\langle v_1^{(i)} \right\rangle_\lambda$$

where $\langle \cdot \rangle_0$ is the average attained from the original data distribution and $\langle \cdot \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution.

Next, the derivations to determine the updates of the biases for the hidden layer (i.e., $b$'s) are described:

$$\frac{\partial}{\partial b^{(j)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial b^{(j)}} \left( - \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2} \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 \right.$$
$$\left. - \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \right)$$

The equation above can then be re-written as:

$$\frac{\partial}{\partial b^{(j)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = -\frac{\partial}{\partial b^{(j)}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial b^{(j)}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\}$$
$$- \frac{\partial}{\partial b^{(j)}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 \right\} - \frac{\partial}{\partial b^{(j)}} \left\{ \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \right\}$$

These yield the following update rules:

$$\implies \Delta b^{(j)} \propto \left\langle h^{(j)} \right\rangle_0 - \left\langle h^{(j)} \right\rangle_\lambda$$

where $\langle \cdot \rangle_0$ is the average attained from the original data distribution and $\langle \cdot \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution.

The above derivation is then repeated to determine the update rules for the biases of the visible target layer to generate:

$$\frac{\partial}{\partial c^{(k)}} E(\mathcal{V}_1, \mathcal{V}_1, \mathcal{H}) = \frac{\partial}{\partial c^{(k)}} \Bigg( - \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2} \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 - \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \Bigg)$$

The equation above can then be re-written as:

$$\frac{\partial}{\partial c^{(k)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = - \frac{\partial}{\partial c^{(k)}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial c^{(k)}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\}$$
$$- \frac{\partial}{\partial c^{(k)}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_T^{(k)} - c^{(k)}}{\sigma_T^k} \right)^2 \right\} - \frac{\partial}{\partial c^{(k)}} \left\{ \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \right\}$$

Therefore, the following update rules for the visible target layer are attained:

$$\implies \Delta c^{(k)} \propto \left\langle v_2^{(k)} \right\rangle_0 - \left\langle v_2^{(k)} \right\rangle_\lambda$$

where $\langle \cdot \rangle_0$ is the average attained from the original data distribution and $\langle \cdot \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution.

Summing the results together in a set of three update equations, culminates in the following:

$$\Delta W_{ijk} \propto \left\langle v_1^{(i)}, h^{(j)}, v_2^{(k)} \right\rangle_0 - \left\langle v_1^{(i)}, h^{(j)}, v_2^{(k)} \right\rangle_\lambda$$
$$\Delta a^{(i)} \propto \left\langle v_1^{(i)} \right\rangle_0 - \left\langle v_1^{(i)} \right\rangle_\lambda$$
$$\Delta b^{(j)} \propto \left\langle h^{(j)} \right\rangle_0 - \left\langle h^{(j)} \right\rangle_\lambda$$
$$\Delta c^{(k)} \propto \left\langle v_2^{(k)} \right\rangle_0 - \left\langle v_2^{(k)} \right\rangle_\lambda$$

where $\langle . \rangle_0$ is the average attained from the original data distribution and $\langle . \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting at the original data distribution. The main factors in these update rules are similar, however the individual constituents are different. For instance, all update equations are a difference between two main terms. The first, is the original data distribution, while the second is that reconstructed from the model itself. In other words, TrRBM is trying to learn the free variables such that the error between the original data distribution and the reconstructed distribution generated from the model is minimized. Although the general form of the equations look similar, the constituents of each vary depending on which weight is being updated. For instance, to update the three-way weight tensor connecting the first and second visible layers and the hidden layer together, the source visible, the target visible, and the hidden units are used. However, to update the biases of the source visible layer, only units from this specific layer are used. Similarly, the two other update rules for the biases of the hidden and the second visible layer could be constructed.

## Factored Transfer Restricted Boltzmann Machine

TrRBM, as proposed in the previous section, is computationally expensive. Because one of the main goals of TL is to speedup learning, any TL method must be efficient. This section presents a factored version of the algorithm, FTrRBM. In particular, the three-way weight tensor is factored into sums of products through a factoring function, thus reducing the computational complexity from $\mathcal{O}(N^3)$ for TrRBM to $\mathcal{O}(N^2)$ for FTrRBM[1].

[1]The cost of this factorization is that some accuracy might be lost. However, in practice as seen in this work, as well as in [?] is not critical.

### Energy of the Factored Model

As mentioned previously, the three-way weight tensor among the different layers is now factored. Therefore, the energy function is now defined as:

$$E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = -\sum_{i=1}^{d_1}\left(\frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}}\right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2}\left(\frac{v_2^{(k)} - c^{(k)}}{\sigma_2^{k}}\right)^2 - \sum_{f=1}^{F}\left(\sum_{i=1}^{d_1}\left[w_{i,f}^{[\mathcal{V}_1]}\frac{v_1^{(i)}}{\sigma_1^{(i)}}\right]\sum_{j=1}^{n_h}\left[w_{j,f}^{[\mathcal{H}]}h^{(j)}\right]\sum_{k=1}^{d_2}\left[w_{k,f}^{[\mathcal{V}_2]}\frac{v_2^{(k)}}{\sigma_2^{(k)}}\right]\right)$$

with $F$ being the number of factors.

The main difference between this and the energy function of the full model is the factoring of the three-way weight tensor, where the weight tensor was factored into sums of products with corresponding weights (i.e., $w_{i,f}^{[\mathcal{V}_1]}$ for the visible source layer, $w_{j,f}^{[\mathcal{H}]}$ for the hidden and $w_{k,f}^{[\mathcal{V}_2]}$ for the visible target layer). Since this new model is described using "new" weights, then inference has to be conducted again. As in the full model the type of the distributions in each of the layer is the same. On the other hand, the inputs for these distributions are different.

### Inference in the Factored Model

Inference in the factored version is done in a similar manner to that of the full model with different inputs for the nodes. In particular, because there are no connections between the units in the same layer, inference is done in parallel for each of the nodes. Mathematically these are derived as:

$$p(\mathcal{V}_1|\mathcal{V}_2, \mathcal{H}) = \prod_{i=1}^{d_1} p(v_1^{(i)}|\mathcal{V}_2, \mathcal{H}) = \prod_{i=1}^{d_1}\mathcal{N}\left(\mu_1^{(i)}, \sigma_1^{(i)2}\right)$$

$$p(\mathcal{H}|\mathcal{V}_1, \mathcal{V}_2) = \prod_{j=1}^{n_h} p(h^{(j)} = 1|\mathcal{V}_1, \mathcal{V}_2) = \prod_{j=1}^{n_h}\text{sigmoid}\left(s^{(j)}\right)$$

$$p(\mathcal{V}_2|\mathcal{V}_1, \mathcal{H}) = \prod_{k=1}^{d_2} p\left(v_2^{(k)}|\mathcal{V}_1, \mathcal{H}\right) = \prod_{k=1}^{d_2}\mathcal{N}\left(\mu_2^{(k)}, \sigma_2^{(k)2}\right)$$

where, $\mu_1^{(i)}$ represents the $i^{th}$ mean for the $i^{th}$ source task visible unit, $s^{(j)}$ denotes the activation of the $j^{th}$ hidden unit, and $\mu_2^{(k)}$ is the mean of the $k^{th}$ visible target unit. These inputs depend on the configurations of the RBM. Mathematically, such inputs are determined according to:

$$\mu_1^{(i)} = \sum_{f=1}^{F} w_{i,f}^{[\mathcal{V}_1]} \left[ \sum_{j=1}^{n_h} w_{j,f}^{[\mathcal{H}]} h^{(j)} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} v_2^{(k)} \right] + a^{(i)}$$
$$\forall i \in \{1, 2, \ldots, d_1\}$$
$$s^{(j)} = \sum_{f=1}^{F} w_{j,f}^{\mathcal{H}} \left[ \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} v_1^{(i)} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} v_2^{(k)} \right] + b^{(j)}$$
$$\forall j \in \{1, 2, \ldots, n_h\}$$
$$\mu_2^{(k)} = \sum_{f=1}^{F} w_{k,f}^{[\mathcal{V}_2]} \left[ \sum_{j=1}^{n_h} w_{j,f}^{[\mathcal{H}]} h^{(j)} \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} v_1^{(i)} \right] + c^{(k)}$$
$$\forall k \in \{1, 2, \ldots, d_2\}$$

**Learning in the Factored Model**

Learning in the factored model is done using a modified version of Contrastive Divergence. As in the full model case, the derivatives of the energy function should be calculated with respect to the free parameters to determine the update equations.

First the energy equation above is rewritten in the following form:

$$E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = E_1(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) + E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H})$$

where,

$$E_1(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = -\sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2} \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2$$

and,

$$E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = -\sum_{f=1}^{F} \left( \sum_{i=1}^{d_1} \left[ w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \right] \sum_{j=1}^{n_h} \left[ w_{j,f}^{[\mathcal{H}])} h^{(j)} \right] \sum_{k=1}^{d_2} \left[ w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right)$$

**Derivatives with respect to the source:** The derivative of this energy function is computed with respect to $w_{i,f}^{[\mathcal{V}_1]}$. Formally, this is done according to:

$$\frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} E_1(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) + \frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H})$$

Unfolding each of the above terms yields the following:

$$\frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} E_1(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = -\frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_S]}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\} - \frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 \right\}$$

$$= 0$$

The derivative of the second term is:

$$\frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} \left\{ -\sum_{f=1}^{F} \left( \sum_{i=1}^{d_1} \left[ w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \right] \sum_{j=1}^{n_h} \left[ w_{j,f}^{(h)} h^{(j)} \right] \sum_{k=1}^{d_2} \left[ w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right) \right\}$$

assuming only one factor (i.e., $F = 1$)

$$\frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = -\frac{\partial}{\partial w_{i,f}^{[\mathcal{V}_1]}} \left\{ \left( \sum_{i=1}^{d_1} \left[ w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \right] \sum_{j=1}^{n_h} \left[ w_{j,f}^{(h)} h^{(j)} \right] \sum_{k=1}^{d_2} \left[ w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right) \right\}$$

This implies that the update for the weights for the visible source layer are:

$$\Delta w_{i,f}^{[\mathcal{V}_1]} \propto \left\langle \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{j=1}^{n_h} w_{j,f}^{[\mathcal{H}]} h^{(j)} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right\rangle_0 - \left\langle \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{j=1}^{n_h} w_{j,f}^{[\mathcal{H}]} h^{(j)} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right\rangle_\lambda$$

$$\forall f \in \{1, 2, \ldots, F\}$$

where $\langle . \rangle_0$ is the average attained from the original data distribution and $\langle . \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution.

**Derivative with respect to the hidden:** The same procedure is repeated to

calculate the updates for the weights of the hidden layer. Formally:

$$\frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} E_1(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) + \frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H})$$

Unfolding each of the above terms yields the following:

$$\begin{aligned}\frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} E_1(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = &- \frac{\partial}{\partial w_{j,f}^{[\mathcal{H}])}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\} \\ &- \frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 \right\} \\ &= 0\end{aligned}$$

The derivative of the second term is:

$$\begin{aligned}\frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} \Bigg\{ &- \sum_{f=1}^{F} \left( \sum_{i=1}^{d_1} \left[ w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \right] \right. \\ &\left. \sum_{j=1}^{n_h} \left[ w_{j,f}^{[\mathcal{H}])} h^{(j)} \right] \sum_{k=1}^{d_2} \left[ w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right) \Bigg\}\end{aligned}$$

assuming only one factor (i.e., $F = 1$)

$$\begin{aligned}\frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = - \frac{\partial}{\partial w_{j,f}^{[\mathcal{H}]}} \Bigg\{ &\left( \sum_{i=1}^{d_1} \left[ w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \right] \right. \\ &\left. \sum_{j=1}^{n_h} \left[ w_{j,f}^{[\mathcal{H}]} h^{(j)} \right] \sum_{k=1}^{d_2} \left[ w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right) \Bigg\}\end{aligned}$$

This yields the following updates for the factored weights of the hidden layer:

$$\begin{aligned}\Delta w_{j,f}^{[\mathcal{H}]} \propto &\left\langle h^{(j)} \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right\rangle_0 \\ &- \left\langle h^{(j)} \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right\rangle_\lambda \\ &\forall f \in \{1, 2, \ldots, F\}\end{aligned}$$

where $\langle . \rangle_0$ is the average attained from the original data distribution and $\langle . \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution.

**Derivative with respect to the target:** Here the derivatives with respect to the target factored weights (i.e., $w_{k,f}^{[\mathcal{V}_2]}$) are determined. Formally:

$$\frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_2]}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_2]}} E_1(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) + \frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_2]}} E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H})$$

Unfolding each of the above terms yields the following:

$$\begin{aligned}\frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_T]}} E_1(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = &- \frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_2]}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_2]}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\} \\ &- \frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_2]}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 \right\} \\ &= 0\end{aligned}$$

The derivative of the second term is:

$$\begin{aligned}\frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_2]}} E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_2]}} \Bigg\{ &- \sum_{f=1}^{F} \Bigg( \sum_{i=1}^{d_1} \left[ w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \right] \\ &\sum_{j=1}^{n_h} \left[ w_{j,f}^{[\mathcal{H}]} h^{(j)} \right] \sum_{k=1}^{d_2} \left[ w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \Bigg) \Bigg\}\end{aligned}$$

Assuming one factor (i.e., $F = 1$):

$$\begin{aligned}\frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_2]}} E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = - \frac{\partial}{\partial w_{k,f}^{[\mathcal{V}_1]}} \Bigg\{ \Bigg( &\sum_{i=1}^{d_1} \left[ w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \right] \\ &\sum_{j=1}^{n_h} \left[ w_{j,f}^{[\mathcal{H}]} h^{(j)} \right] \sum_{k=1}^{d_2} \left[ w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \Bigg) \Bigg\}\end{aligned}$$

This yields the following updates for the factored weights of the hidden layer:

$$\begin{aligned}\Delta w_{k,f}^{[\mathcal{V}_2]} \propto &\left\langle \frac{v_2^{(k)}}{\sigma_2^{(k)}} \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{j=1}^{n_h} w_{j,f}^{[\mathcal{H}])} h^{(j)} \right\rangle_0 \\ &- \left\langle \frac{v_2^{(k)}}{\sigma_2^{(k)}} \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{j=1}^{n_h} w_{j,f}^{[\mathcal{H}]} h^{(j)} \right\rangle_\lambda \\ &\forall f \in \{1, 2, \ldots, F\}\end{aligned}$$

where $\langle . \rangle_0$ is the average attained from the original data distribution and $\langle . \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution.

The previous derivation is next repeated for the remaining free parameters of the visible source layer (i.e., the biases). Note, that the second energy term (i.e., $E_2(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H})$ plays no role in the derivatives of the biases. The derivatives are calculated as follows:

$$\frac{\partial}{\partial a^{(i)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial a^{(i)}} \Bigg( - \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2} \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 - \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \Bigg)$$

Since the sum of the derivatives is the derivatives of the sum, the above equations could be re-written as:

$$\frac{\partial}{\partial a^{(i)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = - \frac{\partial}{\partial a^{(i)}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial a^{(i)}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\}$$
$$- \frac{\partial}{\partial a^{(i)}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^k} \right)^2 \right\} - \frac{\partial}{\partial a^{(i)}} \left\{ \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \right\}$$

This yields the following updates for the biases of the visible source layer:

$$\implies \Delta a^{(i)} \propto \left\langle v_1^{(i)} \right\rangle_0 - \left\langle v_1^{(i)} \right\rangle_\lambda$$

where $\langle \cdot \rangle_0$ is the average attained from the original data distribution and $\langle \cdot \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution.

Next, the derivations to determine the updates of the biases for the hidden layer (i.e., $b$'s) are described:

$$\frac{\partial}{\partial b^{(j)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = \frac{\partial}{\partial b^{(j)}} \Bigg( - \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2} \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^{k}} \right)^2 - \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \Bigg)$$

The equation above can then be re-written as:

$$\frac{\partial}{\partial b^{(j)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = - \frac{\partial}{\partial b^{(j)}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial b^{(j)}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\} - \frac{\partial}{\partial b^{(j)}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^{k}} \right)^2 \right\} - \frac{\partial}{\partial b^{(j)}} \left\{ \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \right\}$$

These yield the following update rules:

$$\implies \Delta b^{(j)} \propto \left\langle h^{(j)} \right\rangle_0 - \left\langle h^{(j)} \right\rangle_\lambda$$

where $\langle \cdot \rangle_0$ is the average attained from the original data distribution and $\langle \cdot \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution.

The above derivation is then repeated to determine the update rules for the biases of the visible target layer to generate:

$$\frac{\partial}{\partial c^{(k)}} E(\mathcal{V}_1, \mathcal{V}_1, \mathcal{H}) = \frac{\partial}{\partial c^{(k)}} \Bigg( - \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 - \sum_{j=1}^{n_h} h^{(j)} b^{(j)} - \sum_{k=1}^{d_2} \left( \frac{v_2^{(k)} - c^{(k)}}{\sigma_2^{k}} \right)^2 - \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \Bigg)$$

The equation above can then be re-written as:

$$\frac{\partial}{\partial c^{(k)}} E(\mathcal{V}_1, \mathcal{V}_2, \mathcal{H}) = -\frac{\partial}{\partial c^{(k)}} \left\{ \sum_{i=1}^{d_1} \left( \frac{v_1^{(i)} - a^{(i)}}{\sigma_1^{(i)}} \right)^2 \right\} - \frac{\partial}{\partial c^{(k)}} \left\{ \sum_{j=1}^{n_h} h^{(j)} b^{(j)} \right\}$$
$$- \frac{\partial}{\partial c^{(k)}} \sum_{k=1}^{d_2} \left\{ \left( \frac{v_T^{(k)} - c^{(k)}}{\sigma_T^k} \right)^2 \right\} - \frac{\partial}{\partial c^{(k)}} \left\{ \sum_{i=1}^{d_1} \left( \sum_{j=1}^{n_h} \left[ \sum_{k=1}^{d_2} \left[ W_{ijk} \frac{v_1^{(i)}}{\sigma_1^{(i)}} h^{(j)} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right] \right] \right) \right\}$$

Therefore, the following update rules for the visible target layer are attained:

$$\implies \Delta c^{(k)} \propto \left\langle v_2^{(k)} \right\rangle_0 - \left\langle v_2^{(k)} \right\rangle_\lambda$$

where $\langle \cdot \rangle_0$ is the average attained from the original data distribution and $\langle \cdot \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution.

Summarizing everything together the update rules for all the free parameters of the factored model are:

$$\Delta w_{i,f}^{[\mathcal{V}_1]} \propto \left\langle \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{j=1}^{n_h} w_{j,f}^{[\mathcal{H}]} h^{(j)} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right\rangle_0 - \left\langle \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{j=1}^{n_h} w_{j,f}^{[\mathcal{H}]} h^{(j)} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right\rangle_\lambda$$
$$\forall f \in \{1, 2, \ldots, F\}$$

$$\Delta w_{j,f}^{[\mathcal{H}]} \propto \left\langle h^{(j)} \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right\rangle_0 - \left\langle h^{(j)} \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{k=1}^{d_2} w_{k,f}^{[\mathcal{V}_2]} \frac{v_2^{(k)}}{\sigma_2^{(k)}} \right\rangle_\lambda$$
$$\forall f \in \{1, 2, \ldots, F\}$$

$$\Delta w_{k,f}^{[\mathcal{V}_2]} \propto \left\langle \frac{v_2^{(k)}}{\sigma_2^{(k)}} \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{j=1}^{n_h} w_{j,f}^{[\mathcal{H}]} h^{(j)} \right\rangle_0 - \left\langle \frac{v_2^{(k)}}{\sigma_2^{(k)}} \sum_{i=1}^{d_1} w_{i,f}^{[\mathcal{V}_1]} \frac{v_1^{(i)}}{\sigma_1^{(i)}} \sum_{j=1}^{n_h} w_{j,f}^{(h)} h^{(j)} \right\rangle_\lambda$$

$$\forall f \in \{1, 2, \ldots, F\}$$

$$\Delta a^{(i)} \propto \left\langle v_1^{(i)} \right\rangle_0 - \left\langle v_2^{(i)} \right\rangle_\lambda$$

$$\Delta b^{(j)} \propto \left\langle h^{(j)} \right\rangle_0 - \left\langle h^{(j)} \right\rangle_\lambda$$

$$\Delta c^{(k)} \propto \left\langle v_2^{(k)} \right\rangle_0 - \left\langle v_2^{(k)} \right\rangle_\lambda$$

where $\langle . \rangle_0$ is the average attained from the original data distribution and $\langle . \rangle_\lambda$ is the reconstruction determined by a Markov Chain of length $\lambda$ attained through Gibbs sampling, starting with the original data distribution. Since the main difference between FTrRBM and TrRBM is the factoring of the weight tensor, the update rules for the biases (i.e., $\Delta a^{(i)}$, $\Delta b^{(j)}$, and $\Delta c^{(k)}$) remain unchanged. The difference to TrRBM however, are in the update rules for the factored weights (i.e., $\Delta w_{i,f}^{[\mathcal{V}_1]}$, $\Delta w_{j,f}^{[\mathcal{H}]}$, and $\Delta w_{k,f}^{[\mathcal{V}_2]}$), where:

- the updates are performed for each of the factors (i.e., $f \in \{1, 2, \ldots, F\}$)
- the update rules of the weight tensor still depend on the units of the connected layers, however these are now factored into three equations corresponding to each of the three layers.

Unfortunately, learning in this model cannot be done with normal CD. The main reason is that if CD divergence was used as is, FTrRBM will learn to correlate *random* samples from the source task to *random* samples in the target. To tackle this problem, as well as ensure computational efficiency, a modified version of CD is proposed. In *Parallel Contrastive Divergence* (PCD), the data sets are first split into batches of samples. Parallel Markov chains run to a certain number of steps on each batch. At each step of the chain, the values of the derivatives are calculated and averaged to perform a learning step. This runs for a certain number of epochs. At the second iteration the same procedure is followed but with randomized samples in each of the batches. Please note that randomizing the batches is important to avoid fallacious matchings between source and target triplets.

## 6.4 Using the Inter-task Mapping

Using FTrRBMs for transfer in RL is done using two phases. First, the inter-task mapping is learned through source and target task samples. Second, samples are transferred from the source to the target, to be used as starting samples for a sample-based RL algorithm (which proceeds normally from this point onward).

### Learning Phase

When FTrRBM learns, weights and biases are tuned to ensure a low reconstruction error between the original samples and the predicted ones from the model. The RBM is initially provided with random samples from both the source and the target tasks. Triplets from the source task (i.e., $\{\langle s_1^{(i)}, u_1^{(i)}, s_1'^{(i)}\rangle\}_{i=1}^{n_1}$) and target task (i.e., $\{\langle s_2^{(j)}, u_2^{(j)}, s_2'^{(j)}\rangle\}_{j=1}^{n_2}$) are inputs to the two visible layers of the RBM. These are then used to learn good hidden and visible layer feature representations. Note that these triplets should come from random sampling—the RBM is attempting to learn an inter-task mapping that covers large ranges in both the source and target tasks' state and actions spaces. If only "good" samples were used the mapping will be relevant in only certain narrow areas of both source and target spaces.

### Transfer Phase

After learning, the FTrRBM encodes an inter-task mapping from the source to the target task. This encoding is then used to transfer (near-)optimal sample transitions from the source task, forming sample transitions in the target task. Given a near optimal source task policy, $\pi_1^\star$, the source task is sampled greedily according to $\pi_1^\star$ to acquire optimal state transitions. The triplets are passed through the visible source layer of FTrRBM and are used to reconstruct initial target task samples at the visible target layer, effectively transferring samples from one task to another. If the source and target task are close enough[2], then the transferred transitions are expected to aid the target agent in learning an (near-)optimal behavior. They are then used in a sample based RL algorithm, such as LSPI to learn an optimal behavior in the target task (i.e., $\pi_2^\star$).

The overall process of the two phases is summarized in Algorithm 23.

The work in this chapter share similarities to Chapter 5, but vary significantly in the procedure the intertask mapping is learned. Both share the same inputs and outputs. Precisely, both frameworks require source and target task random samples, as well as a near-optimal behavior in the source to produce an intertask mapping.

[2]Note that defining a similarity metric between tasks is beyond the scope of this chapter. Please refer to the next chapters for details on such a measure.

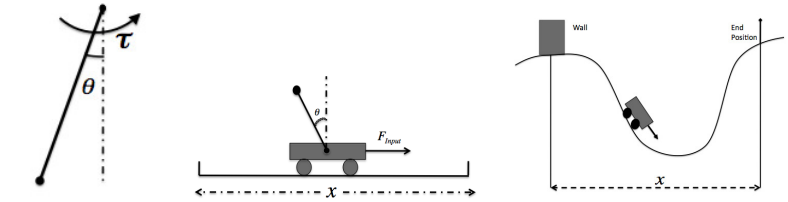
**Algorithm 23** Overall Transfer Framework

1: **Input:** Random source task samples $\mathcal{D}_1 = \{\langle s_1^{(i)}, u_1^{(i)}, s_1^{\prime(i)} \rangle\}_{i=1}^{n_1}$, random target task samples $\mathcal{D}_2 = \{\langle s_2^{(j)}, u_2^{(j)}, s_2^{\prime(j)} \rangle\}_{j=1}^{n_2}$, optimal source task policy $\pi_1^\star$
2: Use $\mathcal{D}_1$ and $\mathcal{D}_2$ to learn the intertask mapping using FTrRBM.
3: Sample source task according to $\pi_1^\star$ to attain $\mathcal{D}_1^\star$.
4: Use the learned RBM to transfer $D_1^\star$ and thus attain $\mathcal{D}_2^0$.
5: Use $\mathcal{D}_2^0$ to learn using a sample-based RL algorithm.
6: **Return:** Optimal target task policy $\pi_2^\star$.

The main intuition of discovering a common space describing both tasks is also shared. However, the procedure of learning such a space is different. In Chapter 5 the idea was to use sparse coding and sparse projections to discover the shared space, opposed to using a unified three layer restricted Boltzmann machine in this chapter. After the space has been achieved, Chapter 5 adopted non-parametric regression (i.e., Sparse Gaussian Processes) to learn the intertask mapping. In TrRBM/FTrRBM such an additional (possibly computationally expensive) step is not required. The learnt hidden space can be used to reconstruct target task transitions from those of the source task $\Big($i.e. by determining $p\left(\mathcal{V}_2 | \mathcal{V}_1, \mathcal{H}\right)\Big)$.

## 6.5 Experiments and Results

To asses the efficiency of the proposed framework, experiments on different RL benchmarks were performed. Three different transfer experiments were conducted using the tasks shown in Figure 6.3[3].




(a) Inverted Pendulum (b) Cart Pole (c) Mountain Car

Figure 6.3: Experimental domains

[3]The samples required for learning the inter-task mapping were not given as extra samples for the random learner in the target. Please note, that even if these were included the results as seen from the graphs will still be in favor of the proposed methods.

## Cart-Pole to Cart-Pole Transfer

The goal of this first experiment is to determine if the proposed method is capable of learning an inter-task mapping between pairs of similar tasks. Thus, transfer was done between two cart-poles (CP) tasks. The state of a CP system are described via the four dimensional state vector, $\langle \theta, \dot{\theta}, x, \dot{x} \rangle$. The first two state variables represent the angle and angular velocity of the pole, respectively, while the last two represent the position and the velocity of the cart. The agent is allowed to choose from a set of discretized actions. The goal of the agent is to stabilize the pole in an upright position. Next the details of the source task and target task are described.

**Source Task** The source task was selected to be easier to learn than the target task. The source task has a long pole, $l = 3$ meters. The action space was a discrete set of two actions $\mathcal{U}_1 = \{-10, 10\}$ in Newtons. The reward was $+1$ if the angle of the pole is in the $-\frac{\pi}{9} < \theta < \frac{\pi}{9}$ range and the position is in $-4 < x < 4$ meters range. The agent is given a negative reward of $-1$ otherwise. Here the source agent learned a near-optimal behavior to balance the pole in an upright position using LSPI.

**Target Task** The target task was also a CP but with a shorter pole length, $l = 0.5$, making the control problem harder. To determine the performance of the proposed framework with varying MDPs, the target action space, transition probability and reward functions were different from the source task. Because the length of the pole was different from the source task, the transition dynamics change. Additionally, the action space of the target agent was changed to $\mathcal{U}_2 = \{-10, 0, 10\}$ and the reward function was changed to $cos(\theta)$, giving the agent a maximum value of $+1$ when the pole is the upright position.

**Experiment** First, 5000 random $\langle s, a, s' \rangle$ samples were collected from the source task and 1000 and 500 were collected from the target task. These were used by FTrRBM, in two separate experiments, to learn the inter-task mapping, where the RBM contained 120 hidden units and 30 factors. Samples were split into batches of 100 each to train FTrRBM using PCD. The whole training process ran for 100 epochs and FTrRBM converged in about 4.5 minutes to the lowest reconstruction error.[4]

Second, Sarsa [99] was used to learn $\pi_1^\star$, and then sample different numbers of transitions from the source task. Each sample set — consisting of 1000, 2000, ..., 10000 triplets — was input to the FTrRBM to construct a set of initial samples in the target task, called transferred samples.

[4]The experiments ran on a 2.3 GHz Intel Core i5 MacBook Pro laptop.

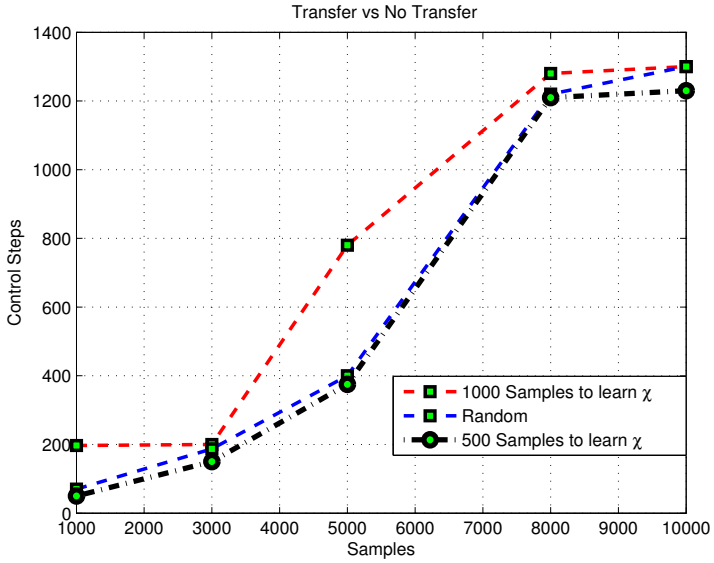



Figure 6.4: Transfer versus no transfer comparison on highly similar tasks

Third, LSPI was used on each of the sample sets to learn a policy, $\pi_2^{(0)}$. The performance of the policy was then determined by executing actions determined by $\pi_2^{(0)}$ for 500 episodes. The behavior of this policy — for each of the sample set sizes — was compared to normal LSPI using random sampling. The results are reported in Figure 6.4. The x-axis shows the sample-set size (i.e., the number of source task samples used) and the y-axis shows the performance of the controller by measuring the number of steps the pole was in the range $-\frac{\pi}{9} < \theta < \frac{\pi}{9}$. Except for the final data point, the performance of $\pi_2^{(0)}$ is higher than that of the policy generated by LSPI with the same number of target task instances that were sampled randomly (instead of being transferred from the source task) when using 1000 samples to learn the intertask mapping. When using 500 samples to learn the intertask mapping the performance is similar to the random case in the source as shown with the "black line" in the figure. Note that the standard errors of the performance are too small to be visible on the graph.

The performance of TrRBM was also tested with respect to learning times in two ways. First, the number of iterations needed for LSPI to converge on a fixed set of samples was measured. LSPI consistently converged with fewer iterations when using the transferred samples. For example, LSPI converged with only 8 iterations on 5000 transferred samples compared to 12 on the random ones and with 17 compared to 21

iterations on 10000 transferred and random samples, respectively. The second timing experiment measured the wall-clock time needed for LSPI to reach (near-)optimal behavior in the target task. For this, LSPI started with initial policy learned on the transferred samples, or on random samples. This policy was then used to sample more from the target environment. The procedure was repeated until an optimal behavior, defined as the behavior of attaining more than 3000 control steps of the pole, was obtained. Starting from the transferred policy with 3000 samples reduced the learning time from 22 minutes to 15 minutes. Therefore,
**ConclusionI:** FTrRBM is capable of learning a relevant inter-task mapping between a pair of similar tasks.
**ConclusionII:** FTrRBM is capable of transferring informative samples that: (1) Produce better performance, (2) Reduce the computational complexity to attain a fixed policy on these samples, and (3) Reduce the computational complexity to reach a (near)-optimal policy.

## Inverted Pendulum to Cart-Pole Transfer

A similar experiment was performed to test the transfer capabilities of FTrRBMs between less similar tasks. This time, while the target task was kept identical the source task was the inverted pendulum of Figure 5.2(a).

**Source Task** The state variables of the pendulum are $\langle \theta, \dot{\theta} \rangle$. The action space is a set of two torques $\{-10, 10\}$ in Newtons. The goal of the agent is again to balance the pole in an upright position with $\langle \theta = 0, \dot{\theta} = 0 \rangle$. A reward of $+1$ is given to the agent when the pole's angle is in $-\frac{\pi}{12} < \theta < \frac{\pi}{12}$ and $-1$ otherwise.

**Experiment** 3000 random source task samples, 1000 and then 500 target task samples were used to learn the inter-task mapping. The RBM contained 80 hidden units and 25 factors. Learning was performed as before, with FTrRBM converging in about 3.5 minutes. Transfer was accomplished and tested similarly to the previous experiment. The results are reported in Figure 6.5. It is again clear that transfer helps the target agent in his learning task when using 1000 samples to learn the intertask mapping. In case, 500 samples were used to learn the intertask mapping the target agent starts better than the random agent in the target task, however this performance decreases with the increase in the number of transferred samples. LSPI again converged with fewer iterations when using transfer. LSPI convergence time also decreased on different transferred samples. For example, LSPI converged with only 9 iterations on 5000 transferred samples compared to 12 using random ones and with 17 compared to 19 on 8000 transferred and random samples, respectively. The time needed to reach near optimal behavior was reduced from 22 to 17 minutes by using a

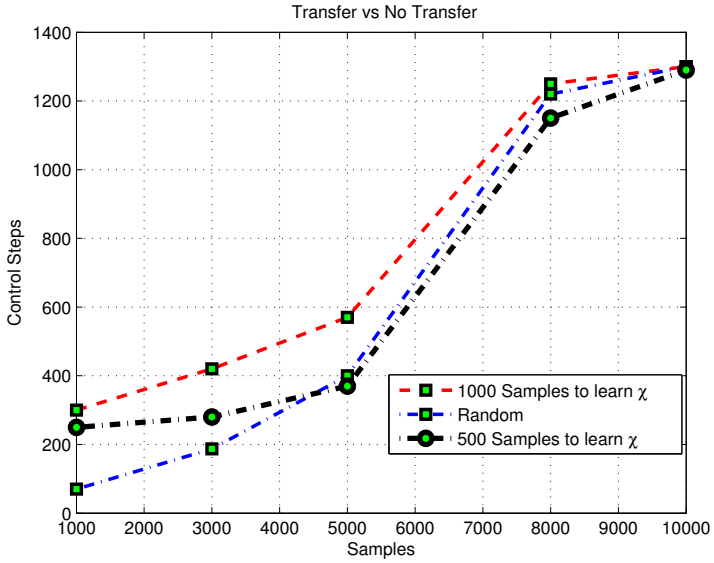



Figure 6.5: Transfer versus no transfer comparison on different tasks.

transferred policy to initialize LSPI. Therefore,
**ConclusionIII:** FTrRBM is capable of learning a relevant inter-task mapping between a pair of dissimilar tasks.

## Mountain Car to Cart-Pole Transfer

A third experiment shows the transfer performance of FTrRBMs between pairs even less similar tasks than in the previous section. The target task remained the same cart-pole as before, while the source task was chosen to be the Mountain-Car (MC) problem. Although very different, successful transfer results between these tasks had previously been shown in the previous chapter.

**Source Task** The system is described with two state variable $\langle x, \dot{x} \rangle$. The agent can choose between two actions $\{-1, 1\}$. The goal of the agent is to drive the car up the hill to the end position. The car's motor is not sufficient to drive the car directly to its goal state — the car has to oscillate in order to acquire enough momentum to drive to the goal state. The reward of the agent is $-1$ for each step the car did not reach the end position. If the car reaches the goal state, the agent receives a positive reward of $+1$ and the episode terminates. Learning in the source task was performed using SARSA.

4000 random source task samples and 1000 and then 500 target task samples were used to lean the inter-task mapping as before. The RBM contained 120 hidden units and 33 factors and converged in about 3.5 minutes to the lowest reconstruction error.

The results of transfer (performed as before) are reported in Figure 7.10. It is clear that transfer helps even when tasks are highly dissimilar and when using 1000 samples to learn the intertask mapping. As before, LSPI converged with fewer itera-

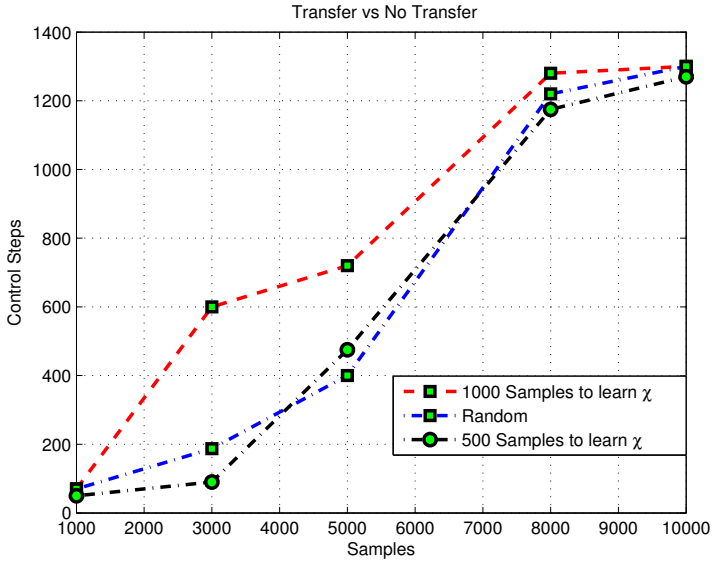



Figure 6.6: Transfer versus no transfer comparison on highly dissimilar tasks.

tions when using transfer than without using transfer. For example, LSPI converged with only 10 iterations on 5000 transferred samples compared to 12 using random ones. LSPI converged to an optimal behavior in about 18 minutes compared to 22 minutes for the non-transfer case. Therefore,

**ConclusionIV:** FTrRBM is capable of learning a relevant inter-task mapping between a pair of highly dissimilar tasks.

## Comparisons to Sparse Coded Inter-task Mapping

To provide a comprehensive comparison between FTrRBM and the work of Chapter 5, two additional experiments were conducted. The source task was either the IP or the MC, while the target task was the CP system. 1000 and 500 target samples were used to learn an intertask mapping using either FTrRBM or the work of

Chapter 5. Having these intertask mappings, (near-) optimal source task samples[5] were then transferred to provide an initial batch for the target RL agent to learn on. Performance, measured by the number of successful control steps in the target, was then reported in Figures 6.7 and 6.8.

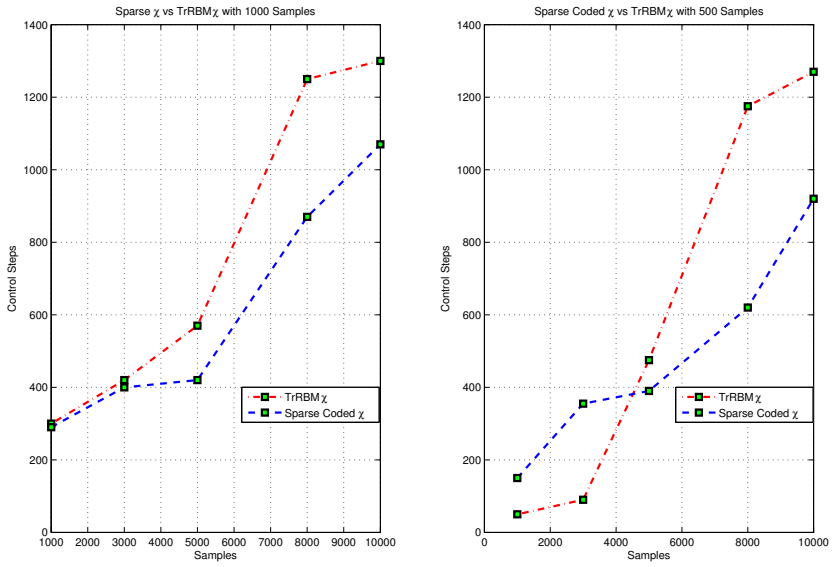



Figure 6.7: Performance comparison of transferring from IP to CP using FTrRBM or Sparse Coded inter-task mappings (i.e., Chapter 5). The left graph shows the results of transfer when using 1000 target samples to learn the intertask mapping, while the right presents the results when using 500 samples to learn such a mapping.

Figure 6.7 shows two comparison graphs. The left graph reports the performance when using 1000 target samples to learn the intertask mapping. These clearly demonstrate that FTrRBM performs better than sparse coded intertask mappings, where for example, FTrRBM attains about 570 control steps compared to 400 in the sparse coded case at 5000 transferred samples. As the number of control steps increases, the performance of both methods also increases, to reach around 1300 control steps for FTrRBM compared to 1080 in the sparse coded case at 10000 transferred samples. The right graph shows the results of the same experiments, however, when using only 500 target samples to learn the intertask mapping. Again these results show that apart from the first two points, FTrRBM outperforms the Sparse coded intertask

[5]The optimal policy in the source was again attained using SARSA.

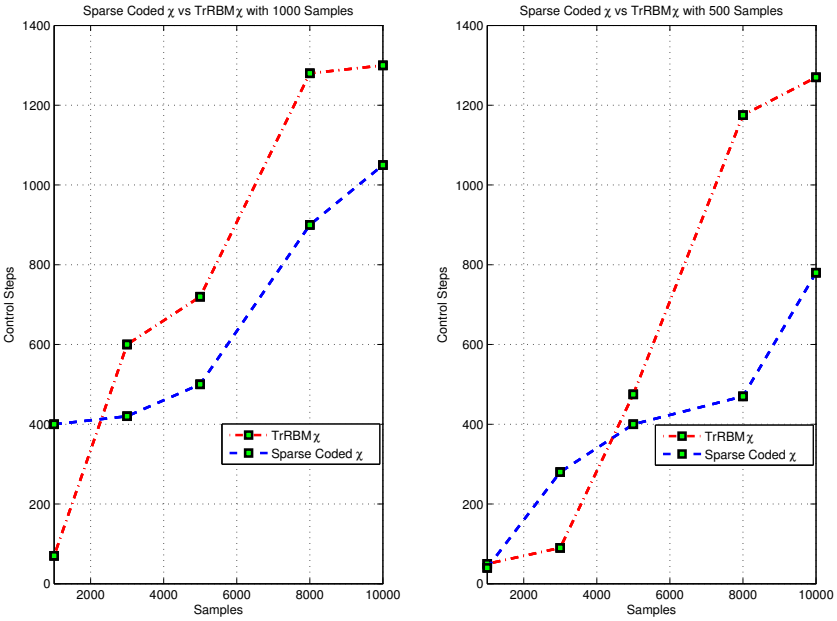



Figure 6.8: Performance comparison of transferring from MC to CP using FTrRBM or Sparse Coded inter-task mappings (i.e., Chapter 5). The left graph shows the results of transfer when using 1000 target samples to learn the intertask mapping, while the right presents the results when using 500 samples to learn such a mapping.

mapping.

In Figure 6.8 the results of the same experiments on highly dissimilar tasks are shown. In the left graph, 1000 target samples were used to learn an intertask mapping using either FTrRBM or the approach of Chapter 5. The results clearly manifest the superiority of FTrRBM compared to the sparse coded approach, where at 5000 transferred samples FTrRBM attains 600 control steps, with 410 steps for the sparse coded intertask mapping. This performance increases to reach about 1300 control steps for FTrRBM with 1050 for the sparse coded approach on 10000 transferred samples. In the right graph the same experiments were repeat using 500 samples to learn the intertask mapping. It is again clear that FTrRBM outperforms the approach of Chapter 5.

From these results the following could be concluded:

**Conclusion V:** The common space learned using FTrRBM is more informative com-

pared to that learned using the approach of Chapter 5, and
**Conclusion VI:** FTrRBM is more sample efficient compared compared to the sparse coded approach as it requires less target samples to learn a successful intertask mapping.

## 6.6 Discussions and Conclusions

This chapter proposes a theoretically grounded method for learning an inter-task mapping, based on RBMs. The merits of the approach were validated through experimental evidence. The proposed technique successfully learned a useful inter-task mapping between pairs of similar, less similar, and highly dissimilar pairs of tasks.

The idea of discovering a common space that is capable of representing different MDPs has been shown effective. It was clear from this and the previous chapter that discovering such space will help target agents in improving their learning behaviors. Interestingly, when following automated schemes surprising results can emerge. For instance, prior to these works, transfer from the MC to the CP, was believed to be impossible as it is hard for a human to hand-code such a relation between these two tasks. Fortunately, if a more abstract and informative space to encode an inter-task mapping was adopted, transfer between highly dissimilar tasks becomes possible. Although successful, the relations between the tasks are still vague and unclear. If an autonomous transfer framework is to be developed, a more comprehensible study of these relations is essential.

So far, different approaches to automatically learn valid mappings between *preset* source and target tasks were discussed. However, for full autonomy, a learner has to be able to autonomously decide on what tasks to use. More specifically, given a specific target task and a bag of different source tasks[6], the learner has to decide on which source to choose from. Therefore, if a similarity measure between the different MDPs was available, transfer learners can make use of such a measure to effectively choose valid source task(s) and thus approach full autonomy.

Aiming at a better understanding of the attained results, as well as at fully autonomous transfer learners, the next chapter proposes a similarity measure between MDPs with different constituents. This measure makes use of deep learning techniques to discover highly informative spaces, that are capable of describing a broad range of tasks. It is of major importance to the transfer and reinforcement learning community, as it will allow practitioners to automatically choose from and potentially generate relevant source task(s) for a given target task.

[6]Please note, that automatically generating relevant source tasks is also possible but is left for future work.

# Part II

# Choosing Relevant Source Tasks

# 7

# Connecting the Dots

**This chapter is based on:** *H. B. Ammar, D. C. Mocanu, M. Taylor, K. Driessens, G. Weiss, and K. Tuyls, "Data-Driven Similarity Measure Between Markov Decision Processes," Journal of Machine Learning Research (JMLR)-in review.*

In the first part of this dissertation, different approaches to learn intertask mappings between reinforcement learning tasks have been proposed. Namely, three different techniques, of which two that are fully automated (Chapters 5, and 6), have been detailed. The first method, discussed in Chapter 5, adopted sparse coding, sparse projection, and sparse gaussian processes to learn such a mapping. The second, described in Chapter 6, made use of high order restricted Boltzmann machine to achieve that goal. Having such mappings, two novel and effective transfer for reinforcement learning algorithms were described (i.e., TrLSPI, and TrFQI). It is worth noting, that the proposed techniques for learning the intertask mapping are not restricted to LSPI and FQI. Essentially, any sample based reinforcement learning algorithm can be used in the target. However, the main advantage of LSPI and FQI is their learning efficiency, justifying their usage as basis RL algorithms for transfer.

At a high level, the shared characteristics between these two methods was the idea to discover a high dimensional common space that is capable of characterizing both tasks. Such a space that allowed for a unified description, enabled transfer. Therefore, these techniques solved the challenges of: (1) automatically relating source and target MDPs, and (2) effectively exploiting such a relation in order to perform transfer.

However, for autonomous transfer, one more challenge needs to be solved. To have a fully autonomous transfer agent, it still has to decide which source task(s) to be chosen for a specific target. From the author's opinion, this is the toughest question to be answered as it requires the definition of similarity measures between tasks, that might potentially have different state and action spaces. To the best of our knowledge, there has been little progress toward this goal thus far.

Few attempts aiming at such a similarity measure between MDPs. Most relevant are works that use bisimulation metrics [20, 27, 28]. Although interesting, these works suffer from restrictive assumptions on the state and/or action spaces, transition probabilities, and/or reward functions of the MDPs. In the work in this chapter the proposed measure are: (i) data-driven (the metric is acquired from MDP transitions), (ii) operational in continuous state space, (iii) computationally tractable[1], and (iv) compatible with MDPs having different action spaces or state features.

Having such a measure is of great importance. First, it will quantify the ad hoc choices made by the designer when selecting source and target tasks for the evaluation of a new TL algorithm. Second, it is the first step in attaining a well-founded performance criterion for TL in RL tasks. For instance, bounds on transfer algorithms can now be attained as a function of this measure that quantifies similarities between tasks. Furthermore, it will serve as a solution for an instant of the last remaining challenge in creating autonomous transfer agents.

In addition to TL, such a measure is of importance to the general RL community. There have been numerous RL algorithms proposed, typically evaluated on a handful of benchmarks, using a set of parameters tuned on each MDP. Unfortunately, this can lead to the problem of empirical over-fitting [25], i.e., an algorithm can work well given specific MDP parameters values, but not when using other parameter values or different MDPs. With the help of an MDP similarity measure, RL algorithms can be evaluated over a well-defined set of MDPs.

Throughout the previous chapters in this thesis successful results have shown on the transfer between three RL benchmarks. Namely, successful transfer between the inverted pendulum to the cart-pole, and mountain car to the cart pole have been presented. Although transfer between the inverted pendulum to the cart-pole is interesting, the results are less surprising compared to the transfer from the mountain car to the cart pole. The reason behind successful transfer results is intuitively based on the idea that tasks appearing far on a low level of abstraction are actually similar at high levels. This claim, where tasks being far at low level of abstractions are actually related and similar once described in higher abstraction levels, can be disproved or validated with the help of such a similarity measure. In the experiments performed in this chapter it becomes clear that indeed such a claim is true.

This chapter presents a method to calculate the similarity between MDPs with the same state features and action spaces by measuring reconstruction error when using a restricted Boltzmann machines. A generalized method is then derived to work when MDPs with different state and/or action spaces are presented by using the additional abstraction of a deep belief network. Experiments show that the proposed measures are capable of capturing and clustering dynamical similarities between MPDs with

[1]The complexity of the algorithm is linear in the number of samples.

multiple differences, including (i) state features, (ii) action space, (iii) transition probability, and/or (iv) reward function. Experiments also show that the jumpstart gained from transferring between different source and target tasks is correlated with both proposed measures.

## 7.1 Problem Definition

In this section the problem definition is detailed. Each of the source and target tasks are MDPs. These vary in each of their constituents. More specifically, the approach makes no restrictive assumptions on the type of variations between the tasks. In other words, not only the probability distributions or reward functions may vary between the tasks, but these variations can also happen between the state and/or action spaces.

Namely, given two MDPs $\mathcal{M}_1 = \langle \mathcal{S}_1, \mathcal{U}_1, \mathcal{T}_1, \mathcal{R}_1, \gamma_1 \rangle$ and $\mathcal{M}_2 = \langle \mathcal{S}_2, \mathcal{U}_2, \mathcal{T}_2, \mathcal{R}_2, \gamma_2 \rangle$, where $\mathcal{S}_i$, $\mathcal{U}_i$, $\mathcal{T}_i$, $\mathcal{R}_i$, and $\gamma_i$ correspond to the state space, actions space, transition model, reward function, and discount factor for all $i \in \{1, 2\}$ representing the source and target tasks' MDP constituents, respectively, the question is to define a similarity measure between $\mathcal{M}_1$ and $\mathcal{M}_2$. Knowledge in form of transitions from both MDPs are available. Two data sets $\mathcal{D}_1 = \langle s_1^{(i)}, u_1^{(i)}, s_1'^{(i)} \rangle_{i=1}^{n_1}$, with $n_1$ being the number of samples in the source, and $\mathcal{D}_2 = \langle s_2^{(j)}, u_2^{(j)}, s_2'^{(j)} \rangle_{j=1}^{n_2}$, with $n_2$ being the number of samples in the target. These are then used learn about the similarities between the tasks.

More specifically, the solution to this problem is spilt into two phases. In the first, restrictive assumptions on the state and actions spaces of the source and target MDPs are imposed. In the first proposed measure the state and action spaces of the MDPs are assumed to be of the same dimensionality. Although successful as shown in the experiments, this measure is restrictive. It only operates for MDPs that have the same domain. In the most interesting cases of transfer learning, the source and target MDPs belong to different domains. Therefore, if the third step for automated transfer is to be solved then a generalization of the previous measure is required. Aiming at a more general framework, this previous measure is generalized by changing the configuration of the proposed RBM. The new configuration is based on deep belief networks. The idea is to generate a space that is capable of describing transitions from different MDPs.

## 7.2 Overall Framework

This section presents a high level description for both proposed measures. Firstly, the measure between MDPs with the same state and actions spaces (i.e., domains) is

discussed. Secondly, the generalization of the measure to operate in different domains setting is described.

## Similarity Measure: Same Domains Case

The first proposed measure, RBDist, operates within the setting were the MDPs have the same state and action spaces. Given a set of transition samples, $\mathcal{D}_1 = \langle s_1^{(i)}, u_1^{(i)}, s_1'^{(i)} \rangle_{i=1}^{n_1}$ and $\mathcal{D}_2 = \langle s_2^{(j)}, u_2^{(j)}, s_2'^{(j)} \rangle_{j=1}^{n_2}$, from both the source and target MDPs, respectively, the question is to determine a measure that is capable of describing similarities between these transitions. Here again the idea of a common space is of major importance. If there existed a highly "informative" high dimensional space, that is capable of describing, for instance, the source MDP, it can also be used to determine similarities between MDPs. More specifically, if two MDPs are similar, then an informative rich space describing the first, should also be capable to some extend of describing the second.

The main question then is how to define, discover, or learn such a common space. In this thesis, two approaches for determining such spaces were investigated. The first was based on sparse coding, while the second adopted high-order restricted Boltzmann machines. At the beginning attempts using sparse coding to determine such a measure were followed. Although appealing, applying sparse coding can indeed discover such a space, but it seemed to be hard to use this space to attain the sought measure. The main problem related to the nature of sparsity induced in the generated data. To clarify, the such a measure can be seen as the measure between two probability distributions in a high dimensional common space representing the transitions of both MDPs. To discover such a space using sparse coding, one has to first sparse code the source transitions, project the target transitions to the discovered space, and finally perform density estimation. The last step is executed to represent the transitional distributions of both MDPs in the discovered space. It is this step that is a hard, since the data is high dimensional and sparse. Normal density estimation techniques such as Gaussian mixture models are not applicable at these dimensions. Therefore, it seemed that such direction of analysis is not optimal.

The second technique that is capable of determining such spaces that was already proposed in Chapter 6 relies on high order restricted Boltzmann machines. The advantages of these techniques is that they are natural density estimators and therefore, are ideal for the targeted problem. RBDist, shown in Figure 7.1, makes used of a restricted Boltzmann machine (RBM) to determine the similarity between MDPs with the same state and action spaces. Given $\mathcal{D}_1$ and $\mathcal{D}_2$, the idea could be split in two steps. First, $\mathcal{D}_1$, is used to train an RBM to determine a hidden and weight configuration capable of describing source transitions (i.e., $\langle s_1, u_1, s_1' \rangle$) in a

high dimensional feature space. If the two MDPs are similar then this space should also be able to construct target transitions. Therefore, the second step is to try to reconstruct transitions in $\mathcal{D}_2$ based on the RBM learned in the first step. The error incurred while reconstructing these samples is proportional to the difference between the two distributions describing both MDPs, and thus can be used as a similarity measure.

### Similarity Measure: Different Domains Case

Although successful, the previous measure suffers from restrictive assumptions as it operates within same domain MDPs. Aiming at extending it to more realistic settings, DRBDist, shown in Figure 7.2, is proposed. The idea is similar to RBDist, where it is again based on the reconstruction error.

However, to allow for the flexibility of having different state and action spaces between the MDPs, the RBM configuration of Figure 7.1 has to be extended. As the two MDPs might require different RBM configurations to be accurately abstracted, the first step is to use $\mathcal{D}_1$ and $\mathcal{D}_2$ to train two separate RBMs. It is worth noting, that these may have different visible and hidden neural configurations (i.e., the first two layers in Figure 7.2). At this level the two MDPs are described in a high dimensional rich feature space. Next, these spaces need to be unified to allow for a common high dimensional space of both MDPs. This space can then be used for sample reconstruction. To have the reconstruction layer, two steps are needed. In the first, the attained hidden configuration of the two RBMs is mapped to a new hidden layer under the constraint that this layer has the same number of units for each of the MDPs. In the second, this newly learned configuration from the second MDP is then projected towards the first. This step allows the description both MDPs in the same space. At this level, any sample from either the source or the target can be reconstructed. The last hidden layer is then added to allow for such a reconstruction. The main intuition behind the idea is that if these MDPs are similar, then these discovered spaces will be able to describe samples from both MDPs. The reconstruction error, which is then proportional to the probability distribution describing both MDPs, is the basis for the similarity measure.

The remainder of the chapter is organized as follows. Section 7.3 describes both proposed measures. Namely, first RBDist, the measure between MDPs with shared state and action spaces is detailed, second, the extension to DRBDist is presented. Section 7.4 presents the experiments showing both measures to be successful and meaningful. Finally, Section 7.5 concludes with a discussion and interesting directions for future work.

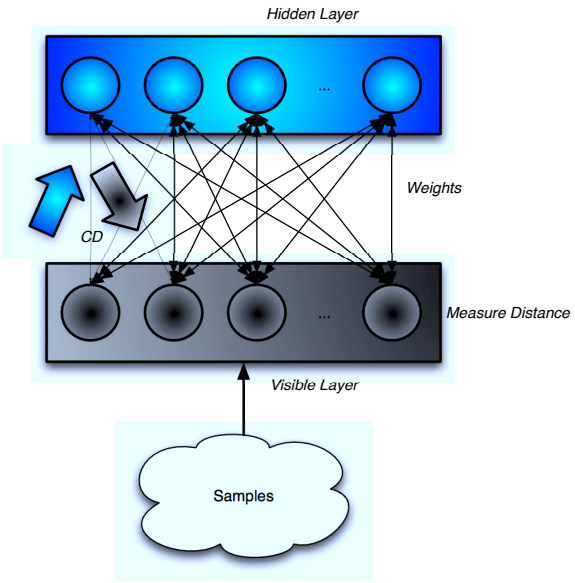



Figure 7.1: This is a high level schematic of the similarity measure between MDPs with shared state-action spaces. Both training and reconstruction use CD.

## 7.3 MDP Similarity Measures

This section details the proposed similarity measures. First, a measure between MDPs with the identical state-action spaces is described. After, it is extended so that the similarity between MDPs with differences in any or all of their five constituent parts can be measured.

### Shared State and Action Spaces: RBDist

The similarity measure is defined by its computation Algorithm 24. The two MDPs are uniform randomly sampled to generate $\mathcal{D}_1 = \{\langle s_1^{(i)}, u_1^{(i)}, s_1'^{(i)}\rangle\}_{j=1}^{n_1}$ and $\mathcal{D}_2 = \{\langle s_2^{(j)}, u_2^{(j)}, s_2'^{(j)}\rangle\}_{j=1}^{n_2}$, where $s_1'^{(i)} \sim \tilde{\mathcal{T}}_1(s_1^{(i)}, u_1^{(i)})$, $s_2'^{(j)} \sim \tilde{\mathcal{T}}_2(s_2^{(j)}, u_2^{(j)})$, $n_1$ and $n_2$ represent the number of samples from the first and second tasks, respectively. The source task data set is used to train an RBM (line 2) to describe the transitions in a richer feature space. The idea behind this is that if the rich feature space is informative enough, the learned RBM will not only be capable of reconstructing samples from the source MDP, but also from similar MDPs. The third step is to reconstruct samples from the other MDPs using this learned RBM as shown in lines 5 and 6 of the algorithm. The reconstruction error of a sample is defined as the Euclidean distance

**Algorithm 24** RBDist: Shared State and Action Spaces

1: **Input:** $\mathcal{M}_1$ samples: $\mathcal{D}_1 = \{\langle s_1^{(i)}, u_1^{(1)}, s_1'^{(i)}\rangle\}_{i=1}^{n_1}$,
$\mathcal{M}_2$ samples: $\mathcal{D}_2 = \{\langle s_2^{(j)}, u_2^{(j)}, s_2'^{(j)}\rangle\}_{j=1}^{n_2}$
2: Use $\mathcal{D}_1$ to train an RBM yielding (**v**,**h**,**W**).
3: **for** k=1 to n **do**
4: Reconstruct each sample from $D_2$ in a single forward-backward Gibbs step using:

$$p(\mathbf{v}|\mathbf{h},\mathbf{W}) = \prod_{i=1}^{n_v} \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma})$$

$$\text{with } \boldsymbol{\mu}_i = \sum_{f=1}^{n_h} w_{i,f} h_f + b_i$$

5: Compute the reconstruction error
$e_j = L_2(\langle s_2^{(j)}, u_2^{(j)}, s_2'^{(j)}\rangle_0, \langle s_2^{(j)}, u_2^{(j)}, s_2'^{(j)}\rangle_1)$
6: **end for**
7: **Return:** the mean of all errors $\mathbf{E} = \frac{1}{n}\sum_{j=1}^{n} e_j$ as the measure between the MDPs.

between the original sample and its reconstruction after a single forward-backward Gibbs step. The difference measure between the two MDPs, referenced from now on as RBDist, is defined as the average reconstruction error of all $\mathcal{M}_2$ samples.

### Different State and/or Action Spaces: DRBDist

RBDist is only applicable for MDPs with an identically represented state and action space. To construct at a measure for MDPs with different state features and/or action spaces, essential for cross domain transfer, RBDist is extended. Different state-action spaces require different visible and hidden configurations to accurately model the MDPs. A new scheme, shown in Figure 7.2, is introduced that consists of one visible and three hidden layers. Each of these layers is necessary. The bottom layer includes a set of visible nodes for both MDPs. Each MDP can use a different number of visible units matching the dimensions of each task. This visible layer is connected to a first hidden layer that still allows a different number of nodes for each MPD. The main goal of this first hidden layer is to find a description of the visible layer in a more representative feature space. The second hidden layer constraints each MDP to use the same number of units. This allows the reconstruction error — proportional to the KL-measure between the different probability distributions in that layer — to be calculated at this level. To measure this error two steps are required: (1) hidden layer three, and (2) a description of the probability distribution of the target MDP in the source MDP space (illustrated as the "left" arrow in Figure 7.2). Once determined,

the reconstruction error can be used as the basis for the similarity measure.

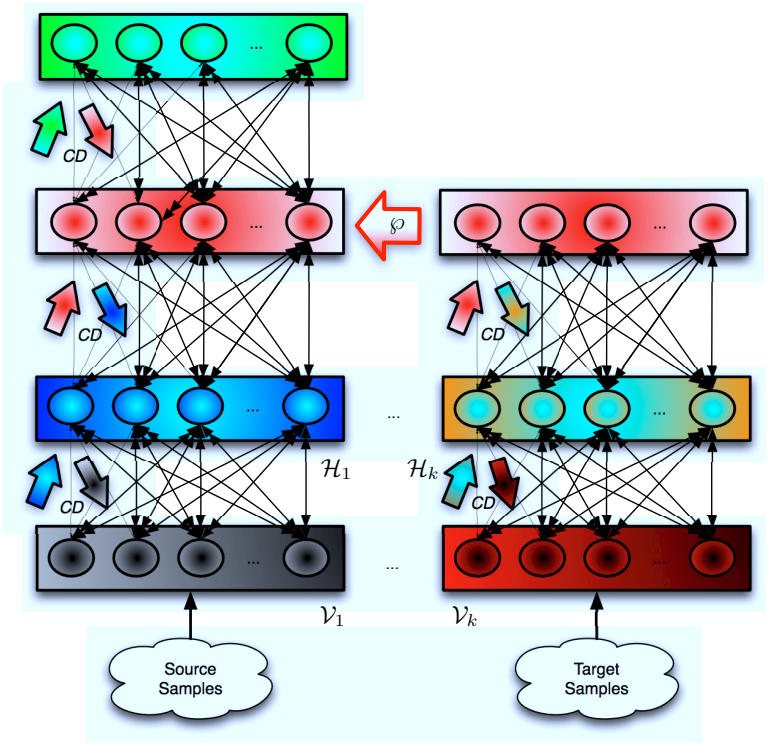



Figure 7.2: This figure diagrams DRBDist. The bottom (visible) layer receives task data. The first hidden layer constructs a more representative feature space of an MDP. Both second hidden layers have the same structure and number of nodes, allowing the target to be matched to the source. The reconstruction error in the third hidden layer measures task similarity.

We now describe the mathematical formulation of inference as well as the update rules for the DBN.

**First Two Layers:** For $l \in \{1, 2\}$, where $l$ indicates the source or target MDP, define $\mathcal{V}_l = [v_l^{(1)}, \ldots, v_l^{(n_l)}]$ for $l \in \{1, 2\}$ with $n_l$ the number of nodes in the visible layer for MDP $l$ and $\mathcal{H}_l = [h_l^{(1)}, \ldots, h_l^{(n_{h_l})}]$ for $L \in \{1, 2\}$ with $n_{h_l}$ the number of hidden nodes for MDP $l$. The energy function for each RBM is given by:

$$E_l(\mathcal{V}_l, \mathcal{H}_l) = -(\mathcal{V}_l - \mathbf{a}_l)^T \mathbf{\Sigma}_l (\mathcal{V}_l - \mathbf{a}_l) - \mathbf{b}_l^T \mathcal{H}_l - \mathcal{V}_l^T \mathbf{W}_l \mathcal{H}_l$$

where $\mathbf{a}_l$ and $\mathbf{b}_l$ are the bias vectors of the corresponding visible and hidden layers respectively, $\mathbf{W}_l \in \mathbb{R}^{n_l \times n_{h_l}}$ is the weight matrix for MDP $l$, and $\mathbf{\Sigma}_l \in \mathbb{R}^{n_l \times n_{h_l}}$

corresponds to the inverse of the covariance matrix.

Visible layer distributions are Gaussians while the hidden layer uses sigmoids. Inference is performed in parallel for each of the nodes with different inputs according to:

$$\mu_l^{(i)} = \sum_{j=1}^{n_{h_l}} w_l^{(i,j)} h_l^{(j)} + a_l^{(i)} \text{ for each visible node } i$$
$$s_l^{(j)} = \sum_{i=1}^{n_l} w_l^{(i,j)} v_l^{(i)} + b_l^{(j)} \text{ for each hidden node } j$$

Learning the weights for each of the RBMs done via normal CD:

$$\Delta \mathbf{W}_l \propto \left\langle \mathcal{H}_l \mathcal{V}_l^T \right\rangle_0 - \left\langle \mathcal{H}_l \mathcal{V}_l^T \right\rangle_\lambda$$

where $\langle \cdot \rangle_0$ is the expectation of $\cdot$ in the original data set (i.e., samples) and $\langle \cdot \rangle_\lambda$ is the attained from a Markov chain starting with the original data distribution and running for $\lambda$ steps.

**Second Two Layers:** The third layer is trained to reconstruct the second learned hidden layer. Each MDP is given the same number of hidden nodes to use in this layer. Define $\mathcal{H}_{l,3} = [h_{l,3}^{(1)}, \ldots, h_{l,3}^{(n_{h_3})}]$ with $n_{h_3}$ the number of nodes used in the third layer for each MDP $l$. The energy function equation becomes

$$E_l(\mathcal{H}_l, \mathcal{H}_{l,3}) = -\mathbf{b}_l^T \mathcal{H}_l - \mathbf{b}_{l,3}^T \mathcal{H}_{l,3} - \mathcal{H}_l^T \mathbf{W}_{l,3} \mathcal{H}_{l,3}.$$

where, $\mathbf{b}_{l,3}^T$ are the biases of the third layer and $\mathbf{W}_{l,3}$ are the weight matrices for $l = \{1, 2\}$. Inference is performed in parallel as

$$s_l^{(j)} = \sum_{k=1}^{n_{h_3}} w_{l,3}^{(j,k)} h_{l,3}^{(k)} + b_l^{(j)} \text{ for each node } j \text{ of layer 2, and}$$
$$s_{l,3}^{(k)} = \sum_{j=1}^{n_{h_l}} w_{l,3}^{(j,k)} h_l^{(j)} + b_{l,3}^{(k)} \text{ for each node } k \text{ of layer 3.}$$

Weight learning is performed by using:

$$\Delta \mathbf{W}_{l,3} \propto \left\langle \mathcal{H}_{l,3} \mathcal{H}_l^T \right\rangle_0 - \left\langle \mathcal{H}_{l,3} \mathcal{H}_l^T \right\rangle_\lambda.$$

**Last Two Layers:** Define $\mathcal{H} = [h^{(1)}, \ldots, h^{(n_h)}]$, where $n_h$ is the index of the last hidden unit in the third hidden layer. The energy function, using only the hidden nodes from the third layer corresponding to the source MDP, becomes:

$$E(\mathcal{H}_{1,3}, \mathcal{H}) = -\mathbf{b}_{1,3}^T \mathcal{H}_{1,3} - \mathbf{b}^T \mathcal{H} - \mathcal{H}_{1,3}^T \mathbf{W} \mathcal{H}$$

where, $\mathbf{b}$ is the bias vector for the last hidden layer, and $\mathbf{W} \in \mathbb{R}^{n_{h_3} \times n_h}$ is the matrix of the weight connections. Inference could be performed using:

$$s^{(j)} = \sum_{k=1}^{n_{h_3}} w^{(k,j)} h_{1,3}^{(k)} + b^{(j)} \quad \text{for each final layer node } j$$

$$s^{(k)} = \sum_{j=1}^{n_h} w^{(k,j)} h^{(j)} + b_{1,3}^{(k)} \quad \text{for each node } k \text{ of the source task's third layer}$$

$\wp : \mathcal{H}_{2,3} \rightarrow \mathcal{H}_{1,3}$ is a mapping of the target tasks's third hidden layer into the source MDP's third level basis. In this work $\wp$ is chosen to be an identity mapping. This can be done since the third hidden level includes the same number of hidden units for both MDPs. Using $\wp$, the target hidden layer is "reflected" towards the source MDP. $\wp(\mathcal{H}_{2,3})$ is then reconstructed using the already learned hidden layer $\mathcal{H}$. The intuition is that if two MDPs are related in their dynamics, the hidden layer describing the source MDP should also be capable of describing the dynamics of the target MDP. The reconstruction, used to determine the MDPs similarity exactly as in RBDist, is computed using

$$p(\mathcal{H}|\wp(\mathcal{H}_{2,3})) = \prod_{j=1}^{n_h} \text{sig}(\sum_{k=1}^{n_{h_3}} w^{(k,j)} \wp(h_{2,3}^{(k)}) + b^{(j)})$$

$$p(\mathcal{H}_{1,3}|\mathcal{H}) = \prod_{k=1}^{n_{h_3}} \text{sig}(\sum_{j=1}^{n_h} w^{(k,j)} h^{(j)} + b_{1,3}^{(k)})$$

where $\text{sig}(x) = \frac{1}{1+\exp(-x)}$ is the sigmoidal function. The resulting similarity measure is called DRBDist.

## 7.4 Experiments and Results

The adequacy of RBDist and DRBDist as MDP similarity measures was tested on three standard reinforcement learning benchmarks: Mountain Car (MC), Inverted Pendulum (IP), and Cart Pole (CP). In a first set of experiments, RBDist was used

(see Section 7.4) to see if it could differentiate the different dynamical phases (e.g., oscillating, damped, critically damped) these systems exhibit using different environmental parameters. We sampled the systems in parametrical setups belonging to each of different phases and used RBDist to cluster the different setups (see Figures 7.3, 7.5 and 7.7). To check for the correspondence between RBDist and transfer learning, optimal behavior for an MDP from one cluster used to transfer to other MDPs of the same system belonging to either the same or different clusters. The jumpstart results show that as RBDist between the MDPs increases the jumpstart decreases (see Figures 7.4, 7.6 and 7.8).

DRBDist was tested in a similar fashion, but using different benchmarks. A set of 3 experiments were performed using a different benchmark (either MC, IP or CP) as the source task. The DRBDist average over all dynamical phases are summarized in Table 7.1. Again, to test for the correspondence to transfer learning, two transfer experiments were performed. In each of these experiments, a number of CPs belonging to different phases were target tasks, while the source tasks were either oscillating IP or oscillating MC. Jumpstart on the CP was measured in correspondence with the similarity between the CP clusters and the source system (see Figure 7.9 and 7.10).

## Experimental Domains

The three experimental domains used are explained next.

**Inverted Pendulum:** The variables describing the state of the IP are the angle $\theta$ and angular velocity $\dot{\theta}$. The action space the agent can choose from is consists of two torques values $\tau = [-10, 10]$ in units of $Nm$. The goal of the agent is to balance the pole in an upright position. The agent's reward is set to $-1$ for every time step outside $-\frac{\pi}{9} < \theta < \frac{\pi}{9}$, and $+1$ for every time step it's angle is in the target region. For RBDist, optimal policies were obtained using SARSA with a Q-table representation. To show that the proposed measures are not restricted to a specific RL algorithm, least-squares policy iteration(LSPI) was used in the DRBDist experiments.

**Cart Pole:** The state of the CP system is described via the angle and angular velocity of the pole and the position and velocity of the cart (i.e., $s = \langle \theta, \dot{\theta}, x, \dot{x} \rangle$). The agent's action space is a set of 11 equally distanced linear forces between $[-1, 1]$. The goal is to stabilize the pole in an upright position. A reward of $+1$ is delivered to the agent at each step the angle is between $-\frac{\pi}{9} < \theta < \frac{\pi}{9}$ and the position is $-4 < x < 4$. In case these conditions are not met, the agent receives a $-1$ reward. The optimal policy for the cart pole in the RBDist experiments was build using SARSA.

**Mountain Car:** The state of the MC is described via the position, $x$, and velocity, $\dot{x}$. The agent can choose from three linear forces $F = \{-1, 0, 1\}$. The car starts at the bottom of the hill and has to drive to the top. The car's motor is insufficient to drive the car straight to the goal state. Therefore, the agent has to oscillate to reach the goal position where it receives a positive reward. The position of the car is bounded between $[-1.5, 1]$ and the velocity is bounded between $[-0.007, 0.007]$. SARSA was used for both RBDist and DRBDist experiments.

## RBDist Experiments

Each of the three systems exhibit different dynamical phases, depending on the parameters values of their transition models: oscillating, damped or critically damped. Each of the phases requires a substantially different control policy in order to attain the desired behavior. To test RBDist, each of the above systems was intentionally set to these different phases by varying their dynamical parameters. Samples from each setting were used to measure the similarity to other settings using Algorithm 24. The details of the experiments are explained next.

### Inverted Pendulum Experiments

To set the system in different phases, the inertia of the rod, $J$, and the damping constant between the rod and the wall-pin, $b$, were varied. This produced three types of behavior: (1) low damping with high inertia, and thus high oscillations, (2) medium damping with high inertia, still oscillating but at medium frequencies and (3) high damping such that the system does not oscillate. A system from phase one was chosen as a reference behavior and Algorithm 24 was used to determine the similarity to the other systems. Results are shown in Figure 7.3. The x-axis corresponds to different MDPs randomly sampled in each of the three different phases of the system. Each sample set contained 5000 transitions. The y-axis represents the similarity between these different MDPs to the reference. Different colors, red, green, blue show the ground truth of the different phases. Figure 7.3 shows that similar phases result in similar differences. The first MDPs with the smallest difference all belong to the highly oscillating phase of the IP system, as does the reference MDP. The second phase (medium oscillation and indicated as green dots of Figure 7.3) results in a bigger difference than the highly oscillating phase, and a smaller difference than the third, damped phase.

A set of experiments were then conducted to test the correlation between the difference measure and transferability. An optimal policy from the reference MDP, learned with SARSA, was used as a source policy. The optimal Q-values were used to initialize the Q tables of the other MDPs. The policies specified by these Q-values were

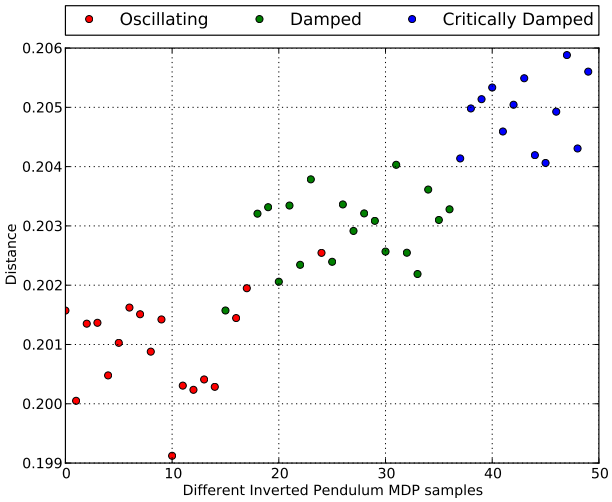



Figure 7.3: Different MDPs from three phases are sampled and the RBDist values are plotted on the y-axis. The ground truth phase is shown by each datapoint's color.

then greedily followed and the jumpstart (e.g., the performance improvement without additional learning over no transfer) was averaged over 300 episodes. The results are shown in Figure 7.4. As the similarity between the source MDP and the target MDPs decreases, i.e., the target task is in phase two or phase three, the jumpstart decreases. This illustrates the correspondence between RBDist and TL performance because jumpstart is highest when the target is in the same cluster as the source (lowest distance), has middle performance when the target exhibits behavior two (middle distance), and has the worst average performance when the target is from behavior three (highest distance).

### Cart Pole and Mountain Car Experiments

Similar experiments were performed on the CP system. Here the length (i.e., the inertia of the pole) and the damping constant were modified to put the system in three different phases. Figure 7.5 again shows that the proposed measure was capable of automatically clustering similar dynamics. Jumpstart experiments were repeated for the CP, where an optimal policy was attained from the first MDP of the first cluster, and used to transfer to other CPs. The results shown in Figure 7.6 also manifest the correspondence between RBDist and transfer performance.

Finally, the same experiments were repeated in the MC system. The mass of the car was varied to set the car in one of the three phases, as described above. Figure 7.7

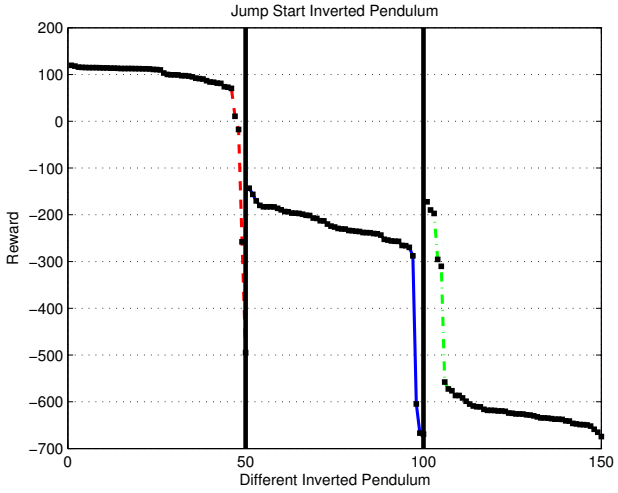



Figure 7.4: Jumpstart results for inverted pendulum where the source is from behavior one and the target is from behavior one (left), behavior two (middle) or behavior three (right). Performance is correlated with the behavior of the target task, and therefore with the RBDist in Figure 7.3.

shows again that systems in similar phases obtain similar RBDist values when compared with the reference MDP. The jumpstart experiments were also repeated for the MC system. Figure 7.8 again shows that as RBDist between the MDPs increases, the jumpstart behavior resulting from transfer decreases. It is possible for MDPs belonging different clusters to results in relatively large jumpstart gains, but the probability of this becomes much lower.

It is clear from the above experiments, that the proposed measure was capable of discovering relevant phases in dynamical systems, and is meaningful for transfer where as the RBDist between the MDPs increases the jumpstart decreases.

## DRBDist Experiments

A second set of experiments was conducted to show the applicability of DRBDist. First, the similarity was measured by training the RBM discussed in Section 7.3 on each of the different tasks, which was then used as the reference MDP to measure the similarity to the others. For instance, the IP systems was used as the reference MDP and the similarity to MC and CP was measured. This procedure was repeated by taking each of the different systems as a reference. Second, this measure was used to test the jumpstart attained from cross domain transfer, where the target tasks were

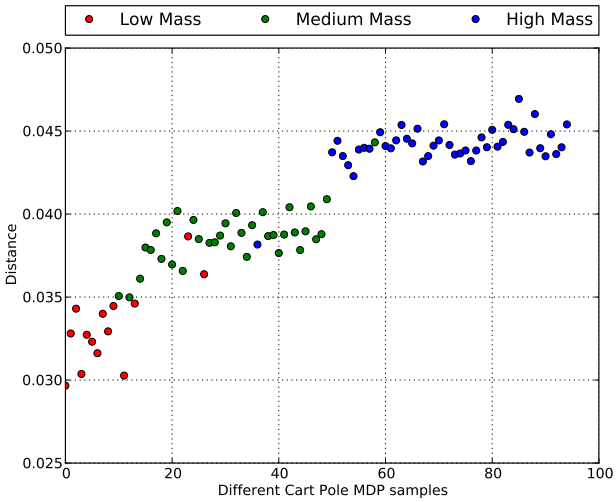



Figure 7.5: RBDist values are measured for the three different behavior phases of the cart pole in reference to an MDP of the red (low mass) phase.

different CPs and the source tasks were MC tasks. Third, the target tasks were again CP tasks, but IPs were used for the source tasks.

**DRBDist Measure:** Different experiments to determine the similarity between different MDPs were conducted. The results of using DRBDist between the MDPs are summarized in Table 7.1. RBMs were trained on one of these systems and then used to measure the similarity to MDPs — the smallest values are along the diagonal, as these represent training and testing on the same MDP variations.

**Cross Domain Transfer** DRBMDist can be used in cross-domain transfer. The first experiment examined the transfer between the most similar systems (i.e., IP to CP), while the second considered less similar systems (i.e., MC to CP). To transfer these behaviors to the CP there exists the need for an inter-task mapping relating the source and target state actions spaces. Learning this inter-task mapping was done using an existing method [14]. The transferred policy was then used in the target task (i.e., CP) and the jump start was measured over 500 episodes each.

Figure 7.9 shows the results of transfer from an MDP from the first IP cluster to different phases in the CP system. The vertical lines separate the different dynamical phases of the corresponding system. It is clear that although there is not a large difference between the jumpstarts in the different clusters, rewards in the first cluster (i.e., similar pole lengths in the IP and CP) show a higher jumpstart on average. This

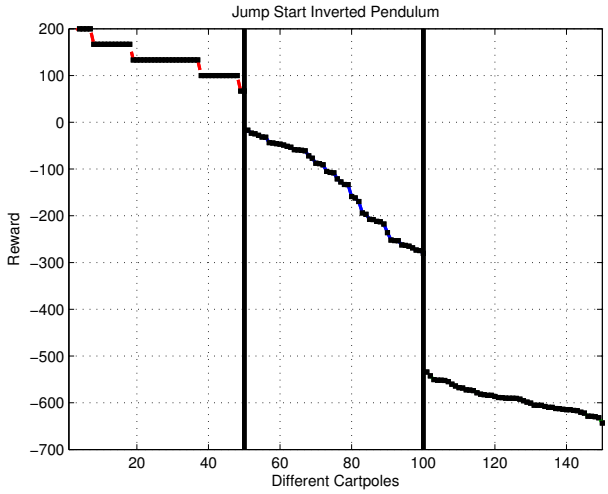



Figure 7.6: Jumpstart results for cart pole show a correlation between the attained jumpstart and RBDist values, where the MDPs on the left are drawn from the same behavior as the source, the MDPs in the middle come from the medium mass phase, and MDPs on the right exhibit high mass behavior.

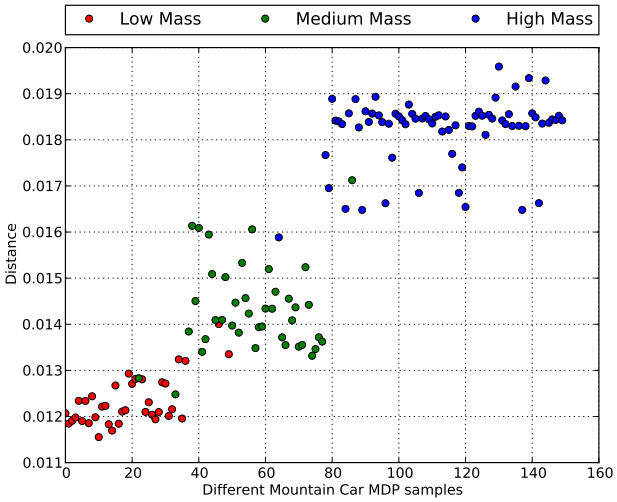



Figure 7.7: RBDist values for the three different phases of the cart pole are shown, measured with respect to a reference MDP from the low mass (red) behavior phase.

is also in accordance with the DRBDist values between the source system (i.e., the IP) and the CP clusters, shown at the top of each column in Figure 7.9.

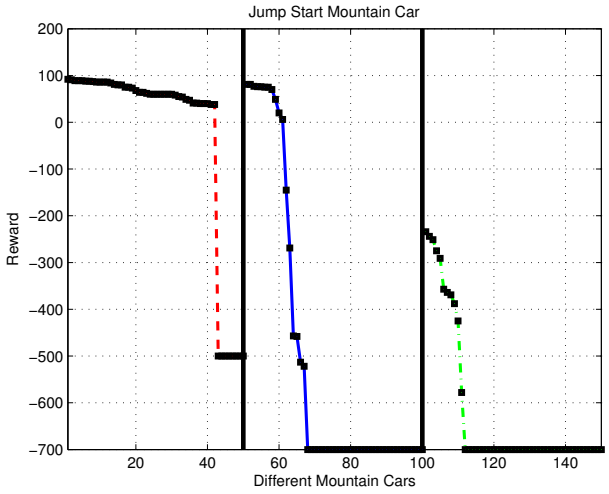



Figure 7.8: Jumpstart results for mountain car also show a similar correlation between jumpstart and RBDist values.

Similar experiments were conducted from MC to CP. Again the first system in the first cluster of the MC was chosen as the source task. The behavior of this transfer was then tested across the different clusters in the CP. This time, the jumpstart is small compared to using the IP. The average jumpstart for transferring from the MC to the CP is $-630$ for the first cluster, $-523$ for the second cluster, and $-678$ for the third cluster. In contrast, random action selection was merely able to attain a reward of $-680$ on average over all the clusters. Therefore, if the cart pole belonged to the first or second cluster, then the agent will benefit from transferring.

In total, these two sets of experiments show us that i) transfer from IP to CP is more useful than MC to CP, as was predicted by the DRBDistance metric, ii) the average jumpstart for IP to CP is correlated with the DRBDist metric, iii) the DRBDist metric is not yet fully predictive, because when transferring from MC to CP, the second cluster has a higher jumpstart than the first, even though its DRBDist is slightly higher than the first.

## 7.5 Discussion and Conclusion

In this paper two measures for MDPs were proposed. These measure are of major interest for the TL and the RL community in general. The first, RBDist, is based on RBMs and suited the problem of measuring the similarity between MDPs having

| | | MC | IP | CP |
|---|---|---|---|---|
| MC | $\mu$ | $\mathbf{1.15}\times10^{-3}$ | $5.71\times10^{-2}$ | $6.13\times10^{-2}$ |
| | $\sigma$ | $1.72\times10^{-5}$ | $4.16\times10^{-4}$ | $1.30\times10^{-4}$ |
| IP | $\mu$ | $4.28\times10^{-2}$ | $\mathbf{1.59}\times10^{-3}$ | $6.42\times10^{-2}$ |
| | $\sigma$ | $1.55\times10^{-5}$ | $6.11\times10^{-6}$ | $6.67\times10^{-3}$ |
| CP | $\mu$ | $4.62\times10^{-2}$ | $5.83\times10^{-2}$ | $\mathbf{2.66}\times10^{-3}$ |
| | $\sigma$ | $1.39\times10^{-5}$ | $5.83\times10^{-5}$ | $8.44\times10^{-4}$ |

Table 7.1: DRBDist between different MDPs. Each row represents where the DBN has been trained (i.e., the reference MDP), while the columns denote the similarity measure to each of the other MDPs. Both the means, $\mu$, as well as the standard deviation, $\sigma$, attained from 150 systems for each of the MDPs are reported.

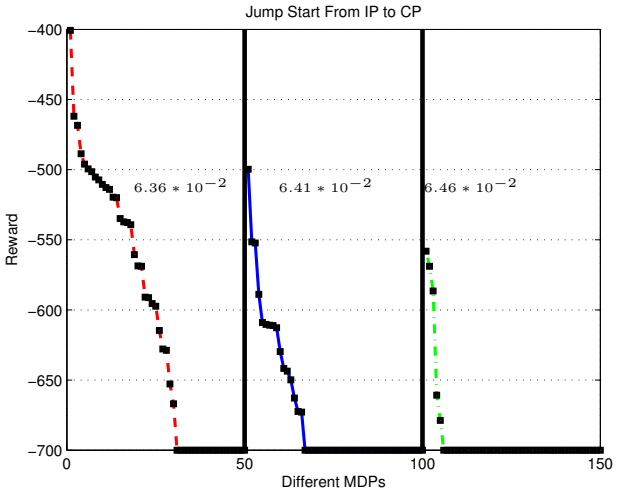



Figure 7.9: Jump start results from transferring from an IP to the CP. The numbers at the top show the average difference between the cluster and the reference MDP.

the same state and action spaces. The second measure, DRBDist, is capable of measuring similarities between MDPs with differences in all of the constituents, making it suitable for cross domain transfer. Different experiments show that these measure are able to predict meaningful transfer learning results. This represents a critical step towards being able to automatically select a source task when given a target task. It can also help to avoid the problem of negative transfer.

There are many interesting future directions of this work. For instance, other transfer learning criteria might have certain correlations with the proposed measure. These could be investigated in more details. Furthermore, these measures could serve

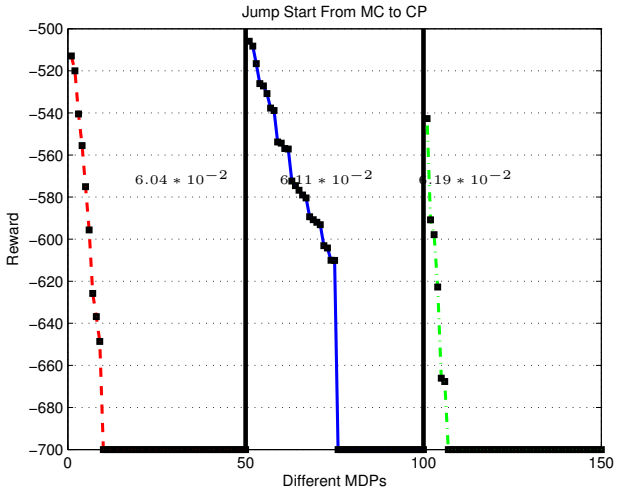



Figure 7.10: Jump start results for transferring from a MC to the CP. The numbers at the top show the average difference between the cluster and the reference MDP.

as a starting point for research in the direction of formally quantifying the performance of different TL for RL algorithms.

It is clear that the proposed measures are capable of capturing similarities between different reinforcement learning tasks. Such measures can now be used by a transfer learning agent in order to assess the similarity between different source task(s)[2] and a target. Therefore, using this measure a transfer agent can now decide on which MDP to transfer from.

[2]The extension of these measures to multiple MDPs is trivial and is left as an exercise for the reader.

# 8

# Conclusions and Future Work

In this chapter the conclusions of this dissertation are presented. The research questions are first answered offering an overall solution to the problem statement. Finally, future work directions are discussed. These are split into improvements over the proposed methods, as well as potential applications in different artificial intelligence fields.

## 8.1 Answers to Research Questions

This dissertation started with the goal of creating autonomous transfer agents for reinforcement learning tasks. Such scenarios required that a transfer agent performs the following successfully:

1. Given a target task choose the relevant source task(s).
2. Learn about the relation between source and target task(s).
3. Effectively use the learned relation to transfer knowledge between the tasks.

Aiming at a solution for each of the steps, seven research questions were formulated. Each chapter contributed by answering some of these questions. These are detailed next.

**Research Question 1:** *What categorization of the current transfer algorithms exist?*

To asses the starting point at which research in this dissertation should start, a survey of different algorithm for the transfer between reinforcement learning tasks has been conducted. Chapter 3 provided a categorization of these algorithms. This categorization constituted a formal framework in which transfer algorithms fit. Mainly, the different transfer schemes were separated according to the procedure in which learning about the inter task mapping was performed. Two categories were first distinguished: (1) deep, and (2) shallow transfer. In deep transfer, the source and target

state and/or action spaces are different, while in the second the domains (i.e., state and action spaces) are the same.

After performing this literature study, it became clear that no autonomous transfer framework existed. Most of the proposed algorithms in transfer literature require substantial human intervention. Trying to reduce the burden on the user in relating source and target reinforcement learning tasks, the next research question was formulated.

**Research Question 2:** *To what extent is it possible to reduce human intervention in reasoning about the relations between different tasks?*

When a source and target task have different state and action spaces, an intertask mapping relating these is essential. Chapter 4 presented an approach that requires a common state subspace to learn an intertask mapping.

The method operates within a deep transfer learning setting and required the following knowledge:

- Random source and target state successor state transitions.
- Optimal source task policy (i.e., $\pi_1^\star$).
- *Human* defined common state subspace.

The overall approach, shown in Figure 8.1, can be split into two phases. The first phase, shown at the top of the figure, corresponds to learning the interstate mapping, while the second phase, shown at the bottom, represents the learning of the interaction mapping. First, the source and target random samples are projected onto the *human* defined common subspace. A similarity measure is then used to correspond these projected samples. The output of such a similarity correspondence, is a data set with target and source states as inputs and outputs, respectively. Here, a regression algorithm is adopted in order to learn the inter state mapping, $\chi_{inter_{\mathcal{S}}}$.

Having the interstate mapping, the next question is to learn an interaction mapping between the two tasks. This problem is approached indirectly. Starting from a specific target state $s_2 \in \mathcal{S}_2$, the interstate mapping $\chi_{inter_{\mathcal{S}}}$ is used to correspond $s_2$ to the source task. From the attained source state, the optimal source policy, $\pi_1^\star$, is used to determine the "optimal source transition". At this stage, the question is to determine the action in the target that produces the closest transition to the optimal source transition. Starting from the target state, all actions are executed to transition to different target successor states. These are then projected to the common subspace in which the similarity to the optimal source transition is determined. The above steps are repeated for a certain number of target transitions leading to an initial policy, $\pi_2^{(0)}$ in the target task.

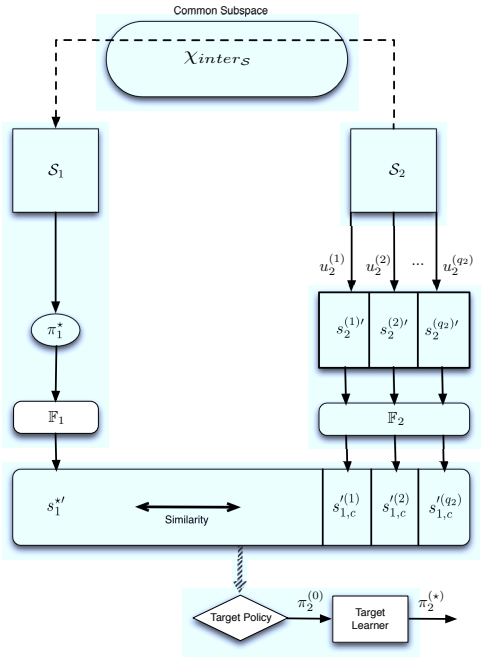



Figure 8.1: High-level schematic of the overall common subspace framework.

Of course, $\pi_2^{(0)}$ is not optimal. However, if the source and target tasks were similar enough, this policy should be a good start to improve on. Therefore, to attain an optimal behavior in the target, a sample based reinforcement learning algorithm is used.

Although successful, this approach required human intervention in defining the common subspace. This is not always easy. Aiming at an automatic framework that is able to autonomously learn an intertask mapping, the next research question was derived.

**Research Question 3:** *Is it possible to automate learning the relation between different reinforcement learning tasks?*

To automate learning the intertask mapping between two reinforcement learning tasks, Chapter 5 provided an approach based on: (1) sparse coding, (2) sparse projection learning, and (3) sparse gaussian processes. The method operates within a deep transfer learning setting and requires the following knowledge:

- Random source and target state-action-successor state transitions.
- Optimal source task policy (i.e., $\pi_1^\star$).

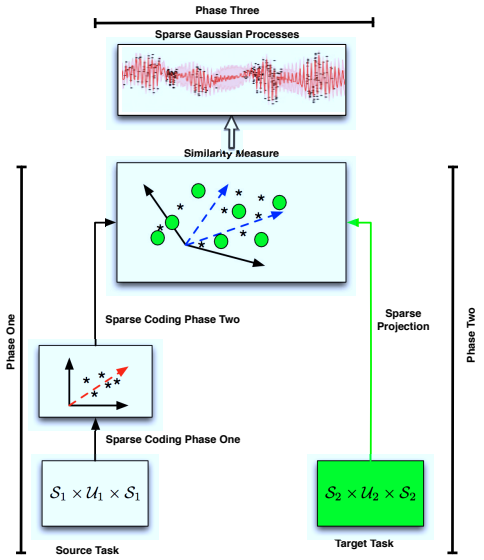



Figure 8.2: A high level schematic of the overall approach. It consists of three major phases. In the first high level features are detected in the source task MDP. In the second, samples from the target task are projected to that space. Finally, in the last phase (i.e., phase three), sparse Gaussian processes are used in order to learn the intertask mapping.

The overall approach is shown in Figure 8.2. There are three essential phases needed to automatically learn the intertask mapping. In the first phase, two sparse coding steps are performed. The first ensures that the source and target tasks have the same dimensionality, while the second step discovers more informative features of the source. In the second phase, target task samples are then projected to this discovered space, where a similarity measure is applied to correspond the source and target triplets. The result of this correspondence is a data set with source transitions as inputs and target transitions as outputs. Having this data set, regression, using sparse gaussian processes, is applied to learn the intertask mapping.

Although successful, the proposed approach might suffer if the samples were not informative enough. Aiming at a more robust framework, the following research question was formulated.

**Research Question 4:** *Are there any possible more robust alternatives to the learning of the relations between the tasks?*

Having automated the intertask mapping, Chapter 6, provide a more robust alternative compared to the previous method. The overall approach is based on a high order

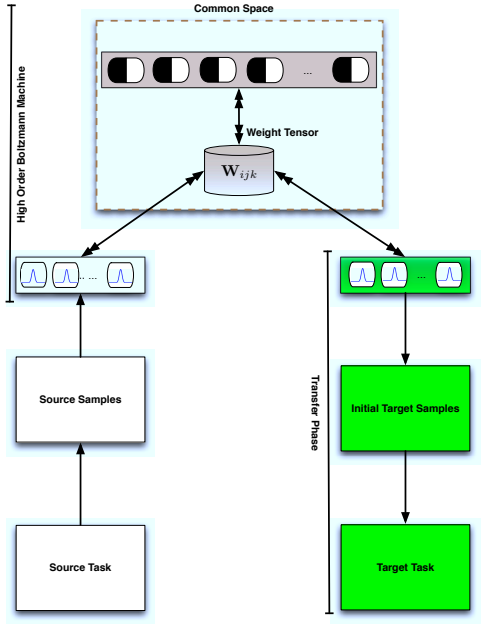



Figure 8.3: The overall framework of learning an intertask mapping using a three way restricted Boltzmann machine.

restricted Boltzmann machine. More specifically, a three way restricted Boltzmann machine is used to learn the intertask mapping. The method operates within a deep transfer learning setting and requires the following knowledge:

- Random source and target state-action-successor state transitions.
- Optimal source task policy (i.e., $\pi_1^\star$).

The overall approach is shown in Figure 8.3. The main idea is to learn a common high dimensional space that is capable of representing both tasks. This space is encoded by the hidden and weight configuration of the high order restricted Boltzmann machine. Using random source and target transitions, the machine is trained such that it is capable of reconstructing transition from both tasks. After convergence, this machine will encode an intertask mapping that can be used for transfer.

So far three different approaches to learn an intertask mapping between source and target reinforcement learning tasks have been proposed. Therefore, the next question is how to effectively use this mapping to ensure successful transfer. To that end, the following research question was formulated.

**Research Question 5:** *How to effectively utilize the learned mapping to successfully transfer between different tasks?*

Having proposed three techniques for learning the intertask mapping between reinforcement learning tasks, the question was how to best transfer knowledge across these tasks. This question was answered by the usage of sample based and efficient reinforcement learning algorithms. To insure that the scope of applicability of the proposed transfer schemes cover a broad range of applications, the focus was on algorithms that operate within continuous state space settings. In Chapters 5, and 6, two novel transfer algorithms TrLSPI and TrFQI were proposed. These algorithms, operating in continuous state spaces, make use of state-of-the-art reinforcement learning algorithms in addition to the intertask mapping to ensure behavioral improvements in the target task. On the contrary, the algorithm proposed in Chapter 4, although efficient, is model based and therefore, is restricted to situations in which a model of the environment is available.

The second part of this dissertation targeted the question of automatically choosing a relevant source task for a given target. It seemed that to tackle this challenge, a similarity measure relating two MDPs was essential.

This problem was split into two research questions (i.e., research questions 6 and 7):

**Research Question 6:** *Is it possible to learn a similarity measure between reinforcement learning tasks with same state and action spaces?*

The first part of Chapter 7 introduced a method that is capable of learning the similarity between two reinforcement learning tasks (i.e., RBDist). The overall framework of this measure is shown in Figure 8.4.

The main intuition behind the approach is that if two MDPs are similar, then an informative space representing the first, should also be able to reconstruct samples from the second. To discover this space, a restricted Boltzmann machine, shown in Figure 8.4, was adopted.

Although successful as shown in the experiments of Chapter 7, the proposed measure is restrictive as it only operates in the setting where the source and target MDPs have the same domain. For autonomous transfer agents, the case in which the source and target tasks have different domains is of major importance. To extend the RBDist the following research question was formulated.

**Research Question 7:** *Is it possible to learn a similarity measure between reinforcement learning tasks with different state and action spaces?*

The second part of Chapter 7 extended RBDist to DRBDist. The overall framework of DRBDist is shown in Figure 8.5. The main intuition behind this measure is

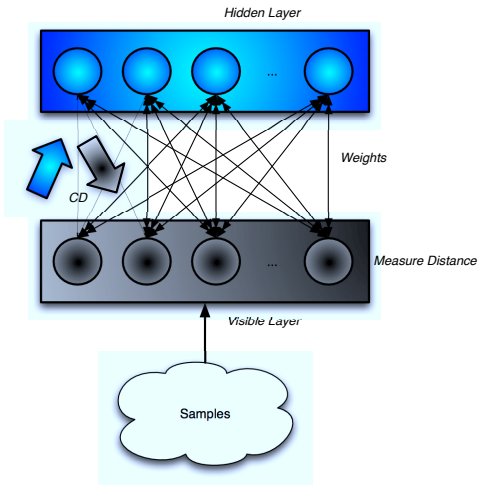



Figure 8.4: This is a high level schematic of the similarity measure between MDPs with shared state-action spaces. Both training and reconstruction use CD.

similar to RBDist, where if two MDPs are similar, then an informative space that is capable of describing the first will also be to some extend able of reconstructing samples from the second. However, attaining this common space is not as easy anymore since the two MDPs have different state and action spaces. In other words, they might require different neural configurations to be accurately abstracted. To guarantee that the similarity measure is capable of accurately abstracting the MDPs under study a four layer deep belief network (DBN) is proposed.

Each layer, shown in Figure 8.5, is essential to the success of DRBDist. Given source and target samples, the first step is to represent these transitions in a high dimensional and informative space. Since these MDPs might require different configurations to be accurately abstracted, then there are no restrictions on the number of units to be used in the first two layers of the DBN. Next, to create a reconstruction layer, the third layer is added. The number of units in this layer is constrained to be the same for both tasks. At this level, all transitions of both MDPs are described using the same representation. However, to enable a meaningful measure, these should be projected to the same space, where reconstruction can be measured. This is done using $\wp(\cdot)$ as shown in Figure 8.5. Here, the target transitions can be reconstructed using the deep Boltzmann machine of the source.

Experiments have shown the correspondence between the proposed measures and transfer. Therefore, these can be easily used to determine which source to choose for a given target.

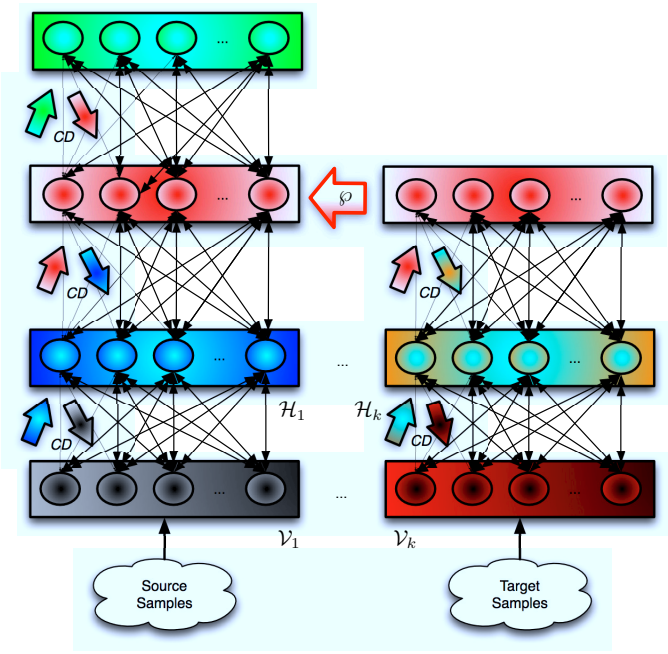



Figure 8.5: This figure diagrams DRBDist. The bottom (visible) layer receives task data. The first hidden layer constructs a more representative feature space of an MDP. Both second hidden layers have the same structure and number of nodes, allowing the target to be matched to the source. The reconstruction error in the third hidden layer measures task similarity.

## 8.2 Answers to the Problem Statement

This dissertation aimed at creating a framework that is capable of autonomously transferring between reinforcement learning tasks. The agent had to fulfill three goals autonomously. Chapters 4, 5, and 6 contributed by proposing methods capable of: (1) autonomously learning about the relations between source and target reinforcement learning tasks, and (2) effectively using this learned relation to efficiently transfer between them. These provided answers to the second and third steps required from an autonomous transfer agent. For full autonomy one more step is needed. Chapter 7 proposed two measures between MDPs and showed that these correspond with transfer. Therefore, RBDist and DRBDist are capable of capturing similarities between reinforcement learning tasks, making them suitable for the use by transfer agents to choose the relevant source task(s) for a given target.

Given all the above, it can be concluded that each of the three steps required by

an autonomous transfer agent have been fulfilled paving the way for real autonomy in transfer learning. There are a lot of interesting future directions for the proposed work in this dissertation. These could be split into theoretical contributions, application in different AI fields, as well as improvements of the currently proposed methods. The next section briefly mentions these ideas.

## 8.3 Ideas for Future Work

This section details various ideas for future work. Three main directions can be differentiated. In the first, improvements of the current proposed frameworks are described. In the second, applications in various AI fields are explained, and finally possible theoretical contributions are distinguished.

### In-Chapter Improvements

This section describes possible future work directions in Chapters 4, 5, 6, and 7.

**Chapter 4:** Future work can involve three major goals. The first is to extend the proposed algorithms to operate in stochastic model-free MDP settings. The second is to learn a common subspace automatically in both the action and state spaces. Various ideas could be used to achieve such a goal, one of which could be a dimensionality reduction scheme constrained by the common characteristics shared by the different tasks. The third is to test the transfer method with multiple algorithms including policy iteration, Sarsa($\lambda$) and Q-learning.

**Chapter 5:** Three interesting future work direction can be differentiated. First, a comparison of different distance metrics (e.g., weighted Euclidean measures) can be performed and their effects on the overall performance of the algorithm can be assessed. Second, the distance measure can be improved by incorporating the rewards in the framework, helping to avoid the problem of negative transfer. The rewards can be related using their own space that can be discovered using the same procedure of sparse coding and sparse projections. After relating the reward functions, transfer can be constrained by such a similarity, where only transitions generating a positive reward in the target is transferred while others are ignored. Third, transfer can be conducted directly using the sparse coded space, without the need for function approximations.

**Chapter 6:** Future work can involve two directions. First, to avoid negative transfer, the rewards might be incorporated within the overall construction of the high

order restricted Boltzmann machine. An additional reward unit in both the source and target tasks' layers can be added. With this addition, the network will converge to learn similarities in both transitions models, as well as reward functions of the two tasks. Second, to improve robustness, the network's construction can be changed into a deep belief network by stacking multiple CRBMs over each other.

**Chapter 7:** There are many interesting future directions of this work. For instance, other transfer learning criteria might have certain correlations with the proposed measure. These could be investigated in more detail. Furthermore, these measures could serve as a starting point for research in the direction of formally quantifying the performance of different TL for RL algorithms. Additionally, because of the structural prediction capabilities of RBMs and deep networks, we believe that the network structure as used by DRBDist could be used in future work to generate a sequence of MDPs that can be used to transfer between two tasks too different to allow successful transfer directly.

## Applications

There are various interesting applications for the proposed methods in different AI fields. This section, sheds the light on these possibilities without diving into all the details.

**Multi-agent Systems and Swarm Intelligence:** There are a lot of interesting applications for the ideas proposed in this dissertation in the fields of Multi-agent systems and swarm intelligence. Transfer in general can be of major help when it comes to learning in a multi-agent setting. For example, when certain agent(s) have acquired a representative behavior in a specific region of an environment, transfer can be helpful for other agent(s) in different areas of that environment. Chapter 7 can be used to asses the similarities between such sub-tasks and Chapters 5 and 6 can then perform autonomous transfer.

**General Game Playing:** The problems encountered in the field of general game playing are hard to solve mainly due to the complexity of the intended tasks. Transfer learning in general, as well as the ideas proposed in this dissertation, might serve as a potential solution to such problems. For example, the distance measure of Chapter 7 can be used to factor a huge MDP into smaller and easier MDPs. After this factoring has been performed, learning can commence on these factored systems. To map back from the factored models to the original game, transfer is required. Ideas from

Chapters 5 and 6 can be used to automatically learn about the intertask mappings and thus solve the original MDP.

**Automated Negotiations:** In automated negotiations one of the toughest challenges is learning opponent models. When faced with new opponents agents typically start from scratch. Transfer might be beneficial in such scenarios. When formulated as a reinforcement learning problem, ideas from this dissertation can be used to: (1) choose the relevant source opponent, and (2) effectively and autonomously transfer.

### Theoretical Contributions

The last direction of future work mentioned here corresponds to theoretical contributions for the proposed techniques. For example, Chapters 5 and 6, first learn an intertask mapping and then use such a mapping in a sample-based RL algorithm to perform transfer. Theoretical analysis can be performed to, for instance, attain finite sample bounds on the errors of such algorithms. Another interesting direction could be to study the transferred model distributions by quantifying the effect of transfer using the ideas of Chapter 7. Starting directions for conducting such theoretical analysis as well as relating transfer to the analysis of exploration in reinforcement learning can be inspired by the work of [64].

### Final Remark

In this dissertation different transfer learning algorithms have been proposed and shown successful in a variety of experiments. These methods relied on learning a space that relates the state and action spaces among different dynamical systems. It became clear, that systems that appear highly dissimilar in one description are actually highly similar when described in "correct" high dimensional spaces. This idea in general, can be further pursued as it might serve as an interesting starting point in studying human transfer.

# Summary

Originally, artificial intelligence (AI) was created to comprehend human intellect. Since then, different fields and sub-domains of AI have been established. Among these fields, maybe the closest to humans, is that of reinforcement learning (RL). In RL, agents try to act in potentially unknown environments using only trial and error, a technique that has proven liable in a variety of human learning settings. In human learning however, neutral information and objective thinking is impossible. Almost always, humans make use directly or indirectly of already acquired knowledge to aid learning in a new task. Albeit, most RL agents are tabula rasa learners. If RL is to mimic human learning, then knowledge reuse is essential. Transfer Learning (TL) is that sub-field of AI dedicated to targeting these challenges. Although successful in different applications, the strive to create autonomous transfer agents is still far-fetched. Taylor and Stone [107], define an autonomous transfer agent as having the following three capabilities: (1) appropriate selection of source task(s), (2) successful learning of the relation between the source and target task(s), and (3) effective knowledge transfer between the tasks.

In this dissertation the previous three problems are targeted and different contributions are made. Firstly, two novel and automated approaches to learn the relationships between RL tasks are proposed. On a high level, these novel methods work by discovering a unifying, rich, and descriptive space that is capable of potentially representing both tasks. These spaces are then used as the basis for learning the intertask mapping between different tasks. Secondly, using this learnt relation, effective and efficient knowledge transfer is executed by using state-of-the-art RL algorithms, such as, least squares policy iteration, and fitted Q-iteration. Here, the transferred knowledge is in form of transitions that are used to bias sample-efficient RL algorithms in their action-selection scheme, thus, leading to an increase in the learning performance. Finally, two data driven similarity measures between different source and target RL tasks are presented. Such measures, allow for the automatic determination of appropriate source task(s) to a given target. The measures adopt and extend Deep Belief Networks (DBNs) for issues of effectiveness and robustness. It is shown that these measure are not only capable of discovering different dynamical phases in the same family of systems, but are also able of relating cross domain similarities between different tasks.

# Samenvatting

Reinforcement learning is een methode waarin agenten proberen om doormiddel van trial-and-error te handelen in een potentieel onbekende omgeving, een techniek die haar waarde heeft bewezen in een verscheidenheid aan menselijke leertaken. Mensen leren echter zelden aan de hand van puur neutrale informatie en objectief denken. In tegendeel, mensen maken bijna altijd direct of indirect gebruik van bestaande kennis wanneer een nieuwe taak geleerd wordt. Als reinforcement learning tot doel heeft om menselijk leren te benaderen, dan is hergebruik van kennis essentieel. Transfer Learning is het gebied binnen kunstmatige intelligentie dat zich met deze kwestie bezighoudt. Hoewel Transfer Learning reeds succesvol is in verschillende toepassingen, is het streven naar volledig autonome lerende agenten die deze techniek gebruiken nog een open probleem.

Dit proefschrift draagt op meerdere manieren bij aan een oplossing die het automatisch hergebruiken van kennis mogelijk maakt. Ten eerste worden er twee technieken voorgesteld om automatisch de relatie (inter-task mapping) tussen verschillende reinforcement learning taken te leren. Ten tweede wordt laten zien hoe effectieve en efficinte kennisoverdracht kan worden bereikt in geavanceerde reinforcement learning technieken. Tot slot worden er twee maatstaven gepresenteerd om verschillende leertaken met elkaar te vergelijken, waarmee automatisch een geschikte brontaak gevonden kan worden voor een gegeven doeltaak.

# Publications List

**Publications 2013:**

M. Chami, H. B. Ammar, H. Voss, K. Tuyls, and G. Weiss, "Swarm-based Evaluation of Nonparametric SysML Mechatronics System Design," in *Proceeding of the International Conference on Mehcatronics (ICM)*, Vicenza, Italy, 2013.

S. Chen, H. B. Ammar, K. Tuyls, and G. Weiss, "Optimizing Complex Automated Negotiations using Sparse Pseudo-Input Gaussian Processes", in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Minnesota, USA, 2013.

S. Chen, H. B. Ammar, K. Driessens, K. Tuyls, and G. Weiss, "Automated Transfer Between Negotiation Tasks," in *Proceedings of the Adaptive Learning Agents (ALA) at the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Minnesota, USA, 2013.

S. Chen, H. B. Ammar, K. Tuyls, and G. Weiss, "Using Conditional Restricted Boltzmann Machines for Highly Competitive Negotiation Tasks," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Beijing, China, 2013.

D. C. Mocanu, H. B. Ammar, K. Driessens, D. Lowet, G. Weiss, and K. Tuyls, "Four Way Conditional Restricted Boltzmann Machines," *in review at the European Conference on Machine Learning (ECML)*, Czech Republic, 2013.

H. B. Ammar, D. C. Mocanu, M. Taylor, K. Driessens, G. Weiss, and K. Tuyls, "Automatically Mapped Transfer Between Reinforcement Learning via Three-Way Restricted Boltzmann Machines," *in review at the European Conference on Machine Learning (ECML)*, Czech Republic, 2013.

S. Chen, H. B. Ammar, K. Tuyls, and G. Weiss, "Transfer for Automated Negotiation," in Künstiche Intelligenz (German Artificial Intelligence), 2013.

H. B. Ammar, S. Chen, K. Tuyls, and G. Weiss, "Reinforcement Learning Transfer a Survey and Formal Framework," in Künstiche Intelligenz (German Artificial Intelligence), 2013.

**Publications 2012:**

H. B. Ammar, M. E. Taylor, K. Tuyls, K. Driessens, and G. Weiss, "Reinforcement Learning Transfer via Sparse Coding," *in Proceedings of the eleventh conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Valencia, Spain, 2012.

H. B. Ammar, M. E. Taylor, K. Tuyls, and G. Weiss, "Reinforcement Learning Transfer using a Sparse Coded Inter-Task Mapping," *in Post proceedings of the European Workshop on Multiagent Systems (EUMAS)*, Berlin, Germany, 2012.

H. B. Ammar, M. Kaisers, and K. Tuyls, "Evolutionary Dynamics of Ant Colony Optimization," *in Proceedings of the Multiagent Technology and Systems (MATES)*, Trier, Germany, 2012[1].

H. B. Ammar and M. Tokic, "Reinforcement Learning using a Physical Robot," *in Proceedings of the Teaching Machine Learning Workshop at the International Conference on Machine Learning (ICML)*, Edinburg, Scotland, 2012.

H. B. Ammar and S. D. Jong, "Advocating an Application-Driven Machine Learning Curriculum," *in Proceedings of the Teaching Machine Learning Workshop at the International Conference on Machine Learning (ICML)*, Edinburg, Scotland, 2012.

H. B. Ammar, M. E. Taylor, K. Tuyls, K. Driessens, and G. Weiss, "Reinforcement Learning Transfer Using a Sparse-Coded Intertask Mapping," *Short paper in Proceedings of the Benelux Conference on Artificial Intelligence (BNAIC),* Maastricht, The Netherlands, 2012.

S. Chen, H. B. Ammar, K. Tuyls, and G. Weiss, "Transfer Learning for Bilateral Multi Issue Negotiations," *in Proceedings of the Benelux Conference on Artificial Intelligence (BNAIC)*, Maastricht, The Netherlands, 2012.

M. Chami, H. B. Ammar, H. Voos, K. Tuyls, and G. Weiss, "An Intelligent SysML-based Conceptual Design Evaluation of Mechatronics Systems," *in Proceedings of the Benelux Conference on Artificial Intelligence (BNAIC)*, Maastricht, The Netherlands, 2012.

**Publications 2011:**

H. B. Ammar, M. E. Taylor, K. Tuyls, and G. Weiss, "Common Subspace Transfer for Reinforcement Learning Tasks (B-Paper)," *in Proceedings of the Benelux Conference on Artificial Intelligence (BNAIC)*, Ghent, Belgium, 2011.

H. B. Ammar, and M. E. Taylor, "Common Subspace Transfer for Reinforcement Learning Tasks," *in Proceedings of the AAMAS 2011 Workshop on Adaptive Learning Agents and Multi-agent Systems (ALA), at the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Taipei, Taiwan, 2011.

H. B. Ammar, M. E. Taylor, K. Tuyls, and G. Weiss, "Reinforcement Learning Transfer Using a Sparse Coded Inter-Task Mapping," *in Proceedings of the European Workshop on Multi-agent Systems (EUMAS)*, Maastricht, The Netherlands, 2011.

[1]This paper also appeared in Lecture Notes on Artificial Intelligence (LNAI).

H. B. Ammar and M. E. Taylor, Common Subspace Transfer for Reinforcement Learning Tasks, *in Lecture Notes on Artificial Intelligence (LNAI)* Ed., Berlin: Springer-Verlag, 2011.

**Publications 2010:**

H. Voos and H. B. Ammar, "Nonlinear Tracking and Landing Controller for Quadrotor Aerial Robots," *in Proceedings of the IEEE Multi-Conference on Systems and Control,* Yokohama, Japan, 2010.

H. B. Ammar, H. Voos, and W. Ertel, "Controller Design for Quadrotor UAVs using Reinforcement Learning," *in Proceedings of the IEEE Multi-Conference on Systems and Control,* Yokohama, Japan, 2010.

# SIKS Dissertation Series

**2009**

1 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*

2 Willem Robert van Hage (VU) *Evaluating Ontology-Alignment Techniques*

3 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*

4 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using Collaboration Engineering*

5 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality*

6 Muhammad Subianto (UU) *Understanding Classification*

7 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*

8 Volker Nannen (VU) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*

9 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*

10 Jan Wielemaker (UVA) *Logic programming for knowledge-intensive interactive applications*

11 Alexander Boer (UVA) *Legal Theory, Sources of Law & the Semantic Web*

12 Peter Massuthe (TUE) *Humboldt-Universitaet Berlin*Operating Guidelines for Services

13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*

14 Maksym Korotkiy (VU) *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)*

15 Rinke Hoekstra (UVA) *Ontology Representation - Design Patterns and Ontologies that Make Sense*

16 Fritz Reul (UvT) *New Architectures in Computer Chess*

17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*

18 Fabian Groffen (CWI) *Armada, An Evolving Database System*

19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*

20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*

21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*

22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*

23 Peter Hofgesang (VU) *Modelling Web Usage in a Changing Environment*

24 Annerieke Heuvelink (VU) *Cognitive Models for Training Simulations*

25 Alex van Ballegooij (CWI) *RAM: Array Database Management through Relational Mapping*

26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*

27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*

28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*

**Abbreviations:** SIKS – Dutch Research School for Information and Knowledge Systems; CWI – Centrum voor Wiskunde en Informatica, Amsterdam; EUR – Erasmus Universiteit, Rotterdam; KUB – Katholieke Universiteit Brabant, Tilburg; KUN – Katholieke Universiteit Nijmegen; RUG – Rijksuniversiteit Groningen; RUL – Rijksuniversiteit Leiden; RUN – Radboud Universiteit Nijmegen; TUD – Technische Universiteit Delft; TU/e – Technische Universiteit Eindhoven; UL – Universiteit Leiden; UM – Universiteit Maastricht; UT – Universiteit Twente; UU – Universiteit Utrecht; UvA – Universiteit van Amsterdam; UvT – Universiteit van Tilburg; VU – Vrije Universiteit, Amsterdam.

29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*

30 Marcin Zukowski (CWI) *Balancing vectorized query execution with bandwidth-optimized storage*

31 Sofiya Katrenko (UVA) *A Closer Look at Learning Relations from Text*

32 Rik Farenhorst and Remco de Boer (VU) *Architectural Knowledge Management: Supporting Architects and Auditors*

33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*

34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method Engineering Approach*

35 Wouter Koelewijn (UL) *Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling*

36 Marco Kalz (OUN) *Placement Support for Learners in Learning Networks*

37 Hendrik Drachsler (OUN) *Navigation Support for Learners in Informal Learning Networks*

38 Riina Vuorikari (OU) *Tags and self-organisation: a metadata ecology for learning resources in a multilingual context*

39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin) *Service Substitution – A Behavioral Approach Based on Petri Nets*

40 Stephan Raaijmakers (UvT) *Multinomial Language Learning: Investigations into the Geometry of Language*

41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*

42 Toine Bogers (UvT) *Recommender Systems for Social Bookmarking*

43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients*

44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*

45 Jilles Vreeken (UU) *Making Pattern Mining Useful*

46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*

**2010**

1 Matthijs van Leeuwen (UU) *Patterns that Matter*

2 Ingo Wassink (UT) *Work flows in Life Science*

3 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*

4 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*

5 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*

6 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*

7 Wim Fikkert (UT) *Gesture interaction at a Distance*

8 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*

9 Hugo Kielman (UL) *A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging*

10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*

11 Adriaan Ter Mors (TUD) *The world according to MARP: Multi-Agent Route Planning*

12 Susan van den Braak (UU) *Sensemaking software for crime analysis*

13 Gianluigi Folino (RUN) *High Performance Data Mining using Bio-inspired techniques*

14 Sander van Splunter (VU) *Automated Web Service Reconfiguration*

15 Lianne Bodenstaff (UT) *Managing Dependency Relations in Inter-Organizational Models*

16 Sicco Verwer (TUD) *Efficient Identification of Timed Automata, theory and practice*

17 Spyros Kotoulas (VU) *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications*

18 Charlotte Gerritsen (VU) *Caught in the Act: Investigating Crime by Agent-Based Simulation*

19 Henriette Cramer (UvA) *People's Responses to Autonomous and Adaptive Systems*

20 Ivo Swartjes (UT) *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative*

21 Harold van Heerde (UT) *Privacy-aware data management by means of data degradation*

22 Michiel Hildebrand (CWI) *End-user Support for Access to Heterogeneous Linked Data*

23 Bas Steunebrink (UU) *The Logical Structure of Emotions*

24 Dmytro Tykhonov () *Designing Generic and Efficient Negotiation Strategies*

25 Zulfiqar Ali Memon (VU) *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective*

26 Ying Zhang (CWI) *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines*

27 Marten Voulon (UL) *Automatisch contracteren*

28 Arne Koopman (UU) *Characteristic Relational Patterns*

29 Stratos Idreos (CWI) *Database Cracking: Towards Auto-tuning Database Kernels*

30 Marieke van Erp (UvT) *Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval*

31 Victor de Boer (UVA) *Ontology Enrichment from Heterogeneous Sources on the Web*

32 Marcel Hiel (UvT) *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems*

33 Robin Aly (UT) *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval*

34 Teduh Dirgahayu (UT) *Interaction Design in Service Compositions*

35 Dolf Trieschnigg (UT) *Proof of Concept: Concept-based Biomedical Information Retrieval*

36 Jose Janssen (OU) *Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification*

37 Niels Lohmann (TUE) *Correctness of services and their composition*

38 Dirk Fahland (TUE) *From Scenarios to components*

39 Ghazanfar Farooq Siddiqui (VU) *Integrative modeling of emotions in virtual agents*

40 Mark van Assem (VU) *Converting and Integrating Vocabularies for the Semantic Web*

41 Guillaume Chaslot (UM) *Monte-Carlo Tree Search*

42 Sybren de Kinderen (VU) *Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach*

43 Peter van Kranenburg (UU) *A Computational Approach to Content-Based Retrieval of Folk Song Melodies*

44 Pieter Bellekens (TUE) *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain*

45 Vasilios Andrikopoulos (UvT) *A theory and model for the evolution of software services*

46 Vincent Pijpers (VU) *e3alignment: Exploring Inter-Organizational Business-ICT Alignment*

47 Chen Li (UT) *Mining Process Model Variants: Challenges, Techniques, Examples*

48 Jahn-Takeshi Saito (UM) *Solving difficult game positions*

49 Bouke Huurnink (UVA) *Search in Audiovisual Broadcast Archives*

50 Alia Khairia Amin (CWI) *Understanding and supporting information seeking tasks in multiple sources*

51 Peter-Paul van Maanen (VU) *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention*

52 Edgar Meij (UVA) *Combining Concepts and Language Models for Information Access*

**2011**

1 Botond Cseke (RUN) *Variational Algorithms for Bayesian Inference in Latent Gaussian Models*

2 Nick Tinnemeier (UU) *Work flows in Life Science*

3 Jan Martijn van der Werf (TUE) *Compositional Design and Verification of Component-Based Information Systems*

4 Hado van Hasselt (UU) *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms*

5 Base van der Raadt (VU) *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.*

6 Yiwen Wang (TUE) *Semantically-Enhanced Recommendations in Cultural Heritage*

7 Yujia Cao (UT) *Multimodal Information Presentation for High Load Human Computer Interaction*

8 Nieske Vergunst (UU) *BDI-based Generation of Robust Task-Oriented Dialogues*

9 Tim de Jong (OU) *Contextualised Mobile Media for Learning*

10 Bart Bogaert (UvT) *Cloud Content Contention*

11 Dhaval Vyas (UT) *Designing for Awareness: An Experience-focused HCI Perspective*

12 Carmen Bratosin (TUE) *Grid Architecture for Distributed Process Mining*

13 Xiaoyu Mao (UvT) *Airport under Control. Multiagent Scheduling for Airport Ground Handling*

14 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*

15 Marijn Koolen (UvA) *The Meaning of Structure: the Value of Link Evidence for Information Retrieval*

16 Maarten Schadd (UM) *Selective Search in Games of Different Complexity*

17 Jiyin He (UVA) *Exploring Topic Structure: Coherence, Diversity and Relatedness*

18 Mark Ponsen (UM) *Strategic Decision-Making in complex games*

19 Ellen Rusman (OU) *The Mind's Eye on Personal Profiles*

20 Qing Gu (VU) *Guiding service-oriented software engineering - A view-based approach*

21 Linda Terlouw (TUD) *Modularization and Specification of Service-Oriented Systems*

22 Junte Zhang (UVA) *System Evaluation of Archival Description and Access*

23 Wouter Weerkamp (UVA) *Finding People and their Utterances in Social Media*

24 Herwin van Welbergen (UT) *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior*

25 Syed Waqar ul Qounain Jaffry (VU) *Analysis and Validation of Models for Trust Dynamics*

26 Matthijs Aart Pontier (VU) *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots*

27 Aniel Bhulai (VU) *Dynamic website optimization through autonomous management of design patterns*

28 Rianne Kaptein (UVA) *Effective Focused Retrieval by Exploiting Query Context and Document Structure*

29 Faisal Kamiran (TUE) *Discrimination-aware Classification*

30 Egon van den Broek (UT) *Affective Signal Processing (ASP): Unraveling the mystery of emotions*

31 Ludo Waltman (EUR) *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality*

32 Nees-Jan van Eck (EUR) *Methodological Advances in Bibliometric Mapping of Science*

33 Tom van der Weide (UU) *Arguing to Motivate Decisions*

34 Paolo Turrini (UU) *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations*

35 Maaike Harbers (UU) *Explaining Agent Behavior in Virtual Training*

36 Erik van der Spek (UU) *Experiments in serious game design: a cognitive approach*

37 Adriana Birlutiu (RUN) *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference*

38 Nyree Lemmens (UM) *Bee-inspired Distributed Optimization*

39 Joost Westra (UU) *Organizing Adaptation using Agents in Serious Games*

40 Viktor Clerc (VU) *Architectural Knowledge Management in Global Software Development*

41 Luan Ibraimi (UT) *Cryptographically Enforced Distributed Data Access Control*

42 Michal Sindlar (UU) *Explaining Behavior through Mental State Attribution*

43 Henk van der Schuur (UU) *Process Improvement through Software Operation Knowledge*

44 Boris Reuderink (UT) *Robust Brain-Computer Interfaces*

45 Herman Stehouwer (UvT) *Statistical Language Models for Alternative Sequence Selection*

46 Beibei Hu (TUD) *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work*

47 Azizi Bin Ab Aziz (VU) *Exploring Computational Models for Intelligent Support of Persons with Depression*

48 Mark Ter Maat (UT) *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent*

49 Andreea Niculescu (UT) *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality*

**2012**

1 Terry Kakeeto (UvT) *Relationship Marketing for SMEs in Uganda*

2 Muhammad Umair (VU) *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models*

3 Adam Vanya (VU) *Supporting Architecture Evolution by Mining Software Repositories*

4 Jurriaan Souer (UU) *Development of Content Management System-based Web Applications*

5 Marijn Plomp (UU) *Maturing Interorganisational Information Systems*

6 Wolfgang Reinhardt (OU) *Awareness Support for Knowledge Workers in Research Networks*

7 Rianne van Lambalgen (VU) *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions*

8 Gerben de Vries (UVA) *Kernel Methods for Vessel Trajectories*

9 Ricardo Neisse (UT) *Trust and Privacy Management Support for Context-Aware Service Platforms*

10 David Smits (TUE) *Towards a Generic Distributed Adaptive Hypermedia Environment*

11 J.C.B. Rantham Prabhakara (TUE) *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*

12 Kees van der Sluijs (TUE) *Model Driven Design and Data Integration in Semantic Web Information Systems*

13 Suleman Shahid (UvT) *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions*

14 Evgeny Knutov (TUE) *Generic Adaptation Framework for Unifying Adaptive Web-based Systems*

15 Natalie van der Wal (VU) *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.*

16 Fiemke Both (VU) *Helping people by understanding them - Ambient Agents supporting task execution and depression treatment*

17 Amal Elgammal (UvT) *Towards a Comprehensive Framework for Business Process Compliance*

18 Eltjo Poort (VU) *Improving Solution Architecting Practices*

19 Helen Schonenberg (TUE) *What's Next? Operational Support for Business Process Execution*

20 Ali Bahramisharif (RUN) *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing*

21 Roberto Cornacchia (TUD) *Querying Sparse Matrices for Information Retrieval*

22 Thijs Vis (UvT) *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?*

23 Christian Muehl (UT) *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction*

24 Laurens van der Werff (UT) *Evaluation of Noisy Transcripts for Spoken Document Retrieval*

25 Silja Eckartz (UT) *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application*

26 Emile de Maat (UVA) *Making Sense of Legal Text*

27 Hayrettin Gurkok (UT) *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games*

28 Nancy Pascall (UvT) *Engendering Technology Empowering Women*

29 Almer Tigelaar (UT) *Peer-to-Peer Information Retrieval*

30 Alina Pommeranz (TUD) *Designing Human-Centered Systems for Reflective Decision Making*

31 Emily Bagarukayo (RUN) *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure*

32 Wietske Visser (TUD) *Qualitative multi-criteria preference representation and reasoning*

33 Rory Sie (OUN) *Coalitions in Cooperation Networks (COCOON)*

34 Pavol Jancura (RUN) *Evolutionary analysis in PPI networks and applications*

35 Evert Haasdijk (VU) *Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics*

36 Denis Ssebugwawo (RUN) *Analysis and Evaluation of Collaborative Modeling Processes*

37 Agnes Nakakawa (RUN) *A Collaboration Process for Enterprise Architecture Creation*

38 Selmar Smit (VU) *Parameter Tuning and Scientific Testing in Evolutionary Algorithms*

39 Hassan Fatemi (UT) *Risk-aware design of value and coordination networks*

40 Agus Gunawan (UvT) *Information Access for SMEs in Indonesia*

41 Sebastian Kelle (OU) *Game Design Patterns for Learning*

42 Dominique Verpoorten (OU) *Reflection Amplifiers in self-regulated Learning*

43 (Withdrawn)

44 Anna Tordai (VU) *On Combining Alignment Techniques*

45 Benedikt Kratz (UvT) *A Model and Language for Business-aware Transactions*

46 Simon Carter (UVA) *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation*

47 Manos Tsagkias (UVA) *A Model and Language for Business-aware Transactions*

48 Jorn Bakker (TUE) *Handling Abrupt Changes in Evolving Time-series Data*

49 Michael Kaisers (UM) *Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions*

50 Steven van Kervel (TUD) *Ontologogy driven Enterprise Information Systems Engineering*

51 Jeroen de Jong (TUD) *Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching*

**2013**

1 Viorel Milea (EUR) *News Analytics for Financial Decision Support*

2 Erietta Liarou (CWI) *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing*

3 Szymon Klarman (VU) *Reasoning with Contexts in Description Logics*

4 Chetan Yadati (TUD) *Coordinating autonomous planning and scheduling*

5 Dulce Pumareja (UT) *Groupware Requirements Evolutions Patterns*

6 Romulo Gonzalves (CWI) *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience*

7 Giel van Lankveld (UT) *Quantifying Individual Player Differences*

8 Robbert-Jan Merk (VU) *Making Enemies: Cognitive Modeling for Opponent Agents in Fighter Pilot Simulators*

9 Fabio Gori (RUN) *Metagenomic Data Analysis: Computational Methods and Applications*

10 Jeewanie Jayasinghe Arachchige (UvT) *A Unified Modeling Framework for Service Design*

11 Evangelos Pournaras (TUD) *Multi-level Reconfigurable Self-organization in Overlay Services*

12 Maryam Razavian (VU) *Knowledge-driven Migration to Services*

13 Mohammad Zafiri (UT) *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly*

14 Jafar Tanha (UVA) *Ensemble Approaches to Semi-Supervised Learning Learning*

15 Daniel Hennes (UM) *Multiagent Learning - Dynamic Games and Applications*

16 Eric Kok (UU) *Exploring the practical benefits of argumentation in multi-agent deliberation*

17 Koen Kok (VU) *The PowerMatcher: Smart Coordination for the Smart Electricity Grid*

18 Jeroen Janssens (UvT) *Outlier Selection and One-Class Classification*

19 Renze Steenhuisen (TUD) *Coordinated Multi-Agent Planning and Scheduling*

20 Katja Hofmann (UVA) *Fast and Reliable Online Learning to Rank for Information Retrieval*

21 Sander Wubben (UvT) *Text-to-text generation by monolingual machine translation*

22 Tom Claassen (RUN) *Causal Discovery and Logic*

23 Patrcio de Alencar Silva (UvT) *Value Activity Monitoring*

24 Haitham Bou Ammar (UM) *Automated Transfer in Reinforcement Learning*

# Bibliography


[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 1–, New York, NY, USA, 2004. ACM.

[2] H. Ackley, E. Hinton, and J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, pages 147–169, 1985.

[3] Alessandro Lazaric and Marcello Restelli. Transfer from multiple MDPs. In *Proceedings of the Twenty-Fifth Annual Conference on Neural Information Processing Systems (NIPS'11)*, 2011.

[4] Haitham Bou Ammar, Decebal Constantin Mocanu, Matthew Taylor, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. Automatically mapped transfer between reinforcement learning tasks via three-way restricted boltzmann machines. In *Proceedings of the European Conference on Machine Learning (ECML)-In review*, Prague, Czech Republic, 2013.

[5] Haitham Bou Ammar and Mathew E. Taylor. Common subspace transfer for reinforcement learning tasks. In *Proceedings of the AAMAS 2011 Workshop on Adaptive Learning Agents and Multi-agent Systems (ALA 2011)*, Taipei, Taiwan, May 2011.

[6] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483, May 2009.

[7] Mehran Asadi and Manfred Huber. Effective control knowledge transfer through learning skill and representation hierarchies. In *Proceedings of the 20 th International Joint Conference on Artificial Intelligence*, pages 2054–2059, 2007.

[8] Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 672–677, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[9] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems 14*, pages 585–591. MIT Press, 2001.

[10] Samy Bengio, Fernando Pereira, Yoram Singer, and Dennis Strelow. Group sparse coding. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 82–89. 2009.

[11] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.

[12] Daniel S. Bernstein. Reusing old policies to accelerate learning on new mdps. Technical report, 1999.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[14] Blind. In *Under submission — if not accepted by ICML-13, we will create and cite a tech report.*

[15] Andrea Bonarini, Ro Lazaric, and Marcello Restelli. Incremental skill acquisition for self-motivated learning animats. In *Lecture Notes in Computer Science*, page 2006.

[16] Haitham Bou-Ammar, Matthew Taylor, Karl Tuyls, Kurt Driessens, and Gerhard Weiss. Reinforcement learning transfer via sparse coding. In *Proceedings of the eleventh conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Valencia, Spain, 2012.

[17] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators.* CRC Press, Boca Raton, Florida, 2010.

[18] Miguel A. Carreira-Perpinan and Geoffrey E. Hinton. On contrastive divergence learning.

[19] Miguel A. Carreira-Perpinan and Geoffrey E. Hinton. On contrastive divergence learning, 2005.

[20] Pablo S. Castro and Doina Precup. Using bisimulation for policy transfer in mdps. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 1399–1400, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[21] Özgür Şimşek, Alicia P. Wolfe, and Andrew G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 816–823, New York, NY, USA, 2005. ACM.

[22] Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.

[23] Finale Doshi-Velez and Zoubin Ghahramani. A Comparison of Human and Agent Reinforcement Learning in Partially Observable Domains. 2011.

[24] Chris Drummond. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *CoRR*, abs/1106.1796, 2011.

[25] Emanuel Falkenauer. On method overfitting. *Journal of Heuristics*, 4, 1998.

[26] Kimberly Ferguson and Sridhar Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. In *ICML Workshop on Transfer Learning*, 2006.

[27] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In David Maxwell Chickering and Joseph Y. Halpern, editors, *UAI*, pages 162–169. AUAI Press, 2004.

[28] Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM J. Comput.*, 40(6):1662–1714, 2011.

[29] Eliseo Ferrante, Alessandro Lazaric, and Marcello Restelli. Transfer of task representation in reinforcement learning using policy-based proto-value functions ? (short paper).

[30] Asia Fischer and Christian Igel. Bounding the bias of contrastive divergence learning. *Neural Comput.*, 23(3):664–673, March 2011.

[31] David Foster and Peter Dayan. Structure in the space of value functions. *Mach. Learn.*, 49(2-3):325–346, November 2002.

[32] Thomas Furmston and David Barber. Lagrange dual decomposition for finite horizon markov decision processes. In *ECML/PKDD (1)*, pages 487–502, 2011.

[33] Judy Goldsmith, Chris Lusena, and Martin Mundhenk. The complexity of deterministically observable finite-horizon markov decision processes. Technical report, 1996.

[34] Bernhard Hengst. Discovering hierarchy in reinforcement learning with hexq. In *In Maching Learning: Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250. Morgan Kaufmann, 2002.

[35] Geoffrey Hinton. A Practical Guide to Training Restricted Boltzmann Machines. Technical report, 2010.

[36] Geoffrey E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800, August 2002.

[37] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[38] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013.

[39] Seung jean Kim, Kwangmoo Koh, Michael Lustig, Stephen Boyd, and Dimitry Gorinevsky. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine Learning Research*, 2007, 2007.

[40] Zsolt Kalmr, Csaba Szepesvri, and Konkoly Thege M. U. An evaluation criterion for macro learning and some results.

[41] W. Bradley Knox, Brian D. Glass, Bradley C. Love, W. Todd Maddox, and Peter Stone. How humans teach agents: A new experimental perspective. *International Journal of Social Robotics*, 4:409–421, 2012.

[42] W. Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *The Fifth International Conference on Knowledge Capture*, September 2009.

[43] W.B̃radley Knox and Peter Stone. Augmenting reinforcement learning with human feedback. In *ICML 2011 Workshop on New Developments in Imitation Learning*, July 2011.

[44] W. Bradley Knox and Peter Stone. Reinforcement learning from human reward: Discounting in episodic tasks. In *In Proceedings of the 21st IEEE International Symposium on Robot and Human Interactive Communication (Ro-Man)*, September 2012.

[45] W. Bradley Knox and Peter Stone. Reinforcement learning from simultaneous human and MDP reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, June 2012.

[46] Jens Kober and Jan Peters. Policy search for motor primitives in robotics. *Mach. Learn.*, 84(1-2):171–203, July 2011.

[47] George Konidaris and Andrew Barto. Autonomous shaping: knowledge transfer in reinforcement learning. In *In Proceedings of the 23rd Internation Conference on Machine Learning*, pages 489–496, 2006.

[48] George Konidaris and Andrew Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 895–900, 2007.

[49] Yehuda Koren and Robert M. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. 2011.

[50] H. S. Kuyuk, E. Yildirim, E. Dogan, and G. Horasan. Application of k-means and gaussian mixture model for classification of seismic activities in istanbul. *Nonlinear Processes in Geophysics*, 19(4):411–419, 2012.

[51] Michail G. Lagoudakis, Ronald Parr, and L. Bartlett. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:2003, 2003.

[52] Michail G. Lagoudakis, Ronald Parr, and Michael L. Littman. Least-squares methods in reinforcement learning for control. In *In SETN 02: Proceedings of the Second Hellenic Conference on AI*, pages 249–260. Springer-Verlag, 2002.

[53] Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning*. PhD thesis, Champaign, IL, USA, 2004. AAI3130966.

[54] Alessandro Lazaric. Transfer in Reinforcement Learning: a Framework and a Survey. In Martijn van Otterlo Marco Wiering, editor, *Reinforcement Learning - State of the art*, volume 12, pages 143–173. Springer, 2012.

[55] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *In: Proceedings of the 25th Annual ICML*, pages 544–551, 2008.

[56] Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012.

[57] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *In NIPS*, pages 801–808. NIPS, 2007.

[58] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *In NIPS*, pages 801–808. NIPS, 2007.

[59] Honglak Lee, Rajat Raina, Alex Teichman, and Andrew Y. Ng. Exponential family sparse coding with application to self-taught learning. In *IJCAI*, pages 1113–1119, 2009.

[60] Offer Lieberman, Roy Rosemarin, and Judith Rousseau. Asymptotic theory for maximum likelihood estimation of the memory parameter in stationary gaussian processes. *Econometric Theory*, 28(02):457–470, 2012.

[61] Richard Maclin, Jude Shavlik, Trevor Walker, and Lisa Torrey. A simple and effective method for incorporating advice into kernel methods. In *In AAAI*, 2006.

[62] Sridhar Mahadevan. Proto-value functions: developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 553–560, New York, NY, USA, 2005. ACM.

[63] Sridhar Mahadevan, Mauro Maggioni, and Carlos Guestrin. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8:2007, 2006.

[64] Timothy Arthur Mann and Yoonsuck Choe. Scaling up reinforcement learning through targeted exploration. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

[65] Amy Mcgovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *In Proceedings of the eighteenth international conference on machine learning*, pages 361–368. Morgan Kaufmann, 2001.

[66] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning*, ECML '02, pages 295–306, London, UK, UK, 2002. Springer-Verlag.

[67] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[68] Decebal Mocanu, Haitham Bou Ammar, Deitwig Dowel, Kurt Dreissens, Gerhard Weiss, and Karl Tuyls. Four way conditional restricted boltzmann machines. In *ECML/PKDD (under review)*, 2013.

[69] Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *J. Mach. Learn. Res.*, 9:815–857, June 2008.

[70] Peter Mller and David Rios Insua. Issues in bayesian analysis of neural network models, 1998.

[71] Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47:79–91, 2004.

[72] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *International Conference on Machine Learning*, 2004.

[73] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.

[74] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Inverted autonomous helicopter flight via reinforcement learning. In *In International Symposium on Experimental Robotics*. MIT Press, 2004.

[75] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, August 2000.

[76] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, October 2010.

[77] Jan Peters. Machine learning for motor skills in robotics. *KI*, 22(4):41–43, 2008.

[78] Jan Peters and Stefan Schaal. Natural actor-critic. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pages 280–291. springer, 2005.

[79] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomput.*, 71(7-9):1180–1190, March 2008.

[80] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[81] Joaquin Quionero-candela, Carl Edward Rasmussen, and Ralf Herbrich. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:2005, 2005.

[82] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the Twenty-fourth International Conference on Machine Learning*, 2007.

[83] Roger Grosse Rajat Raina, Helen Kwong, and Andrew Y. Ng. Shift-invariant sparse coding for audio classification. In *UAI*.

[84] Carl Edward Rasmussen. The infinite gaussian mixture model. In *In Advances in Neural Information Processing Systems 12*, pages 554–560. MIT Press, 2000.

[85] Carl Edward Rasmussen. In *Gaussian processes for machine learning*. MIT Press, 2006.

[86] Carl Edward Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.

[87] Balaraman Ravindran and Andrew G. Barto. Relativized options: Choosing the right transformation. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 608–615. AAAI Press, 2003.

[88] Martin Riedmiller. Neural fitted q iteration first experiences with a data efficient neural reinforcement learning method. In *In 16th European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[89] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. Technical report, 2007.

[90] Alexander A. Sherstov and Peter Stone. Improving action selection in mdp's via knowledge transfer. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 2*, AAAI'05, pages 1024–1029. AAAI Press, 2005.

[91] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 194–281. MIT Press, Cambridge, 1987.

[92] Edward Snelson. Local and global sparse gaussian process approximations. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, 2007.

[93] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264. MIT press, 2006.

[94] Vishal Soni and Satinder Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pages 494–499. AAAI Press, 2006.

[95] Ashok Narain Srivastava and Jiawei Han, editors. *Machine learning and knowledge discovery for engineering systems health management.* Chapman & Hall/CRC data mining and knowledge discovery series. CRC Press, Boca Raton, FL, 2012.

[96] Peter Stone, Gregory Kuhlmann, Matthew E. Taylor, and Yaxin Liu. Keepaway soccer: From machine learning testbed to benchmark. In Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld, and Yasutake Takahashi, editors, *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020, pages 93–105. Springer-Verlag, Berlin, 2006.

[97] Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[99] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction, 1998.

[100] Richard S. Sutton, Anna Koop, and David Silver. On the role of tracking in stationary environments. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 871–878, New York, NY, USA, 2007. ACM.

[101] Csaba Szepesvári. *Algorithms for Reinforcement Learning.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

[102] Graham W. Taylor, Geoffrey E. Hinton, and Sam T. Roweis. Two distributed-state models for generating high-dimensional time series. *Journal of Machine Learning Research*, 12:1025–1068, 2011.

[103] Matthew E. Taylor, Nicholas K. Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 488–505, September 2008.

[104] Matthew E. Taylor, Gregory Kuhlmann, and Peter Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 283–290, May 2008.

[105] Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 53–59, July 2005.

[106] Matthew E. Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML)*, June 2007.

[107] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, December 2009.

[108] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via intertask mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[109] Matthew E. Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2011.

[110] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, pages 640–646. The MIT Press, 1996.

[111] Lisa Torrey, Jude W. Shavlik, Trevor Walker, and Richard Maclin. Skill acquisition via transfer learning and advice taking. In *ECML*, pages 425–436, 2006.

[112] Lisa Torrey, Jude W. Shavlik, Trevor Walker, and Richard Maclin. Transfer learning via advice taking. In *Advances in Machine Learning I*, pages 147–170. 2010.

[113] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *in Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000*, pages 1079–1086, 2000.

[114] Thomas J. Walsh, Lihong Li, and Michael L. Littman. Transferring state abstractions between mdps. In *In ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[115] Jun Zhang. A note on the tau-method approximations for the bessel functions. *Computers Math. Applications*, 30:5–14, 1996.