Deep Reinforcement Learning for POMDPs

Master-Thesis von Hong Linh Thai aus Darmstadt
Januar 2018

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Deep Reinforcement Learning for POMDPs

Vorgelegte Master-Thesis von Hong Linh Thai aus Darmstadt

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Joni Pajarinen

Tag der Einreichung:

Erklärung zur Abschlussarbeit gemäß §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Hong Linh Thai, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum / Date: Unterschrift / Signature:

_____ _____

Abstract

Deep reinforcement learning has made a big impact in recent years by achieving human level game play in the Atari 2600 games just using images as input and by learning robot control policies end-to-end from images to motor signals - tasks, which were previously intractable for classical reinforcement learning. For learning these control policies based on deep neural networks policy search methods, such as deep deterministic policy gradient (Lillicrap et al., 2015) and especially trust region policy optimization (Schulman et al., 2015), have been very successful, also in transferring insights and past approaches of classical reinforcement learning to deep neural networks. In this thesis we build up this previous work and derive a new algorithm for deep neural networks, called compatible policy search (COPS), based on the idea of the natural gradient, compatible value function approximation, entropy regularization and relative entropy policy search (Peters et al., 2010). In our experiment we investigated the capability of COPS and other policy search methods in challenging partially observable environments: RockSample, where the agent needs to take information gathering into account and Pocman, a large scale partially observable Markov decision processes (POMDP) with ca. 10^{56} underlying states. We present results where COPS outperforms all other six policy search methods and where the additional entropy regularization constraint, bounding the entropy of the new policy, is essential for exploration and for finding a good policy in these partially observable environments. Furthermore, to encourage additional exploration in these partially observable environments we will propose a factored context tree switching model for POMDPs, which we use for a pseudocount based exploration bonus and leads to additional performance gains.

Zusammenfassung

In den letzten Jahren hat Deep Reinforcement Learning große Wellen geschlagen. Es hat zum Beispiel nur auf Basis der Bildschirmausgabe die Atari 2600 Konsolenspiele gemeistert und wurde erfolgreich zum Lernen einer künstlichen Intelligenz, die End-zu-End von Kamerabildern die Steuersignale eines Roboters lernen soll, eingesetzt - Aufgaben, die für klassisches Reinforcement Learning bis jetzt unlösbar waren. Für das Lernen dieser künstlichen Intelligenzen zur Steuerung der Roboter oder Agenten mit Hilfe tiefer neuronaler Netze haben sich Policy Search Methoden wie „Deep Deterministic Policy Gradient“ (Lillicrap et al., 2015) und besonders „Trust Region Policy Optimization“ (Schulman et al., 2015) als erfolgreich erwiesen und waren in der Lage, vergangene Erkenntnisse und Ansätze aus dem Bereich des klassischen Reinforcement Learning auf tiefe neuronale Netze zu transferieren. In dieser Arbeit schließen wir an diese Methoden an und leiten einen neuen Policy Search Algorithmus für tiefe neuronale Netze her, genannt „Compatible Policy Search“ (COPS), basierend auf der Idee von „Natural Gradient“, „Compatible Value Function Approximation“, „Entropy Regularization“ und „Relative Entropy Policy Search“ (Peters et al., 2010). In Experimenten untersuchten wir die Anwendungsmöglichkeiten dieses neuen Algorithmus und anderer Policy Search Methoden für herausfordernde partiell observierbare Umgebungen: Für die „RockSample“ Umgebung, die das Sammeln von Informationen über den wahren Zustand der Welt erfordert und für die „Pocman“ Umgebung, welche ein riesiger „Partially Observable Markov Decision Process“ (POMDP) mit ca. 10^{56} Weltzuständen ist. Wir zeigen Ergebnisse, in denen COPS alle anderen sechs Policy Search Algorithmen übertrifft und in welchen die Nebenbedingung zur Begrenzung der Entropie der neuen Policy essentiell zur Exploration und zum Lernen einer guten Policy in diesen partiell observierbaren Umgebungen ist. Des Weiteren, um zusätzliche Exploration in diesen partiell observierbaren Umgebungen zu fördern, stellen wir in dieser Arbeit ein Modell, das auf dem „Context Tree Switching“ Algorithmus basiert, für POMDPs vor. Dieses Modell kann verwendet werden, um einen Explorationsbonus zu berechnen, basierend auf der Idee von einem „Pseudocount“ und führt zu zusätzlichen Leistungssteigerung.

Acknowledgments

I would like to thank Prof. Gerhard Neumann and Dr. Joni Parjarinen for the interesting topic, their patient supervision and for our countless discussions and meetings, that have been very informative for me. I am grateful for all their ideas and input that went into this thesis and are the base of this work. I am also much obliged to Prof. Jan Peters and Prof. Gerhard Neumann, who gave me both the opportunity to work in their excellent research groups 'Intelligent Autonomous Systems' and 'Computational Learning and Autonomous Systems'. I really appreciate the time spent in the autonomous system labs with their inspiring and motivating research atmosphere.

A big thanks goes to my parents for their support, not only throughout the thesis but for all their support and love through my whole life. Finally, I sincerely thank Maria for her love, encouragement and the emotional support whenever I needed it.

Contents

Figures and Tables

List of Figures

List of Tables

Abbreviations, Symbols and Operators

The following abbreviations, symbols and operators are used throughout this thesis:

List of Abbreviations

Notation	Description
CEM	Cross entropy method [12]
CMA-ES	Covariance matrix adaption evolution strategy [21]
CNN	Convolutional neural network
COPS	Compatible policy search
CTS	Context tree switching [80]
DDPG	Deep deterministic policy gradient [66, 39]
DQN	Deep Q-Network [44, 45]
FVRS	Field Vision Rock Sample [56]
KL	Kullback-Leibler divergence
MDP	Markov decision process
MORE	Model-based relative entropy stochastic search [2]
MOTO	Model-free trajectory optimization for reinforcement learning [3]
POMCP	Partially observable Monte-Carlo planning algorithm [67]
POMDP	Partially observable Markov decision process
REPS	Relative entropy policy search [52]
RL	Reinforcement learning
SARSA	State-action-state-reward-action algorithm [59]
TD	Temporal difference
TNPG	Truncated natural policy gradient [17, 63]
TRPO	Trust region policy optimization[63, 62]

List of Operators

Notation	Description	Operator
∇_{θ}	Derivative with respect to paramterer $\boldsymbol{\theta}$	$\nabla_{\boldsymbol{\theta}}(\bullet)$
H	Entropy of a probability distributions	$H(\bullet)$
J	Expected long term reward	$J(\bullet)$
KL	Kullback-Leibler divergence between two probability distributions	$KL(\bullet \mid\mid \bullet)$
π	Policy, probability distribution	$\pi(\bullet)$
T	Transpose	$(\bullet)^T$

1 Introduction

In today's world, artificial intelligence and autonomous intelligent systems have increasingly gained more importance and interest in various fields due to recent breakthroughs. For example in the domain of board games Go was previously believed to be the borderline for AI due to its enormous count of possible game states. But in 2016 DeepMind has made a big impact with their AI AlphaGo [65] by defeating Lee Sedol, winner of 18 world titles, in the board game Go four to one. In the domain of computer games DeepMind has also made a big splash in 2015 by training an AI, that achieved human level gameplay in the Atari 2600 games just using image pixels as input, known as deep Q-Network [44, 45] (DQN). Furthermore, in 2017 OpenAI trained a AI, which defeated the world's top professional player in a special 1 vs 1 gamemode of the computer game Dota 2, which is generally played 5 vs 5. But also in other real world domains there has been huge advances, for example the automotive industry. Autonomous driving and autonomous cars have reached nowadays a state where the technology is ready to be deployed on real streets for testing purposes. Moreover, robots are not only used for industrial applications, but are slowly coming to our daily life in the form of autonomous vacuum cleaners, autonomous mowing machines and toys.

For many of these stated applications we do not want to pre-programm the behavior of the robot / agent for every possible situation, since it would be too complex to consider every possible case. Instead we want the artificial intelligence to be able to learn the optimal behavior for a given task through trial and error and experiences of their interactions with the environment, known as reinforcement learning. However, to apply reinforcement learning on these stated kind of problems it is usually combined with powerful function approximators and representation learning, provided through deep neural networks, in order to be able to process high dimensional input, such as the game state of Go or images. This step is needed because classical reinforcement learning methods are intractable to high dimensions. The idea of using neural networks for reinforcement learning is not completely new and has been for example already successfully applied to Backgammon with TD-Gammon [76] in beginning of the 90s. But after that there was a long moment of silence before the idea re-emerged recently with the success of deep learning under the name deep reinforcement learning. The reason for this period of silence is that training of these networks is difficult due to memory, computational and sample complexity. But this problem could be overcome by breakthroughs in deep learning, such as using more simple activation functions, more available computation power and the usage of convolutional layers combined with the idea of weight sharing. However, one problem that remains is that learning the network parameters using deep reinforcement learning can be highly unstable and lead to premature convergence or catastrophic failure. To address this problem there have been several prominent ideas such as the application of a replay memory, weight freezing and target networks in DQN and deep deterministic policy gradient [66, 39] (DDPG) or the application of the Kullback-Leibler divergence to bound the difference between the results of the new and previous update step in trust region policy optimization [63, 62] (TRPO). There has been also successes in robotics, where control policy were learned end-to-end directly from camera inputs to robot movements such in [38, 20, 86].

Another problem we face in these real world applications is partial observability. In the reinforcement learning framework it is usually assumed that the agent is in a fully observable environment, where the agent knows the true state of the environment and hence can act optimal by just reacting to the current state of the environment. This holds for the current common industrial robot applications, where the robot works in a controlled environment. However, this assumption is usually violated in the real world. For example if we have a household robot in a real world setting, the robot will perceive the state of the real world only through sensors, which show only parts of the environment and are usually noisy. Hence, for decision making the robot also needs to take missing information into account and needs to gather actively information about the uncertain parts to be more sure about the true state of the world. This idea is captured in the partially observable Markov decision process (POMDP) framework, which is an extension of the Markov decision process and allows to model a sequential decision process for an agent in an uncertain environment. Most of the deep reinforcement learning approaches have been only applied to Markov decision processes, while their partially observable counterpart has been under-explored. For example after the success of DQN there have been several approaches applying recurrent neural networks and LSTMs for POMDPs. However, the partially observable problems, they have been applied to, have been mainly focusing on remembering some important observations, e.g. the goal position that is only given at the start of an interaction chain or the last position of a ball, which is needed to compute the velocity given the current position of the ball [22, 24]. But, there is also another class of partially observable environments where the agent needs to take information gathering into account and hence needs much more exploration and interaction with these environments to learn how to gather information and how to use them.

1.1 Contribution

In this work, we address the problems of unstable learning updates and exploration in POMDPs by proposing a new deep reinforcement learning algorithm in Chapter 3, which uses the idea of the Kullback-Leibler divergence and entropy regularization to ensure exploration while preventing catastrophic failure and premature convergence. These ideas have been already explored for classical reinforcement learning in relative entropy policy search and many of its variants [52, 3]. However, not so much for deep reinforcement learning, yet. Hence, we show in this work how these ideas can be transferred and implemented for deep neural networks and discuss connections to other related algorithms, such as TRPO. We derive a new algorithm based on the natural gradient and compatible value function approximation, named compatible policy search, with close form policy update equations for neural networks. In our experiments we investigate the performance of related policy search methods and our new algorithm on two difficult POMDP environments: the RockSample environment, where the agent needs to take information gathering into account and the Pocman environment, a large scale POMDP with ca. 10^{56} underlying states. We further address the problem of exploration in POMDP environments by proposing in Chapter 4 how a pseudocount exploration bonus with factored CTS models could be applied to POMDP histories.

1.2 Outline

In this section we provide an overview and outline of this thesis' structure. In Chapter 2, we first give an introduction to reinforcement learning and POMDP and show the common approaches to solve them. Afterwards, we give an introduction to deep learning, show how it is applied to the reinforcement learning framework and give an brief overview over prominent deep reinforcement learning algorithms: DQN, DDPG and TRPO. In Chapter 3, we introduce our new algorithm compatible policy search, a general framework to solve trust region optimization problems with an additional entropy regularization constraint in close form using compatible value function approximation and show how it is derived. We compare this new method with other state of the art approaches and show results on two difficult POMDP problems. We will also investigate different neural network structures for POMDPs. In Chapter 4, we introduce for additional exploration a factored context tree switching model POMDPs, which can be used for a pseudocount based exploration bonus. In Chapter 5, we summarize our results and present future work.

2 Background

In this chapter we will first give a brief introduction to reinforcement learning, based on the current literature. We will describe the general idea of reinforcement learning, the mathematical framework behind it and show how a reinforcement learning problem is modeled formally as a Markov decision process. We will also discuss their counterpart for partially observable environments, known as POMDP, and explain their differences. For both cases we will present the challenges and the main reinforcement learning approaches. For Markov decision processes we will focus on model-free approaches, while for POMDPs we will present mainly model-based approaches. Thereby we will use definitions, formulations and structure our chapter based on [73, 60, 15, 5, 27, 64]. In the second part of this chapter we will move on to deep learning and deep reinforcement learning. We will first briefly explain the history of deep learning and the most used neural network structures in deep learning: feedforward neural networks and convolutional neural networks. Then we will explain how deep learning is applied to reinforcement learning. Here, we will show recent successful algorithms: DQN, DDPG and TRPO, focusing on the techniques the algorithms used to stabilize their learning.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a general mathematical framework to specify learning problems, based on the idea that an individual learns to solve a given task by interacting with its environment [73]. This individual is usually called an agent, which interacts with its environment by executing actions sequentially and learns by altering its behavior based on the observed consequences of the executed actions. Historically RL has two foundations: trial and error learning from behavioral psychology starting with the work of Edward Thorndike in the late 1800s and the problem of optimal control starting with the work of Richard Bellman in the late 1950s. The solution of Bellman known as dynamic programming and later works such as the policy iteration algorithm gave birth to various essential mathematical formulations that are still used in RL today, for example Markov decision processes, which we discuss in more detail, shortly.
In the typical RL setup we have an *agent* that interacts with its unknown *environment* (everything not under control of the agent) at any time step t by choosing an action $\mathbf{a}_t$ based on the current state $\mathbf{s}_t$ of the environment. The environment will change its state based on the chosen action $\mathbf{a}_t$ and present a new situation $\mathbf{s}_{t+1}$ to which the agent needs to react again. Furthermore, based on the chosen action the environment will output a numerical *reward* signal r_t, that the agent tries to maximize over time by self-improvement. Figure 2.1 illustrates this interaction between the agent and the environment.

2.1.1 Markov Decision Process (MDP)

Formally, a RL problem can be defined as a Markov decision process (MDP), which includes [61]:

- the state space S containing all possible states $\mathbf{s} \in S$ describing the environment
- the action space A containing all valid actions $\mathbf{a} \in A$ that can be performed by the agent

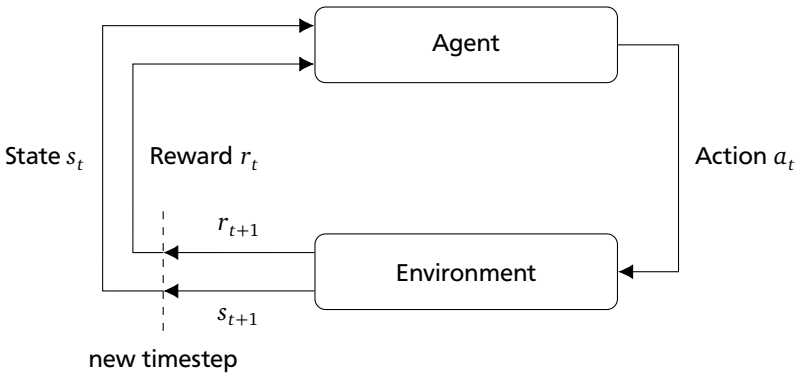


Figure 2.1.: Typical RL interaction loop between agent and environment (recreated from [73])

- the transition model $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ specifying the probability that the environment will be in new state $\mathbf{s}_{t+1}$ after action $\mathbf{a}_t$ has been performed in the current state $\mathbf{s}_t$
- the reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ specifying the immediate reward that is received for performing action $\mathbf{a}_t$ in state $\mathbf{s}_t$
- the initial state distribution $\mu_0(\mathbf{s})$ specifying the probability of state $\mathbf{s}$ being the initial state of the decision process

Furthermore, the transition model of an MDP must always satisfy the *Markov property* $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_t, \mathbf{a}_t) = \mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, i.e., the next state $\mathbf{s}_{t+1}$ depends exclusively on the current state $\mathbf{s}_t$ and the performed action $\mathbf{a}_t$. How the current state $\mathbf{s}_t$ was reached, meaning any previous action or state, does not have any impact on the transition probability.

Depending on the textbook / paper the discount factor γ and the horizon H can be also part of the MDP, while in other definitions they are regarded as additional parameters [5, 63, 17]. The discount factor $\gamma \in [0, 1]$ is responsible how much emphasis the agent puts on immediate rewards compared to long-time rewards. The horizon H describes the maximum number of time steps in which an agent has to perform the task. If the time step H is reached, the environment will reset its current state to an initial state again and the total interaction chain ends. This total interaction chain is usually referred as *episode*. Note that it is possible to choose $H = \infty$. In this case the MDP is called *non-episodic*; we will only use episodic MDPs in this thesis.

2.1.2 Policy

To represent the agent's behavior, a policy π is used to define the agent's action $\mathbf{a}$ given some input, usually the current world state $\mathbf{s}$. The policy can be either a *deterministic* function $\pi(\mathbf{s})$ that maps a unique action $\mathbf{a} = \pi(\mathbf{s})$ to a given state $\mathbf{s}$ or *stochastic* defined as conditional $\pi(\mathbf{a}|\mathbf{s})$, which defines how likely action $\mathbf{a}$ is taken given state $\mathbf{s}$. For the optimal behavior it is sufficient to specify it in a deterministic policy, but when we are learning the optimal behavior from interaction with an unknown environment we will need stochastic polices to encode different decisions. For example if we are still uncertain about the best action for a given state $\mathbf{s}$, we would like to have a policy, which has the possibility of taking either action 1 or action 2 in $\mathbf{s}$, such that it can learn from its experience if action 1 or action 2 is better in state $\mathbf{s}$. This is known as exploration. For this reason stochastic policies are usually used for exploration by starting with a random policy and slowly reducing the stochasticity during the learning process [15].

In this thesis we will use *parameterized stochastic policies* $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$, where $\boldsymbol{\theta}$ is a parameter vector that specifies the policy. For example, if we have a Gaussian policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mathbf{a}|\mu(\mathbf{s}), \sigma^2)$, where the mean $\mu(\mathbf{s})$ is a linear combination of the state features $\phi(\mathbf{s})$, i.e $\mu(\mathbf{s}) = \phi(\mathbf{s})^T\mathbf{w}$. Then $\boldsymbol{\theta}$ corresponds to the parameters of the mean $\mathbf{w}$ and the variance σ^2 of the Gaussian distribution. Another common policy representation are neural networks, which will be introduced later in Section 2.3.1. In this case $\boldsymbol{\theta}$ corresponds to the concatenated flattened vector of all weights and biases of the network.

2.1.3 Optimal Decision Making

The goal of RL is to find a policy $\pi(\mathbf{a}|\mathbf{s})$ that maximizes the expected reward over time $J(\pi)$. This expected reward over time is called *return* and defined as following:

$$J(\pi) = \mathbb{E}_{\mu_0(\mathbf{s}_0), \mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), \pi(\mathbf{a}_t|\mathbf{s}_t)} \left[\sum_{t=0}^{H} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right]. \tag{2.1}$$

Note that in a non-episodic MDP ($H = \infty$) the choice $\gamma = 1$ is invalid, because this would lead to an infinite sum [73]. But since we only use the episodic setting in this thesis, we will reformulate (2.1) to a slightly more handy representation. For this we use the notation of *trajectories*, also called *rollouts* or *paths*, following [15]

$$\boldsymbol{\tau} = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T) \tag{2.2}$$

which is a series of interactions between the environment and agent containing all taken actions, observed states and rewards of one episode. Note, that depending on the paper, the reward is not always part of a trajectory [15, 5]. Independent from this choice, we can define the total accumulated reward $R(\boldsymbol{\tau})$ for one trajectory as:

$$R(\boldsymbol{\tau}) = \left[\sum_{t=0}^{H} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right] = \left[\sum_{t=0}^{H} \gamma^t r_t \right]. \tag{2.3}$$

Using this definition we can rewrite (2.1) to:

$$J(\pi) = \mathbb{E}_{\mathcal{P}_\pi(\tau)}[R(\boldsymbol{\tau})] = \int \mathcal{P}_\pi(\boldsymbol{\tau})\, R(\boldsymbol{\tau})\, d\boldsymbol{\tau} \tag{2.4}$$

where $\mathcal{P}_\pi(\boldsymbol{\tau})$ is the distribution over trajectories for the policy π:

$$\mathcal{P}_\pi(\boldsymbol{\tau}) = \mu_0(\mathbf{s}_0) \prod_{t=0}^{T-1} \mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\, \pi(\mathbf{a}_t|\mathbf{s}_t). \tag{2.5}$$

As we are using parametrized stochastic policies in this thesis the optimization goal corresponds to finding the optimal parameter $\boldsymbol{\theta}$ that maximizes the return

$$\arg\max_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}). \tag{2.6}$$

2.1.4 Partially Observable Markov Decision Process (POMDP)

In an MDP we assume that we can always observe the true state of the environment and that the next state of the environment is only dependent on the current state and chosen action. This assumption gives us the possibility to learn a reactive policy, which only takes the current state to compute the next action and can still be optimal. However, the assumption often does not hold for applications in the real world. For example in the real world robots perceive the world through noisy sensors and can only see parts of the world through a camera. Or in real-time strategy games, players usually only see parts of the map where the players' units are, while the rest is hidden under the so-called fog of war or unknown before units have entered the unknown part. In these cases a reactive policy found by an MDP will not be sufficient to represent the optimal policy as the observations do not hold the Markov property. For this reason POMDPs were designed to tackle this problem. They can be seen as an extension to MDPs and are defined according to [27] by:

- the state space S containing all possible states $\mathbf{s} \in S$ describing the environment, which is unknown to the agent
- the action space A with all valid actions $\mathbf{a} \in A$ that can be performed by the agent
- the observation space O describing all possible observations $\mathbf{o} \in O$ that the agent receives
- the transition model $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ specifying the probability that task environment will be in new state $\mathbf{s}_{t+1}$ when action $\mathbf{a}_t$ is performed in the current state $\mathbf{s}_t$
- the observation dynamics $\mathcal{O}(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{a}_t)$ specifying the probability of observing $\mathbf{o}_{t+1}$ given that the agent took the action $\mathbf{a}_t$ resulting in the true underlying state $\mathbf{s}_{t+1}$
- the reward function $r(\mathbf{s}, \mathbf{a})$ specifying the immediate reward that is received for performing action $\mathbf{a}$ in state $\mathbf{s}$
- the initial state distribution $\mu_0(\mathbf{s})$ specifying the probability of state $\mathbf{s}$ being the initial state of the decision process.

Furthermore, we define the history $h_t = (\mathbf{o}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{t-1}, \mathbf{o}_t)$ as a vector containing all past observations and taken actions. It can be easily seen that this history is needed for the optimal policy in a POMDP since the single observations in each timestep do not hold the Markov property [64]. For example we could have two different underlying states s_1 and s_2 which have the same observation o, but different next states with different next observations for the same action a. Therefore, to predict the next state and next observation it is not sufficient to use only o and a, since we cannot differ between s_1 and s_2 if we have only given o.

To solve POMDPs there are two main possibilities, which transform a POMDP to an equivalent, but computational demanding MDP. This can be seen if we look at the complexity to obtain a optimal policy for MDPs and POMDPs. Papadimitriou et al. have shown that solving a finite horizon or infinite horizon MDP is P-complete, while solving a finite horizon POMDP is PSPACE-hard [51].

- Information state: In this approach the full history h_t is used as the state of the MDP, called information state. It can be easily seen that the full history fulfills the Markov property: $P(h_{t+1} = (\mathbf{o}_0, \mathbf{a}_1, \ldots, \mathbf{a}_t, \mathbf{o}_{t+t})|h_t, \mathbf{a}_t) = P(\mathbf{o}_{t+t}|h_t, \mathbf{a}_t)$, because all past histories and actions are already in the current history h_t per definition. Hence, we obtain an information state MDP. But the information state space will be much larger than the original state space, since with each timestep the history grows exponentially. This problem is also known as the curse of history [54].

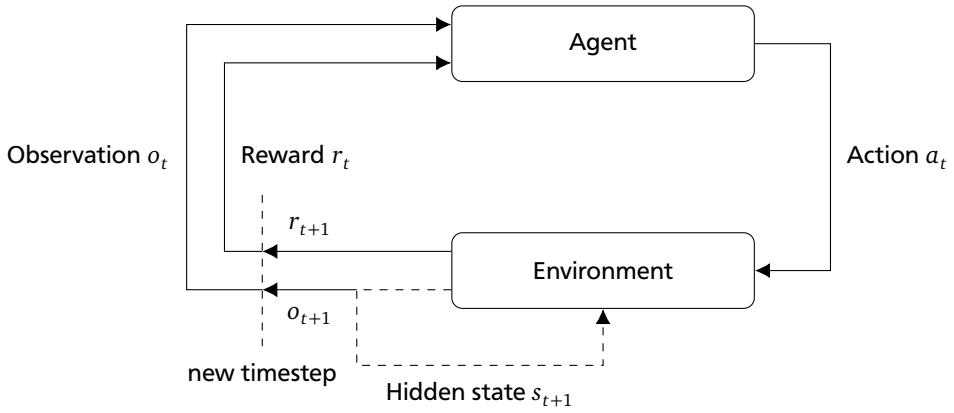


Figure 2.2.: Interaction loop between agent and environment in a POMDP. The agent can only observe the observations o_t while the true state s_t of the environment is hidden from the agent.

- Belief state: In this approach instead of remembering the whole history, which is not always feasible, a sufficient statistic of history is used. This statistic is the belief space, which represents the current belief of the agent in which current underlying state the agent is given the past observations and actions: $b_t(\mathbf{s}_t) = p(\mathbf{s}_t|\mathbf{o}_{1:t}, \mathbf{a}_{1:t})$. For example if we have a POMDP with two discrete states s_0 and s_1 the belief state would be given by an one-dimensional vector between 0 and 1, where 0 corresponds to the belief that the environment is in s_0 and 1 corresponds to s_1. For any other value between 0 and 1, $0 < p < 1$ corresponds to the belief that the environment is with probability $1 - p$ in s_0 and probability p in s_1. Using this belief state one can derive a reward function and transition function based on beliefs. Hence, we obtain a belief state MDP [27]. But in this case we will have an MDP with continuous states and D-dimensional belief vector, where $D = |S|$. Furthermore, we need a method to update the current belief given a new observation and action. More information about the belief state can be also found in Section 2.2.3.

In this thesis we will focus on solving POMDPs using the history of observations and actions. However, there are various algorithms, that use the belief state. In Section 2.2.3 we will show solution methods for POMDPs.

2.1.5 The Challenges of Reinforcement Learning in Fully and Partially Observable Environments

Before we discuss how these RL problems can be solved, we explain why solving MDPs and POMDPs are challenging from a model-free RL perspective [73, 5, 27]:

- The optimal policy must only be learned through interactions with the environment. The reward is the only learning signal that the agent receives. Other information such as the transition model of the environment is not given.
- Temporal credit assignment problem: In many environments it is difficult to determine which actions are good, since they could have influence on results many transitions later. Thus, the agent needs somehow to be able to deal with long-term dependencies.
- Exploration vs exploitation: In RL problems the agent needs to interact with the environment to explore the environment's reward structure. However, the agent will never know if its current found solution is the best or not. Hence, the agent is always in the dilemma if it should keep an policy, which is focused on exploration and might be suboptimal under the current knowledge, but may ultimately find a better solution. Or if it just should stop exploration and use a greedy policy which exploits the current knowledge.
- Curse of dimensionality: When Bellman applied optimal control to high dimensional problems, he faced an exponential explosion of necessary samples to cover the whole state and action space. This is known nowadays as the term 'curse of dimensionality'. In POMDPs we suffer even harder from the curse, since we saw in the last section, that we can transform them to MDPs, but with a much higher dimensional discrete or continuous state space.
- Information gathering vs exploitation: In RL problems with partial observability we have an additional challenge; an exploration vs exploitation dilemma on the policy level between information gathering and exploitation. In a POMDPs the agent does not know the exact state. Often it needs first to gather information about the true

underlying state of the world and can only exploit its knowledge if its sure enough about the true world state. Furthermore, the agent needs to learn this information gathering process only from interaction with the environment. That is why POMDPs are more difficult to their MDP counterpart and why more exploration / interactions are needed in order to find the optimal policy.

2.2 Reinforcement Learning Algorithms

After having described the RL problem in general and having showed how the problem can be modeled as an MDP, we will show in the first part of this section how one can obtain an optimal policy for a given MDP or RL problem. We will look at the two main classes of RL algorithms: *value function* methods and methods based on *policy search*. In the second part of this section we will take a look at the partially observable case and show how one can obtain an optimal policy for a given POMDP. Here, we will differ between model-based and model-free approaches.

2.2.1 Value Function Methods

Value function methods have their roots in control theory, where mapping states to numerical values to describe the effects of control decisions is a key part of the theory [73]. This mapping is called value function $V^\pi(\mathbf{s})$ and defined as the expected the long-term reward for state $\mathbf{s}$ when following policy π from state $\mathbf{s}$. Therefore, $V^\pi(\mathbf{s})$ is defined as the immediate reward for state $\mathbf{s}$ and the expected discounted reward of the next states

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t),\pi(\mathbf{a}_t|\mathbf{s}_t)}\left[\sum_{t=0}^{\infty}\gamma^t r(\mathbf{s}_t,\mathbf{a}_t)|\mathbf{s}_0=\mathbf{s}\right]. \tag{2.7}$$

In addition to this, there is a very similar definition, which does not only take the state but also the action into account, called state-action value function $Q^\pi(\mathbf{s},\mathbf{a})$ for policy π or better known as Q-function. This function is defined as the immediate reward for taking action $\mathbf{a}$ in state $\mathbf{s}$ and the expected reward of the next states, when following policy π:

$$Q^\pi(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t),\pi(\mathbf{a}_t|\mathbf{s}_t)}\left[\sum_{t=0}^{\infty}\gamma^t r(\mathbf{s}_t,\mathbf{a}_t)|\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a}\right]. \tag{2.8}$$

From the viewpoint of quality measure we can see both functions as a measurement how good it is to be in state $\mathbf{s}$ under the policy π and how good it is to take action $\mathbf{a}$ in state $\mathbf{s}$ under the policy π, respectively. Given $Q^\pi(\mathbf{s},\mathbf{a})$ we also can always obtain the best policy by just choosing the best action $\mathbf{a}$ for every state $\mathbf{s}$. Furthermore, it is easy to obtain the value function from the state-action value function and vice versa using following equations:

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\pi(\mathbf{a}|\mathbf{s})}\left[Q^\pi(\mathbf{s},\mathbf{a})\right] \quad Q^\pi(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \mathbb{E}_{\mathcal{P}(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[V^\pi(\mathbf{s}')\right]. \tag{2.9}$$

Following the theory of dynamic programming and optimal control one can use the definitions of a MDP to formulate both functions recursively, which is better known as the *Bellman equation*:

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\pi(\mathbf{a}|\mathbf{s})}\left[r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\mathcal{P}(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')]\right] \qquad Q^\pi(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\mathcal{P}(\mathbf{s}'|\mathbf{s},\mathbf{a}),\pi(\mathbf{a}|\mathbf{s})}[Q^\pi(\mathbf{s},\mathbf{a}')]. \tag{2.10}$$

Furthermore, it can be shown that there is only one unique solution to the equations (2.10), which is either the value or the state-action value function, depending on the used equation [73].

To obtain the optimal value function $V^{\pi^*}(s)$ of the optimal policy π^* one can use the *policy iteration* algorithm, which is a simple algorithm consisting out of two steps: *policy evaluation* and *policy improvement*. In the policy evaluation step one estimates the current value and Q-function of the current policy. This is done for example by computing alternately the Q-function from the value function and the value function from the Q-function using equations (2.10) and (2.9) until convergence. In the policy improvement step the current policy is updated by taking the actions which give the highest Q-values. By iterating these two steps the value function and policy will ultimately converge towards the optimal value function and optimal policy. Instead of computing the value function in the policy evaluation step until convergence one can also compute only one step of following equation

$$V^*(\mathbf{s}) = \max_{\mathbf{a}}(r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\mathcal{P}(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^*(\mathbf{s}')]) \tag{2.11}$$

known as *Bellman optimality equation* instead of using (2.10) and will still converge to the optimal value function and optimal policy. This is known as *value iteration* [73].

Since both algorithms, policy and value iteration, are based on optimal control, they assume that the transition model $\mathcal{P}(\mathbf{s}'|\mathbf{s},\mathbf{a})$ is known. However, in the typical RL-setting this model is unknown and needs to be estimated from the results of the interaction with the environment. In model-free RL we do not want to learn the transition model of the environment, but want to use the collected samples $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ of our trajectories directly for the update of the Q-function or value function. [1] This leads us to Monte-Carlo methods and temporal difference (TD) learning. TD learning methods are based on the recursive formulation of the Bellman equation, that is reformulated such that the prediction of the next value function is improved based on the current one. This strategy is known as *bootstrapping*. The most well-known algorithms of this method are Q-learning [82] and the state-action-state-reward-action algorithm [59] (SARSA), which both use the same approach based on the temporal difference error and have the same update rule. To estimate the value function they use Monte-Carlo estimates. Let $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ be a collected sample from the interaction with the environment using the current policy then the update rule is formulated as

$$Q^{\pi}_{new}(\mathbf{s}_t, \mathbf{a}_t) = Q^{\pi}_{old}(\mathbf{s}_t, \mathbf{a}_t) + \alpha\delta_t \tag{2.12}$$

where δ_t is the TD error and defined as

$$\delta_t = r_t + \gamma Q^{\pi}_{old}(\mathbf{s}_{t+1}, \mathbf{a}') - Q^{\pi}_{old}(\mathbf{s}_t, \mathbf{a}_t) \tag{2.13}$$

and can be described as the 1-step prediction error between the estimate of the current state and the next state. Using the estimated Q-function one can again apply the policy improvement step. The difference between Q-learning and SARSA is which action $\mathbf{a}'$ is used in the TD error. In SARSA one uses only collected samples from the policy, i.e., $\mathbf{a}' = \mathbf{a}_{t+1}$, where $\mathbf{a}_{t+1}$ is the next action that has been sampled in the next state $\mathbf{s}_{t+1}$ of the same trajectory. Therefore, SARSA is referred as an *on-policy* algorithm. Q-learning instead uses actions that were not necessary generated by the policy and is considered as an *off-policy* algorithm for this reason. In Q-learning the best action is picked $\mathbf{a}' = \arg\max_{\mathbf{a}_{t+1}} Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$. But as we don't know the optimal Q-values nor the optimal policy and use only samples to find them, we need to use an stochastic exploration policy, since with a deterministic or greedy policy we may never visit relevant state-action pairs. Common exploration policies for the discrete action case are for example the epsilon greedy policy, which takes a random action with probability ϵ and with probability $1-\epsilon$ the best action, or a softmax policy, based on the current Q-values [73, 60]. While Q-learning will eventually approximate the state-value function of the optimal policy, SARSA will approximate the state-value function of the exploration policy. However, by reducing the exploration rate of the exploration policy SARSA will eventually converge to the optimal policy.

2.2.2 Policy Search

Policy search is the other main class of RL algorithms, which are not based on any value function model, but try to improve the policy by directly searching in the policy parameter space. Improvement in the policy search framework is defined that the new found parameters $\boldsymbol{\theta}_{new}$ have a larger return (see Equation (2.1)) than the previous ones $\boldsymbol{\theta}_{old}$:

$$J(\pi_{\boldsymbol{\theta}_{new}}) > J(\pi_{\boldsymbol{\theta}_{old}}). \tag{2.14}$$

According to [15] policy search can be generally sketched as an algorithm consisting out of three steps.

- Explore: Generate multiple trajectories $\tau^{(i)} = (s_0^{(i)}, a_0^{(i)}, r_0^{(i)}, s_1^{(i)}, a_1^{(i)}, r_1^{(i)}, \ldots, s_T^{(i)})$, defined as a sequence of states, actions and rewards ending with a terminal state s_T and starting in an initial state s_0 using the current policy $\pi_{\boldsymbol{\theta}}$.
- Evaluate: Evaluate the quality of the sampled trajectories or actions.
- Update: Compute the new policy $\pi_{\boldsymbol{\theta}_{new}}$ from the sampled trajectories and evaluations.

In the evaluation step one can distinguish between two different evaluation strategies, *episode-based* and *step-based*. In episode-based methods the quality of a parameter vector is measured by the returns of the whole trajectory $R(\tau^{(i)}) = \sum_{t=0}^{T} r_t^{(i)}$ resulting in one data point $(\boldsymbol{\theta}^{(i)}, R(\tau^{(i)}))$ per trajectory $\tau^{(i)}$
While in step-based methods the quality of each single state action pair of the trajectory is measured by the returns to come $Q(s_t^{(i)}, a_t^{(i)}) = \sum_{h=t}^{T} r_h^{(i)}$ resulting in one data point $(s_t^{(i)}, a_t^{(i)}, Q(s_t^{(i)}, a_t^{(i)}))$ per state-action pair. Furthermore, in the update step there are different update strategies such as non-derivative, gradient-based methods or information-theoretic-based methods. In this thesis we will mainly focus on step-based policy gradient and information-theoretic-based methods, but for completeness non-derivative episode-based methods are for example the cross entropy method [12] (CEM) or the covariance matrix adaption evolution strategy [21] (CMA-ES).

[1] There are of course model-based RL methods, but they are not the focus of this thesis. Therefore, we won't introduce them here. But, we will see in Section 2.2.3 some model-based methods for POMDPs.

Policy gradient methods are a class of model-free policy search algorithms, that use gradient ascent as optimization strategy and update the policy parameters in the direction of the gradient. This gradient is calculated from the expected return with respect to all policy parameters, since the goal is RL is to find a policy that maximizes the expected reward (see Section 2.1.3). To compute the policy gradient the "likelihood-ratio" trick is usually applied; however, there are also other methods (see [15]). Assume that we have a random variable x and $f(x)$ any derivable function, then following equation holds:

$$\nabla \log f(x) = \frac{1}{f(x)} \nabla f(x) \tag{2.15}$$

using the chain rule. We can reformulate this into

$$\nabla f(x) = f(x) \nabla \log f(x). \tag{2.16}$$

Therefore, instead of computing directly the gradient of $f(x)$ we can also compute the gradient of $\nabla \log f(x)$ and just multiply this with $f(x)$ to get the gradient of $f(x)$, which is also known as "likelihood-ratio" trick. This trick can be especially helpful if we have an expectation of a function $h(x)$ (e.g. the return), where x is a random variable with probability density $f(x|\theta)$, and want to maximize this expectation with respect to θ or want to calculate the gradient, respectively. Using the "likelihood-ratio" trick we write the gradient as:

$$\begin{aligned} \nabla_\theta \mathbb{E}_{f(x|\theta)}[h(x)] &= \nabla_\theta \int f(x|\theta)\, h(x)\, dx = \int \nabla_\theta f(x|\theta)\, h(x)\, dx = \int f(x|\theta)\, \nabla_\theta \log f(x|\theta)\, h(x)\, dx \\ &= \mathbb{E}_{f(x|\theta)}[\nabla_\theta \log f(x|\theta)\, h(x)]. \end{aligned} \tag{2.17}$$

Thus, we can estimate the gradient of the function $f(x)$ with respect to θ using Monte-Carlo sampling

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log f(x_i|\theta)\, h(x_i) \tag{2.18}$$

where we sample $x_1, x_2, \ldots, x_N$ from $f(x|\theta)$. Now we can put everything into the RL framework, where we want to compute the gradient of $J(\pi_{\boldsymbol{\theta}})$ with respect to the parameter $\boldsymbol{\theta}$ of the parametrized policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{P}_\pi(\boldsymbol{\tau}|\boldsymbol{\theta})}[R(\boldsymbol{\tau})] = \mathbb{E}_{\mathcal{P}_\pi(\boldsymbol{\tau}|\boldsymbol{\theta})}[\nabla_\theta \log \mathcal{P}_\pi(\boldsymbol{\tau}|\boldsymbol{\theta})\, R(\boldsymbol{\tau})] \tag{2.19}$$

where $\mathcal{P}_\pi(\boldsymbol{\tau}|\boldsymbol{\theta})$ is the probability distribution for sampling trajectory $\boldsymbol{\tau}$ given $\pi_{\boldsymbol{\theta}}$. Inserting the definition of $\mathcal{P}_\pi(\boldsymbol{\tau}|\boldsymbol{\theta})$ (2.5) into the expectation results in a logarithm of a product sum

$$\mathbb{E}_{\mathcal{P}_\pi(\boldsymbol{\tau}|\boldsymbol{\theta})}\left[\nabla_\theta \log\left(\mu_0(\mathbf{s}_0) \prod_{t=0}^{T-1} \mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\, \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\right) R(\boldsymbol{\tau})\right]. \tag{2.20}$$

When taking the logarithm of this expression the product sum turns to a regular sum and when differentiating this sum with respect to $\boldsymbol{\theta}$ the terms $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and $\mu_0(\mathbf{s}_0)$ will disappear, resulting in

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathcal{P}_\pi(\boldsymbol{\tau}|\boldsymbol{\theta})}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\, R(\boldsymbol{\tau})\right] \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})\, R(\boldsymbol{\tau}^{(i)}). \tag{2.21}$$

This equation is also known as the REINFORCE rule, which was one of the first policy gradient methods [85]. This equation can be intuitively interpreted as collecting trajectories from the current policy and then move in direction of the gradient of log policy weighted by the return of the collected trajectories. By using the knowledge that the reward of the past are not correlated with actions in the future one can rewrite the equation to the equation, known as policy gradient theorem [74]

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathcal{P}_\pi(\boldsymbol{\tau}|\boldsymbol{\theta})}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t) \sum_{h=t}^{T} \gamma^h r_h\right] \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})\, Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}). \tag{2.22}$$

Instead of computing the Q-values using Monte-Carlo estimates, which suffer from high variance, they can also be estimated using previous introduced techniques such as TD learning methods, which typically have lower variance but

suffer from bias induced by the bootstrapping methods [5, 15]. We can further subtract a baseline b from the returns to reduce the variance and while keeping the estimation unbiased [85]

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathcal{P}_\pi(\tau|\boldsymbol{\theta})}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\ (Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t, \mathbf{a}_t) - b)\right]. \tag{2.23}$$

It is even possible to subtract a state-dependent baseline $b(\mathbf{s})$ without changing the expectation. For this usually the value function $V^{\pi_{\boldsymbol{\theta}}}(\mathbf{s})$ is used [74]. This results in the advantage function

$$A^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a}) = Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a}) - V^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}) \tag{2.24}$$

and following policy gradient

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathcal{P}_\pi(\tau|\boldsymbol{\theta})}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\, A^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t, \mathbf{a}_t)\right]. \tag{2.25}$$

For the update of the policy parameters, we use following equation regardless of the chosen estimation $J(\pi_{\boldsymbol{\theta}})$:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} + \alpha\, \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) \tag{2.26}$$

where α is some positive step size, called learning rate. The difficulty of this approach is that we need now to chose an appropriate learning rate α, which regulates how much we do want to follow our gradient. If we choose α too large then we might update our policy too greedily and only try to maximize the reward on the given samples, since we need to keep in mind that small changes in $\boldsymbol{\theta}$ can lead to large changes of the policy $\pi_{\boldsymbol{\theta}}$. This would stop exploration, because we would update our current parameters, such that new policy only outputs the best action with the highest reward on the seen samples, and would lead to premature convergence. If we choose α too small then we will keep the exploration rate very high and converge only very slowly towards the optimal solution. Furthermore, the plain gradient is not invariant to rescaling and transformation of the parameters. In the later section we will see several approaches, known as information theoretic methods, which aim to solve this problem based on a information theoretic 'distance' measure between probability distributions, called Kullback-Leibler (KL) divergence and defined as:

$$KL(\, p(x) \,||\, q(x)\,) = \int p(x)\, \log \frac{p(x)}{q(x)}\, dx \tag{2.27}$$

where $p(x)$ and $q(x)$ are probability distributions of the variable x. The reason why we put distance into quotation marks, is that the KL divergence is not a real distance metric, because it is not symmetric. If it was symmetric

$$KL(\, p(x) \,||\, q(x)\,) = KL(\, q(x) \,||\, p(x)\,) \tag{2.28}$$

needs to hold for every possible $p(x)$ and $q(x)$. However, this equation holds only for $p(x) = q(x)$. It still can be seen as a measure of how off two probabilistic distributions $p(x)$ and $q(x)$ are. The idea of these information theoretic methods is to use the KL divergence to bound the difference between the policy before and after the update [15].

2.2.3 POMDP Methods

In this section we will give a brief introduction to common solution methods for POMDPs: belief-based approaches, which use the belief state (see Section 2.1.4) and update these beliefs based on a model, and model-free approaches, which try to learn a policy directly from interaction with the environment. In this thesis we won't use belief / model-based approaches, however, as they have been used much more extensively than model-free methods for POMDPs, we will present them first before discussing model-free approaches.

Belief-based approaches for POMDPs

In the previous Section 2.1.4 we saw that using a belief state b, which is a probability distribution over S describing the probability of being at every state s, such that $0 \leq b(s) \leq 1\ \forall s \in S$ and $\sum_{s \in S} b(s) = 1$, one can create a belief MDP. In this subsection we will discuss this approach in more detail and show how belief MDP can be solved. We will first explain how one can obtain new belief b' for a given belief b, action a and observation o, known as belief update, before we

define formally a belief MDP according to [27]. Assume that the agent is in belief b and executes action a, which leads to observation o then we can obtain the new probability for being in the state s', $b'(s')_{b,a,o}$ by applying Bayes rule

$$\begin{aligned} b'(s')_{b,a,o} &= P(s' \mid b, a, o) \\ &= \frac{P(o \mid s', b, a)\, P(s' \mid b, a)}{P(o \mid b, a)} \\ &= \frac{P(o \mid s', a)\, \sum_{s \in S} P(s' \mid s, b, a)\, P(s \mid b, a)}{P(o \mid b, a)} \\ &= \frac{\mathcal{O}(o \mid s', a)\, \sum_{s \in S} \mathcal{P}(s' \mid s, a)\, b(s)}{P(o \mid b, a)} \end{aligned} \tag{2.29}$$

where $P(o \mid b, a)$ is a normalization factor independent from s' and defined as

$$P(o \mid b, a) = \sum_{s \in S} b(s) \sum_{s' \in S} \mathcal{P}(s' \mid s, a)\, \mathcal{O}(o \mid s', a). \tag{2.30}$$

Hence, given $\mathcal{P}$ and $\mathcal{O}$ from the original POMDP we can compute the new belief from an old one for a given action, observation pair. With this we can specify a continuous space belief MDP from the original POMDP $(S, A, O, \mathcal{P}, \mathcal{O}, r, \mu_0)$ containing:

- B, the set containing all possible belief states $\mathbf{b} \in B$ over the states S from the original POMDP.
- A, the set of possible actions from the original POMDP.
- $\tau(b_{t+1} | b_t, a_t)$ the belief transition function, describing the probability that the task environment is in the new belief b_{t+1} when action a_t is performed in belief b_t. This transition function is defined as

  $$\tau(b_{t+1} | b_t, a_t) = \sum_{o_{t+1} \in O} P(b_{t+1} | b_t, a_t, o_{t+1})\, P(o_{t+1} | b_t, a_t)$$

  where

  $$P(b_{t+1} | b_t, a_t, o_{t+1}) = \begin{cases} 1 & \text{if } b'_{b_t, a_t, o_t} = b_{t+1} \\ 0 & \text{otherwise} \end{cases}.$$

  Hence, what this transition function does is to sum the probabilities of all observation o_{t+1} which are observed after executing action a_t in belief b_t that result in the next belief b_{t+1}.
- The reward function $r_B(b, a)$ defined on the beliefs $r_B(b, a) = \sum_{s \in S} b(s)\, R(s, a)$ specifying the immediate reward of action a in belief b.

To solve this belief MDP one can apply value function methods and formulate the Bellman optimality equation (2.11) for beliefs

$$\begin{aligned} V^*(b) &= \max_a (r_B(b, a) + \gamma \mathbb{E}_{\tau(b' | b, a)}[V^*(b')]) \\ &= \max_a (r_B(b, a) + \gamma \sum_{b' \in B} \tau(b' | b, a)\, V^*(b')) \\ &= \max_a (r_B(b, a) + \gamma \sum_{o \in O} \tau(o | b, a)\, V^*(b'_{b,a,o})). \end{aligned} \tag{2.31}$$

Since this equation is hard to solve due to the huge continuous belief space, it is common to use a different representation of the value function based on linear convex functions, which have been shown that they can approximate V^* arbitrarily close [71]. Hence, the common approach is to write the value function with a finite set of linear functions, known as α-vectors

$$V = \{\alpha_1, \ldots, \alpha_n\} \tag{2.32}$$

where one α-vector is a function over S and describes the value function for one belief b. Hence, we can write the Bellman optimality equation (2.11) also as

$$V^*(b) = \max_{\alpha \in V} b \cdot \alpha \tag{2.33}$$

where $b \cdot \alpha = \sum_{s \in S} b(s)\ \alpha(s)$ is the inner product. Furthermore each α-vector is associated with an action defining the best immediate action for this belief assuming optimal behavior for the following timesteps [54]. Using this definitions one can derive the value iteration algorithm for belief MDPs. But as the complexity for a single iteration is $O(|V| \times |A| \times |O| \times |S|^2 + |A| \times |S| \times |V|^{|O|})$ this method is only feasible for small POMDPs as the number of α-vectors $|V|$ grows exponentially with every iteration [64].

Approximate Methods:

For this reason a variety of approximate methods have been proposed. One prominent class of them are point-based methods, which are based on the idea that many points of the belief space will be never reached. Hence, these methods represent and update only the α-vectors at certain belief points $B = \{b_1, b_2, \ldots, b_n\}$ spanned over the whole belief space. The advantage of only using the finite set of belief points is that one does not need to store all α-vectors and can for example only store the best α-vector at each belief point. The disadvantage of this approach is that the accuracy of the value function will suffer [64]. Well known point-based algorithms are for example point-based value iteration [54], heuristic search value iteration [68], Perseus [72] and SARSOP [32].

Online Methods:

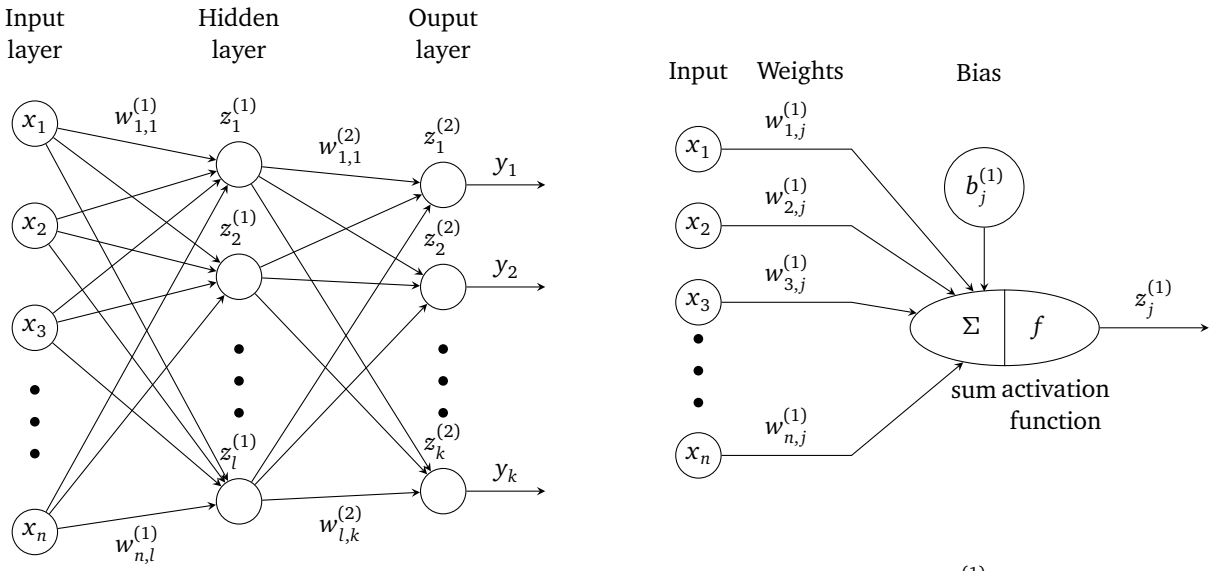
The methods to solve POMDPs we have presented until now, are also known as offline planning methods. In these methods the agent computes the best actions for all possible situations beforehand. Hence, they struggle to scale up to very large POMDPs. Furthermore, they need to use the dynamics of the environment for planning. Therefore, small changes lead to the necessity of recomputing the full policy [57]. For this reason online methods were designed to avoid the complexity of planning a policy for all possible situations and only to plan online for the current information state. The advantage of this approach, known as online, is that to find a good local policy for the current belief state, we only need to consider the belief states that are reachable from the current one. According to [57] an online algorithm is typically divided into two steps: the *planning step* and *execution step*. In the planning step the algorithm starts from the current belief and computes the best action to take. This is done either by following the same procedure as the shown offline methods, which is to compute all reachable belief states from the current one using belief update and to do value estimation. Or by using a black box simulator as in the partially observable Monte-Carlo planning algorithm [67] (POMCP) or determinized sparse partially observable tree algorithm [70] (DESPOT), which is used to sample sequences of states, observations and rewards given the current belief and an action. Based on these samples, also known as particles, one can approximate the belief state and update its value. In the execution step the best found action is chosen and the current belief state is updated according to the received observation. An alternative related algorithm is the policy graph improvement algorithm, which is an online approximate algorithm that works on policy graphs instead of as POMCP on policy trees or as point-based methods on the value function [50, 49].

Model-free approaches for POMDPs

In the last subsection we have seen methods, which either use an explicit model of the POMDP, $M = (\mathcal{P}, \mathcal{O}, r, \mu_0)$ taken from the POMDP definition or a black box simulator of the POMDP to compute the belief and value function. In this subsection we will show now model-free approaches, which directly learn the policy from interactions with the environment. However, there have not been so many model-free approaches, as it is very difficult to learn the solution only by interactions with the environment without any planning. To give some examples for model-free approaches: In 1993 Lin et al. [40] have proposed several different neural network structures, which take the POMDP history into account, to approximate the Q-function and trained them using TD-methods (see Section 2.2.1). In 2007 Wiestra et al. [83] successfully applied the policy gradient framework (see Section 2.2.2) on recurrent neural networks with LSTMs to train them for different tasks that require memorization. More information about neural networks can be found in the next section.

2.3 Deep Learning

Having seen the powerfulness of deep reinforcement learning in the introduction and since we are using this approach in this thesis, we will give in this section an introduction to the other foundation of deep reinforcement learning: deep learning. In the recent years the usage of deep learning and its popularity has increased enormously. It has even become the main approach in several machine learning fields such as computer vision, natural language processing and speech recognition after showing very remarkable results in these research fields. In computer vision everything took off when Alex et al. [31] trained a large deep convolutional neural network on tons of images that won the ImageNet Large-Scale



(a) Feedforward neural network

(b) Neuron $z_j^{(1)}$

Figure 2.3.: left: Sketch of a fully connected multilayer perceptron / feedforward neural network with one hidden layer, n-dimensional input $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and k-dimensional output $\mathbf{y} = (y_1, y_2, \ldots, y_k)$, blank circles represent neurons (enlargement on the right), arrows represent the connections and data flow between the neurons, biases are omitted for clarity, right: Sketch of one example neuron z_j of the hidden layer with n-dimensional input $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and output z_j

Visual Recognition Challenge in 2012 with archiving a top-5 error of 15.3%, 10.8% ahead of the runner up. Similar in speech recognition [26] beat all the previous records on phonetic classification for speech recognition and was the first major industrial application of deep learning [35]. This trend of deep learning has mainly started in the early 2000s and its core idea can be summarized as a supervised learning method, that is able to learn very expressive functions and representations end-to-end from tons of data, without needing to design any features by hand [19]. Or summarized in a simple recipe: choose a very powerful function approximator such as *deep neural networks (multilayer perceptrons)*, choose a loss function and then train the function approximator using gradient decent on a huge amount of labeled data [62]. Therefore, we need to take a look at the root of these powerful function approximators: artificial neural networks. Research on artificial neural networks started actually already in the 1940s beginning with the attempt to find a mathematical representation for biological learning systems [41, 23]. One of the first successful implementation of these systems was the perceptron of Rosenblatt, which allowed the training of a single neuron [55]. The interest in this field first died though, when 1969 Minsky and Papert [42] showed that a single layer perceptron is not able to learn simple functions as XOR. However, the second wave emerged in the 1980s with the idea of multilayer perceptrons and their successful training using backpropagation - both ideas, which are still used nowadays [58]. Deep Learning is now the third wave of this neural network research with new techniques and advances such as:

- convolutional neural networks and the idea of weight sharing [37],
- breakthroughs using other activation functions such as relu [46, 18], which made training easier,
- new momentum-based gradient decent methods such as ADAM [29] or RMS-prop [79],
- the availability of much larger labeled datasets and computation power
- and the existence of new libraries such as Theano [77] and Tensorflow [1], which made the implementation and training of these networks much easier and even allow training on GPUs.

2.3.1 Feedforward Neural Networks

Multilayer perceptrons or nowadays also known as (deep) feedforward neural networks are the core of deep learning models. They aim to approximate some unknown function f, which maps from an input x to some output y. To model

this function f multilayer perceptrons use, so called *neurons*, which are single computation units with multiple inputs and one output, inspired by neuroscience ("*neural*"). Furthermore, these neurons are connected to each other and arranged in multiple *layers*, such that the output of one neuron will be the input of another neuron ("*network*"). A neural network is called a "*feedforward*" neural network, if the connections in the network are aligned such that from input to output each neuron is only visited once, i.e., there are no feedback loops in the model. If a neural network has feedback loops in its structure it is called *recurrent* neural network. Summarized, a feedforward neural network approximates an unknown function f from input $\mathbf{x}$ to output $\mathbf{y}$ by using multiple intermediate computations, represented by its neurons and by its network structure. Figure 2.3 (a) shows an example of a feedforward neural network. Mathematically, the computation of a single neuron in this network (depicted in Figure 2.3 (b)) can be described as

$$z_j = f\left(\sum_{i=0}^{N} w_{i,j}^{(k)} x_i + b_j^{(k)}\right) \tag{2.34}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ is the input, z_j is the output, $w_{i,j}^{(k)}$ are the corresponding weights for input component x_i, $b_j^{(k)}$ is the bias, f is the activation function and j the index that z_j is the j-th neuron in its layer k. If all neurons in one layer share the same activation function $f^{(k)}$, which is usually the case, the output for all neurons of this layer can be written as a simple matrix-vector expression:

$$\mathbf{z} = f^{(k)}\left({\mathbf{W}^{(k)}}^T \mathbf{x} + \mathbf{b}^{(k)}\right) \tag{2.35}$$

where $\mathbf{z} = (z_1, z_2, \ldots, z_l)$ is a vector containing all outputs of all neurons, $\mathbf{b}^{(k)} = (b_1^{(k)}, b_2^{(k)}, \ldots, b_l^{(k)})$ is a vector containing all biases and $\mathbf{W}^{(k)}$ is the weight matrix containing all weights $w_{i,j}^{(k)}$ for all neurons of this layer k. Furthermore, in this case the whole computation of a neural network can be written as a concatenation of matrix operations and the application of the activation functions. For example the output of the neural network of Figure 2.3 can be expressed as

$$\mathbf{y} = f^{(2)}\left({\mathbf{W}^{(2)}}^T f^{(1)}\left({\mathbf{W}^{(1)}}^T \mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right). \tag{2.36}$$

where $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ is the output of the neural network. One additional insight is that it has been shown that a feedforward neural network is a universal function approximator, i.e., it can model any smooth function, given enough hidden units, to any desired level of accuracy [33]. The problem though is to find the right number of hidden units and the training of their weights, while preventing overfitting.

2.3.2 Convolutional Neural Networks

Convolutional neural network (CNN) emerged in the third wave of neural networks, deep learning, and were proposed by LeCun et al. initially for character recognition [36, 37]. Nowadays they have become the bread and butter of computer vision and natural language processing. According to Bishop CNNs (example illustrated in Figure 2.4) can be seen as a special case of feedforward neural networks with three main differences: local connectivity and parameter sharing and subsampling [10].

- Local connectivity / receptive field: When dealing with high-dimensional input such as images, it can be advantageous to have neurons that are only connected to a local region of the input instead of fully connecting them with all input neurons. This is also known under the hyperparameter *receptive field* (as in the primary visual cortex), which defines how large this local region is. In our example we have a receptive field of 5x5. One additional hyperparameter is the *stride*, which defines how much overlap and difference is between the input neurons of the receptive fields of two neighboring neurons in the convolutional layer, respectively. In our example we use a stride of 1, i.e., the difference of neighboring neurons (red and orange) is only one row or column into the corresponding direction.

- Parameter sharing: To reduce the parameters that need to be learned for a CNNs, usually the weights for the same receptive field are shared across the whole input. Intuitively, this idea can be seen, that if we consider a neuron in the convolutional layer as feature detector, that all neuron of the same feature map should detect the same pattern, but just at different locations of the input. Since we will need to detect multiple features to obtain a good model, there are usually multiple feature maps in one convolutional layer. The amount of feature maps is defined as the hyperparameter *depth*. In our example we have a depth of 5, indicated by the 5 different feature maps.

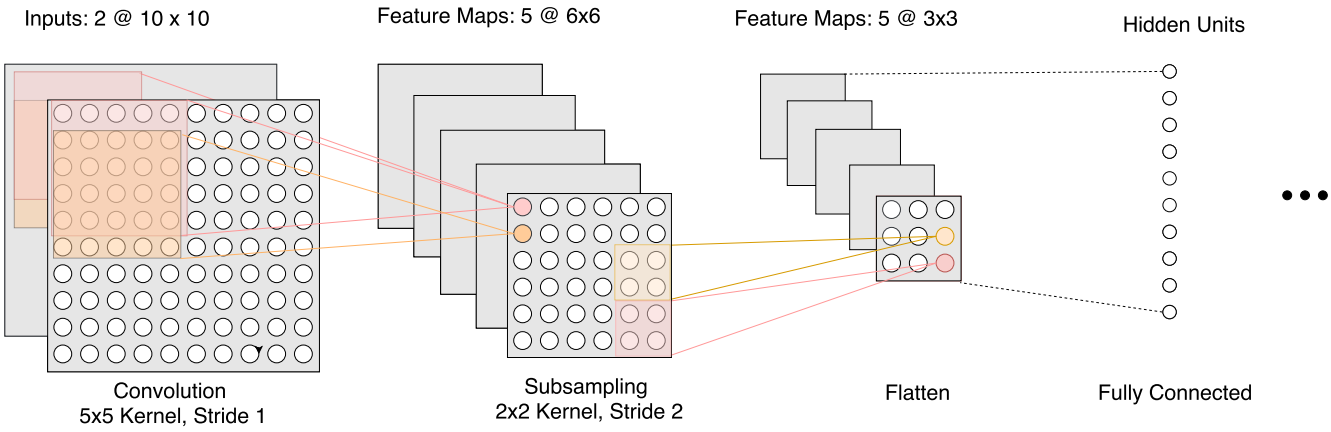


Figure 2.4.: Example for a convolutional neural network.

- Subsampling: Often after a convolutional layer a subsampling layer is applied to reduce the the dimensionality of the data of each feature map. This method is very common when processing images for object recognition, since this does not only reduce the dimensionality but also introduces insensitivity of the learned features to small shifts inside the images. Subsampling is also known in computer vision as pooling. In deep reinforcement learning pooling is often omitted, because translation invariance can be sometimes bad for a reinforcement learning task. For instance, if we think of the video game Pong and have a ball feature detector, the location of the ball is important. Hence, we do not want that this information is lost due to pooling.

2.3.3 Training of Neural Networks

To train the parameters / weights of the network usually gradient decent is used in combination with the chain rule with respect to a predefined loss function E. In this area this method is also known under the term backpropagation, which is an algorithm that can perform gradient decent in a computational efficient way. To show the idea of backpropagation we will follow [10]. Assume that E is a quadratic loss function[2] over the each data point in the training set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_N, \mathbf{t}_N)\}$

$$E(\mathbf{w}) = \sum_{i=1}^{N} E_n(\mathbf{w}) = \sum_{i=1}^{N} \frac{1}{2}(\mathbf{y}_n - \mathbf{t}_n)^2 \tag{2.37}$$

where $\mathbf{w}$ is a vector containing all the parameters of the neural network, $\mathbf{y}_n$ is the output of the neural network given $\mathbf{x}_n$ and $\mathbf{w}$ for the training data point n and $\mathbf{t}_n = (t_{n1}, \ldots, t_{nk})$ the target output for the training data point n. To update the weights using gradient descent

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + \alpha \nabla_{\mathbf{w}} E_n(\mathbf{w}) \tag{2.38}$$

where α is the learning rate, we will need to compute the gradient of the error function $\nabla_{\mathbf{w}} E_n(\mathbf{w})$. But before we compute the exact form of the gradient we will introduce an additional variable $a_j^{(k)}$

$$z_j^{(k)} = f^{(k)}\left(\sum_{i=0}^{N} w_{i,j}^{(k)} z_i^{(k-1)}\right) = f^{(k)}\left(a_j^{(k)}\right) \tag{2.39}$$

which describes the input of non-linear activation function of the j-th neuron. Note that we omitted in this equation the biases and don't need to consider them explicitly, since one can include them into the sum by introducing an extra unit with fixed activation [10]. If we take now the derivative of E with respect to a particular weight $w_{i,j}^{(k)}$, we see that we need to take the derivative of E_n

$$\frac{\partial E}{\partial w_{i,j}^{(k)}} = \sum_{n=1}^{N} \frac{\partial E_n}{\partial w_{i,j}^{(k)}} \tag{2.40}$$

[2] It is also possible to derive the backpropagation for other error functions. In this thesis we will use the quadratic loss function as example to show the idea of backpropagation.

since we have a sum over E_n in Equation (2.37). Note that the outputs of the neurons depend on the input of the training point n; however, we will follow [10] and omit the subscript n from the network variables for readability. To derive now the partial derivative we note that the weight $w_{i,j}^{(k)}$ influences E_n only via $a_j^{(k)}$ of neuron j. Hence, we can apply the chain rule and obtain

$$\frac{\partial E_n}{\partial w_{i,j}^{(k)}} = \frac{\partial E_n}{\partial a_j^{(k)}} \frac{\partial a_j^{(k)}}{\partial w_{i,j}^{(k)}}. \tag{2.41}$$

To obtain a simpler formulation for this derivative, we introduce an additional second variable

$$\delta_j^{(k)} = \frac{\partial E_n}{\partial a_j^{(k)}} \tag{2.42}$$

where the δ's are often called errors. Using the definition of $a_j^{(k)}$ in (2.39) we can write

$$\frac{\partial a_j^{(k)}}{\partial w_{i,j}^{(k)}} = z_i^{(k-1)} \tag{2.43}$$

and obtain

$$\frac{\partial E_n}{\partial w_{i,j}^{(k)}} = \delta_j^{(k)} z_i^{(k-1)}. \tag{2.44}$$

Therefore, we only need to calculate the value of $\delta_j^{(k)}$ for every neuron to evaluate the derivative, since we can get $z_i^{(k-1)}$ easily just by evaluating the neural network. For the output layer L we have

$$\delta_j^{(L)} = f'^{(L)}\left(a_j^{(L)}\right)\left(y_j - t_j\right) \tag{2.45}$$

for the j-th output neuron. To derive $\delta_j^{(k)}$ for the hidden layers we now can apply the chain rule again

$$\delta_j^{(k)} = \frac{\partial E_n}{\partial a_j^{(k)}} = \sum_k \frac{\partial E_n}{\partial a_i^{(k+1)}} \frac{\partial a_i^{(k+1)}}{\partial a_j^{(k)}} \tag{2.46}$$

where i is a sum over all neurons, which use neuron j as input. By inserting (2.39) and (2.42) one can obtain

$$\delta_j^{(k)} = f'^{(k)}\left(a_j^{(k)}\right) \sum_i w_{ji}\, \delta_i^{(k+1)}. \tag{2.47}$$

Hence, to obtain all δ's we can just start with computing $\delta_j^{(L)}$ for all output neurons and use its result to compute $\delta_j^{(L-1)}$ and so on.

2.4 Deep Reinforcement Learning

Similar to deep learning the idea of using the idea of using neural networks to approximate the policy π and / or the value functions V or Q is not new, but with recent success of deep learning this idea was revitalized, too. One main advantage of deep reinforcement learning methods compared to prior work in RL is that they are able to scale to high-dimensional problems. For example by using CNNs for approximation these approaches are able to learn directly from raw, high-dimensional vision data and are able to extract low-dimensional feature representations from the input on their own. Similar to the reinforcement learning section, we will take a look on both value function and policy search methods and introduce briefly some prominent related works. For a more detailed survey on this methods we refer to [5].

2.4.1 Value Function Methods

One big recent breakthrough in model-free deep reinforcement learning was the application of DQN, starting from 2013, to a set of the classical Atari 2600 video games, which was ultimately able to learn policies that could archive human-level performance or even better directly from visual data [44, 45]. In [45] a deep neural network, consisting out of several stacked convolutional layers followed by some fully connected layers, was used to approximate the Q-function $Q^{\pi}(s,a)$, where s in this case were large 210 x 160 pixel 8 bit RGB-images. To train the this network DQN follows the same equations (2.12) and (2.13) as in standard Q-learning and uses the mean squared error of the TD-error as loss function for stochastic gradient descent

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i}^{N}\left(r_i + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s_i', a_i') - Q_{\boldsymbol{\theta}}(s_i, a_i)\right)^2, \tag{2.48}$$

where s_i' is the next state when taking action a_i in state s_i of sample i. But usually in practice this would diverge or lead to bad results due to correlations of the samples and non-stationary targets. To deal with both problems the paper utilized two techniques: experience replay and weight freezing, also known as target networks. Experience replay is a first-in-first-out memory buffer, where past sampled transitions are stored in the form $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i)$ and sampled uniform randomly for the training batch of the stochastic gradient decent, breaking temporal correlations. One additional advantage of the experience replay is that by reusing the transitions, fewer interactions with the environment are needed. Weight freezing or target networks, describe the idea that an additional target network $Q_{\boldsymbol{\theta}^-}$ is introduced, where the parameters are fixed for a certain number of iterations, to stabilize the oscillations in the TD-error. This leads to following loss function

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i}^{N}\left(r_i + \gamma \max_{a'} Q_{\boldsymbol{\theta}^-}(s_i', a_i') - Q_{\boldsymbol{\theta}}(s_i, a_i)\right)^2. \tag{2.49}$$

In this equation we can also see the limitations of DQN. DQN is only applicable for a discrete action space, since we need to be able to take the max operator over all actions. Another limitation is the simple epsilon greedy exploration policy Mnih et al. used in [44]. For example DQN is still struggling to solve the infamous Atari game "Montezuma's Revenge". There have been several ideas attempting to solve this problem [8, 48, 75], however, there has been no breakthrough, yet. In Chapter 4 we will show one of these methods and apply this to POMDPs. There has been also an extension to the DQN algorithm, which uses recurrent neural networks instead of feedforward neural networks and has been applied to a partially observable environment [22]. This extension is called deep recurrent Q-networks and based on the idea that by using recurrent connections the network is able to store information over long time periods. It has been applied to the videogame Pong and was able to learn to play the game even when frames are randomly blacked out.

2.4.2 Policy Search Methods

Inspired by the success of DQN there has been several research on policy search methods and deep learning. For this thesis we will briefly introduce two related approaches: DDPG [66, 39] and TRPO [63].

In the previous section we saw that DQN can be only applied to discrete actions. DDPG solves this problem by applying an actor-critic approach with a deterministic policy and policy gradient. This circumvents the necessity of taking the max operator over all actions. The actor-critic approach combines the idea of policy search and value-function methods by using explicit representations for both the value function and the policy. In this approach the value function ("critic") is learned from the samples of the rollout and used for the update of the policy ("actor"), instead of directly using the Monte-Carlo sampled returns for the policy gradient equations. Furthermore, in DDPG a deterministic policy is used instead of a stochastic policy. The advantage of using a deterministic policy is that Silver et al. have extended the policy gradient theorem to deterministic policies, which needs less samples, since it only needs to integrate over the state space, while with stochastic policies it is necessary to integrate both over state and action space [66]. In [39] DDPG has been successfully applied in an off-policy setting on high-dimensional vision input to solve more than 20 different simulated physics tasks. Similar to the discrete case, the idea of DDPG has been also applied to POMDPs by using recurrent neural networks and LSTMs [24].

The disadvantage of DDPG is that the training of actor-critic can be unstable, i.e., a bad approximation of the critic can lead to a bad update of the policy. To prevent this DDPG also applies techniques, such as replay memory and soft weight-freezing. However, it has been shown in [13] on real robot experiments that these techniques can still fail and lead to degeneration even after a decent policy was found if the replay memory does not contain experience samples that are

diverse enough. To circumvent all these stability issues with these techniques there is also another approach, proposed by Schulman et al. in 2015, called TRPO. TRPO is a policy search method, that does not use any of these techniques, but instead is based on the core idea to avoid parameter updates, which would change the policy too much by adding a constraint on the KL divergence (introduced in Section 2.2.2) to bound the difference between the old and new policy in an policy update step. This idea is not new in RL and has been used in several older RL approaches such as relative entropy policy search [52] and its variants e.g. [2, 3]. However, TRPO was the first one to bring this idea successfully into the world of deep reinforcement learning. Mathematically, TRPO optimizes in practice following objective function

$$\begin{aligned} &\max_{\theta} L_{\theta_{\text{old}}}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathbf{s}\sim\pi_{\boldsymbol{\theta}_{\text{old}}},\mathbf{a}\sim\pi_{\boldsymbol{\theta}}}\left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}|\mathbf{s})} A^{\pi_{\text{old}}}(\mathbf{s},\mathbf{a})\right] \\ &\text{subject to } \mathbb{E}_{\mathbf{s}\sim\pi_{\boldsymbol{\theta}_{\text{old}}}}\left[KL\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot\,|\,\mathbf{s}) \,||\, \pi_{\boldsymbol{\theta}}(\cdot\,|\,\mathbf{s})\right)\right] < \epsilon \end{aligned} \tag{2.50}$$

where $L_{\theta_{\text{old}}}(\pi_{\boldsymbol{\theta}})$ is a surrogate loss function, that has the same gradient as the policy gradient loss function, $A^{\pi_{\text{old}}}(\mathbf{s},\mathbf{a})$ is the advantage and ϵ is a hyperparameter, defining the maximum allowed KL difference in one update step. To solve this constrained trust region problem Schulman et al. proposed a method, consisting out of two steps:

1. Compute a search direction, using a linear approximation to the objective and quadratic approximation to the KL constraint.

2. Perform a line search in the direction, found in step 1, ensuring that the objective is improved while satisfying the KL constraint. Subsequently, update the policy by adding the search direction scaled by the found line search parameter to the old policy.

To compute the search direction in step 1 TRPO approximately solves the equation $\mathbf{As} = \mathbf{g}$, where $\mathbf{g}$ is the derivative of the surrogate loss with respect to the policy parameters, $\mathbf{s}$ the search direction and $\mathbf{A}$ the Fisher information matrix. The reason why the Fisher information matrix is used is, because it is a quadratic approximation to the KL constraint (we will see the usage of the same insight later in Section 3.2.2)

$$KL\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}|\mathbf{s})||\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\right) \approx \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}_{\text{old}})^T \mathbf{A}(\boldsymbol{\theta}-\boldsymbol{\theta}_{\text{old}}) \quad \mathbf{A}_{ij} = \frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j}\mathbb{E}_{\mathbf{s}\sim\pi_{\boldsymbol{\theta}_{\text{old}}}}\left[KL\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot\,|\,\mathbf{s}) \,||\, \pi_{\boldsymbol{\theta}}(\cdot\,|\,\mathbf{s})\right)\right]. \tag{2.51}$$

However, in large-scale problems, such as when using deep neural networks for the policy, it is very difficult from a computational and memory perspective to compute the full matrix $\mathbf{A}$ and its inverse $\mathbf{A}^{-1}$, which is needed to obtain the search direction. Therefore, TRPO uses the conjugate gradient algorithm [25] to avoid this computation. This algorithm allows to approximately solve equations of type $\mathbf{Ax} = \mathbf{b}$ iteratively without needing to compute the full matrix and only requires a function that computes matrix-vector products $\mathbf{y} \rightarrow \mathbf{Ay}$.

3 Compatible Policy Search (COPS)

3.1 Introduction

In this chapter we will present our own compatible policy search (COPS) approach, which is based on the natural gradient, compatible value function approximation and entropy regularization, bringing these ideas to deep reinforcement learning combined with the already successful trust region updates we saw in the last chapter in section 2.4.2. We will show that when using the natural gradient with compatible value function approximation that the natural gradient is the optimal (and not just the approximate) solution to a trust region problem for log-linear models. Furthermore, we will show that using compatible value function approximation, we can derive a similar algorithm to TRPO, but with closed form policy update equations and less approximations. Similar to TRPO this algorithm is scalable and can optimize non-linear policies up to tens of thousand of parameters. In empirical experiments on challenging POMDPs we investigate the application of policy search methods for POMDPs and highlight the advantage of the additional entropy regularization constraint of our method, comparing them with TRPO, TRPO where an entropy regularization term is added to the returns and other common approaches.

3.2 Preliminaries

To derive our algorithm we reformulate the objective of the step-based policy search framework, introduced in Section 2.2.2, slightly

$$J(\pi_{\boldsymbol{\theta}}) = \int \int \mu(\mathbf{s})\, \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\, Q^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\mathbf{s},\mathbf{a})\, d\mathbf{s}\, d\mathbf{a} \tag{3.1}$$

where $\mu(\mathbf{s})$ denotes the stationary state distribution of the MDP with policy π and $Q^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\mathbf{s},\mathbf{a})$ denotes the future accumulated reward values of the current policy $\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{s},\mathbf{a})$. To bound the difference between the new and old policy trust region optimization methods such as relative entropy policy search [52] (REPS) use the KL divergence, introduced in Section 2.2.2. This can be formulated as finding a policy which maximizes the expected reward while satisfying the KL constraint

$$\begin{aligned} \arg\max_{\pi_{\boldsymbol{\theta}}} J(\pi_{\boldsymbol{\theta}}) &= \int \int \mu(\mathbf{s})\, \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\, Q^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\mathbf{s},\mathbf{a})\, d\mathbf{s}\, d\mathbf{a} \\ \text{subject to } \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}&\left[KL\left(\pi_{\boldsymbol{\theta}}(\,\cdot\,|\,\mathbf{s}) \,||\, \pi_{\boldsymbol{\theta}_{\text{old}}}(\,\cdot\,|\,\mathbf{s})\right)\right] < \epsilon \end{aligned} \tag{3.2}$$

where ϵ is a hyperparameter, defining the maximum allowed KL difference in one update step. This KL bound has two good properties. On the one hand it prevents unstable updates, i.e., the new policy will not move too far towards areas it has not seen before. On the other hand it prevents the updates from being too greedy which would decrease exploration too aggressively and lead to premature convergence issues. There are several ways to solve this optimization problem, which we will show below.

3.2.1 Relative Entropy Policy Search

REPS was the first method to introduce this trust region optimization method to the policy search framework [52]. The approach of REPS to solve these class of problems is to disregard the parametrization and distribution of the policy and first solve the constraint optimization problem for any class of distributions. Using the method of Lagrangian multipliers [11] REPS reformulates the optimization problem to a dual problem

$$L(\eta) = \eta\epsilon + \eta \log \int \int \pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}|\mathbf{s})\, \exp\left(\frac{Q^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\mathbf{s},\mathbf{a})}{\eta}\right) d\mathbf{s}\, d\mathbf{a} \tag{3.3}$$

where $\eta > 0$ is a Lagrangian multiplier. We can approximate the expectation over $\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}|\mathbf{s})$ using Monte-Carlo sampling, resulting in

$$L(\eta) \approx \eta\epsilon + \eta \log \sum_{i=0}^{N} \pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_i|\mathbf{s}_i)\, \exp\left(\frac{Q^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\mathbf{s}_i,\mathbf{a}_i)}{\eta}\right). \tag{3.4}$$

Thus, we can disregard the parametrization of the policy and only need to be able to sample from it. Minimizing this log-sum-exp term gives us the Lagrangian multiplier η. This η is used subsequently for the computation of the optimal policy, which is given by

$$\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \propto \pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}|\mathbf{s}) \exp\left(\frac{Q^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\mathbf{s},\mathbf{a})}{\eta}\right), \tag{3.5}$$

i.e., we reweight the old policy by a softmax transform of the Q-function, where η sets the temperature of the softmax transformation according to the KL-constraint. To obtain the new policy the optimal solution is projected back to its parametric policy class, which corresponds to fitting a weighted maximum estimate [15]. More variants of this algorithm can be found in the same survey.

3.2.2 Natural Gradient

The natural gradient [4] was initially motivated that the plain gradient is not invariant to rescaling and transformation of the parameters and has been applied in several policy gradient methods, such as [28, 53]. However, we will see later that the natural gradient is actually the approximate solution to a trust region update and see that when used with compatible value function approximation that it is in fact the optimal solution for a given trust region problem. To avoid the problems of the plain gradient Amari proposed to use a metric which is invariant on the space of parameters of probability distributions. One popular choice for this metric is the Fisher metric, however it is possible to choose any Riemannian metric for the natural gradient [78]. The idea of the natural gradient is to follow the steepest direction with respect to that metric instead of following the directly the gradient in parameter space. Applied to the policy gradient framework we follow $\nabla_{\boldsymbol{\theta}} J_{\text{NAC}}(\pi_{\boldsymbol{\theta}})$ instead of the vanilla gradient $\nabla_{\boldsymbol{\theta}} J_{\text{PG}}(\pi_{\boldsymbol{\theta}})$, which is given by

$$\nabla_{\boldsymbol{\theta}} J_{\text{NAC}}(\pi_{\boldsymbol{\theta}}) = \mathbf{F}^{-1}(\pi_{\boldsymbol{\theta}})\, \nabla_{\boldsymbol{\theta}} J_{\text{PG}}(\pi_{\boldsymbol{\theta}}) \tag{3.6}$$

where $\mathbf{F}$ is the Fisher information matrix, defined as

$$\mathbf{F}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})^T\right]. \tag{3.7}$$

The resulting policy update is

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta} + \boldsymbol{\alpha} \nabla_{\boldsymbol{\theta}} J_{\text{NAC}} = \boldsymbol{\theta} + \boldsymbol{\alpha} \mathbf{F}^{-1}(\pi_{\boldsymbol{\theta}})\, \nabla_{\boldsymbol{\theta}} J_{\text{PG}}(\pi_{\boldsymbol{\theta}}) \tag{3.8}$$

where $\boldsymbol{\alpha}$ denotes the learning rate. The link to trust region optimization can be now drawn by using the knowledge that the Kullback-Leibler divergence can be actually approximated by the Fisher information matrix (2nd order Taylor approximation)

$$KL\left(\pi_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) || \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\right) \approx \Delta\boldsymbol{\theta}^T \mathbf{F}(\pi_{\boldsymbol{\theta}}) \Delta\boldsymbol{\theta} \tag{3.9}$$

for some $\boldsymbol{\theta} + \Delta\boldsymbol{\theta}$ around $\boldsymbol{\theta}$. Thus, the natural gradient can be obtained by an approximate trust region update and can be seen as the update direction that is most correlated to $\nabla_{\boldsymbol{\theta}} J_{\text{PG}}(\pi_{\boldsymbol{\theta}})$ and has a limited approximative KL

$$\arg\max_{\Delta\boldsymbol{\theta}} \Delta\boldsymbol{\theta}^T \nabla_{\boldsymbol{\theta}} J_{\text{PG}}(\pi_{\boldsymbol{\theta}}) \quad \text{s.t.} \quad KL\left(\pi_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) || \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\right) \approx \Delta\boldsymbol{\theta}^T \mathbf{F}(\pi_{\boldsymbol{\theta}}) \Delta\boldsymbol{\theta} \leq \epsilon \tag{3.10}$$

according to [15]. The result of this optimization is given by

$$\Delta\boldsymbol{\theta} = \eta^{-1} \mathbf{F}^{-1}(\pi_{\boldsymbol{\theta}})\, \nabla_{\boldsymbol{\theta}} J_{\text{PG}} = \eta^{-1} \nabla_{\boldsymbol{\theta}} J_{\text{NAC}}(\pi_{\boldsymbol{\theta}}) \tag{3.11}$$

where η is a Lagrangian multiplier and can be seen as the learning rate. With that we end up with the same equation as (3.8).

3.2.3 Compatible Value Function Approximation

Compatible value function approximation [74] is the idea to use a critic based on special features, which are the gradient of the log-policy, to approximate the reward, which are plugged into the policy gradient equations (see Section 2.2.2). Mathematically, we have a function $\tilde{F}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s},\mathbf{a})$, which approximates the Monte-Carlo estimates $Q^{\pi_\theta}(\mathbf{s},\mathbf{a})$ of the returns as following

$$Q^{\pi_\theta}(\mathbf{s},\mathbf{a}) \approx \tilde{F}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s},\mathbf{a}) = \phi(\mathbf{s},\mathbf{a})^T\mathbf{w} \qquad \phi(\mathbf{s},\mathbf{a}) = \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \tag{3.12}$$

where $\mathbf{w}$ is a vector containing all the weights of $\tilde{F}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s},\mathbf{a})$. Similar to applying a baseline to the policy gradient, according to [28] this method leads to smaller variance while keeping the gradient unbiased, when $\mathbf{w}$ of the approximation is chosen such that it minimizes following least squares problem

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathbb{E}_{\mu(\mathbf{s}),\pi_\theta(\mathbf{a}|\mathbf{s})}\Big[\big(Q^{\pi_\theta}(\mathbf{s},\mathbf{a}) - \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})^T\mathbf{w}\big)^2\Big]. \tag{3.13}$$

When using this compatible value function in combination with the natural gradient it has been shown in [53] that the inverse of the Fisher information matrix cancels with the matrix spanned by the compatible features resulting in a very simple formulation for the natural gradient

$$\nabla_\theta J_{\text{NAC}}(\pi_\theta) = \mathbf{w}^* \tag{3.14}$$

and the update rule

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta} + \eta^{-1}\mathbf{w}^*. \tag{3.15}$$

This can be also be explained if we look at $\mathbf{w}^*$ given by the least squares solution

$$\begin{aligned}\mathbf{w}^* &= (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{Q} \\ &= \left(\sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)^T\right)^{-1}\sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)Q^{\pi_\theta}(\mathbf{s}_i,\mathbf{a}_i) \\ &= \mathbf{F}^{-1}\nabla_\theta J_{\text{PG}}(\pi_\theta).\end{aligned} \tag{3.16}$$

which contains the inverse of the fisher information matrix. Furthermore, it has been observed that the compatible value function approximation is in fact an approximation for the advantage function and not for the Q-function as it has zero mean [53]. Thus, it would be better not to regress directly on the Q-values but on the advantage values. But, this would need the knowledge of the actual value function to compute the advantage. This value function is difficult to obtain as there is no compatible feature basis for the corresponding value function. However, we will see later that there is more structure in the compatible features, allowing us to extract features for the value and Q-function individually.

3.2.4 Entropy Regularization

Some recent trust region approaches use an additional regularization term for the entropy of the new policy to ensure exploration and avoid premature convergence, such as model-free trajectory optimization for reinforcement learning [3] (MOTO), model-based relative entropy stochastic search [2] (MORE). This results in following optimization problem

$$\begin{aligned}\arg\max_{\pi_\theta} \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[\int \pi_\theta(\mathbf{a}|\mathbf{s})\, Q^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a})\, d\mathbf{a}\right] \\ \text{subject to } \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\Big[KL\big(\pi_\theta(\cdot\,|\,\mathbf{s}) \,||\, \pi_{\theta_{\text{old}}}(\cdot\,|\,\mathbf{s})\big)\Big] < \epsilon \\ \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\Big[H\big(\pi_\theta(\cdot\,|\,\mathbf{s})\big) - H\big(\pi_{\theta_{\text{old}}}(\cdot\,|\,\mathbf{s})\big)\Big] < \beta\end{aligned} \tag{3.17}$$

where the second constraint bounds the loss in entropy between the new and old policy in one update step and β is a constant entropy reduction hyperparameter. Note, that in Equation (3.17) and in the following equations of this thesis we use a slightly different notation instead of using the one of the original papers: $\mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}[H(\pi_\theta(\cdot\,|\,\mathbf{s}))] > \beta$, where β is set in each iteration to the entropy of the previous policy minus some constant factor. Once again one can obtain a closed form solution for the new policy using the methods of Lagrangian multipliers and given by

$$\pi_\theta(\mathbf{a}|\mathbf{s}) \propto \pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})^{\frac{\eta}{\eta+\omega}} \exp\left(\frac{Q^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a})}{\eta+\omega}\right) \tag{3.18}$$

where ω is an additional Lagrangian multiplier, controlling the entropy of the new policy. In general, entropy regularization has also been applied to other non trust region approaches, such as [43, 47], by augmenting the reward with an additional entropy term

$$J_{\text{ENT}}(\pi) = \mathbb{E}_{\mu_0(\mathbf{s}_0),\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t),\pi(\mathbf{a}_t|\mathbf{s}_t)}\left[\sum_{t=0}^{H}\gamma^t r(\mathbf{s}_t,\mathbf{a}_t) + \alpha\, H\left(\pi(\,\cdot\,|\,\mathbf{s_t})\right)\right] \tag{3.19}$$

where α is a hyperparameter, controlling the relative importance between the reward and entropy. There is also an implementation of a variant of TRPO in the OpenAI baselines [16] with entropy regularization. In our experiments we will show the advantages of bounding the entropy loss in an additional constraint for the trust region optimization problem instead of augmenting an additional entropy term.

3.3 Compatible Policy Search with Natural Parameters

In this section we will put all the related work together and analyze the natural gradient equations in more detail for distributions of the exponential family. We will first start with distributions, which are log-linear in its parameters, such as softmax distributions and then move on to distributions, which are log-non-linear such as neural network policies. In this analysis we will see an important connection between the natural gradient and trust region optimization for distributions, which are log-linear in its parameters: both are equivalent if we use the natural parametrization of the exponential distribution in combination with compatible value function approximation. This is an important insight as the natural gradient provides the optimal solution for a given trust region, not just an approximation, which is commonly believed. Furthermore, the use of compatible value function approximation has potentially several advantages in terms of variance reduction, which can not be achieved with the plain Monte-Carlo estimates. Finally, we will present our new algorithm COPS and show how it can be practically implemented for the discrete action case.

3.3.1 Equivalence of Natural Gradient and Trust Region Optimization

We will first start with softmax distributions that are log-linear in the parameters and commonly used in the discrete action case. Afterwards we will move on and extend our results to non-linear softmax distributions, for example a neural network, where the last layer is a softmax. Softmax distributions can be represented as

$$\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) = \frac{\exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)}{\int \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}} \tag{3.20}$$

where $\psi(\mathbf{s},\mathbf{a})$ are the features and $\boldsymbol{\theta}$ are the parameters of the log-linear distribution. On a sidenote the Gaussian distribution can be also represented in this log-linear form using the natural parametrization of the exponential family; however, this representation is not commonly used for Gaussian distributions. Typically, the Gaussian is parametrized by the mean $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$. In natural parametrization the Gaussian is parametrized by $\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}^{-1}$, where $\boldsymbol{\Sigma}^{-1}$ is also known as the precision matrix. In this thesis we will focus on the discrete case, though.

To continue with our analysis, we will investigate the exact form of the compatible features for log-linear models starting with (3.12):

$$\begin{aligned}
\boldsymbol{\phi}(\mathbf{s},\mathbf{a}) &= \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \\
&= \nabla_{\boldsymbol{\theta}} \log \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right) - \nabla_{\boldsymbol{\theta}} \log \int \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a} \\
&= \nabla_{\boldsymbol{\theta}}\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta} - \frac{\nabla_{\boldsymbol{\theta}} \int \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}}{\int \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}} \\
&= \psi(\mathbf{s},\mathbf{a}) - \frac{\int \psi(\mathbf{s},\mathbf{a}) \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}}{\int \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}} \\
&= \boldsymbol{\psi}(\mathbf{s},\mathbf{a}) - \int \boldsymbol{\psi}(\mathbf{s},\mathbf{a})\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\, d\mathbf{a} = \boldsymbol{\psi}(\mathbf{s},\mathbf{a}) - \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s})}[\boldsymbol{\psi}(\mathbf{s},\cdot)]\,.
\end{aligned} \tag{3.21}$$

It can be observed that the compatible feature space is zero mean with respect to the action distribution, i.e.,

$$\int \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\boldsymbol{\phi}(\mathbf{s},\mathbf{a})\, d\mathbf{a} = \int \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\boldsymbol{\psi}(\mathbf{s},\mathbf{a})\, d\mathbf{a} - \int \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\mathbb{E}_{\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s})}[\boldsymbol{\psi}(\mathbf{s},\cdot)]\, d\mathbf{a} = 0. \tag{3.22}$$

This fits to the insight that the compatible approximation function $\tilde{F}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s},\mathbf{a})$ is an advantage function. Furthermore, the structure of the features suggests that the advantage function is composed of a term for the Q-function and for the value function, i.e.,

$$\tilde{F}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s},\mathbf{a}) = \underbrace{\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\mathbf{w}}_{\tilde{Q}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s},\mathbf{a})} - \mathbb{E}_{\pi_\theta(\cdot|\mathbf{s})}\left[\boldsymbol{\psi}(\mathbf{s},\cdot)^T\mathbf{w}\right] = \tilde{Q}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s},\mathbf{a}) - \underbrace{\mathbb{E}_{\pi_\theta(\cdot|\mathbf{s})}\left[\tilde{Q}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s},\cdot)\right]}_{\tilde{V}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s})}. \tag{3.23}$$

We now can use the compatible advantage function in our trust region optimization problem given in (3.2) for $Q^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a})$. From REPS we know that the resulting optimal policy follows equation (3.5) and is given by:

$$\begin{aligned}
\pi_\theta(\mathbf{a}|\mathbf{s}) &\propto \pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})\ \exp\left(\frac{\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a})}{\eta}\right) \\
&\propto \pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})\ \exp\left(\frac{\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\mathbf{w} - \mathbb{E}_{\pi_{\theta_{\text{old}}}(\cdot|\mathbf{s})}\left[\boldsymbol{\psi}(\mathbf{s},\cdot)^T\mathbf{w}\right]}{\eta}\right) \\
&\propto \frac{\exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}_{\text{old}}\right)}{\int \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}_{\text{old}}\right)\, d\mathbf{a}} \exp\left(\frac{\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\mathbf{w}}{\eta}\right) \exp\left(\frac{\mathbb{E}_{\pi_{\theta_{\text{old}}}(\cdot|\mathbf{s})}\left[\boldsymbol{\psi}(\mathbf{s},\cdot)^T\mathbf{w}\right]}{\eta}\right)^{-1} \\
&\propto \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}_{\text{old}} + \frac{\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\mathbf{w}}{\eta}\right) \\
&\propto \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\left(\boldsymbol{\theta}_{\text{old}} + \eta^{-1}\mathbf{w}\right)\right).
\end{aligned} \tag{3.24}$$

Note that the value function part of $\tilde{F}_{\mathbf{w}}^{\pi_\theta}(\mathbf{s},\mathbf{a})$ is only dependent from the state and not the action. Therefore, we can see it as a additional normalization constant of the policy. Looking at the result of our analysis, we see that if we use the the natural parametrization of the exponential family distribution in combination with compatible function approximation, we can get directly to a parametric update for the new policy of the form

$$\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{old}} + \eta^{-1}\mathbf{w}. \tag{3.25}$$

We observe that no projection to the parameter space of the policy is required, since the new policy has the same structure as the nominator of a softmax. Thus, the resulting new policy is just

$$\pi_\theta(\mathbf{a}|\mathbf{s}) = \frac{\exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\left(\boldsymbol{\theta}_{\text{old}} + \eta^{-1}\mathbf{w}\right)\right)}{\int \exp\left(\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\left(\boldsymbol{\theta}_{\text{old}} + \eta^{-1}\mathbf{w}\right)\right)\, d\mathbf{a}}. \tag{3.26}$$

Furthermore, this suggested update rule is equivalent to the natural gradient update (3.15). This insight has an important implication: the natural gradient is the optimal solution for a given trust region problem and not just an approximation. However, this statement only holds if we use natural parameters and compatible value function approximation. We can also see, that only the Q-function part of the compatible function approximation is needed for the update.

3.3.2 Updates with Entropy Regularization

For the trust region problem with entropy regularization given in (3.17) we can do the same analysis. This results in following equation for the optimal policy given the compatible value function approximation

$$\pi_\theta(\mathbf{a}|\mathbf{s}) \propto \pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})^{\frac{\eta}{\eta+\omega}}\ \exp\left(\frac{\boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\mathbf{w} - \mathbb{E}_{\pi_{\theta_{\text{old}}}(\cdot|\mathbf{s})}\left[\boldsymbol{\psi}(\mathbf{s},\cdot)^T\mathbf{w}\right]}{\eta+\omega}\right) \propto \exp\left(\psi(\mathbf{s},\mathbf{a})^T\frac{\eta\boldsymbol{\theta}_{\text{old}} + \mathbf{w}}{\eta+\omega}\right) \tag{3.27}$$

which suggest the update rule

$$\boldsymbol{\theta} = \frac{\eta\boldsymbol{\theta}_{\text{old}} + \mathbf{w}}{\eta+\omega}. \tag{3.28}$$

If we compare this update rule to the standard natural gradient, we see that in the entropy regularization case, we diminish the old parameters by the factor $\eta/(\eta+\omega)$.

3.3.3 Compatible Approximation for Deep Neural Networks

Having seen that the natural gradient and trust region optimization are equivalent when used with natural parameters, compatible value function approximation and log-linear models, we want now to extend this insights for complex non-linear models, such as deep neural networks. Therefore, we introduce additional non-linear parameters $\boldsymbol{\beta}$ to our features, i.e., $\boldsymbol{\psi}(\mathbf{s},\mathbf{a}) = \boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})$ and

$$\pi_{\boldsymbol{\theta},\boldsymbol{\beta}}(\mathbf{a}|\mathbf{s}) = \frac{\exp\left(\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)}{\int \exp\left(\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}}. \tag{3.29}$$

Hence, in this model we also consider the non-linear parameters $\boldsymbol{\beta}$, from which the feature basis is dependent on, while in the log-linear model it is assumed that we are just given some fixed feature basis. Especially, we want to learn these non-linear parameters $\boldsymbol{\beta}$.

From the deep neural network perspective $\boldsymbol{\theta}$ represent the weights of the last softmax layer and the new introduced parameters $\boldsymbol{\beta}$ represent all the weights and biases of the previous layers, i.e., $\boldsymbol{\beta} = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \cdots, \mathbf{W}^{(k-1)}, \mathbf{b}^{(k-1)})$, where k is the number of layers of the network. $\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})$ describes in this case the input of the last softmax layer, which is easily seen dependent on the weights of the previous layers, hence dependent on $\boldsymbol{\beta}$. This can be seen if we write $\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})$ as input of the last layer

$$\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a}) = f^{(k-1)}\left(\mathbf{W}^{(k-1)^T} f^{(k-2)}\left(\cdots f^{(2)}\left(\mathbf{W}^{(2)^T} f^{(1)}\left(\mathbf{W}^{(1)^T}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right)\cdots\right) + \mathbf{b}^{(k-1)}\right) \tag{3.30}$$

To obtain the update rule for the non-linear parameters $\boldsymbol{\beta}$, we have to compute again the compatible features, similar to (3.21), in this case only with respect to $\boldsymbol{\beta}$

$$\begin{aligned}
\nabla_{\boldsymbol{\beta}} \log \pi_{\boldsymbol{\theta},\boldsymbol{\beta}}(\mathbf{a}|\mathbf{s}) &= \nabla_{\boldsymbol{\beta}} \log \exp\left(\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right) - \nabla_{\boldsymbol{\beta}} \log \int \exp\left(\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a} \\
&= \frac{\partial \boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})}{\partial \boldsymbol{\beta}}\boldsymbol{\theta} - \frac{\nabla_{\boldsymbol{\beta}} \int \exp\left(\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}}{\int \exp\left(\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}} \\
&= \frac{\partial \boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})}{\partial \boldsymbol{\beta}}\boldsymbol{\theta} - \frac{\int \frac{\partial \boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})}{\partial \boldsymbol{\beta}}\boldsymbol{\theta} \exp\left(\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}}{\int \exp\left(\boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})^T\boldsymbol{\theta}\right)\, d\mathbf{a}} \\
&= \frac{\partial \boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})}{\partial \boldsymbol{\beta}}\boldsymbol{\theta} - \int \frac{\partial \boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})}{\partial \boldsymbol{\beta}}\boldsymbol{\theta}\, \pi_{\boldsymbol{\theta},\boldsymbol{\beta}}(\mathbf{a}|\mathbf{s})\, d\mathbf{a} \\
&= \mathbf{J}(\mathbf{s},\mathbf{a})\boldsymbol{\theta} - \mathbb{E}_{\pi_{\boldsymbol{\theta},\boldsymbol{\beta}}(\cdot|\mathbf{s})}\left[\mathbf{J}(\mathbf{s},\cdot)\boldsymbol{\theta}\right]
\end{aligned} \tag{3.31}$$

where $\mathbf{J}(\mathbf{s},\mathbf{a})$ denotes the Jacobian matrix of the feature basis $\psi_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})$, i.e., $\mathbf{J}(\mathbf{s},\mathbf{a}) = \partial \boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})/\partial \boldsymbol{\beta}$. From the implementation point of view the Jacobian can be easily computed using auto-differentiation tools, such as Theano. For the further derivation we will denote $C_{\boldsymbol{\theta},\boldsymbol{\beta}}(\mathbf{s},\mathbf{a}) = \boldsymbol{\psi}_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a})\boldsymbol{\theta}$ as the energy function of the softmax distribution with $\nabla_{\boldsymbol{\beta}} C_{\boldsymbol{\theta},\boldsymbol{\beta}}(\mathbf{s},\mathbf{a}) = J(\mathbf{s},\mathbf{a})\boldsymbol{\theta}$. Furthermore, we denote $\mathbf{w}_{\boldsymbol{\beta}}$ as the weights of the compatible approximation feature vector that are responsible for $\boldsymbol{\beta}$. To derive now the update rule for the new policy we will only concentrate on the non-linear parameters $\boldsymbol{\beta}$, since we already have an update rule for the linear parameters $\boldsymbol{\theta}$, which does not change if we add $\boldsymbol{\beta}$. For the entropy regularization case we can write the new policy as

$$\begin{aligned}
\pi_{\boldsymbol{\theta},\boldsymbol{\beta}}(\mathbf{a}|\mathbf{s}) &\propto \pi_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\mathbf{a}|\mathbf{s})^{\frac{\eta}{\eta+\omega}} \exp\left(\frac{\mathbf{J}(\mathbf{s},\mathbf{a})\boldsymbol{\theta}_{\text{old}}\mathbf{w}_{\boldsymbol{\beta}} - \mathbb{E}_{\pi_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\cdot|\mathbf{s})}\left[\mathbf{J}(\mathbf{s},\cdot)\boldsymbol{\theta}_{\text{old}}\mathbf{w}_{\boldsymbol{\beta}}\right]}{\eta+\omega}\right) \\
&\propto \left(\frac{\exp\left(C_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\mathbf{s},\mathbf{a})\right)}{\int \exp\left(C_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\mathbf{s},\mathbf{a})\right)\, d\mathbf{a}}\right)^{\frac{\eta}{\eta+\omega}} \exp\left(\frac{\nabla_{\boldsymbol{\beta}_{\text{old}}} C_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\mathbf{s},\mathbf{a})\mathbf{w}_{\boldsymbol{\beta}} - \mathbb{E}_{\pi_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\cdot|\mathbf{s})}\left[\nabla_{\boldsymbol{\beta}_{\text{old}}} C_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\mathbf{s},\cdot)\mathbf{w}_{\boldsymbol{\beta}}\right]}{\eta+\omega}\right) \\
&\propto \exp\left(\frac{\eta}{\eta+\omega} C_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\mathbf{s},\mathbf{a})\right) \exp\left(\frac{\nabla_{\boldsymbol{\beta}_{\text{old}}} C_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\mathbf{s},\mathbf{a})\mathbf{w}_{\boldsymbol{\beta}}}{\eta+\omega}\right) \\
&\propto \exp\left(\frac{\eta C_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\mathbf{s},\mathbf{a}) + \nabla_{\boldsymbol{\beta}_{\text{old}}} C_{\boldsymbol{\theta}_{\text{old}},\boldsymbol{\beta}_{\text{old}}}(\mathbf{s},\mathbf{a})\mathbf{w}_{\boldsymbol{\beta}}}{\eta+\omega}\right).
\end{aligned} \tag{3.32}$$

Thus, we realize that the new energy function $C_{\theta,\beta}(\mathbf{s},\mathbf{a})$ is given by

$$\begin{aligned} C_{\theta,\beta}(\mathbf{s},\mathbf{a}) &= \frac{\eta C_{\theta_{\text{old}},\beta_{\text{old}}}(\mathbf{s},\mathbf{a}) + \nabla_{\beta_{\text{old}}} C_{\theta_{\text{old}},\beta_{\text{old}}}(\mathbf{s},\mathbf{a})\mathbf{w}_\beta}{\eta+\omega} \\ &= \frac{\eta}{\eta+\omega}\left(C_{\theta_{\text{old}},\beta_{\text{old}}}(\mathbf{s},\mathbf{a}) + \nabla_{\beta_{\text{old}}} C_{\theta_{\text{old}},\beta_{\text{old}}}(\mathbf{s},\mathbf{a})\eta^{-1}\mathbf{w}_\beta\right) \\ &= \frac{\eta}{\eta+\omega}\left(C_{\theta_{\text{old}},\beta_{\text{old}}}(\mathbf{s},\mathbf{a}) + \nabla_{\beta_{\text{old}}} C_{\theta_{\text{old}},\beta_{\text{old}}}(\mathbf{s},\mathbf{a})\left((\boldsymbol{\beta}_{\text{old}} + \eta^{-1}\mathbf{w}_\beta) - \boldsymbol{\beta}_{\text{old}}\right)\right) \quad (3.33) \\ &\approx \frac{\eta}{\eta+\omega} C_{\theta_{\text{old}},\beta_{\text{old}}+\eta^{-1}\mathbf{w}_\beta}(\mathbf{s},\mathbf{a}) \quad (3.34) \end{aligned}$$

where (3.33) represents the first order Taylor approximation of (3.34) at $\boldsymbol{\beta}_{\text{old}}$. This suggest the update rule

$$\boldsymbol{\beta} = \boldsymbol{\beta}_{\text{old}} + \eta^{-1}\mathbf{w}_\beta \quad (3.35)$$

since the rescaling of the energy function is implemented by the update of the parameters $\boldsymbol{\theta}$ and hence can be ignored for the update of $\boldsymbol{\beta}$. Note that the update rule is again the natural gradient. With that we can conclude that the natural gradient is an approximate trust region solution for the non-linear parameters $\boldsymbol{\beta}$. In the next section we will show how all these steps of our analysis can be put together in an algorithm and practically implemented for deep neural networks.

3.4 Compatible Policy Search in Practice

In this section we will present two practical algorithms based on our ideas above, which differ in the way how the weights $\mathbf{w}$ of the compatible value function are computed (step 2). Otherwise both algorithm follow the steps we showed in Section 3.3:

1. Compute the compatible feature base $\boldsymbol{\phi}(\mathbf{s},\mathbf{a})$ for our current policy $\pi_{\theta,\beta}(\mathbf{a}|\mathbf{s})$.
2. Use $\boldsymbol{\phi}(\mathbf{s},\mathbf{a})$ to compute $\mathbf{w} = \left(\mathbf{w}_\theta, \mathbf{w}_\beta\right)$ by solving $\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathbb{E}_{\mu(\mathbf{s}),\pi_{\theta,\beta}(\mathbf{a}|\mathbf{s})}\left[\left(A^{\pi_{\theta,\beta}}(\mathbf{s},\mathbf{a}) - \boldsymbol{\phi}(\mathbf{s},\mathbf{a})^T\mathbf{w}\right)^2\right]$ or any equivalent equation.
3. Compute $\tilde{F}_{\mathbf{w}}^{\pi_{\theta,\beta}}(\mathbf{s},\mathbf{a})$ using $\mathbf{w}$ and put this into our optimization problem. Solve the trust region optimization problem using the dual obtained using the methods of Lagrangian multiplier to compute η and ω.
4. Use our derived update rules to update $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ using η and ω.

In Section 3.3 we also saw that the compatible value function approximation actually denotes an advantage function and not the Q-function. Therefore, the weights $\mathbf{w}$ of the compatible value function approximation can not be directly estimated by least squares or TD learning from the Monte-Carlo estimates as it would require the knowledge of the value function. It has been shown though that it can even work when using the compatible value function approximation with the Monte-Carlo estimates of the Q-function [74]. However, we propose to use a slightly different value function approximation to estimate the Q-values

$$\tilde{F}_{\mathbf{w},\mathbf{v}}^{\pi_{\text{old}}}(\mathbf{s},\mathbf{a}) = \boldsymbol{\psi}(\mathbf{s},\mathbf{a})^T\mathbf{w} + \mathbb{E}_{\pi_{\text{old}}(\cdot|\mathbf{s})}\left[\psi(\mathbf{s},\cdot)^T\mathbf{v}\right] \quad (3.36)$$

where $\mathbf{v}$ denotes the parameters of the action independent part of the Q-function, which is not used for updating the policy, but needed to obtain robust estimates of $\mathbf{V}$. This is equivalent to viewing the two parts of the original compatible value function as individual terms with different parameters and hence also compatible to the policy. Moreover, it is still unbiased as is it equivalent to adding another baseline. With this we can learn $\tilde{F}_{\mathbf{w},\mathbf{v}}^{\pi_{\text{old}}}(\mathbf{s},\mathbf{a})$ either directly via regression (least squares or any gradient descent method) using the Monte-Carlo estimates to minimize the quadratic loss or via linear TD learning methods. For this thesis we will use gradient descent with a additional regularization term, but using TD learning methods might be more preferable since these methods have lower variance and allow sample reuse of old transitions using importance weighting.

The second variant is first to learn the action independent value function part using the Monte-Carlo estimates and use this as a baseline to compute the advantages from which we subsequently can compute $\mathbf{w}$ using the original compatible value function. Doing so we can use (3.16) to compute $\mathbf{w}$, which uses the inverse of the Fisher information matrix and the standard policy gradient. This approach allows us to use the computation techniques of TRPO, such as the computation of $\mathbf{w}$ using conjugate gradient or the computation of the Fisher information matrix using the Hessian of the KL divergence instead of using the covariance matrix of the gradients. Note that it is also possible to use a different feature base to learn the value function, however it may be difficult to find such a good feature base. To obtain $\mathbf{v}$ of the baseline one can again either learn $V_{\mathbf{v}}^{\pi_{\text{old}}}(\mathbf{s})$ using regression on the direct estimates of the return or using TD methods.

The pseudocode for both variants is presented in the appendix in Algorithm 1 and Algorithm 2.

3.5 Connections with Previous Algorithms

Natural Actor Critic [53]: We showed in Section 3.3.1 that the natural gradient is not only an approximation as previously believed, but the optimal solution for trust region optimization problems for log-linear models, when used with compatible value function approximation. Furthermore, we showed how the ideas of natural actor critic can be applied to deep neural networks and obtain simple updates rules, that are very similar to the standard natural gradient.

REPS: Our approach uses the same approach as REPS to solve the trust region optimization problem by solving the dual. However, by using COPS with the right parametrization one can obtain closed form policy update rules without needing to project the result back into the policy's parametric class as in REPS. This makes life much easier when learning policies parametrized by neural networks, since we can update the parameters directly using COPS and don't need any projection.

TRPO: Our derived COPS conjugate gradient variant (Algorithm 2) is very similar to TRPO, as it also inverts the Fisher information matrix. One can even obtain the TRPO algorithm (with a baseline using the compatible features) when using COPS conjugate gradient without entropy regularization. This gives us the insight that one can derive TRPO from a different perspective and end up with a similar algorithm. Moreover, by using our derivations we see that TRPO is actually computing the natural gradient with compatible value function approximation. The big difference is that our algorithm uses entropy regularization embedded as an additional constraint in the trust region optimization problem. This constraint prevents too fast entropy drops in the learned policy and premature convergence, ultimately leading to better performance. Moreover, the updates of COPS are more precise, since we use the knowledge about the exponential family to derive two different update rules, one for the log-linear, which does not need line search, and one for the non-log-linear parameters. While in TRPO there is only one update rule with line search for all parameters. Hence, the update for the log-linear parameters will be suboptimal. All in all, following our derivation we end up with a very similar algorithm to TRPO, but from a different perspective, which even allows an off-policy setting.

3.6 Experiments

In this section we will present our empirical results of our method, where we designed the experiments to answer following research questions:

1. How does COPS perform compared to other related approaches? Especially, how does entropy regularization embedded in the trust region optimization affect the performance of the algorithm?

2. Is there any significant difference between the two variants of COPS? And is there any gain using only the action independent part of the Q-function of the estimate $\tilde{F}^{\pi_{\text{old}}}_{\mathbf{w},\mathbf{v}}(\mathbf{s},\mathbf{a})$ for the update of the policy compared to using $\tilde{F}^{\pi_{\text{old}}}_{\vec{w}}(\mathbf{s},\mathbf{a})$? How much is the gain between using the advantage instead of the Q-values?

To answer the first question we compared COPS with other trust region algorithms, such as TRPO with the same KL-bound and with or without additional augmented entropy term (see Section 3.2.4). For this question, we were able to show that entropy regularization embedded in the trust region optimization has an impact on the performance. For the second question we compared both variants of COPS and compared the performance of COPS using $\mathbf{w}$ from the full compatible features to using $\mathbf{w}$ from the action independent part of the compatible features for the update of the parameters. For both research questions we chose to investigate POMDP environments, since on the one hand they are the focus of the thesis. On the other hand we hope that by using challenging POMDP environments, where a lot of exploration is needed, we can show the advantages of our approach, such as entropy regularization to prevent premature convergence.

3.6.1 Field Vision Rock Sample

For our first experiments we used an environment with discrete actions and partial observations, named FieldVisionRockSample (FVRS), that was used in [56]. FVRS is a variant of the RockSample task, which is a common benchmark task for POMDP algorithms and was introduced in [69]. In an FVRS(n,k) environment we have a robot that has to explore a $n \times n$ grid containing k different rocks with unknown random initialized values. To perceive the values of the rocks, which are either good or bad, the robot uses a noisy sensor. The noise of the sensor depends on the distance between the robot and the rock checked. In FVRS the robot perceives all rocks at the same time, while in standard RockSample it can only perceive one rock in one timestep. Thus in FVRS, there are in total 2^R observations, where R is the number of rocks in the environment, while the standard RockSample has only three observations (either good or bad if a check action was taken and no observation for all other actions). If the robot picks up a good rock it gets a reward; if it picks up a bad rock it gets a punishment. Furthermore, the robot receives a reward if it leaves the grid on the right

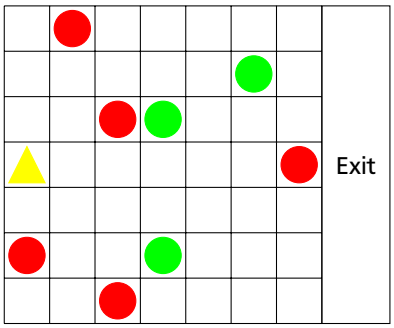


Figure 3.1.: Sketch of the RockSample $(7, 8)$ environment. The agent (yellow triangle) is in a grid and has to collect the valuable rocks (green), while avoiding the bad rocks (red) before leaving the grid to the right side. However, the agent does not know the rock value and only has a noisy sensor to sense them. This noise depends on the distance between the rock and the agent.

Algorithm	$(5, 5)$ full	$(5, 5)$ noise	$(5, 7)$ full	$(5, 7)$ noise	$(7, 8)$ full	$(7, 8)$ noise
COPS conjugate gradient	**2.14 ± 0.08**	**1.94 ± 0.12**	**2.66 ± 0.14**	**2.45 ± 0.10**	**1.66 ± 0.17**	**1.32 ± 0.15**
TRPO	1.50 ± 0.23	1.24 ± 0.02	1.80 ± 0.20	2.01 ± 0.02	1.28 ± 0.32	**1.22 ± 0.28**
TRPO aug ent reg	1.47 ± 0.22	1.24 ± 0.01	1.92 ± 0.17	2.02 ± 0.02	1.31 ± 0.20	**1.36 ± 0.16**
TNPG	1.43 ± 0.28	1.24 ± 0.01	1.87 ± 0.16	2.01 ± 0.03	1.19 ± 0.26	**1.18 ± 0.22**

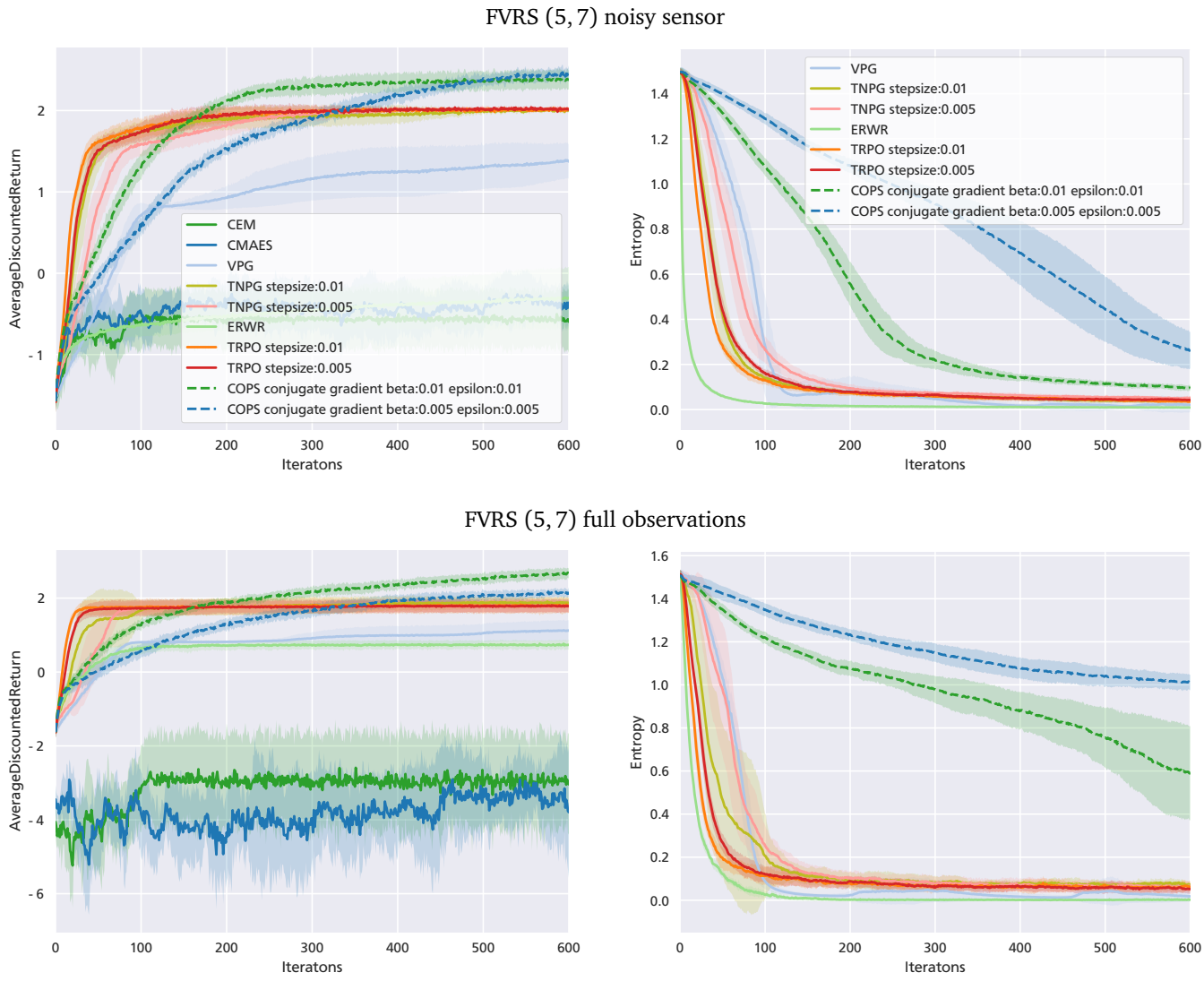
Table 3.1.: Discounted average reward on different FVRS instances (fully and partially observable) averaged over 10 runs after 600 iterations (same seeds for every algorithm). Best results are marked in bold. All other results, that are not statistically significantly different from the best result (Welch's t-test with $p < 0.05$) are also marked bold. Comparison between COPS, TRPO, TNPG and TRPO with augmented entropy regularization on different FVRS instances.

side and is punished if it leaves the grid on a different direction. The robot also gets a punishment if it chooses the pickup action on a field without rock or if the rock already have been picked up. All in all the task of the robot can be summarized to explore the grid and gather information to determine which rock is valuable, pickup the valuable rocks and leave the area, if there is no appreciable reward left. A sketch of the RockSample environment is shown in Figure 3.1.

Generally, FVRS is very a challenging problem for model-free policy gradient approaches due to premature convergence. It is very easy to learn a policy, which just moves to the right side of the grid. But to learn a good policy that is able to pickup rocks properly, one needs a lot of exploration, since one need first to find the correct position of the rocks and afterwards need to understand how to obtain a proper belief for the rock value from the history of noisy observations and match them to the correct rock.

For the experiments we used a fully connected feed forward neural network consisting out of 2 hidden layers with tanh nonlinearity and each layer containing of 30 neurons as policy. The input is the history of observations and the current position of the agent in the grid. To obtain the hyperparameters β and ϵ a grid search was performed on smaller instances of FVRS, where the rewards were averaged over 10 runs and the best were taken. Further details about the setup and neural network can be found in Appendix B. Note, that our presented rewards are scaled down by a factor of 0.1 compared to the standard rewards in FVRS. For the comparison we followed [63, 17] and compared COPS with four gradient based methods: TRPO, truncated natural policy gradient [17, 63] (TNPG), REINFORCE (VPG) [85], Reward-Weighted Regression (RWR) [30] and two gradient-free black box optimization methods: CEM [12] and CMA-ES [21]. The implementation for these algorithm were taken from rllab.

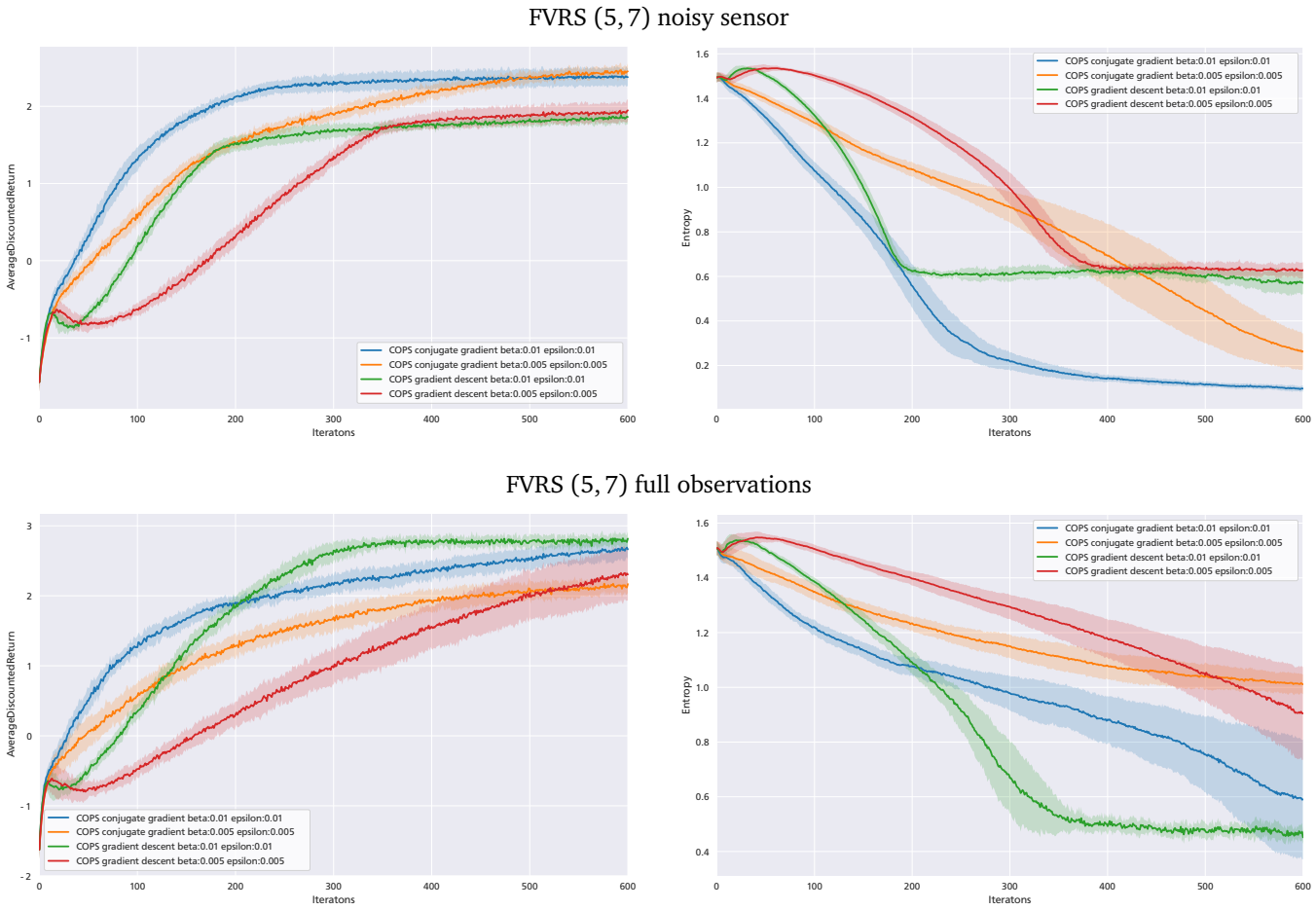
Figure 3.2 and Table 3.1 show that COPS has a significantly better performance than the other methods on almost every tested FVRS instances. TRPO and TNPG performed well on all instances, but were worse than COPS. Looking at the entropy curve in Figure 3.2 (b) it can be observed that for all other algorithms except COPS the entropy is decreasing very fast. On the contrary the learned policy of COPS maintains much higher entropy due to the entropy constraint. These



(a) Discounted average reward

(b) Entropy

Figure 3.2.: Learning curves for FVRS $(5, 7)$ with noisy sensor and full observations, averaged over 10 runs (same seeds for every algorithm). Shaded area shows standard deviation. For a more clear visualization the learning curves for the best variant of COPS are shown only. a) Comparison of the discounted averaged reward between different gradient-based and gradient-free black box optimization methods. b) Comparison of the average current entropy of the learned policy for all gradient-based methods.



(a) Discounted average reward

(b) Entropy

Figure 3.3.: Comparison between both COPS variant gradient descent and conjugate gradient on FVRS. Learning curves for FVRS $(5, 7)$ with noisy sensor and full observations, averaged over 10 runs (same seeds for every algorithm). Shaded area shows standard deviation.

results provide empirical evidence that adding an additional entropy regularization constraint to a trust region optimization problem can help to prevent premature convergence and encourage exploration by preventing that the entropy of the policy is reduced to fast. Especially, if we compare the results of COPS with TRPO and TNPG, we see the effect of the entropy constraint, since they all use the same stepsize / β for the KL constraint. This insight is also strengthened by Table 3.1 where one can observe that COPS outperforms TRPO with augmented entropy regularization. On FVRS $(7, 8)$ noise we see however diminishing returns of COPS compared to TRPO and TNPG. This shows that for very difficult exploration problems one still needs to apply additional exploration techniques in addition to entropy regularization to obtain the optimal policy.

Figure 3.3 and Table 3.2 show the results of the different variants of COPS on several fully and partially observable FVRS instances. In Figure 3.3 it can be seen that COPS gradient descent performs better in the fully observable environment and worse in the partially observable environment than COPS conjugate gradient. This trend is also visible in Table 3.2 for all the other fully and partially observable instances.

Table 3.3 shows a comparison between COPS conjugate gradient with a baseline from a action independent part of the Q-function and without using a baseline to answer the second research question. The results show that on the partially observable environments both variants are performing very similar. But on the fully observable FVRS $(5, 5)$ and $(5, 7)$ environment COPS without baseline tends to perform better than COPS with baseline. This difference can be explained, that COPS with baseline still has a higher entropy after 600 iterations than COPS without baseline. Hence, the results show that on FVRS it is actually better to use the direct estimates. However, we will see in the next section a different environment where using the baseline increases the performance.

	Algorithm	(5, 5) full	(5, 5) noise	(5, 7) full	(5, 7) noise	(7, 8) full	(7, 8) noise
(a)	COPS conjugate gradient	2.14 ± 0.08	1.94 ± 0.12	2.66 ± 0.14	2.45 ± 0.10	1.66 ± 0.17	1.32 ± 0.15
	COPS gradient descent	2.34 ± 0.09	1.20 ± 0.02	2.82 ± 0.10	1.95 ± 0.14	1.95 ± 0.24	0.85 ± 0.21
(b)	COPS conjugate gradient	0.53 ± 0.24	0.16 ± 0.4	0.59 ± 0.22	0.26 ± 0.08	0.05 ± 0.02	0.05 ± 0.02
	COPS gradient descent	0.40 ± 0.02	0.39 ± 0.04	0.45 ± 0.02	0.63 ± 0.04	0.50 ± 0.04	0.27 ± 0.21

Table 3.2.: Comparison between both COPS variant gradient descent and conjugate gradient on different FVRS instances. Discounted average reward (a) and entropy (b) on different FVRS instances (fully and partially observable) averaged over 10 runs after 600 iterations (same seeds for every algorithm).

	Algorithm	(5, 5) full	(5, 5) noise	(5, 7) full	(5, 7) noise	(7, 8) full	(7, 8) noise
(a)	COPS cg no baseline	2.14 ± 0.08	1.94 ± 0.12	2.66 ± 0.14	2.45 ± 0.10	1.66 ± 0.17	1.32 ± 0.15
	COPS cg with baseline	1.74 ± 0.10	1.85 ± 0.12	1.89 ± 0.06	2.33 ± 0.13	1.61 ± 0.14	1.26 ± 0.17
(b)	COPS cg no baseline	0.53 ± 0.24	0.16 ± 0.4	0.59 ± 0.22	0.26 ± 0.08	0.05 ± 0.02	0.05 ± 0.02
	COPS cg with baseline	0.95 ± 0.05	0.16 ± 0.03	1.08 ± 0.02	0.19 ± 0.02	0.34 ± 0.34	0.16 ± 0.1

Table 3.3.: Discounted average reward (a) and entropy (b) on different FVRS instances (fully and partially observable) averaged over 10 runs after 600 iterations (same seeds for every algorithm). Comparison between using the baseline from the action independent part of the Q-function for the value function as proposed in 3.4 (with baseline) and direct usage of the Monte-Carlo estimates (no baseline) for COPS conjugate gradient (gc).

3.6.2 Partially Observable Pacman

The second environment we used in our experiments is a partially observable variant of the well known video game "Pacman", called Pocman, first introduced in [67]. In this task, similar to the original Pacman, an agent (Pocman) is in a 17×19 maze and has to eat all the food pellets that are randomly placed inside of the maze, while not dying to the roaming enemies (ghosts). The main difference between Pacman and Pocman are the observations. Unlike in Pacman the agent in Pocman only receives local observations and does not receive the full game screen as information. For example the agent does not know its global position nor the structure of the maze and only receives a 4 bit observation describing the local wall configuration at its local position. Furthermore, the agent does not know the position of the ghosts. It only receives a 4 bit observation if a ghost is visible via direct line of sight in one of the four directions. Additionally, the agent can hear a ghost, which is a 1 bit observation that is set to one if the agent is within a Manhattan distance of two of a ghost. Finally, the agent can smell food pallets, which is a 1 bit observation that is set to one if if the agent is within a Manhattan distance of one of the food pallet. In total the pocman has approximately 10^{56} underlying states, which can generate 2^{10} different observations, with four actions. Except these changes the environment behaves similar to the Pocman game. For instance the four ghost roam the maze randomly, select a new direction at a junction point without going back and chase the agent aggressively if the agent is closer than certain a Manhattan distance. If the agent touches a ghost it dies and the episodes terminates. There are four power pills on fixed locations in the maze though, which give the Pocman invulnerability and the ability to eat ghosts for 15 time steps. When Pocman is under the effect of a power pill the ghosts will flee from him, when he is in a certain range. A sketch of the Pocman environment is depicted in Figure 3.4. For comparison reasons we chose the same rewards as in [67] and show the undiscounted returns for the experiments.

We chose this environment for our experiments, since it is a challenging large scale POMDP. For the experiments we tried out two different policy parametrizations: one based on a fully connected feed forward neural network and one based on a convolutional neural network. The inputs for both parametrizations are the histories of observations and past chosen actions, however, the neural networks differ in the structure. While the fully connected feed forward neural network was chosen for comparison reasons to our experiments with FVRS and does not take any temporal structure of the observations and actions into account, the convolutional neural network structure was chosen such that it can exploit eventual temporal dependencies in the history. Therefore, we order the history of observations and actions in a 2D array similar to an image and chose a receptive field for the CNN such that it can apply the convolution operation over past observations and actions of multiple timesteps. This idea is depicted in Figure 3.5. For comparison we chose TRPO and TRPO with augmented entropy regularization, since both algorithm are the closest competitors to COPS. For more details on the experiment setup we refer to Appendix B.

Figure 3.7 shows the results of this comparison for different baselines. It can be observed that COPS is outperforming TRPO and TRPO with augmented entropy regularization for all utilized baselines. Furthermore, the results show that when applying a baseline COPS is able to achieve a performance, which is even competitive with POMCP. POMCP is one of the best known algorithms for large scale POMDPs based on Monte-Carlo tree search and particle filters and its results can be seen as an lower bound for the one of the best known result on this problem [67] (see also Section 2.2.3). We distinguish between POMCP with and without using domain knowledge, because POMCP has two variants, where in one the rollout policy has no information over the environment (without domain knowledge) and the other, where the rollout policy is some simple hand-crafted policy tailored to the specific environment (with domain knowledge). For our comparison, we are mainly interested in the results without using domain knowledge, since policy search methods, also do not have any information about the environment. Note that the shown results of POMCP are learned with a discount factor of 0.95, while we are using a discount factor of 0.99. However, since we are comparing the undiscounted returns, the different discount factor do not influence the visualization. Comparing the policy parametrization the results show that applying a CNN to perform the convolution operation over the observations and actions of multiple timesteps leads to better results, especially when used with COPS. This could be explained that COPS has two different update rules, one for the last layer, which is exact and one for the other layers, which are approximate solutions. While TRPO only has an approximate solution for all parameters. Comparing the baselines it can be observed that in the Pocman environment that the baseline helps to reduce the variance of the Monte-Carlo estimates of the returns and increases the performance. However, using the compatible feature baseline does not lead to additional gain when compared to a simple linear feature baseline.

Figure 3.6 (a) shows the result of the learned policy for COPS using the CNN policy and compatible baseline. It can be observed that the agent has learned, how to navigate in the maze and visit every cell in the maze. Furthermore, it has learned a good strategy to traverse the maze, which is to try collect the blue power pills in clockwise direction and all pallets on its way. However, the agent is also able to adapt his trajectory if ghosts are blocking his path. This can be seen in Figure 3.6 (b). Here, we modified the environment for the rollouts of the learned policy by an additional ghost, that

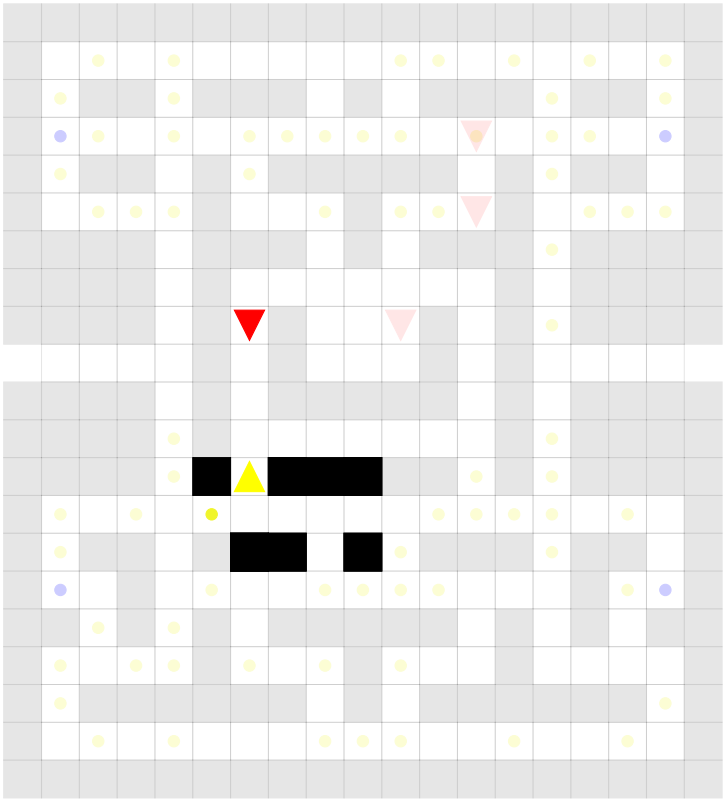


Figure 3.4.: Sketch of the Pocman environment. The agent (yellow triangle) is inside a maze and needs to collect all the food pallets (yellow circles), while avoiding the roaming enemies (red triangles). The agent has only local observations. Hence, in the current visualized position he can only see the enemy north of him, the current wall configuration (black rectangles right and left from the agent) and smell nearby pallets. Furthermore, he has some information about the past observations. For example in this visualization the last four timesteps (other black rectangles). Everything else (transparent color) is unknown to the agent.

blocks the path to the power pill in the lower left corner. It can be observed that, the agent is capable of adapting his trajectory by skipping the second power pill in the lower left corner, avoiding the ghost, and continues to move towards the power pill in the upper left corner. But without the second power pill and its invulnerability effect the agent will get cornered more easily and die more often on its way to the upper left corner. This explains why the lines in Figure 3.6 (b) towards the upper left corner are thinner than in Figure 3.6 (a).

3.7 Discussion

We have proposed and analyzed a new trust region optimization method for deep neural network policies. In our analysis we have showed that using the natural gradient and compatible value function approximation one can derive an algorithm for policies based on the exponential family distribution with two close form update rules, where one is the exact and the other one the approximate solution of a trust region optimization problem. Using this insight we provide a different perspective on TRPO, showing that TRPO is actually computing the natural gradient with compatible value function approximation. We have showed that entropy regularization embedded to the trust region problem archives good promising empirical results on several challenging partially observable policy learning tasks outperforming the related state of the art competitor TRPO. The results suggest that for exploration it is better to have entropy regularization embedded as an additional constraint to the trust region problem, which controls the entropy loss in each update step, than augmenting an additional entropy regularization term to the loss function. In the domain of POMDPs we have successfully learned policies for two difficult tasks using model-free reinforcement learning and a generic policy search method without tailoring it to POMDPs and any belief update just by using the information state. Furthermore, we have seen on the Pocman environment that applying a CNN policy, which exploits can the history structure, leads to better results and are even competitive to state of the art large-scale POMDP algorithms.

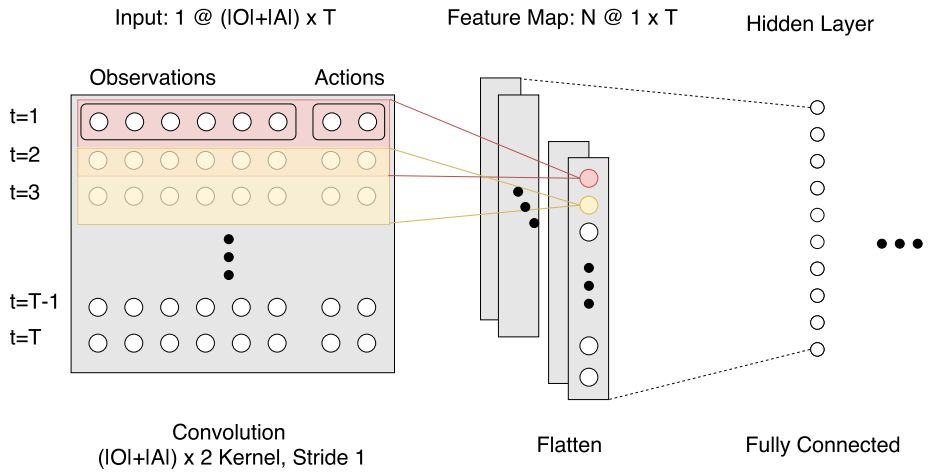
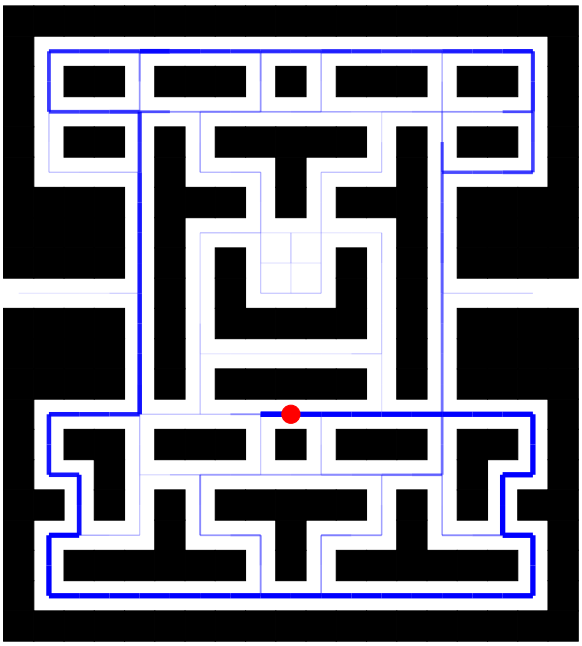
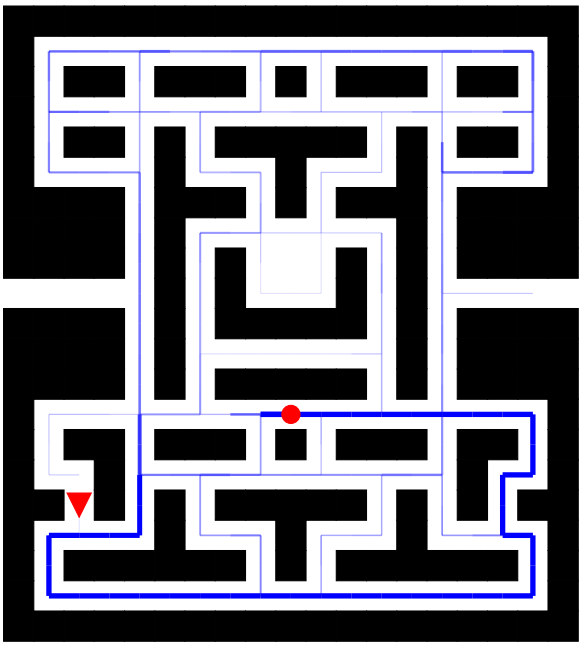


Figure 3.5.: Sketch of a convolutional neural network policy for POMDP histories, which convolves over multiple time steps followed by a fully connected network. The $|O|$ is the dimension of observations, $|A|$ is the dimension of the actions and T are the number of timesteps stored in the history.

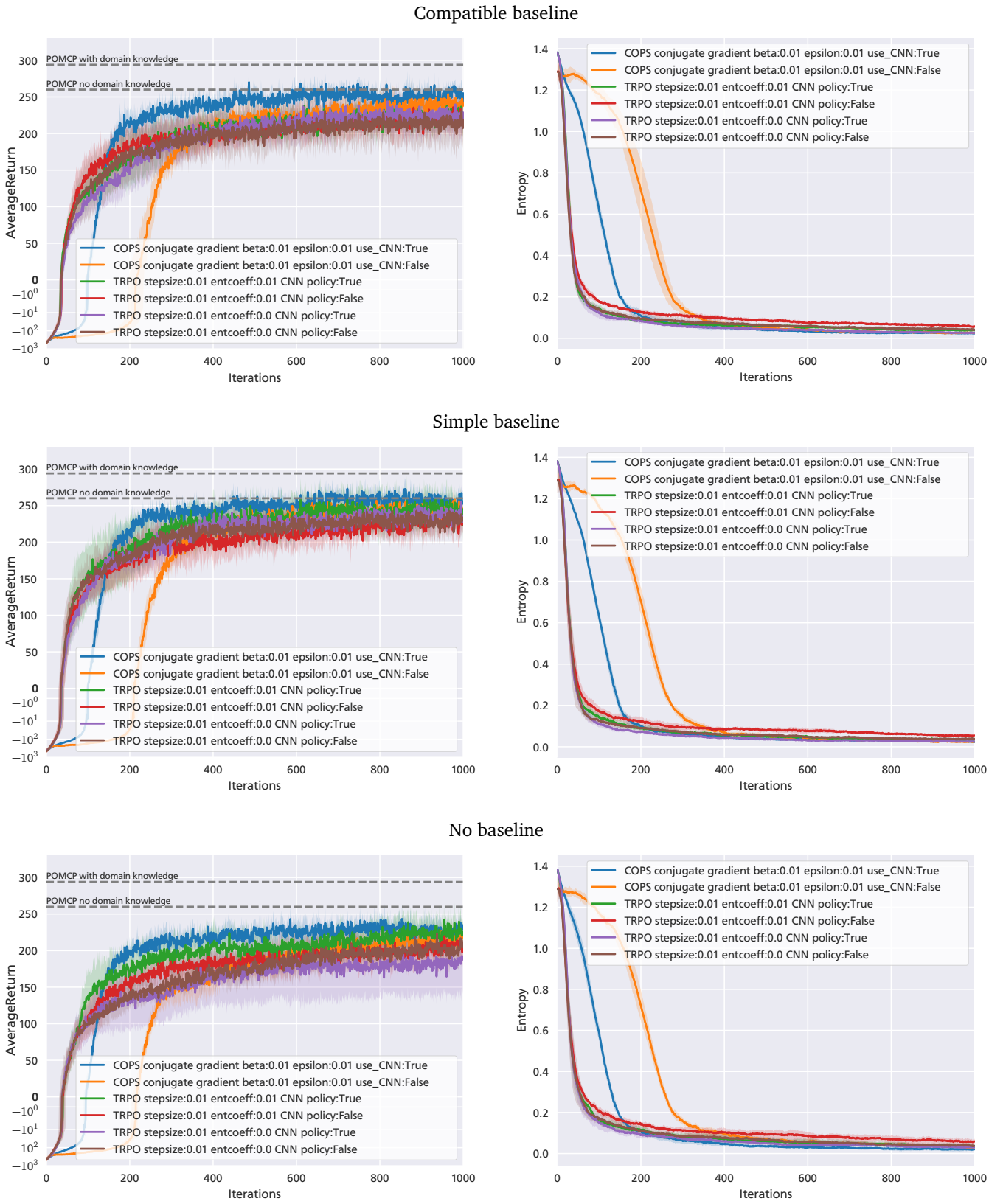


(a) standard environment



(b) modified environment with additional ghost at fixed position

Figure 3.6.: Visualization of the learned policy for COPS using CNN and compatible baseline for 100 trajectories. Blue lines show the path that the learned policy takes. Thinner and transparent lines are showing paths that have been taken less frequent, while thicker, darker lines are showing paths that have been taken more frequent. Starting point is the red dot. (a) shows the trajectories on the standard Pocman environment without any modifications. (b) shows the trajectories for a modified environment, where we added an additional ghost blocking the way to the power pill in the lower left corner.



(a) Undiscounted average reward

(b) Entropy

Figure 3.7.: Learning curves on the Pocman environment averaged over 5 runs (same seeds for every algorithm) for COPS, TPRO (entcoeff$= 0.0$) and TRPO with augmented entropy regularization (entcoeff$= 0.01$) and different baselines. Shaded area shows standard deviation. a) shows a comparison of the undiscounted averaged reward and b) a comparison of the average current entropy of the learned policy. Note that in a), we chose a log scale for negative returns and a linear scale for positive returns.

4 Factored Context Tree Switching (CTS) Pseudocount-Based Exploration

4.1 Introduction

Having seen in the last chapter and Section 2.4.1 that the exploration of the policy plays a large role for model-free RL, especially in POMDPs, we want to push our results of COPS further by adding an additional technique for exploration. In RL exploration has been always one of the greatest difficulties represented in the fundamental dilemma of exploration vs exploitation. One of the main approaches for exploration is the upper confidence bound algorithm (UCB) [34], based on the principle of "optimism in the face of uncertainty" and can also be motivated from an intrinsic motivation viewpoint [5]. One example for this principle can be found in [6], where an exploration bonus is augmented to the returns that is highest for rarely tried actions

$$Q_{\text{aug}}(s,a) = Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \tag{4.1}$$

where $N(s)$ and $N(s,a)$ are functions that tell how often the state or state-action pair was visited, respectively, and c is a hyperparameter regulating the ratio between exploration bonus and the current return. The problem of this approach is that it does not scale really well outside of the tabular case, since we would need a function, which keeps track of all visit counts. Furthermore, an additional issue with visit counts is that in large domains many states are rarely visited more than once. Bellemare et al. [8] were however able to apply this approach to images and the infamous Atari game "Montezuma's Revenge" by the usage of powerful density models and the application of them to do so called "pseudo-counting". For this model the authors chose a context tree switching [80] (CTS) density model, inspired from [9], which treats each pixel of the image as a factor and computes the probability of the whole image by the product of the probability of its factors. Our idea is now to use these factored CTS density models and apply them to the history structure of POMDPs to obtain a similar pseudocount-based exploration for POMDPs.

4.2 Preliminaries

In this section we first explain the idea of Bellemare et al. and show briefly how pseudocounts can be used for exploration and how this pseudocounts can be computed from densities. We give a brief introduction to the CTS model in general, which are used to model the densities, before we move on to explain how these models could be applied to images by factorizing the state space. Afterwards we propose how these models could be applied to histories of actions and observations to obtain a pseudocount-based exploration framework for POMDPs.

4.2.1 From Densities to Pseudocounts

In this section we explain the pseudocounts, introduced in [8] and show how they can be derived from densities and how they relate to intrinsic motivation and exploration. Hence, we will follow their definitions and notation.

Let ρ be a density model on a finite space $x \in \mathcal{X}$ and $\rho_n(x)$ the probability assigned by the model to the state x after being trained on a sequence of states $x_1, x_2, \ldots, x_n$

$$\rho_n(x) := \rho(x; x_{1:n}) \tag{4.2}$$

then we can define an additional probability, called recording probability of a state x, defined as the probability assigned by the model if it were trained once more on x after being trained on $x_1, x_2, \ldots, x_n$ and x

$$\rho'_n(x) := \rho(x; x_{1:n}x). \tag{4.3}$$

If $\rho_n(x) > 0$ for all x and n then both expressions may be interpreted as the usual conditional probability, i.e. the probability of observing $X_{n+1} = x$ given $X_1, \ldots, X_n = x_{1:n}$ and vice versa for $X_{n+2} = x$. Furthermore, the density ρ

is called learning positive if $\rho'_n(x) \geq \rho_n(x)$ for all $x_1, \ldots, x_n, x \in \mathcal{X}$. Using these probabilities the authors define the pseudocount function $\hat{N}_n(x)$ and the pseudocount total $\hat{n}$ by:

$$\rho_n(x) = \frac{\hat{N}_n(x)}{\hat{n}}, \quad \rho'_n(x) = \frac{\hat{N}_n(x)+1}{\hat{n}+1} \tag{4.4}$$

for learning positive distributions. Intuitively this means, after that seeing x for the second time the density model's increase should correspond to an increase of a unit in pseudocount. The pseudocount can be derived by solving this equations for $\hat{N}_n(x)$

$$\hat{N}_n(x) = \frac{\rho_n(x)(1-\rho'_n(x))}{\rho'_n(x)-\rho_n(x)}. \tag{4.5}$$

Given a learning positive density model one can now use the derived pseudocount and plug them into Equation (4.1) to obtain the exploration bonus.

4.2.2 Factored CTS Models for Images

Having seen how we can obtain pseudocounts for exploration from density models, we will move on now to briefly explaining the model Bellemare et al. used. The CTS density model was originally proposed by Veness et al. [80] and comes from a slightly different motivation: code compression of binary sequences drawn from an unknown distribution. To perform this code compression on these binary, stationary, n-Markov sources the CTS algorithm aims to learn a density model based on all past observations such that it minimizes the average code length for the seen sequences. Or in other words from a more machine learning perspective in the CTS algorithm we want to learn a density model in an online setting, which is able to predict the next observation given that the model has been trained on a finite sequence of previous ones. Context tree weighting, the predecessor of CTS, learns this density via Bayesian model averaging over the the class of all prediction suffix trees of bounded depth efficiently. Hence it computes at each time point t the probability

$$P(x_{1:t}) = \sum_M P(M)P(x_{1:t}|M) \tag{4.6}$$

where $(x_{1:t})$ the binary sequence seen so far and M is the model, which is in this case a prediction suffix tree [81]. Note that the summation goes over all possible models of bounded depth D. Thus, this probability model covers all D-order Markov processes. However, a native computation of this sum would take $O(2^{2D})$ time, while the context tree weighting algorithm [84] only needs $O(D)$ [81]. CTS is an extension of the context tree weighting algorithm that has the same bound, but performs generally better in practice [80].

To apply these methods on images e.g. frames of the Atari video games there are two important changes necessary. First to use a non-binary approach, since CTS in its vanilla version assumes that we only have a binary alphabet consisting out of 0s and 1s. Second to factorize the state $x \in \mathcal{X}$, since without factorization $\mathcal{X}$ would be the set of all possible state - in a real world setting with images this would be just impractical. The idea of factorization is to split up the state / image into a product of smaller subspaces

$$\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_j \tag{4.7}$$

where $\mathcal{X}_i$ is the ith subspace and j the number of total subspaces. Using this factored state space Bellemare et al. proposed to approximate the density model over $\mathcal{X}$ using its factors. Moreover, they define a conditional probability for x and $x_{1:n}$

$$P(x|x_{1:n}) = \prod_{i=1}^{j} P_n^i(x^i|x^{\pi(i)}) \tag{4.8}$$

where $x^{\pi(i)} \subseteq \{1, \ldots, i-1\}$ is a parent set associated to each factor x^i and $P_n^i(x^i|x^{\pi(i)}) = P^i(x^i|x_{1:n}, x^{\pi(i)})$ is the factor model for each factor. Note that the condition is chosen such that the probability of each factor also depends on the value of $x^{\pi(i)}$ and not only on the past seen states $x_{1:n}$.

Applied to images this means that $x \in \mathcal{X}$ is a frame, where each pixel (i, j) corresponds to a factor $x^{i,j}$. For the parents Bellemare et al. chose the pixels $(i-1, j), (i, j-1), (i-1, j), (i+1, j-1)$, i.e. the upper-left neighbours of pixel (i, j), assuming that the left upper corner is $(0, 0)$ (depicted in Figure 4.1). The probability of the frame x is then given by (4.8). For each $P^i(x|x_{1:n}, x^{\pi(i)})$ a non-binary CTS model is used, which predicts the pixel's colour value conditioned on its parents.

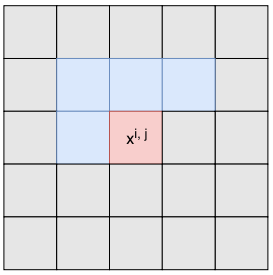


Figure 4.1.: Sketch of the location-dependent CTS model for images. Each pixel is dependent on the pixels direct neighbours (in blue). Recreated from [8]

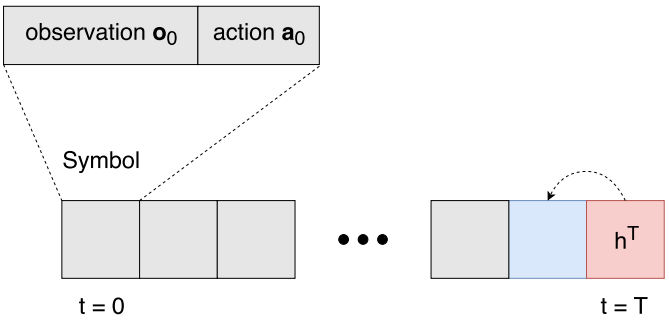


Figure 4.2.: Sketch of the timestep dependent factored CTS model for POMDP histories. Each symbol consisting out of observation and action is dependent on symbols of the previous timesteps (in blue). For the sketch we chose $h^{\pi(t)} = \{t-1\}$ for simplicity, but other choices are possible.

4.3 Factored CTS Model Applied to POMDPs

After having showed how factored CTS models can be applied to images, we will explain now how they can be applied to POMDP histories. Following the idea of Bellemare et al. we can either directly transfer the idea by factorizing the history $h = (\mathbf{o}_0, \mathbf{a}_0, \ldots, \mathbf{a}_{T-1}, \mathbf{o}_T, \mathbf{a}_T)$ simply at each timestep t where $(\mathbf{o}_t, \mathbf{a}_t)$ corresponds to the factor h^t. Then we can compute the conditional probability similar to (4.8) as

$$P(h|h_{1:n}) = \prod_{t=1}^{T} P^t(h^t|h_{1:n}, h^{\pi(t)}) = \prod_{t=1}^{T} P_n^t(h^t|h^{\pi(t)}) \tag{4.9}$$

where $h_{1:n}$ are the previous past observed histories and $h^{\pi(t)} \subseteq \{1, \ldots, t-1\}$ is the parent set of factor t. Hence, we have in the end for each timestep one individual non-binary CTS model, which predicts the action observation pair of its timestep conditioned on its parents. This model we call timestep dependent factored CTS model and is depicted in Figure 4.2.

Or we could also choose a different factorization, for example based on the idea of Veness et al. [81], who proposed the factored action-conditional CTW (FAC-CTW) model, which we will use instead with the CTS algorithm. Hence, we will refer it as FAC-CTS. In this model one does not factor over different timesteps, but over individual observation bits, assuming that each factor is independent. Hence, we have in this model for each observation bit one individual binary CTS model, which predicts the corresponding observation bits of each timestep and uses the actions only for conditioning. This has the advantage that one does not need to use a multi-alphabet approach, where the alphabet consists of the full observation space, but can use directly the CTS algorithm. The authors proposed this model originally for a different purpose as environment model for prediction of the next observation, but it can also be used as density model for pseudocounts. Following Equation (4.8) one can write the model as:

$$P(h|h_{1:n}) = \prod_{j=1}^{K} P^j(h^j|h_{1:n}, h^{\pi(j)}) = \prod_{j=1}^{K} P_n^j(h^j|h^{\pi(j)}) = \prod_{j=1}^{K}\prod_{t=1}^{T} P_n^j(o_t^j|h^{\pi(j,t)}) \tag{4.10}$$

where h^j denotes $(o_0^j, o_1^j \ldots o_T^j)$, $h^{\pi(j)}$ denotes the parent set of factor j and K is the number of total observation bits. Figure 4.3 shows a sketch of the FAC-CTS model. More information about this approach can be found in [81].

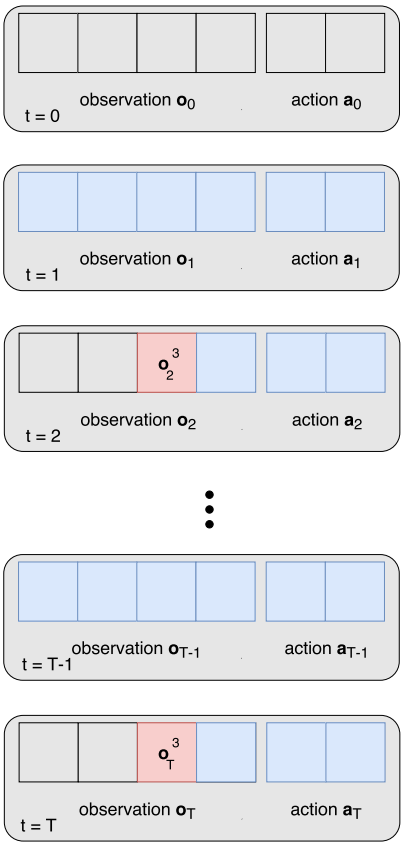


Figure 4.3.: Sketch of the factored action-conditional CTS model for POMDP histories. Each bit j from the observation $\mathbf{o}_t = (o_t^1, o_t^2, \ldots, o_t^k)$ at each timestep t is dependent on its right neighbour observations, the action $\mathbf{a}_t$ and the previous timesteps (in blue). For the sketch we chose the previous timestep as parent for simplicity, but other choices are possible.

Algorithm	$(4,4)$	$(5,5)$	$(5,7)$
COPS cg	1.19 ± 0.4	1.0 ± 0.0	2.04 ± 0.23
COPS cg with time dependent CTS	1.39 ± 0.49	1.0 ± 0.0	2.26 ± 0.22
COPS cg with FAC-CTS	1.19 ± 0.4	1.07 ± 0.15	1.98 ± 0.27

Table 4.1.: Undiscounted average reward on different RockSample instances averaged over 5 runs after 1200 iterations (same seeds for every algorithm) of the best found hyperparameter. Comparison between COPS conjugate gradient using the timestep dependent model and using the FAC-CTS model.

Algorithm	$(5,5)$ noise	$(5,7)$ noise	$(7,8)$ noise
COPS cg	2.76 ± 0.23	3.38 ± 0.19	1.80 ± 0.21
COPS cg with time dependent CTS	2.87 ± 0.02	3.57 ± 0.20	1.89 ± 0.19
COPS cg with FAC-CTS	2.75 ± 0.17	3.53 ± 0.20	-
TRPO	1.48 ± 0.01	2.46 ± 0.03	1.67 ± 0.39
TRPO with time dependent CTS	1.48 ± 0.01	2.48 ± 0.02	1.75 ± 0.23
TRPO ent reg	1.50 ± 0.01	2.46 ± 0.02	1.81 ± 0.21
TRPO ent reg with time dependent CTS	1.48 ± 0.02	2.45 ± 0.04	1.82 ± 0.20

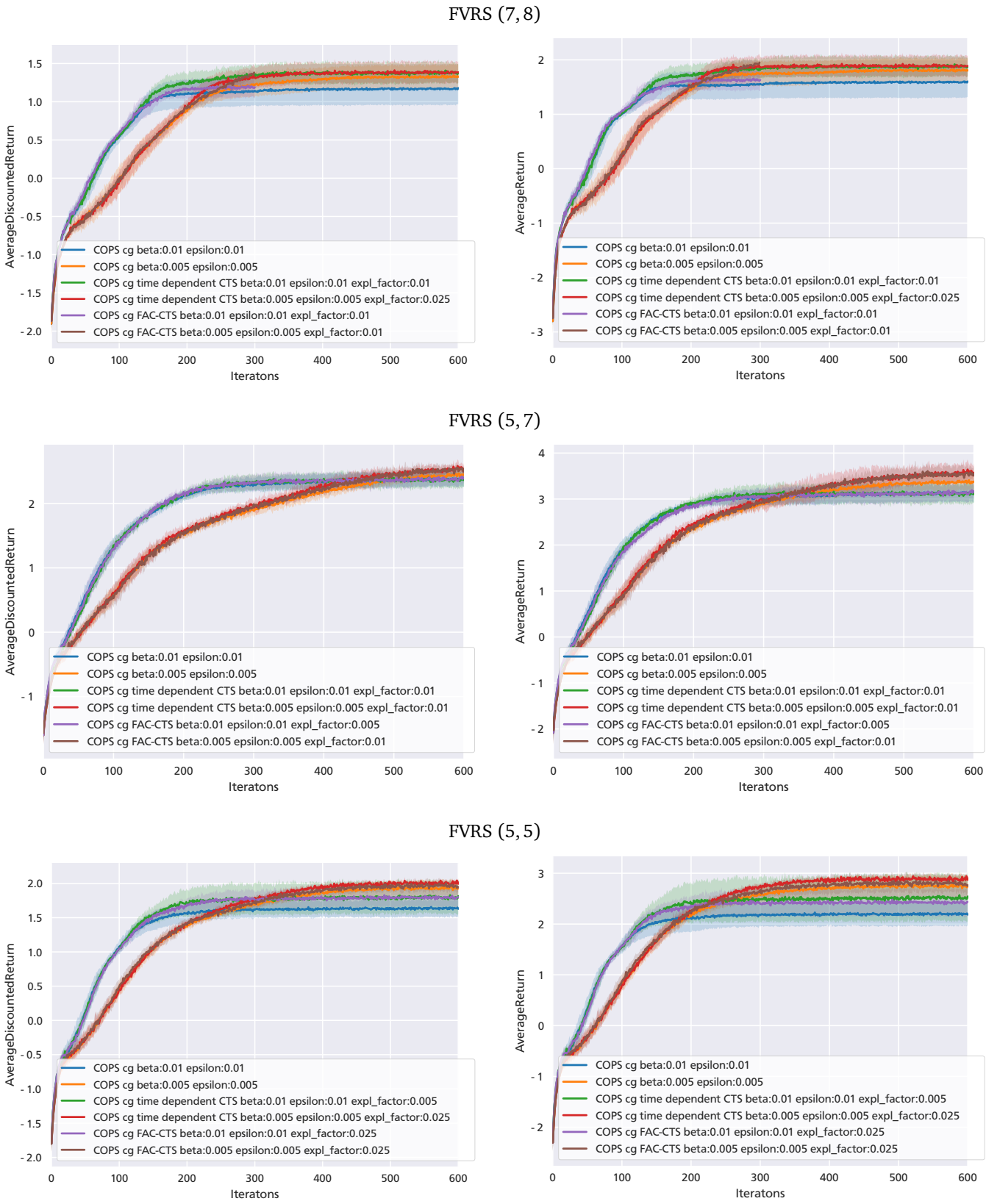
Table 4.2.: Undiscounted average reward on different partially observable FVRS instances averaged over 5 runs after 600 iterations (same seeds for every algorithm) of the best found hyperparameter. Comparison between COPS conjugate gradient, TRPO and TRPO with augmented entropy regularization for different exploration bonuses. For the FAC-CTS model there are no results on FVRS $(7,8)$, since it ran out of memory.

4.4 Experiments

For the experiments we took again the FieldVisionRockSample and RockSample environment, aiming to answer the research question how does the factored CTS pseudocount-based exploration applied to POMDP histories influence the performance of the learning algorithms, especially COPS. Again we performed a hyperparameter search for the best parameter c of Equation (4.1). Figure 4.4 shows the best result of the factored timestep dependent and factored action-conditional CTS model for POMDP histories applied to COPS compared to COPS without any exploration bonus. On all three tested FVRS instances it can be observed that the exploration bonus from both factored CTS models leads to better results, especially on the larger $(7,8)$ problem. We also show the undiscounted average reward, since the exploration bonus leads to policies, where the agent still picks up additional rocks in later timesteps - timesteps, where the agent may have already left the field when trained without exploration bonus. This behavior is better visible in the undiscounted average reward, because in the discounted average reward the rewards of later timesteps are diminished due to the discount factor. Table 4.2 shows the same results not only for COPS, but also for TRPO and TRPO with an augmented entropy regularization loss. It can be observed that the exploration bonus does not really influence the results of both TRPO variants and there is only a small gain in FVRS$(7,8)$ for TRPO using an exploration bonus compared to not using an exploration bonus. As additional experiment Table 4.1 shows the results for COPS with the different CTS models and without on the standard RockSample problem. Here, we can observe that the factored timestep dependent CTS model leads to slightly better results, while the FAC-CTS has the same performance as COPS without any exploration bonus.

4.5 Discussion

In this chapter we have presented a new approach how to apply the idea of Bellemare et al. to POMDP environments for exploration. We have showed two different ways to apply factored CTS models to histories of observations and actions. On FVRS the results show that the exploration bonus leads to better results when applied to COPS and almost no change when applied to TRPO variants except for vanilla TRPO on FVRS$(7,8)$. The difference between COPS and TRPO could be explained by the embedded entropy regularization constraint in the trust region problem, however needs more investigation. On the standard RockSample problem we saw results suggesting that the factored timestep dependent CTS is better than the FAC-CTS model.



(a) Discounted average reward

(b) Undiscounted average reward

Figure 4.4.: Learning curves for FVRS $(7, 8)$, $(5, 7)$ and $(5, 5)$ with noisy sensor, averaged over 5 runs (same seeds for every algorithm). Shaded area shows standard deviation. Comparison of the discounted and undiscounted averaged reward between COPS conjugate gradient with and without exploration bonus. Note that the FAC-CTS model was run on FVRS $(7, 8)$ only for 300 iterations, since it ran out of memory for 600 iterations.

5 Conclusion & Outlook

In this work, we first gave a brief introduction to classical reinforcement learning for fully and partially observable environments, followed by a introduction to deep learning and short overview over prominent state of the art algorithms for deep reinforcement learning. In this overview we looked at different value function and policy search methods, focusing on the techniques deep reinforcement learning algorithms use to stabilize their learning.

In the main part of this thesis, we introduced a new policy search algorithm for policies based on the exponential family and especially deep neural networks, called compatible policy search. We have showed how one can derive this new algorithm using former work on the natural gradient, compatible value function approximation and trust region optimization. We have further showed that we can add one additional entropy regularization constraint to the trust region optimization problem, which prevents the learning algorithm from reducing the entropy of the new policy too fast, encouraging exploration. Experiment on different POMDPs environments have confirmed that the additional entropy regularization constraint indeed helps to obtain a better policy with COPS outperforming other state of the art policy gradient algorithms, such as TRPO and TNPG. The results also have shown that the additional entropy regularization constraint embedded in the trust region optimization problem is better than augmenting additional entropy regularization term to the rewards. Furthermore, we have seen in our analysis a different perspective on TRPO, showing that TRPO is actually computing the natural gradient with compatible value function approximation, just without any entropy regularization.

In the domain of POMDPs we have successfully learned policies using neural networks for two difficult tasks using model-free reinforcement learning and a generic policy search method without tailoring it to POMDPs and any belief update just by using the information state. We also have seen that by exploiting the history structure in a POMDP one can derive a factored CTS model, which can be used for a pseudocount-based exploration bonus. We have showed two different factorization methods and seen in experiments that both methods can lead to an additional performance gain for COPS, while having almost no effect on TRPO. Furthermore, we have seen on the Pocman environment that the application of CNN policies, which can exploit the history structure, lead to better results, that are even competitive to state of the art large-scale POMDP algorithms.

For our experiments we only used on-policy samples for the update of the policy, which is less sample efficient to off-policy methods, such as DQN or DDPG. Hence, it would be interesting to extend COPS to the off-policy setting and to be able to reuse old samples. This could be for example be archived by using a TD learning method instead of using the quadratic loss to obtain $\mathbf{w}$ in equation (3.13) and reusing samples of the previous n iterations for the optimization of the dual. Another point for improvement is that the environments in the experiment sections have been so far only limited to discrete actions. Therefore, the next step for future work is to apply COPS to continuous actions, since the real world is not only limited to discrete actions. Here, the challenge is to find a suitable form for the dual, which is easy to optimize, since we have integrals over all states and actions.

In this thesis so far we only have used feedforward neural networks and convolutional neural networks for the policy and a history of past observation and actions as input. But for future work it would be interesting to extend COPS to recurrent neural networks or give the policy some additional actions to store information in a memory, which would be beneficial in the POMDP setting and would allow the policy to implicitly store some sort of belief state about the true world state or important observations.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] A. Abdolmaleki, R. Lioutikov, J. Peters, N. Lau, L. Reis, and G. Neumann. Model-based relative entropy stochastic search. In *Advances in Neural Information Processing Systems (NIPS)*. mit press, 2015.

[3] R. Akrour, A. Abdolmaleki, H. Abdulsamad, and G. Neumann. Model-free trajectory optimization for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

[4] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[5] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017.

[6] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, May 2002.

[7] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo. Monte Carlo value iteration for continuous-state POMDPs. In *Algorithmic foundations of robotics IX*, pages 175–191. Springer, 2010.

[8] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1471–1479. Curran Associates, Inc., 2016.

[9] M. Bellemare, J. Veness, and E. Talvitie. Skip context tree switching. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1458–1466, 2014.

[10] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[11] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[12] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[13] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška. The importance of experience replay database composition in deep reinforcement learning. In *Deep Reinforcement Learning Workshop, NIPS*, 2015.

[14] T. Degris, M. White, and R. S. Sutton. Off-policy actor-critic. 2012.

[15] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[16] P. Dhariwal, C. Hesse, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

[17] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016.

[18] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[20] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pages 3389–3396, 2017.

[21] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[22] M. J. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[23] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949.

[24] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. *CoRR*, abs/1512.04455, 2015.

[25] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436, 1952.

[26] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[27] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[28] S. Kakade. A natural policy gradient. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 1531–1538. MIT Press, 2001.

[29] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Machine Learning*, 84(1-2):171–203, July 2011.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[32] H. Kurniawati, D. Hsu, and W. S. Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems (R:SS)*, volume 2008, 2008.

[33] Kurt and Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[34] T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.*, 6(1):4–22, Mar. 1985.

[35] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[36] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Winter 1989.

[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[38] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, Jan. 2016.

[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[40] L.-J. Lin and T. M. Mitchell. Reinforcement learning with hidden states. In *From Animals to Animats 2: Second International Conference on Simulation of Adaptive Behavior*, volume 2. MIT Press, 1993.

[41] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[42] M. Minsky and S. Papert. Perceptrons. 1969.

[43] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[45] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

[46] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010.

[47] B. O'Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih. PGQ: combining policy gradient and q-learning. *CoRR*, abs/1611.01626, 2016.

[48] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos. Count-based exploration with neural density models. *CoRR*, abs/1703.01310, 2017.

[49] J. Pajarinen and V. Kyrki. Robotic manipulation of multiple objects as a POMDP. *Artificial Intelligence*, 2015.

[50] J. Pajarinen and J. Peltonen. Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2636–2644, 2011.

[51] C. Papadimitriou and J. N. Tsitsiklis. The complexity of markov decision processes. *Math. Oper. Res.*, 12(3):441–450, Aug. 1987.

[52] J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. In *AAAI*, pages 1607–1612. Atlanta, 2010.

[53] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, Mar. 2008.

[54] J. Pineau, G. J. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. In G. Gottlob and T. Walsh, editors, *IJCAI*, pages 1025–1032. Morgan Kaufmann, 2003.

[55] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[56] S. Ross and B. Chaib-draa. AEMS: an anytime online search algorithm for approximate policy refinement in large POMDPs. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2592–2598, 2007.

[57] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[58] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[59] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering, 1994.

[60] S. Rusell and P. Norvig. Artificial intelligent: A modern approach. 2003.

[61] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[62] J. Schulman. *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2016.

[63] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[64] G. Shani, J. Pineau, and R. Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

[65] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[66] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[67] D. Silver and J. Veness. Monte-carlo planning in large pomdps. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2164–2172. Curran Associates, Inc., 2010.

[68] T. Smith and R. Simmons. Heuristic search value iteration for pomdps. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 520–527, Arlington, Virginia, United States, 2004. AUAI Press.

[69] T. Smith and R. Simmons. Heuristic search value iteration for pomdps. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 520–527. AUAI Press, 2004.

[70] A. Somani, N. Ye, D. Hsu, and W. S. Lee. DESPOT: Online POMDP planning with regularization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1772–1780, 2013.

[71] E. J. Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Oper. Res.*, 26(2):282–304, Apr. 1978.

[72] M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[73] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[74] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[75] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. *CoRR*, abs/1611.04717, 2016.

[76] G. Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.*, 6(2):215–219, Mar. 1994.

[77] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[78] P. S. Thomas, B. C. Da Silva, C. Dann, and E. Brunskill. Energetic natural gradient descent. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2887–2895. JMLR.org, 2016.

[79] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[80] J. Veness, K. S. Ng, M. Hutter, and M. H. Bowling. Context tree switching. *CoRR*, abs/1111.3182, 2011.

[81] J. Veness, K. S. Ng, M. Hutter, W. Uther, and D. Silver. A Monte-Carlo AIXI approximation. *Journal of Artificial Intelligence Research*, 40:95–142, 2011.

[82] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[83] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *International Conference on Artificial Neural Networks (ICANN)*, pages 697–706. Springer, 2007.

[84] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens. The context tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, 41:653–664, 1995.

[85] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[86] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pages 528–535, 2016.

A Pseudocode of Compatible Policy Search

Algorithm 1 COPS gradient descent

Initialize policy network $\pi_{\boldsymbol{\theta},\boldsymbol{\beta}}$ with non-linear parameters $\boldsymbol{\beta}$ and linear parameters $\boldsymbol{\theta}$ and $\Theta = (\boldsymbol{\theta}, \boldsymbol{\beta})$
for episode $\leftarrow 1$ **to** maxEpisode **do**
  Initialize empty batch $\mathscr{B}$
  while collected samples $<$ batchsize **do**
    Run policy $\pi_{\boldsymbol{\theta},\boldsymbol{\beta}}(\mathbf{a}|\mathbf{s})$ for T timesteps or until termination: Draw action $\mathbf{a_t} \sim \pi_{\boldsymbol{\theta},\boldsymbol{\beta}}(\mathbf{a_t}|\mathbf{s_t})$, observe reward r_t
    Put collected samples into trajectories $\tau^{(i)} = (\mathbf{s}_0^{(i)}, \mathbf{a}_0^{(i)}, \mathbf{r}_0^{(i)}, \mathbf{s}_1^{(i)}, \mathbf{a}_1^{(i)}, \mathbf{r}_1^{(i)}, \ldots, \mathbf{s}_T^{(i)})$
    Add trajectories $\tau^{(i)}$ to $\mathscr{B}$
  end while
  Use sampled trajectories $\tau^{(i)}$ from $\mathscr{B}$ to estimate $\tilde{Q}^{\pi_{\text{old}}}(\mathbf{s_t^{(i)}}, \mathbf{a_t^{(i)}}) = \sum_{h=t}^{T} \gamma^h r_h^{(i)}$
  Compute $\mathbf{w} = (\mathbf{w}_{\boldsymbol{\theta}}, \mathbf{w}_{\boldsymbol{\beta}})$ and $\mathbf{v} = (\mathbf{v}_{\boldsymbol{\theta}}, \mathbf{v}_{\boldsymbol{\beta}})$ by minimizing

$$\arg\min_{\mathbf{w},\mathbf{v}} \sum_{i}^{|\mathscr{B}|} \left(Q^{\pi_{\text{old}}}(\mathbf{s}_i, \mathbf{a}_i) - \tilde{F}_{\mathbf{w},\mathbf{v}}^{\pi_{\text{old}}}(\mathbf{s}_i, \mathbf{a}_i)\right)^2 + \lambda \left(\sum_{\mathbf{w}} \mathbf{w}^2 + \sum_{\mathbf{v}} \mathbf{v}^2\right)$$

  Use $\tilde{F}_{\mathbf{w}}^{\pi_{\text{old}}}(\mathbf{s}, \mathbf{a})$ to solve for $\eta > 0$ and $\omega > 0$ using the dual to the corresponding trust region optimization problem:

$$\begin{aligned} \arg\max_{\pi_{\boldsymbol{\theta}}} \; & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[\int \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\, \tilde{F}_{\mathbf{w}}^{\pi_{\text{old}}}(\mathbf{s}, \mathbf{a})\, d\mathbf{a}\right] \\ \text{subject to } \; & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[KL\left(\pi_{\boldsymbol{\theta}}(\cdot \mid \mathbf{s}) \,||\, \pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot \mid \mathbf{s})\right)\right] < \epsilon \\ & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[H\left(\pi_{\boldsymbol{\theta}}(\cdot \mid \mathbf{s})\right) - H\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot \mid \mathbf{s})\right)\right] < \beta \end{aligned}$$

  Apply updates for the new policy:

$$\boldsymbol{\theta}_{\text{new}} = \frac{\eta \boldsymbol{\theta}_{\text{old}} + \mathbf{w}_{\boldsymbol{\theta}}}{\eta + \omega} \quad \boldsymbol{\beta}_{\text{new}} = \boldsymbol{\beta}_{old} + s\frac{\mathbf{w}_{\boldsymbol{\beta}}}{\eta}$$

  where s is a rescaling factor found by line search

end for

Algorithm 2 COPS conjugate gradient

Initialize policy network $\pi_{\theta,\beta}$ with non-linear parameters $\boldsymbol{\beta}$ and linear parameters $\boldsymbol{\theta}$ and $\Theta = (\boldsymbol{\theta}, \boldsymbol{\beta})$
for episode $\leftarrow 1$ **to** maxEpisode **do**
  Initialize empty batch $\mathscr{B}$
  while collected samples $<$ batchsize **do**
    Run policy $\pi_{\theta,\beta}(\mathbf{a}|\mathbf{s})$ for T timesteps or until termination: Draw action $\mathbf{a_t} \sim \pi_{\theta,\beta}(\mathbf{a_t}|\mathbf{s_t})$, observe reward r_t
    Add samples $(\mathbf{s_t}, \mathbf{a_t}, r_t)$ to $\mathscr{B}$
  end while
  Use samples from $\mathscr{B}$ to estimate $V_{\mathbf{v}}^{\pi_{\text{old}}}(\mathbf{s}) = (\, \boldsymbol{\psi}_{\beta}(\mathbf{s}, \mathbf{a}), \nabla_{\beta} C_{\theta,\beta}(\mathbf{s}, \mathbf{a}) \,)\, \mathbf{v}$
  $V_{\mathbf{v}}^{\pi_{\text{old}}}(\mathbf{s})$ as baseline to compute advantages $A^{\pi_{\text{old}}}(\mathbf{s}, \mathbf{a})$
  Compute $\mathbf{w} = (\mathbf{w}_{\theta}, \mathbf{w}_{\beta})$ using conjugate gradient to solve

$$\mathbf{w} = \mathbf{F}^{-1} \nabla_{\Theta} J_{\text{PG}}(\pi_{\Theta}) \,|_{\Theta=\Theta_{\text{old}}} \qquad \nabla_{\Theta} J_{\text{PG}}(\pi_{\Theta}) = \sum_{i}^{|\mathscr{B}|} \nabla_{\Theta} \log \pi_{\Theta}(\mathbf{a}_i|\mathbf{s}_i)\, A^{\pi_{\text{old}}}(\mathbf{s}_i, \mathbf{a}_i)$$

  Use $\tilde{F}_{\mathbf{w}}^{\pi_{\text{old}}}(\mathbf{s}, \mathbf{a})$ to solve for $\eta > 0$ and $\omega > 0$ using the dual to the corresponding trust region optimization problem:

$$\begin{aligned} \arg\max_{\pi_{\theta}} \; & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})} \left[\int \pi_{\theta}(\mathbf{a}|\mathbf{s})\, \tilde{F}_{\mathbf{w}}^{\pi_{\text{old}}}(\mathbf{s}, \mathbf{a})\, d\mathbf{a} \right] \\ \text{subject to } \; & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})} \left[KL\left(\pi_{\theta}(\,\cdot \mid \mathbf{s}) \,||\, \pi_{\theta_{\text{old}}}(\,\cdot \mid \mathbf{s})\right) \right] < \epsilon \\ & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})} \left[H\left(\pi_{\theta}(\,\cdot \mid \mathbf{s})\right) - H\left(\pi_{\theta_{\text{old}}}(\,\cdot \mid \mathbf{s})\right) \right] < \beta \end{aligned}$$

  Apply updates for the new policy:

$$\boldsymbol{\theta}_{\text{new}} = \frac{\eta \boldsymbol{\theta}_{\text{old}} + \mathbf{w}_{\theta}}{\eta + \omega} \quad \boldsymbol{\beta}_{\text{new}} = \boldsymbol{\beta}_{old} + s \frac{\mathbf{w}_{\beta}}{\eta}$$

  where s is a rescaling factor found by line search

end for

B Technical Details for Experiments

	(5×5) full	$(5,5)$ noise	$(5,7)$ full	$(5,7)$ noise	$(7,8)$ full	$(7,8)$ noise
Input space dim.	15	85	17	115	22	134
Output space dim.	5	5	5	5	5	5
Total num. of policy parameters	1410	3510	1650	4560	1620	4980
Sim. step per iteration	5000	5000	5000	5000	5000	5000
Total num. of iterations	600	600	600	600	600	600
Discount γ	0.95	0.95	0.95	0.95	0.95	0.95
History Length	1	15	1	15	1	15
Horizon	25	25	35	35	50	50

Table B.1.: Parameters used for FVRS instances

Experiment setup for the Pocman environment: For Pocman we used two different types of neural networks as policy. A feedforward neural network with 2 hidden layers with tanh nonlinearity and 30 units per layer. And a CNN with one convolutional layer of depth 20 and stride 1 and afterwards a fully connected layer with 64 units.

	feedforward neural network policy	CNN policy
Input space dim.	210	210
Output space dim.	4	4
Total num. of policy parameters	7380	18820
Sim. step per iteration	10000	10000
Total num. of iterations	1000	1000
Discount γ	0.99	0.99
History Length	15	15
Horizon	400	400

Table B.2.: Parameters used for the Pocman environment

Comparison between COPS conjugate gradient and COPS gradient descent: For COPS conjugate gradient we follow Algorithm 2 and computed $\mathbf{w}$ using the inverse of the Fisher information matrix and conjugate gradient similar to TRPO. But in our case the Fisher information matrix is analytically computed using the negative second derivative of the log policy, since we can integrate over all actions in the discrete case, i.e. we use

$$F = \mathbb{E}_{\mu^{\pi}(\mathbf{s}),\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})}\left[-\frac{\partial^2}{\partial \boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\right] \approx \frac{1}{N}\sum_i \sum_{\mathbf{a}} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}_i)\left(-\frac{\partial^2}{\partial \boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}_i)\right).$$

For the computation of COPS gradient descent we follow Algorithm 1 and Section 3.4 using stochastic gradient descent (ADAM) to solve the least squares problem. Here we used a batchsize of 32, a learning rate of 0.001, beta1 0.9 and beta2 0.999 and ran the optimization for 4710 iterations in each policy search iteration.

C Derivation of the Dual

To derive the dual of our trust region optimization problem with entropy regularization and discrete actions we start with equation (3.17):

$$\begin{aligned} \arg\max_{\pi_\theta} \; & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[\int \pi_\theta(\mathbf{a}|\mathbf{s})\, \tilde{F}_{\mathbf{w}}^{\pi_{\text{old}}}(\mathbf{s},\mathbf{a})\, d\mathbf{a}\right] \\ \text{subject to } & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[KL\left(\pi_\theta(\cdot\,|\,\mathbf{s}) \,||\, \pi_{\theta_{\text{old}}}(\cdot\,|\,\mathbf{s})\right)\right] < \epsilon \\ & \mathbb{E}_{\mathbf{s}\sim\mu(\mathbf{s})}\left[H\left(\pi_\theta(\cdot\,|\,\mathbf{s})\right) - H\left(\pi_{\theta_{\text{old}}}(\cdot\,|\,\mathbf{s})\right)\right] < \beta \\ & 1 = \int\int \mu^\pi(\mathbf{s})\pi_\theta(\mathbf{a}|\mathbf{s})\, d\mathbf{a}\, d\mathbf{s} \end{aligned} \tag{C.1}$$

To solve this constrained optimization problem we use the method of Lagrangian multipliers. Therefore, similar to [52] we change the integrals to sums, as we are in the discrete case and for brevity of the derivations, but the proof can be done with integrals similarly. To obtain the Lagrangian we follow the notations of [11]. Hence, we reformulate the problem to a constrained optimization problem, where we need to minimize the objective and only have constrains with zero

$$\begin{aligned} \arg\min_{\pi_\theta} \; & -\sum_{\mathbf{s},\mathbf{a}} \mu^\pi(\mathbf{s})\pi_\theta(\mathbf{a}|\mathbf{s})\tilde{F}_{\mathbf{w}}^{\pi_{\text{old}}}(\mathbf{s},\mathbf{a}) \\ \text{s. t.} \; & \sum_{\mathbf{s}} \mu^\pi(\mathbf{s}) \sum_{\mathbf{a}} \pi_\theta(\mathbf{a}|\mathbf{s}) \log\left(\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})}\right) - \epsilon \leq 0 \\ & \sum_{\mathbf{s}} \mu^\pi(\mathbf{s}) H(\pi_{\theta_{\text{old}}}(\cdot|\mathbf{s})) + \sum_{\mathbf{s},\mathbf{a}} \mu^\pi(\mathbf{s})\pi_\theta(\mathbf{a}|\mathbf{s}) \log(\pi_\theta(\mathbf{a}|\mathbf{s})) - \beta \leq 0 \\ & 0 = \sum_{\mathbf{s},\mathbf{a}} \mu^\pi(\mathbf{s})\pi_\theta(\mathbf{a}|\mathbf{s}) - 1. \end{aligned} \tag{C.2}$$

With this we receive the following Lagrangian

$$\begin{aligned} L(\pi_\theta(\mathbf{a}|\mathbf{s}), \eta, \omega, \lambda) = & -\sum_{\mathbf{s},\mathbf{a}} \mu^\pi(\mathbf{s})\pi_\theta(\mathbf{a}|\mathbf{s})\tilde{F}_{w}^{\pi_{old}}(\mathbf{a}|\mathbf{s}) \\ & + \eta\left[\sum_{\mathbf{s}} \mu^\pi(\mathbf{s}) \sum_{\mathbf{a}} \pi_\theta(\mathbf{a}|\mathbf{s}) \log\left(\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})}\right) - \epsilon\right] \\ & + \omega\left[\sum_{\mathbf{s}} \mu^\pi(\mathbf{s}) H(\pi_{\theta_{\text{old}}}(\cdot|\mathbf{s})) + \sum_{\mathbf{s},\mathbf{a}} \mu^\pi(\mathbf{s})\pi_\theta(\mathbf{a}|\mathbf{s}) \log(\pi_\theta(\mathbf{a}|\mathbf{s})) - \beta\right] \\ & + \lambda\left[\sum_{\mathbf{s},\mathbf{a}} \mu^\pi(\mathbf{s})\pi_\theta(\mathbf{a}|\mathbf{s}) - 1\right] \end{aligned} \tag{C.3}$$

$$\begin{aligned} = & \sum_{\mathbf{s},\mathbf{a}} \mu^\pi(\mathbf{s})\pi_\theta(\mathbf{a}|\mathbf{s})\left[-\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a}) + \eta \log\left(\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})}\right) + \omega \log(\pi_\theta(\mathbf{a}|\mathbf{s})) + \lambda\right] \\ & - \eta\epsilon - \omega\beta + \omega \sum_{\mathbf{s}} \mu^\pi(\mathbf{s}) H(\pi_{\theta_{\text{old}}}(\cdot|\mathbf{s})) - \lambda. \end{aligned} \tag{C.4}$$

We differentiate now the Lagrangian with respect to $\pi_\theta(\mathbf{a}|\mathbf{s})$ and obtain following system

$$\begin{aligned} \delta_{\pi_\theta(\mathbf{a}|\mathbf{s})} L &= \mu^\pi(\mathbf{s})\left[-\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a}) + \eta \log\left(\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})}\right) + \eta\pi_\theta(\mathbf{a}|\mathbf{s})\frac{1}{\pi_\theta(\mathbf{a}|\mathbf{s})} + \omega \log(\pi_\theta(\mathbf{a}|\mathbf{s})) + \omega\pi_\theta(\mathbf{a}|\mathbf{s})\frac{1}{\pi_\theta(\mathbf{a}|\mathbf{s})} + \lambda\right] \\ &= \mu^\pi(\mathbf{s})\left[-\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a}) + \eta \log\left(\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})}\right) + \eta + \omega \log(\pi_\theta(\mathbf{a}|\mathbf{s})) + \omega + \lambda\right]. \end{aligned} \tag{C.5}$$

Setting this to zero and solving for $\pi_\theta(\mathbf{a}|\mathbf{s})$ to obtain the optimal solution $\pi^*_\theta(\mathbf{a}|\mathbf{s})$ yields

$$0 = \mu^\pi(\mathbf{s})\left[-\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a}) + \eta \log\left(\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})}\right) + \eta + \omega \log(\pi_\theta(\mathbf{a}|\mathbf{s})) + \omega + \lambda\right] \tag{C.6}$$

for all $\mathbf{s}$ and $\mathbf{a}$. This gives us either $\mu^\pi(\mathbf{s})$ or the second term is zero. If $\mu^\pi(\mathbf{s})$ is zero this means that the optimal policy does not visit state $\mathbf{s}$ at all and therefore not relevant. Thus, we only look at the case setting the second term to zero

$$0 = \frac{\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a})}{\eta+\omega} - \log(\pi_\theta(\mathbf{a}|\mathbf{s})) + \frac{\eta}{\eta+\omega}\log\left(\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})\right) - \frac{-\eta-\omega-\lambda}{\eta+\omega} \tag{C.7}$$

$$\exp(0) = \exp\left(\frac{\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a})}{\eta+\omega}\right)\frac{1}{\exp\log\pi_\theta(\mathbf{a}|\mathbf{s})}\exp\left(\frac{\eta}{\eta+\omega}\log\left(\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})\right)\right)\exp\left(\frac{-\eta-\omega-\lambda}{\eta+\omega}\right) \tag{C.8}$$

$$\pi_\theta(\mathbf{a}|\mathbf{s}) = \pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})^{\frac{\eta}{\eta+\omega}}\exp\left(\frac{\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a})}{\eta+\omega}\right)\exp\left(\frac{-\eta-\omega-\lambda}{\eta+\omega}\right). \tag{C.9}$$

This gives us our solution

$$\pi_\theta(\mathbf{a}|\mathbf{s}) \propto \pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})\frac{\eta}{\eta+\omega}\exp\left(\frac{\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a})}{\eta+\omega}\right). \tag{C.10}$$

To obtain the dual and the close form solution, we reformulate equation (C.9), using our knowledge that $\sum_{\mathbf{a}}\pi_\theta(\mathbf{a}|\mathbf{s}) = 1$, to

$$1 = \sum_{\mathbf{a}}\pi_\theta(\mathbf{a}|\mathbf{s}) = \sum_{\mathbf{a}}\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})^{\frac{\eta}{\eta+\omega}}\exp\left(\frac{\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{s},\mathbf{a})}{\eta+\omega}\right)\exp\left(\frac{-\eta-\omega-\lambda}{\eta+\omega}\right) \tag{C.11}$$

$$\exp\left(\frac{-\eta-\omega-\lambda}{\eta+\omega}\right) = \left(\sum_{\mathbf{a}}\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})^{\frac{\eta}{\eta+\omega}}\exp\left(\frac{\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{a}|\mathbf{s}}{\eta+\omega}\right)\right)^{-1}. \tag{C.12}$$

Inserting equation (C.9) into the Lagrangian (C.5) and subsequently (C.12) eliminates λ and gives us the dual, that has the same structure as MOTO [3] dual

$$L(\eta,\omega) = -\eta\epsilon - \omega\beta + \omega\sum_{\mathbf{s}}\mu^\pi(\mathbf{s})H(\pi_{\theta_{\text{old}}}(\cdot|\mathbf{s})) - (\eta+\omega)\sum_{\mathbf{s}}\mu^\pi(\mathbf{s})\log\left(\sum_{\mathbf{a}}\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})^{\frac{\eta}{\eta+\omega}}\exp\left(\frac{\tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{a}|\mathbf{s})}{\eta+\omega}\right)\right). \tag{C.13}$$

We can reformulate that to a log sum exp form, which is a much nicer expression when we want to use this with Theano, because Theano has special inbuilt optimization functions when compiling expressions with log sum exp

$$L(\eta,\omega) = -\eta\epsilon - \omega\beta + \omega\sum_{\mathbf{s}}\mu^\pi(\mathbf{s})H(\pi_{\theta_{\text{old}}}(\cdot|\mathbf{s})) - (\eta+\omega)\sum_{\mathbf{s}}\mu^\pi(\mathbf{s})\log\left(\sum_{\mathbf{a}}\exp\left(\frac{\eta\log\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s}) + \tilde{F}_{\mathbf{w}}^{\pi_{\theta_{\text{old}}}}(\mathbf{a}|\mathbf{s})}{\eta+\omega}\right)\right). \tag{C.14}$$