KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Spiking Reinforcement Learning for Robust Robot Control Under Varying Operating Conditions

PHILIPP MONDORF

Stockholm, Sweden, 2022

Spiking Reinforcement Learning for Robust Robot Control Under Varying Operating Conditions

PHILIPP MONDORF

Master's Programme, Systems, Control and Robotics, 120 credits
Date: August 12, 2022

Supervisor: Jörg Conradt
Co-Supervisor: Jens Egholm Pedersen
Examiner: Arvind Kumar
School of Electrical Engineering and Computer Science
Swedish title: Spikande Förstärkningsinlärning för Robust Robotreglering under Varierande Driftsförhållanden

“I have always been convinced that the only way to get artificial intelligence to work is to do the computation in a way similar to the human brain.”

– GEOFFREY HINTON

Abstract

Over the last few years, deep reinforcement learning (RL) has gained increasing popularity for its successful application to a variety of complex control and decision-making tasks. As the demand for deep RL algorithms deployed in challenging real-world environments grows, their robustness towards uncertainty, disturbances and perturbations of the environment becomes more and more important. However, most traditional deep reinforcement learning methods are not inherently robust. Instead, many state-of-the-art deep RL algorithms have been observed to struggle with uncertainties and unforeseen changes in the environment.

Spiking neural networks (SNNs) represent a new generation of neural networks that bridge the gap between neuroscience and machine learning. Unlike conventional ANNs, SNNs employ biological neuron models as computational units that communicate via sparse, event-driven sequences of electrical impulses. When deployed on dedicated neuromorphic hardware, spiking neural networks exhibit favorable properties such as high energy-efficiency and low latency. Recent research further indicates that SNNs are inherently robust to noise and small input errors. This makes them promising candidates to be applied within the field of reinforcement learning.

In this thesis, we propose the *fully spiking deep deterministic policy gradient (FS-DDPG)* algorithm, a spiking actor-critic network for continuous control that can handle high-dimensional state and action spaces. Unlike conversion-based methods, the FS-DDPG algorithm is trained directly via error backpropagation based on surrogate gradients. We show that the FS-DDPG algorithm can successfully learn control policies for different locomotion problems, each involving the control of complex multi-joint dynamics. Furthermore, we evaluate the algorithm's robustness towards sensor noise and perturbations of the environment, and compare it to the DDPG algorithm, a non-spiking actor-critic network with comparable network architecture. While the DDPG algorithm outperforms the spiking actor-critic network on the control tasks, we find the FS-DDPG algorithm to be more robust to sensor noise and measurement errors. Moreover, both algorithms seem to respond similarly to perturbations of the environment.

Our results align with previous works and indicate that spiking neural networks exhibit desirable robustness properties towards sensor noise and measurement errors. Aside from low latency and high power-efficiency on neuromorphic hardware, this might be a substantial advantage of SNNs, in particular within reinforcement learning.

Sammanfattning

Under de senaste åren har djup förstärkningsinlärning (RL) blivit alltmer populär för sin framgångsrika tillämpning på en mängd komplexa uppgifter inom reglerteknik och beslutsfattning. När efterfrågan på djupa RL-algoritmer som används i utmanande verkliga miljöer växer, blir deras robusthet mot osäkerhet, störningar och brus i miljön allt viktigare. De flesta traditionella metoder för inlärning av djup förstärkning är dock inte i sig robusta. Istället har många toppmoderna djupa RL-algoritmer uppvisat svårigheter med hantering av osäkerheter och oförutsedda förändringar i miljön.

Spikande neurala nätverk (SNN) representerar en ny generation av neurala nätverk som överbryggar klyftan mellan neurovetenskap och maskininlärning. Till skillnad från konventionella ANN använder SNN biologiska neuronmodeller som beräkningsenheter som kommunicerar via glesa, händelsedrivna sekvenser av elektriska impulser. När de distribueras på dedikerad neuromorf hårdvara, uppvisar spikande neurala nätverk gynnsamma egenskaper såsom hög energieffektivitet och låg latens. Ny forskning visar vidare att SNN:er är robusta mot brus och små inmatningsfel. Detta gör dem till lovande kandidater att tillämpas inom området förstärkningsinlärning.

I den här avhandlingen föreslår vi algoritmen *fully spiking deep deterministic policy gradient (FS-DDPG)*, ett spikande aktör-kritiskt nätverk för kontinuerlig kontroll som kan hantera högdimensionella tillstånd och handlingsutrymmen. Till skillnad från konverteringsbaserade metoder tränas FS-DDPG algoritmen direkt via backpropagation baserat på surrogatgradienter. Vi visar att FS-DDPG algoritmen framgångsrikt kan lära sig kontrollregler för olika rörelseproblem, som var och en involverar styrning av komplex dynamik med flera leder. Dessutom utvärderar vi algoritmens robusthet mot sensorbrus och störningar i miljön, och jämför den med DDPG algoritmen, ett icke-spikande aktör-kritiskt nätverk med jämförbar nätverksarkitektur. Samtidigt som DDPG algoritmen överträffar det spikande aktör-kritiska nätverket i de reglertekniska uppgifterna, visar vi att FS-DDPG algoritmen är mer robust mot sensorbrus och mätfel. Dessutom verkar båda algoritmerna svara på liknande sätt på störningar i miljön.

Våra resultat är i linje med tidigare arbeten och indikerar att spikande neurala nätverk uppvisar önskvärda robusthetsegenskaper mot sensorbrus och mätfel. Utöver låg latens och hög energieffektivitet på neuromorf hårdvara, kan detta vara en betydande fördel med SNN, särskilt inom förstärkningsinlärning.

Acknowledgments

First and foremost, I would like to thank my supervisors Jens Egholm Pedersen and Prof. Dr. Jörg Conradt for their support and advice during the course of this thesis. My gratitude goes especially to Jens Egholm Pedersen for the interesting and fruitful discussions we had about spiking neural networks, the learning dilemma and various coding schemes. Furthermore, I would like to thank my family and friends for their encouragement and continuous support throughout my studies and life. My gratitude goes also to Sarah who has to bear with me everyday and whom I love and trust unconditionally. Finally, I would like to say how thankful I am for my studies here in Sweden - it was a wonderful time that I would not want to miss.

Contents

1 Introduction **1**
1.1 Main objectives . . . 3
1.2 Thesis Structure . . . 4

2 Theoretical Background **5**
2.1 Reinforcement Learning . . . 5
2.1.1 Markov Decision Process . . . 7
2.1.2 Action-Value Methods . . . 10
2.1.2.1 Q-learning . . . 11
2.1.2.2 Deep Q-learning . . . 12
2.1.3 Policy Gradient Methods . . . 13
2.1.3.1 The Policy Gradient Theorem . . . 13
2.1.3.2 Deterministic Policy Gradient . . . 14
2.1.4 Actor-Critic Methods . . . 15
2.1.4.1 Deep Deterministic Policy Gradient Method 15
2.2 Spiking Neural Networks . . . 17
2.2.1 Neuron Models . . . 17
2.2.2 The Neural Code . . . 21
2.2.3 Training Spiking Neural Networks . . . 24
2.2.4 Simulators for Spiking Neural Networks . . . 27

3 Related Work **29**
3.1 Robustness in Reinforcement Learning . . . 29
3.2 Spiking Reinforcement Learning . . . 33

4 Method **37**
4.1 Fully Spiking DDPG Network . . . 37
4.1.1 Spiking Actor Network . . . 38
4.1.2 Spiking Critic Network . . . 40
4.1.3 Addressing Stability . . . 41

4.2 Input Encoding 44
4.2.1 Min-Max Current Coding 44

5 Experimental Setup **51**
5.1 Simulation Environment 51
5.2 Implementation Details 55
5.3 Robustness Evaluation 57
5.3.1 Sensor Noise and Measurement Errors 57
5.3.2 Perturbations of the Environment 60

6 Results **63**
6.1 Training Results 63
6.2 Robustness Evaluation 65
6.2.1 Sensor Noise and Measurement Errors 65
6.2.2 Perturbations of the Environment 70

7 Discussion **73**
7.1 Training Performance 73
7.2 Spiking Robustness 76
7.3 Limitations and Future Work 81

8 Conclusion **83**

Appendices **85**

Appendix A Robustness Evaluation **86**
A.1 Sensor Noise and Measurement Errors 86
A.1.1 HalfCheetah Environment 86
A.1.2 Hopper Environment 91

Appendix B Hyperparameters **96**
B.1 HalfCheetah Environment 97
B.2 Hopper Environment 98

References **101**

List of Figures

2.1 Schematic overview of agent-environment interactions within the MDP framework 8
2.2 Schematic overview of the DDPG algorithm 16
2.3 RC-circuit representing the LIF model 19
2.4 Schematic overview of a Peri-Stimulus-Time histogram and spike density function 22
2.5 Surrogate gradient approach to approximate partial derivatives 26

4.1 Schematic overview of the FS-DDPG algorithm 38
4.2 Training framework of the FS-DDPG algorithm 43
4.3 Membrane potential of a silent and over-excited LIF neuron . . 45
4.4 Schematic overview of the min-max current coding scheme . . 47
4.5 Entropy profiles of coding schemes with different parameter configurations 49

5.1 Renderings of different control tasks defined within the MuJoCo simulator 52
5.2 Schematic overview of the state-noisy Markov Decision Process 58

6.1 Learning curves of the FS-DDPG and non-spiking DDPG algorithm for the different control tasks 64
6.2 Average return for a halfcheetah test environment in which the state observation of the agents' front tip angle is perturbed by AWGN 66
6.3 Average return for a halfcheetah test environment in which the state observations of the agents' front tip's angular velocity and velocity in x and y-direction are perturbed by AWGN . . . 67

6.4 Average return for a halfcheetah test environment in which state observations regarding the angle of the agents' back thighs and shins are perturbed by impulse noise with probability p = 0.2 68
6.5 Average return for a halfcheetah test environment in which state observations regarding the angle of the agents' back thighs and shins are perturbed by impulse noise with probability p = 0.4 68
6.6 Average return for a halfcheetah test environment in which state observations regarding the angle of the agents' back thighs and shins are perturbed by impulse noise with different noise levels 69
6.7 Average return for a halfcheetah test environment with different torso masses 71
6.8 Average return for a halfcheetah test environment with different joint stiffness matrices 71

7.1 Membrane voltage of a LIF neuron affected by noisy input current 79

A.1 Average return for a halfcheetah test environment in which the state observations of the rotor's angular velocity at the agents' back thighs, shins and feed are perturbed by AWGN 86
A.2 Average return for a halfcheetah test environment in which state observations regarding the angle of the agents' back thighs and shins are perturbed by AWGN 87
A.3 Average return for a halfcheetah test environment in which state observations regarding the angular velocity of the agents' front thighs, shins and feet are perturbed by impulse noise with probability p = 0.2 87
A.4 Average return for a halfcheetah test environment in which state observations regarding the angular velocity of the agents' front thighs, shins and feet are perturbed by impulse noise with probability p = 0.4 87
A.5 Average return for a halfcheetah test environment in which state observations regarding the angular velocity of the agents' front thighs, shins and feet are perturbed by impulse noise with different noise levels 88

A.6 Average return for a hopper test environment in which state observation regarding the angular velocity of the robot's top is perturbed by AWGN 91
A.7 Average return for a hopper test environment in which the state observations of the robot's height and the angle of its top are perturbed by AWGN 91
A.8 Average return for a hopper test environment in which state observations regarding the angle of the robot's thigh joint, leg joint and foot joint are perturbed by AWGN 92
A.9 Average return for a hopper test environment in which the state observation of the robot's height is perturbed by impulse noise with probability p = 0.1 92
A.10 Average return for a hopper test environment in which the state observation of the robot's height is perturbed by impulse noise with probability p = 0.2 92
A.11 Average return for a hopper test environment in which the state observation of the robot's height is perturbed by impulse noise with probability p = 0.4 93
A.12 Average return for a hopper test environment in which state observation regarding the angular velocity of the robot's top is perturbed by impulse noise with probability p = 0.2 93
A.13 Average return for a hopper test environment in which state observation regarding the angular velocity of the robot's top is perturbed by impulse noise with probability p = 0.4 93
A.14 Average return for a hopper test environment in which the state observations of the robot's height and the angle of its top are perturbed by impulse noise with different noise levels 94

B.1 Schematic overview of the FS-DDPG's final network architecture 99
B.2 Schematic overview of the DDPG's final network architecture 100

List of Tables

4.1 Parameters θ of the min-max current coding scheme $\mathcal{F}_\theta$ 48

5.1 Overview of the dimensions and boundaries of the state and action space for each control task 54

6.1 Average return of the algorithms' best models 64

A.1 Average return of the DDPG algorithm evaluated on a halfcheetah test environment in which state observations regarding the angle of the agent's back thigh and shin are perturbed by impulse noise with different noise levels 89

A.2 Average return of the FS-DDPG algorithm evaluated on a halfcheetah test environment in which state observations regarding the angle of the agent's back thigh and shin are perturbed by impulse noise with different noise levels 89

A.3 Average return of the DDPG algorithm evaluated on a halfcheetah test environment in which state observations regarding the angular velocity of the agents' front thighs, shins and feet are perturbed by impulse noise with different noise levels 89

A.4 Average return of the FS-DDPG algorithm evaluated on a halfcheetah test environment in which state observations regarding the angular velocity of the agents' front thighs, shins and feet are perturbed by impulse noise with different noise levels 90

A.5 Average return of the DDPG algorithm evaluated on a hopper test environment in which the state observations of the robot's height and the angle of its top are perturbed by impulse noise with different noise levels 95

A.6 Average return of the FS-DDPG algorithm evaluated on a hopper test environment in which the state observations of the robot's height and the angle of its top are perturbed by impulse noise with different noise levels 95

B.1 Hyperparameters for the LIF neurons of the FS-DDPG algorithm 96
B.2 Hyperparameters for the LI neurons of the FS-DDPG algorithm 96
B.3 Hyperparameters of the FS-DDPG and non-spiking DDPG algorithm for the halfcheetah environment 97
B.4 Hyperparameters of the FS-DDPG and non-spiking DDPG algorithm for the hopper environment 98

Acronyms and Abbreviations

ANN	Artificial Neural Network
AWGN	Additive White Gaussian Noise
DC	Direct Current
DDPG	Deep Deterministic Policy Gradient
DNN	Deep Neural Network
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DSQN	Deep Spiking Q-Network
DVS	Dynamic Vision Sensor
FGSM	Fast Gradient Sign Method
FS-DDPG	Fully Spiking Deep Deterministic Policy Gradient
GMM	Gaussian Mixture Model
GPI	Generalized Policy Iteration
LI	Leaky Integrate
LIF	Leaky Integrate-and-Fire
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
MRP	Markov Reward Process
MuJoCo	Multi-Joint Dynamics with Contact
NR-MDP	Noisy Action Robust Markov Decision Process
PR-MDP	Probabilistic Action Robust Markov Decision Process
PSTH	Peri-Stimulus-Time Histogram
RARARL	Risk-Averse Robust Adversarial Reinforcement Learning
RARL	Robust Adversarial Reinforcement Learning
ReLU	Rectified Linear Unit

RL	Reinforcement Learning
RNN	Recurrent Neural Network
SAN	Spiking Actor Network
SCN	Spiking Critic Network
SDDPG	Spiking Deep Deterministic Policy Gradient
SN-MDP	State-Noisy Markov Decision Process
SNN	Spiking Neural Network
STBP	Spatiotemporal Backpropagation
STDP	Spike-Timing-Dependent Plasticity
TD	Temporal Difference
TPE	Tree-Structured Parzen Estimator

Chapter 1

Introduction

Throughout life, humans and other animals learn by interacting with their environment [103]. For instance, when an infant learns to crawl, it explores a series of body movements that eventually lead to a forward motion. Based on the environmental feedback, favorable movements are repeated and reinforced until the locomotion has been successfully learned [3].

Inspired by such behavioral patterns, *reinforcement learning (RL)* is a computational approach to goal-directed learning from interaction [129]. It describes a learning paradigm in which an agent (for instance the infant) seeks to maximize a reward signal (such as its forward velocity) by exploring a set of actions, while processing corresponding feedback of the environment (like its current position). The main idea is that by balancing the amount of new exploratory movements and actions that have proven to be beneficial, the agent will eventually learn an efficient way to reach its goal [65].

Recent advances in incorporating *deep neural networks* into reinforcement learning [80, 93] have achieved remarkable results in solving complex control and decision-making tasks. For instance, algorithms such as AlphaGo [120] or OpenAI Five[15] have exceeded human performance in sequential decision-making tasks and games like Go and Dota 2, respectively. Over the last few years, this has lead to an increasing popularity of *deep reinforcement learning* [21]. Nowadays, (deep) RL is used within various fields of application, ranging from robotics over natural language processing to finance [79]. Due to the growing demand for deploying RL algorithms in complex real-world environments, their robustness towards uncertainty, disturbances, and changes in the environment has become increasingly important. However, most traditional reinforcement learning methods are not inherently robust [94]. Instead, RL methods have been observed to struggle with sensor noise

and structural changes in the environment [33, 104]. This is particularly problematic within the field of robotics. Robot actuators might degrade over time or fail entirely [83]. Furthermore, robotic agents often have to navigate through dynamically changing environments [104]. Hence, respective control algorithms must be able to cope with these changes.

Spiking neural networks (SNNs) represent a new generation of neural networks that employ biological neuron models as computational units [84]. Inspired by information processing within the brain, SNNs communicate via sparse, event-driven sequences of electrical impulses, also denoted as *spike trains* [112]. When deployed on dedicated neuromorphic hardware, spiking neural networks exhibit desirable properties such as low latency and high energy-efficiency [98, 136, 153]. Furthermore, recent work indicates that SNNs are inherently robust towards noise and measurement errors [76, 109]. This makes spiking neural networks promising candidates to be applied within the field of reinforcement learning.

As of now, *(deep) spiking reinforcement learning* is still in an early stage with only few spiking RL methods existing yet [28]. Common approaches rely on converting trained deep neural networks to SNNs [109, 134], or local learning rules that are limited to small state and action spaces [11, 39]. In this thesis, we introduce the *fully spiking deep deterministic policy gradient (FS-DDPG)* algorithm, a spiking actor-critic network for continuous control that can successfully handle high-dimensional state and action spaces (see chapter 4). In contrast to conversion-based spiking RL approaches, the FS-DDPG network is trained directly via SuperSpike [155], a backpropagation scheme using surrogate gradients [99].

We evaluate the FS-DDPG algorithm with respect to its robustness towards sensor noise and perturbations of the environment. To the best of our knowledge, this is the first work that studies the robustness properties of a spiking actor-critic network for continuous control. With this thesis, we want to pave the way for future research on robustness within deep spiking reinforcement learning. Building robust RL-agents that can be deployed in real-world settings is a step towards making this technology accessible to society. This is not only of scientific, but also of societal interest.

1.1 Main objectives

The main objective of this degree project is to evaluate the robustness of a spiking actor-critic network such as the FS-DDPG algorithm towards sensor noise and perturbations of the environment. In particular, we examine the following three research questions:

RQ1 How does the FS-DDPG algorithm trained on a continuous control task respond to sensor noise and perturbations of the environment?

RQ2 How does a non-spiking actor-critic network with comparable network architecture trained on the same control task respond to sensor noise and perturbations of the environment?

RQ3 How do the spiking and non-spiking networks differ in their robustness towards sensor noise and perturbations of the environment?

As previous work has shown, traditional actor-critic networks struggle with dynamically changing environments and sensor noise [31, 90]. On the other hand, spiking neural networks have shown to be inherently robust towards noise and gradually varying input data [76, 109]. This gives reason to the following two hypotheses:

H1 Both spiking and non-spiking actor-critic networks examine a drop in performance when exposed to sensor noise and/or perturbations of the environment.

H2 The spiking actor-critic network is more robust towards sensor noise and perturbations of the environment than the respective non-spiking network.

To test our hypotheses, we conduct experiments on two different continuous control tasks defined within the MuJoCo simulator [140]. Each task represents a locomotion problem of different difficulty that involves the control of complex multi-joint dynamics (see chapter 5). After successful training of the agents on the respective tasks, we evaluate the networks' robustness towards sensor noise by gradually introducing noise to the agents' observations. In addition, we vary parameters such as the agents' joint stiffness or mass to analyze the networks' response to perturbations of the environment. Finally, we compare the performances of both networks with each other and look at the quality of sampled behaviors.

1.2 Thesis Structure

We begin this thesis by establishing the theoretical foundation of this degree project in chapter 2. For this, we explain important theoretical concepts and ideas related to reinforcement learning and spiking neural networks. In chapter 3, we provide a brief overview of related work and recent advances within the fields of robust RL. Furthermore, we discuss existing approaches to spiking reinforcement learning. In chapter 4, we present the details of the fully spiking DDPG algorithm and discuss coding schemes used to encode continuous signals into discrete spike patterns. Subsequently, we introduce experiments designed to evaluate the robustness properties of the FS-DDPG network towards sensor noise and perturbations of the environment in chapter 5. Chapter 6 presents respective results, which are further discussed in chapter 7. Finally, we summarize the main results and discuss key findings in chapter 8.

Chapter 2

Theoretical Background

In this chapter, we establish the theoretical foundations of this thesis. The first part of the chapter elaborates on reinforcement learning. We explain what reinforcement learning is, how it is formulated and what it can be used for. We further discuss the *Markov Property*, a fundamental assumption most reinforcement learning algorithms are based on. Finally, we outline three relevant reinforcement learning frameworks: action-value methods, policy gradient methods and actor-critic methods.

The second part of this chapter addresses spiking neural networks. We present different spiking neuron models and discuss their role within the field. Furthermore, we describe techniques of encoding and decoding discrete spike signals and elaborate on the challenges of training spiking neural networks. Finally, we talk about frameworks for simulating SNNs.

2.1 Reinforcement Learning

Reinforcement learning is a paradigm that aims at learning how to behave in certain situations. It seeks to find a mapping from perceived states of the environment to actions to be taken within those states. However, unlike supervised learning, where the learning agent is provided *instructive feedback* that gives information about the correct action to take, reinforcement learning mostly relies on *evaluative feedback* that indicates how good the performed action was. The main idea of reinforcement learning is that the agent learns the desired behavior mapping by maximizing the evaluative feedback signal. However, evaluating actions instead of instructing the agent has important implications that distinguishes reinforcement learning from other learning paradigms. It requires the agent to actively interact with

the environment, exploring actions that might lead to desired or undesired behavior. Furthermore, chosen actions that might not immediately seem to be good could turn out to be beneficial in the long run. This results in an exploration-exploitation dilemma, where the agent has to trade off exploring new actions that might be beneficial against exploiting actions that have proven to be good.

In general, reinforcement learning systems comprise the following six elements [129]:

- **Agent**. The learning and decision-making unit. An agent can choose to take an action $a \in \mathcal{A}$ from the set of possible actions $\mathcal{A}$.
- **Environment**. Everything outside the agent that cannot be changed arbitrarily by the agent. The environment can be in different states $s \in \mathcal{S}$, where $\mathcal{S}$ denotes the set of possible states.
- **Policy**. Mapping from perceived states of the environment to actions to be taken within those states. Defines the behavior of the agent.
- **Reward signal**. Represents the immediate desirability of an action given a state of the environment.
- **Value function**. Total amount of reward an agent can expect to accumulate given a state s or a state-action pair (s, a). Indicates the long-term desirability of a state or action.
- **Model of environment**. Describes the behavior of the environment. Allows to predict how the environment will behave under certain actions. Not present in model-free RL algorithms.

In most reinforcement learning settings, an agent interacts with its environment to find a policy that maximizes short and long-term rewards connected to a predefined goal. For this, it uses value functions that estimate the desirability of states. In this sense, reinforcement learning is described as goal-directed learning from interaction. To formalize such settings, most reinforcement learning methods use markov decision processes (MDPs) [129]. In the next section, we mathematically describe MDPs and explain how they can be used to model sequential decision making problems. Furthermore, we derive important concepts of reinforcement learning based on the MDP formulation.

2.1.1 Markov Decision Process

A Markov decision processes is a time-discrete stochastic process used to model sequential decision making, where actions influence immediate and future rewards. Bellman proposed MDPs as a stochastic version of the optimal control problem and showed that under certain assumptions, such problems can be solved using dynamic programming [13, 14, 69]. Markov decision processes build upon Markov processes and extend Markov reward processes. To better understand MDPs, we shall briefly discuss these frameworks.

Markov Process. A Markov process or Markov chain describes a stochastic process that satisfies the Markov property. In other words, it is a memoryless sequence of states in which each new state depends only on the present, but not on past states [60]. The Markov process can be represented by the tuple $\langle \mathcal{S}, \mathcal{P} \rangle$ where $\mathcal{S}$ denotes a (finite) set of states and $\mathcal{P}$ represents the environment's state transition probability function $\mathcal{P} : \mathcal{S} \to \mathbb{P}(\mathcal{S})$ satisfying:

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s] \tag{2.1}$$

Markov Reward Process. A Markov reward process extends the Markov Process by a (stochastic) reward function $r : \mathcal{S} \to \mathbb{R}$ and a discount factor $\gamma \in [0, 1]$ [119]. The MRP can be described by the tuple $\langle \mathcal{S}, \mathcal{P}, r, \gamma \rangle$, where the reward function calculates the expected reward of a state $s \in \mathcal{S}$:

$$r = \mathbb{E}\left[R_{t+1} | S_t = s\right] \tag{2.2}$$

While the reward R_{t+1} represents the immediate desirability of a state $s \in \mathcal{S}$, the return G_t is defined to account for delayed rewards. It represents the total discounted reward beginning from time step t and is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

where the discount factor γ determines how to value delayed rewards. In particular, for $\gamma \to 0$ the return is said to be *myopic*, considering only the immediate reward, whereas for $\gamma \to 1$ it is described to be *far-sighted*, weighting immediate and delayed rewards equally [129].

Markov Decision Process. A Markov decision process extends the Markov reward process by a (finite) set of actions $\mathcal{A}$. It is defined as tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, in which all states are *Markov* and the transition between states

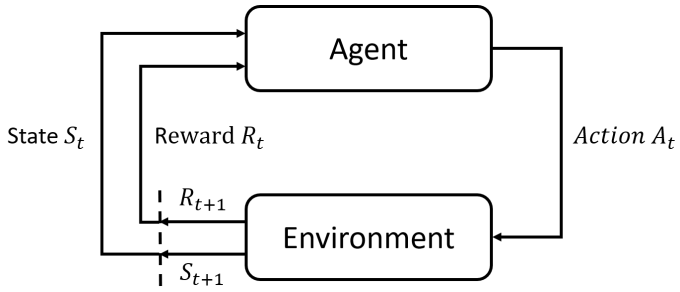


Figure 2.1: Schematic overview of the interaction between agent and environment. At each time step t, the agent chooses an action $A_t \in \mathcal{A}$, given a representation of the environment's state $S_t \in \mathcal{S}$. Based on the action performed, the agent finds itself in a new state S_{t+1}, receiving a reward R_{t+1} [129].

is described by the transition probability function:

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] \tag{2.4}$$

where the next state does not only depend on the current state S_t, but also on the performed action A_t. In addition, the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ can be used to calculate the expected reward of a state-action pair:

$$r = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] \tag{2.5}$$

In reinforcement learning, an *agent* aims at learning goal-directed behavior by interacting with its *environment*. Markov decision processes formalize this problem by discretizing the agent-environment interaction in time. A schematic overview of this process is depicted in figure 2.1. At each time step t of a sequence of discrete time steps $t = 0, 1, 2, 3 \ldots$, the agent receives a representation of the environment's *state* $S_t \in \mathcal{S}$. Based on this representation, the agent selects an *action* $A_t \in \mathcal{A}(s)$ that modifies the current situation and results in a new state $S_{t+1} \in \mathcal{S}$. In addition, the agent receives a *reward* signal $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ that evaluates the agent's action. Based on this reward and the new state observed, the agent chooses a new action. This process repeats until a terminal state has been reached.

The behavior of an agent is fully defined by its policy $\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})$. The policy denotes a (stochastic) mapping from states of the environment to actions the agent can perform:

$$\pi(a|s) = \mathbb{P}\left[A_t = a | S_t = s\right] \tag{2.6}$$

where π is assumed to be stationary, i.e. $A_t \sim \pi\left(\cdot | S_t\right) \forall\, t > 0$. To

estimate the amount of reward an agent can expect to accumulate in a state s when following a policy π, the state-value function $v_\pi : \mathcal{S} \to \mathbb{R}$ is defined as:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] \tag{2.7}$$

where G_t is the return as described in equation 2.3. Similarly, the action-value function $q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ describes the expected return in state s when performing an action a and following policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{2.8}$$

Value functions indicate how desirable certain states or actions are. In this regard, we can define the optimal state-value function $v_*(s)$ as the maximum state-value function over all policies:

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.9}$$

Similarly, the optimal action-value function $q_*(s, a)$ is defined as the maximum action-value function over all policies:

$$q_*(s, a) = \max_\pi q_\pi(s, a) \tag{2.10}$$

The partial ordering over value functions give rise to a partial ordering over policies: $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s),\ \forall\, s$. This allows us to define the optimal policy π_* for which $\pi_* \geq \pi,\ \forall\, \pi$. Reinforcement learning aims at finding the optimal policy π_* that maximizes the value functions v_{π_*} and q_{π_*}. An important theorem states that there exists always at least one optimal policy π_* that is better or equal than all other policies $\pi_* \geq \pi,\ \forall \pi$ [129]. Furthermore, all optimal policies achieve optimal value and action-value functions $v_{\pi_*} = v_*(s)$, $q_{\pi_*} = q_*(s, a)$.

A fundamental property of value functions is that they satisfy recursive relationships that express the value of a state in terms of the immediate reward and discounted value of the successor state. These relationships are denoted as Bellman equations and are defined as:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s \right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi \left(S_{t+1} \right) | S_t = s \right] \\
q_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a \right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi \left(S_{t+1}, A_{t+1} \right) | S_t = s, A_t = a \right]
\end{aligned}
\tag{2.11}
$$

The Bellman equations hold also for optimal value functions and, hence, give rise to the Bellman optimality equations:

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} \mathbb{E}_\pi \left[R_{t+1} + \gamma v_* \left(S_{t+1} \right) | S_t = s \right] \\
q_*(s) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma \max_{a'} q_* \left(S_{t+1}, a' \right) | S_t = s, A_t = a \right]
\end{aligned}
\tag{2.12}
$$

Once v_* or q_* is known, it is straightforward to determine an optimal policy. The Bellman optimality equations form the basis for a variety of algorithms that aim to solve the reinforcement learning problem [129]. While in general, there exists no closed-form solution of the Bellman optimality equation, many iterative solution methods have been designed. In the next sections, we discuss some of these methods. In particular, we present ideas and concepts around Q-learning, elaborate on policy gradient methods and, finally, elaborate on the DDPG method, an actor-critic framework that can successfully solve continuous control tasks.

2.1.2 Action-Value Methods

As described in section 2.1.1, action-value functions describe the desirability of an action in terms of its expected return. Action-value methods use this information to determine optimal policies that select appropriate actions. For instance, let us assume that the action-values q_π of all state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ are known. An optimal policy π_* would then choose those actions that maximize q_π, hence maximizing the expected return for every state-action pair. However, in general, action-values of state-action pairs are not known. Most action-value methods therefore aim at estimating the action-

value function correctly. This is usually done by generalized policy iteration (GPI), an alternating process between estimating action-values for a given policy (policy evaluation) and updating the policy based on the new action-values estimated (policy improvement) [129].

2.1.2.1 Q-learning

An important action-value method that resulted in a breakthrough in reinforcement learning in 1989 is Q-learning [147]. Besides its frequent use, the algorithm forms the basis for a variety of successful reinforcement learning algorithms [41, 52, 53, 93]. Q-learning can be classified as an off-policy temporal difference (TD) method, in which the action-value is directly approximated in terms of the optimal action-value q_*, independently of the current policy π. It is defined as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (2.13)$$

where Q denotes the estimate of q_π and α represents the learning rate. Temporal difference methods approximate the Bellman equation defined in equation 2.11 in two ways: i) they use *samples* of experience instead of calculating an expectation over policies, and ii) they *bootstrap*, i.e. they base their estimates on already learned estimates. Nevertheless, Q-learning has shown to converge with probability 1 to q_* if the learning rate fulfills:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \text{and} \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty \quad (2.14)$$

and all state-action pairs are continuously updated [129]. Directly approximating q_* in 2.13 instead of using an estimate of $q_\pi(S_{t+1}, A_{t+1})$ dramatically simplifies the analysis of Q-learning and, therefore, allowed early convergence proofs of the algorithm.

Q-learning as described originally [146, 147] was designed to handle small finite action and state spaces. However, most reinforcement learning problems approaching real-world complexity require the agent to handle high-dimensional states or continuous action spaces. Over the years, there have been several approaches to account for high-dimensional state or action spaces using variations of Q-learning [22, 86, 130, 131]. One such approach is to use function approximations, a technique to generalize from known examples to unseen data. However, using function approximations gives rise to a series of

challenges such as instability and divergence, especially when combined with off-policy methods such as Q-learning [127, 143]. In the next section, we describe a method in which function approximation in form of a deep neural network was applied to Q-learning. This technique could successfully solve a variety of classic Atari 2600 games [12].

2.1.2.2 Deep Q-learning

Based on advances in training deep neural networks, Mnih et al. [93] propose to use a deep convolutional neural network to approximate the action-value function $Q(s, a; \theta)$, where θ are the parameters of the network. The convolutional neural network serves as a nonlinear function approximator that derives action-values based on images. In this manner, the authors' algorithm, termed deep Q-network (DQN), is able to solve a variety of classic Atari 2600 games using only pixels and the game score as input.

As mentioned in section 2.1.2.1, function approximations give rise to a series of challenges concerning the stability of reinforcement learning methods. Nonlinear function approximators are known to be unstable or even diverge when used to represent action-value functions [143]. In their work, Mnih et al. [93] address such stability issues by the following two fundamental ideas:

1. **Experience replay**: The agent's experience $e_t = (s_t, a_t, r_t, s_{t+1}) \in \mathcal{S} \times \mathcal{A} \times \mathcal{R} \times \mathcal{S}$ is stored at each time t in the dataset $D_t = \{e_1, e_2, ..., e_t\}$. During learning, Q-learning applies updates on mini-batches of experience sampled uniformly from this dataset $(s, a, r, s') \sim U(D)$.

2. **Target network**. An iterative update rule is used that adjusts action-values towards target values which are only periodically updated. The target values are maintained by a separate target network, a duplicate of the original network whose parameters θ^- are synchronized with the learning network every C steps.

While the former technique reduces correlations in the sequence of observations and smooths over changes in the data distribution, the later one reduces correlations with the target values. The action-value function is learned by an iterative update rule that aims at minimizing the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a';\, \theta_i^-) - Q(s, a;\, \theta_i \right)^2 \right] \quad (2.15)$$

where $Q(s', a';\, \theta_i^-)$ denotes the estimate of the target network. While the deep Q-network is able to process high-dimensional sensory input, outperforming most linear learners of previous works on a suite of Atari games, it is still restricted to a discrete set of actions. In the following, we discuss reinforcement learning methods that focus on approximating the policy function π directly instead of first estimating action-values from which the policy can be deduced.

2.1.3 Policy Gradient Methods

In contrast to action-value methods where policies are deduced from estimated action-value functions, policy gradient methods learn parameterized policies directly without requiring action-values for the action selection. We define a (stochastic) parameterized policy $\pi_\theta : \mathcal{S} \times \theta \to \mathbb{P}(\mathcal{A})$ as:

$$\pi_\theta(a|s) = \mathbb{P}\left[A_t = a | S_t = s, \theta_t = \theta\right] \quad (2.16)$$

where $\theta \in \mathbb{R}^d$ represents the policy parameter vector. An advantage of parameterizing policies is that they allow continuous actions spaces. In addition, action probabilities change smoothly compared to other techniques such as ϵ-greedy selection which is often used in action-value methods. This leads to stronger convergence guarantees of policy-gradient methods [129]. Most policy gradient methods seek to learn policies by maximizing a performance measure $J(\theta)$ with respect to the policy parameter. This is usually done by approximating gradient ascent in J, resulting in the update:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla_\theta J(\theta_t)} \quad (2.17)$$

where $\widehat{\nabla_\theta J(\theta_t)}$ denotes a stochastic estimate whose expectation approximates $\nabla_\theta J(\theta_t)$. In practice, the performance is often defined as the expected cumulative discounted reward from the start state $J(\pi) = \mathbb{E}\left[G_0|\pi\right]$.

2.1.3.1 The Policy Gradient Theorem

A challenge with policy gradient methods is that the performance $J(\theta)$ depends both on action selection and the distribution of states in which

those actions are selected. In particular, the effect of the policy on the state distribution depends on the environment and is, therefore, usually unknown [129].

The policy gradient theorem [128] is an important theoretical answer to this challenge. It provides an analytical expression of the policy gradient ∇J with respect to the parameter θ:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_{\mathcal{S}} \rho^{\pi_\theta}(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s)\, q_{\pi_\theta}(s,a)\, da\, ds \\ &= \mathbb{E}_{s\sim\rho^{\pi_\theta}(s), a\sim\pi_\theta} \left[\nabla_\theta log\, \pi_\theta(a|s)\, q_{\pi_\theta}(s,a)\right] \end{aligned} \tag{2.18}$$

where $\rho^{\pi_\theta}(s)$ denotes the state distribution under π_θ. Note that although the state distribution depends on the policy, and hence on the policy parameters, the policy gradient does not depend on the gradient of the state distribution. By using a sample-based estimate of the expectation 2.18, the policy gradient theorem has been used to derive a variety of policy gradient methods [30]. One example is the REINFORCE algorithm where the action-value q_{π_θ} in equation 2.18 is estimated by a sample return G_t [151].

2.1.3.2 Deterministic Policy Gradient

Analogously to the (stochastic) policy gradient theorem, Silver et al. [121] derive the deterministic policy gradient theorem, a theorem that provides an analytic expression for the policy gradient of deterministic policies $\mu_\theta : \mathcal{S} \times \theta \rightarrow \mathcal{A}$. Compared to the stochastic case, deterministic policy gradients resolve in a simpler form that is more efficient to be estimated. Given a deterministic policy μ_θ for which the gradient $\nabla_\theta \mu_\theta$ exists, and assuming that the gradient $\nabla_a q_\mu$ for the action-value function following μ_θ exists, the deterministic policy gradient theorem can be defined as:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_{\mathcal{S}} \rho^{\mu}(s)\, \nabla_\theta \mu_\theta(s)\, \nabla_a q_\mu(s,a)|_{a=\mu_\theta(s)}\, ds \\ &= \mathbb{E}_{s\sim\rho^{\mu}(s)} \left[\nabla_\theta \mu_\theta(s)\, \nabla_a q_\mu(s,a)|_{a=\mu_\theta(s)}\right] \end{aligned} \tag{2.19}$$

Similar to the stochastic policy gradient, the theorem shows that the deterministic policy gradient does not depend on the gradient of the state distribution. Interestingly, Silver et al. show that under certain conditions, the deterministic policy gradient can be considered as a limiting case of the stochastic policy gradient. For a stochastic policy $\pi_{\mu_{\theta,\sigma}}$ where σ denotes the variance, the stochastic policy gradient approaches the deterministic policy as the variance goes to zero:

$$\lim_{\sigma \downarrow 0} \nabla_\theta J(\pi_{\mu_{\theta,\sigma}}) = \nabla_\theta J(\mu_\theta) \tag{2.20}$$

Based on these findings, Silver et al. [121] develop the deterministic policy gradient (DPG) method, a RL algorithm that can handle continuous state and action spaces. However, despite its success on high-dimensional action spaces over earlier stochastic policy gradient methods, DPG is considered computationally expensive and has been observed to show high variance [101].

2.1.4 Actor-Critic Methods

So far, we have seen action-value methods where the policy is implicitly deduced from learned action-values, and policy gradient methods where policies are directly learned by maximizing a performance measure. Actor-critic methods learn approximations to both policy and value function. Parameters of the policy approximator (actor) are learned based on values predicted by the value approximator (critic). Actor-critic methods have been successfully applied to high-dimensional state and continuous action spaces [72, 85]. The framework is particularly popular in deep reinforcement learning where neural networks serve as function approximators [41, 51, 80, 92].

2.1.4.1 Deep Deterministic Policy Gradient Method

A widely used actor-critic algorithm is the deep deterministic policy gradient (DDPG) method presented by Lillicrap et al. [80]. DDPG is a model-free, off-policy actor-critic algorithm that combines the deep Q-network [93] described in section 2.1.2.2 with an actor network to handle continuous action spaces. Lillicrap et al. [80] make use of the deterministic policy gradient theorem [121] to update the actor's parameters as described in equation 2.19. Following the approach of deep Q-learning, the critic is learned by minimizing the loss function:

$$L(\theta^Q) = \mathbb{E}_{s\sim\rho^\beta, a\sim\beta}\left[\left(Q(s,a|\theta^Q) - y\right)^2\right] \tag{2.21}$$

where ρ^β denotes the state distribution under the behavior policy β and y represents the target value:

$$y = r + \gamma\, Q(s', \mu(s')|\theta^Q) \tag{2.22}$$

with s' being the subsequent state observation. A simplified schematic overview of the DDPG method can be found in figure 2.2. Similar to DQN

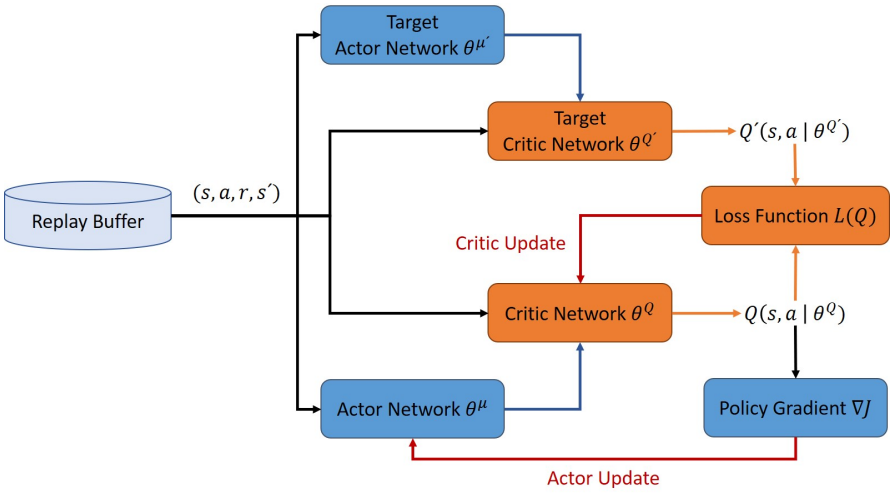


Figure 2.2: Simplified schematic overview of the deep deterministic policy gradient method. The DDPG is an actor-critic method where the actor network approximates a policy based on the predicted values of the critic network. Both networks are updated using gradient-based update schemes. Similar to the work done by Mnih et al. [93], copies of the networks are used to calculate the target values.

[93], Lillicrap et al. [80] maintain a copy of the actor network $\mu'(s|\theta^{\mu'})$ and critic network $Q'(s, a|\theta^{Q'})$ with which the targets for the temporal difference update are computed. However, Lillicrap et al. do not update these networks once every C time steps, but slowly track the learned networks using:

$$\theta' = \tau\theta + (1 - \tau)\,\theta' \qquad \text{with} \quad \tau \ll 1 \tag{2.23}$$

Furthermore, the authors apply batch normalization [61] to normalize each feature across a batch of samples to have zero mean and unit variance. This accounts for different scales and varying physical units in the network's input. To encourage exploration, noise sampled from a noise process $\mathcal{N}$ is added to the actor policy:

$$\tilde{\mu}(s) = \mu(s|\theta^{\mu}) + \mathcal{N} \tag{2.24}$$

where $\mathcal{N}$ is chosen according to the environment. The DDPG algorithm has been successfully applied to a variety of physical control problems with high-dimensional state and continuous action space [80]. In chapter 4, we introduce a spiking actor-critic network that builds upon important concepts of the DDPG network.

2.2 Spiking Neural Networks

The second part of this chapter is dedicated to spiking neural networks (SNNs), a new generation of brain-inspired neural networks that employ spiking neuron models as computational units. Spiking neural networks are designed to process sparse, event-driven data that is encoded via impulses of electrical signals [37]. In this manner, SNNs aim at mimicking biological neural networks in their spike-based communication [35]. When employed on specific neuromorphic hardware, SNNs have proven to be low latency and more energy-efficient than their respective non-spiking counterparts [98, 112, 124, 136]. This makes spiking neural networks a promising candidate for power-efficient computing. However, training spiking neural networks turns out to be a challenging task. Due to the discrete nature of spikes, conventional gradient-based methods that have proven to be successful within artificial neural networks are not practical for SNNs. In the following, we discuss SNNs in more detail by first presenting a set of neuron models used within SNNs. Subsequently, we introduce different coding schemes for spiking data and, finally, discuss challenges around training spiking neural networks.

2.2.1 Neuron Models

Neurons are considered the basic units of information processing within the brain. Connected via synapses, spiking neurons propagate information efficiently through action potentials, i.e. rapid, transitory changes of the resting membrane potential. Over the years, various neuron models have been developed to better understand the behavior of individual neurons as well as their dynamics within a network. In general, these models differ greatly based on the information they want to convey [141]. In this section, we review three popular classes of neuron models: more detailed biophysical models such as the Hodgkin-Huxley model, simplified integrate-and-fire models such as the LIF neuron, and noisy input models that account for the stochastic nature of neurons. For an extensive overview of other neuron models, we refer to [43, 70, 71].

Biophysical Models

Biophysical models describe the dynamics of neurons in terms of their electrical properties. From a biophysical point of view, action potentials are generated because of ionic currents that flow through ion channels in the cell membrane. In 1952, Hodgkin and Huxley [58] conducted a series

of experiments on the giant axon of the squid to measure the flow of currents through the neurons' cell membranes. In their experiments, Hodgkin and Huxley discovered that membranes contain voltage-dependent sodium, potassium and calcium ion channels that control the flow of those ions through the cell membrane. Furthermore, a leakage current was described that is independent of the voltage level. Based on their results, Hodgkin and Huxley succeeded to define the electrical activity of the cell in a set of coupled nonlinear differential equations. These equations resemble what is nowadays known as the Hodgkin-Huxley model [141]:

$$\begin{aligned} C_m \frac{dv(t)}{dt} =& g_L \left[v_L - v(t)\right] + g_{Na}\, m(t)^3 h(t) \left[v_{Na} - v(t)\right] \\ &+ g_K\, n(t)^4 \left[v_K - v(t)\right] + I \\ \frac{dm(t)}{dt} =& \frac{m_\infty(v_(t)) - m(t)}{\tau_m(v(t))} \\ \frac{dh(t)}{dt} =& \frac{h_\infty(v(t)) - h(t)}{\tau_h(v(t))} \\ \frac{dn(t)}{dt} =& \frac{n_\infty(v(t)) - n(t)}{\tau_n(v(t))} \end{aligned} \tag{2.25}$$

where $v(t)$ describes the membrane potential and $m(t)$, $h(t)$, and $n(t)$ denote empirical variables associated with ion channel activation, and inactivation. There are two active channels, a Na^+ and a K^+ channel, and a passive leakage channel L. The external current I acts as a stimulus to the model. C_m represents the capacitance of the membrane, while g_L, g_{Na} and g_K represent the maximum conductance of the ionic channels. Finally, m_∞, h_∞ and n_∞ denote steady-state values, and τ_m, τ_h, and τ_h are experimental time constants. The Hodgkin-Huxley model is regarded as one of the groundbreaking works of the 20th-century in biophysics [44]. Since its occurrence in the 1950s, the Hodgkin-Huxley model has been applied and extended by several other works [38, 105].

Integrate-and-fire Models

While the Hodgkin-Huxley model is an impressive conductance-based neuron model that can reproduce empirical biophysical measurements of neuron activities, its computational complexity is significant. This is particularly problematic when simulating large-scale networks [64]. To reduce the complexity of the model, a variety of simplified neuron models have been

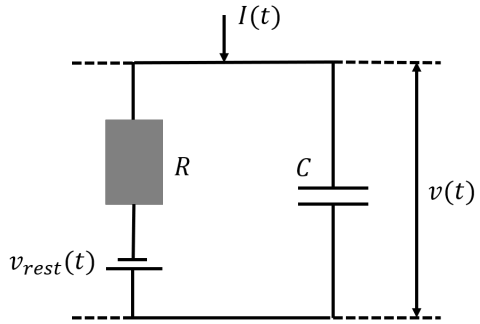


Figure 2.3: Schematic overview of the RC-circuit that represents the leaky integrate-and-fire model as presented by Lapicque [24].

introduced over the years. Integrate-and-fire models are simplified spiking neuron models where the membrane voltage evolves in a subthreshold regime until a threshold v_{th} is reached and an action potential is emitted [141]. One of the first integrate-and-fire model reaches back to 1907 and was introduced by Louis Lapicque [1, 24]. Lapicque modeled the dynamics of a neuron using a simple electric circuit consisting of a parallel capacitor and a resistor (see figure 2.3). Lapicque's approach represents a leaky integrate-and-fire (LIF) model that intends to describe the capacitance and leakage resistance of the cell membrane. When a current $I(t)$ is injected into the neuron or when the neuron receives input from presynaptic neurons, the membrane voltage $v(t)$ begins to rise from its resting value v_{rest} due to the integration of current over time. However, as ions diffuse through the membrane, the voltage potential leaks, driving it slowly back to neuron's resting potential. Lapicque's model captures these dynamics following basic laws of electrical theory, yielding the linear differential equations:

$$\begin{aligned} \tau_m \frac{dv(t)}{dt} &= (v_{rest} - v(t)) + RI(t) \\ \tau_s \frac{dI(t)}{dt} &= -I(t) \end{aligned} \tag{2.26}$$

where R represents the membrane resistance, τ_s describes the synaptic time constant and τ_m denotes the membrane time constant, representing the characteristic time of the membrane potential decay [44]. Lapicque suggested that an action potential was emitted once the membrane capacitor was charged to a certain threshold v_{th}. Subsequently, the capacitor would discharge and the membrane potential would reset to its resting level v_{rest} [1]. Mathematically, this can be described by the following jump condition and transition equation:

$$z(t) = \theta\left(v(t) - v_{th}\right) \qquad \text{with} \quad \theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases} \tag{2.27}$$
$$v(t) = \left(1 - z(t)\right) v(t) + z(t)\, v_{rest}$$

where z denotes the spike emitted, and $\theta(\cdot)$ is a step function such as the Heaviside step function. Although the leaky integrate-and-fire model does not account for behaviors such as neuronal adaption or refractoriness, it is still an extremely useful and popular scheme for studying network dynamics, memory properties or neural coding, especially when simulating a large population of neurons [44]. Furthermore, integrate-and-fire models are not restricted to linear voltage dynamics. It is possible to formulate more general nonlinear integrate-and-fire models. The dynamics of a nonlinear LIF model could be expressed as:

$$\tau_m \frac{dv(t)}{dt} = f(v) + RI(t) \tag{2.28}$$

where $f(v)$ denotes a nonlinear voltage-dependent function.

Noisy Input Models

In contrast to the neuron models discussed so far, the neuronal activity of biological neurons exhibit high degrees of irregularity when exposed to constant stimulus. Whether this flickering activity is the result of thermal noise, microscopic chaos, or the essence of an efficient way of coding is still unclear [44].

Experiments in vitro have shown that neurons, when driven by a current with large amplitude fluctuations, behave more or less deterministically [87]. However, when exposed to slowly changing external stimulus, cortical neurons show noisy behavior [25]. To model the stochastic nature of biological neural networks, a noise term is typically added in the differential equation of the membrane voltage. In this manner, a leaky integrate-and-fire model with noisy input would resemble the following dynamics:

$$\tau_m \frac{dv}{dt} = f(v) + RI^{det}(t) + RI^{noise}(t) \tag{2.29}$$

where I^{det} describes the deterministic input current and I^{noise} describes the noisy input signal. One common approach to model the noise term RI^{noise} is to use white noise $\xi(t)$ characterized by zero mean and a flat power spectrum.

Stein's model [122] describes a noisy leaky integrate-and-fire model with stationary white noise. In the subthreshold regime, the neuron dynamics yield the equation of an Ornstein-Uhlenbeck process (in Langevin form) [18, 144]:

$$\tau_m \frac{dv}{dt} = (v_{rest} - v) + RI^{det}(t) + R\xi(t) \tag{2.30}$$

where v_{rest} denotes the resting potential. The above equation can be rewritten in terms of a Wiener process dW. For discrete time, this yields the iterative update rule:

$$dv = \left(v_{rest} - v + RI^{det}(t)\right) \frac{dt}{\tau_m} + \sigma\sqrt{dt}\, y \tag{2.31}$$

with $dW = \sqrt{dt}\, y$, where y is a random number drawn from a zero-mean Gaussian distribution of unit variance and σ describes the amplitude of the noise input. Analyses based on Stein's model show that stochastic spike arrivals in the input can lead to a broader interspike interval distribution of a neuron [44].

2.2.2 The Neural Code

The question how neurons encode information using spikes is one of the fundamental issues in neuroscience. While a variety of theories and mathematical models have been developed that address this question, it is yet an ongoing debate and no definite answer is known. In computational neuroscience, neural coding schemes can in general be classified into two categories: rate coding and spike coding schemes [43]. In this thesis, we focus on rate coding schemes. Nevertheless, we will also provide a brief overview of popular spike coding schemes.

Rate Coding

Rate coding schemes encode information in terms of the mean firing rate of a neuron. These schemes reach back to the 1920s, when Edgar Adrian showed that the firing rate of neurons in the muscles are related to the force applied to the muscles [4, 5]. There exists at least three different definitions of the mean firing rate of a neuron, depending on the averaging procedure used. In particular, the average could be taken over time, over several repetitions of the same experiment, or over a population of neurons.

The first and most commonly used definition of a neuron's firing rate refers to the temporal average and is denoted as *rate as spike count*. It represents the

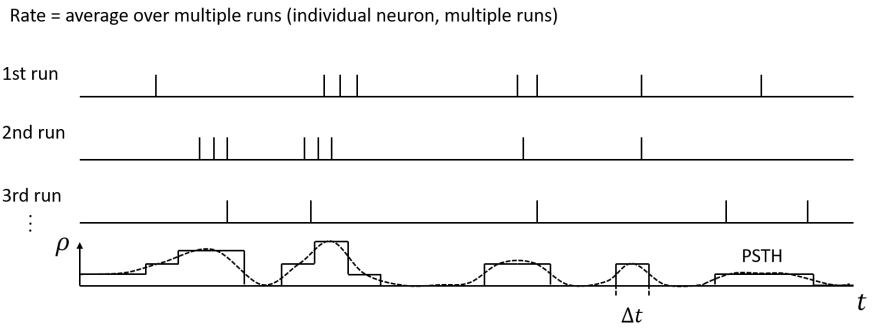


Figure 2.4: Schematic overview of a Peri-Stimulus-Time histogram (PSTH) and the respective spike density ρ function [43].

average of spike counts that occurred within a time interval T and is defined as:

$$\nu = \frac{n_{sp}(T)}{T} \tag{2.32}$$

where $n_{sp}(T)$ denotes the number of spikes that occurred within time interval T. The rate as spike count has been successfully used in many experiments, especially when the input stimuli is stationary or changes slowly.

To account also for time-dependent stimuli, a second definition of rate denoted as *rate as spike density* has been introduced. This type of rate coding takes an average over several repetitions of the same experiment. The spikes emitted during a set of K experiments are reported in a Peri-Stimulus-Time Histogram (PSTH) with bin width Δt (see figure 2.4). The number of spikes $n_K(t; t + \Delta t)$ emitted within a time interval $[t, t + \Delta t]$ across K experiments are divided by the number of experiments K to obtain the activity of the neuron within that time frame. By further division by the interval length Δt, the spike density is obtained:

$$\rho(t) = \frac{1}{\Delta t} \frac{n_K(t; t + \Delta t)}{K} \tag{2.33}$$

the spike density is typically reported in Hz and often called time-dependent firing rate. However, it is considered as biologically implausible as neurons do not need to repeat several runs of the same stimulus to produce information.

Finally, the firing rate might be defined as an average over a population of neurons m. It is denoted as *rate as population activity* and defined as follows:

$$A(t) = \frac{1}{\Delta t} \frac{n_{act}(t; t + \Delta t)}{N} \tag{2.34}$$

where N is the size of the population and n_{act} is the number of spikes summed over all neurons in the population that occur between t and $t+\Delta t$. The population activity is able to reflect temporal changes in the stimulus nearly instantaneously. However, it generally assumes a homogeneous population of neurons with identical connections. In real neural systems, this is hardly given [43]. For a more detailed discussion on rate codes, we refer to [17].

Spike Coding

Due to their simplicity, rate coding schemes became a standard tool for encoding and decoding information conveyed via action potentials [36]. However, behavioral studies have shown that reaction times can be rather short. For instance, a fly is able to react to a stimuli and change the direction of flight within 30-40ms [17]. Humans are able to recognize visual scenes in under 150ms [139]. Such time intervals are not long enough to collect the statistics required for rate coding schemes.

Spike coding schemes base their coding of information on the time of spikes occurring. The most straightforward approach is denoted as *time-to-first-spike*, where information is encoded in the timing of the first spike after a new stimulus has been introduced [43]. Experimental data has shown that most information about a new stimulus is indeed processed during the first few milliseconds of a neuronal response [116, 142]. While a coding scheme based on the latency of the first spike can convey a large amount of information, it is certainly an idealization. There exists a variety of other spike coding schemes such as *phase coding*, which encodes information in the phase of an oscillation of for example the population activity [50, 67, 82].

Experimental evidence indicate the importance of spike timings and suggest that schemes such as the rate as spike count might be too simplistic for encoding information within the brain [2, 75]. Spike coding has proven to be an efficient way of accounting for spike times. However, these coding schemes exhibit a higher complexity, therefore making them less practical than the commonly used rate coding schemes [35].

Other Coding Schemes

Besides the two main neural coding schemes presented in the previous sections, there exists are variety of other techniques to encode or decode

information conveyed via action potentials, which exceed the scope of this thesis. Examples are delta modulation [37] or population codes [107]. Furthermore, it is possible to treat input data to an SNN as direct current (DC) input, assuming constant stimulus over various time steps. However, such (DC) inputs do not exploit the sparse characteristics of spike trains [37].

2.2.3 Training Spiking Neural Networks

Despite their computational efficiency on dedicated neuromorphic hardware, spiking neural networks generally exhibit lower accuracy performances than their respective non-spiking counterparts on typical benchmarks [112]. On the one hand, this can be attributed to the nature of these benchmarks, which resemble conventional data formats such as frame-based images. Thus, an encoding scheme, as described in section 2.2.2, has to be applied that obtains the desired sparse and discrete properties of neuromorphic datasets. This conversion is often lossy and suboptimal.

Another important factor are challenges related to training SNNs [37]. Conventional deep neural networks have shown great success using error backpropagation and gradient-based learning rules [74]. However, such training methods require differentiable activation functions. Spiking neural networks communicate via discrete action potentials, where a discontinuous spike is emitted once a certain voltage threshold is reached (see equation 2.27). In this thesis, we focus on supervised learning methods for SNNs, where a variant of backpropagation is used to minimize a certain loss signal. In particular, we focus on SuperSpike [156], a learning technique that uses surrogate gradients to train multi-layer spiking neural networks. However, there exists a variety of other learning approaches for training SNNs. In general, these approaches can be classified as follows:

- **Conversion from ANNs to SNNs**. Non-spiking ANNs are trained using gradient-based backpropagation. Once a model has been successfully trained, it is converted into an SNN by interpreting its activation functions as firing rates or spike times. Examples are given in [32, 59, 117].

- **Supervised learning using spikes**. The SNN is directly trained using a variation of error backpropagation. Respective rules are given in [10, 19, 20, 145].

- **Local learning rules**. Instead of a global error signal, local spatio-temporal signals govern the weight update. While promising high

hardware-efficiency, local learning rules exhibit difficulties in training lower levels of the model's hierarchy. Popular examples of local learning rules are spike-timing-dependent plasticity (STDP) [16], Hebbian learning [55] or e-prop [11].

In the following, we present the theoretical details of SuperSpike. For a more thorough review of other training algorithms for SNNs, we refer to [132, 145].

SuperSpike

In 2018, Zenke and Ganguli [155] introduced SuperSpike, a nonlinear voltage-based learning rule to train multi-layer spiking neural networks of deterministic leaky integrate-and-fire models. Their work bases on a surrogate gradient approach where the partial derivative of a hidden unit is approximated by the product of the presynaptic spike input and a nonlinear function dependent of the postsynaptic voltage level.

The problem of directly applying gradient-based backpropagation rules to SNNs is that the output of a spiking neuron i, i.e. the spike train $S_i(t) = \sum_f \delta(t - t_i^{(f)})$ with spike times $t_i^{(f)}$, is not differentiable. In particular, the partial derivative $\frac{\partial S_i}{\partial W_{ij}}$ of the neuron's output with respect to the synaptic weights W_{ij} is zero except at spike times $t_i^{(f)}$ at which it is not defined. To overcome this problem, Zenke and Ganguli replace the spike train $S_i(t)$ with a continuous monotonic function $\sigma(v_i(t))$ of the membrane voltage potential, which increases steeply and peaks at the voltage threshold v_{th}. In this manner, the partial derivative of the neuron's output is approximated by:

$$\frac{\partial S_i}{\partial W_{ij}} \approx \sigma'(v_i) \frac{\partial v_i}{\partial W_{ij}} \tag{2.35}$$

where $\sigma'(v_i)$ denotes a smooth surrogate gradient and v_i represents the membrane potential of neuron i. The approach of using surrogate gradients to approximate the gradient of the non-differentiable neuron output is depicted in figure 2.5. The Heaviside step function is approximated by a sigmoid function for which a smooth gradient $\sigma'(v_i)$ exists. Given the dynamics of the LIF neuron as described in equation 2.26, the membrane voltage v_i depends on its own past through its output. Therefore, the derivative $\frac{\partial v_i}{\partial W_{ij}}$ cannot be computed directly. In their derivation, Zenke and Ganguli argue that, assuming (regularized) low firing rates, the impact of this dependence is negligible. In this regard, the filtered presynaptic activity is obtained and the partial derivative of the neuron's output can be approximated by:

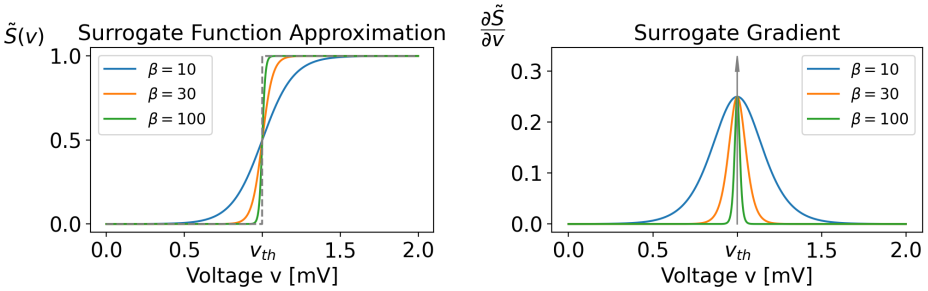


Figure 2.5: Surrogate gradient approach to approximate the partial derivative $\frac{\partial S_i}{\partial W_{ij}}$ of the neuron's spike output by the product of a smooth surrogate gradient $\sigma'(v_i) = \frac{\partial \tilde{S}}{\partial v}$ and the partial derivative of the neuron's membrane voltage with respect to the weights W_{ij} (see equation 2.35). Depicted on the left, the discrete spike output $S(v(t))$ is approximated by a continuous function $\tilde{S}(v)$ for which the respective surrogate gradient can be computed (see figure on the right). With increasing β (see equation 2.38), the surrogate approaches the discrete Heaviside step function $\theta(v_{th})$.

$$\frac{\partial v_i}{\partial W_{ij}} \approx \epsilon * S_j(t) \quad \Rightarrow \quad \frac{\partial S_i}{\partial W_{ij}} \approx \underbrace{\sigma'(v_i)}_{post} \underbrace{(\epsilon * S_j(t))}_{pre} \tag{2.36}$$

where ϵ represents a causal membrane kernel (see [155] for more detail) and $S_j(t)$ denotes a presynaptic spike train. Note that the above equation combines pre- and postsynaptic activity in a multiplicative manner. It is therefore considered as nonlinear Hebbian term. Finally, the full learning rule can be written as an integral over finite time intervals:

$$\Delta W_{ij}^k = r_{ij} \int_{t_k}^{t_{k+1}} \underbrace{e_i(s)}_{\text{error signal}} \underbrace{\alpha * \left(\underbrace{\sigma'(v_i(s))}_{post} \underbrace{(\epsilon * S_j)(s)}_{pre} \right)}_{\lambda_{ij}(s)} ds \tag{2.37}$$

where r_{ij} is the learning rate, $e_i(s)$ denotes an output error signal and α represents a normalized smooth temporal convolutional kernel. Note that the causal convolution with α acts as an eligibility trace λ_{ij}, which transiently stores coincidences between pre- and postsynaptic behavior at the synapse W_{ij}. For efficiency reasons, the surrogate gradient is computed as the partial derivative of the negative half of a fast sigmoid function:

$$\sigma'(v_i) = (1 + |h_i|)^{-2} \qquad \text{with} \quad h_i = \beta(v_i - v_{th}) \tag{2.38}$$

where β can be treated as hyperparameter. In summary, SuperSpike is a biologically plausible learning algorithm for spiking neural networks that, based on two approximations, solves the temporal credit assignment for deterministic LIF neurons.

2.2.4 Simulators for Spiking Neural Networks

To conduct experiments with bio-inspired neural networks, software packages are required that simulate the information flow and dynamics within such systems. In recent years, a variety of software frameworks have been developed to efficiently build and simulate models of biological neurons [27, 45, 47, 123]. However, many frameworks are customized towards a specific field of application. Only recently, Intel introduced Lava [73] - an open-source, platform-agnostic neuromorphic computing framework that aims at uniting the various approaches of simulating SNNs within the neuromorphic community. In general, simulators for spiking neural networks can be differentiated with respect to their hardware support, biological realism, learning algorithms and richness of features. An example of a simulator with high biological realism is `NEURON` [27]. The simulation environment allows for modeling complex biological multi-compartment models, as well as building large-scale networks of point neurons. In contrast, `NEST` [45] focuses on the dynamics and structure of large heterogeneous networks of point neurons rather than the biophysical properties of single neurons. Furthermore, `NEST` supports STDP learning rules and provides a variety of visualization tools. Other popular software frameworks are `Brian` [47, 123], `PyNN` [29] and `BindsNET` [54]. For a more detailed review of state-of-the art simulation packages, we refer to [54].

In this thesis, we use `Norse` [111] to develop, train and evaluate networks of spiking neurons. Similar to `BindsNET` [54], `Norse` is built upon PyTorch [108] and extends the deep learning library by a set of neural components such as coding schemes, bio-inspired neuron models or neuromorphic dataset integrations. Preliminary benchmark tests indicate that `Norse`'s performance on conventional hardware is comparable with frameworks such as `BindsNET` or `GeNN` [111]. `Norse` allows to train SNNs via SuperSpike [155] and other surrogate gradient methods. Furthermore, it supports a variety of neuron models such as standard LIF neurons, Izhikevich neurons [63] or long short-term memory neurons [10]. To solve the dynamics of LIF neurons (see equation 2.26), `Norse` uses a one-step Euler-integration scheme, resulting

in the following update rule:

$$\begin{aligned} v_t &= v_{t-1} + \Delta t \, \frac{1}{\tau_m} \, ((v_{rest} - v_{t-1}) + R I_{t-1}) \\ I_t &= I_{t-1} - \Delta t \, \frac{1}{\tau_s} \, I_{t-1} \end{aligned} \tag{2.39}$$

where Δt denotes the size of the time step, v_t represents the membrane voltage at time t, and the jump equation for the final output is defined as:

$$\begin{aligned} &z_t = \theta \left(v_t - v_{th} \right) \qquad \text{with} \quad \theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases} \\ &v_t = (1 - z_t) \, v_t + z_t \, v_{rest} \end{aligned} \tag{2.40}$$

This concludes the theoretical background of this thesis. Before applying the theory presented in this chapter, it is crucial to look at related work within the area. In the next chapter, we thus provide a brief overview of existing work and recent advances in the field of robust reinforcement learning. Furthermore, we examine approaches that introduce spiking neural networks to reinforcement learning.

Chapter 3

Related Work

Reinforcement learning is a field of research with a rich history and influences from various domains such as optimal control, animal psychology and neuroscience [129]. In recent years, in particular deep reinforcement learning has gained increasing popularity for its successful application to a variety of complex control and decision-making tasks [78]. Notable examples are AlphaGo [120] and OpenAI Five [15], algorithms that defeated human world champions in games such as Go and Dota 2, respectively. However, despite their successes on constrained environments, most reinforcement learning algorithms struggle with generalization and robustness towards uncertainties and unforeseen changes in the environment [94].

In this chapter, we survey the literature on robust reinforcement learning, a field of study that addresses robustness within reinforcement learning. We elaborate on different approaches of robust RL and discuss most recent advances within the area. Finally, we look at existing work that introduces spiking neural networks to reinforcement learning.

3.1 Robustness in Reinforcement Learning

In reinforcement learning, agents learn by interacting with their environment. They try to maximize a goal-directed reward by exploring different states and actions. This trial-and-error learning can be time consuming and might lead to dangerous or unwanted actions. To address these drawbacks, most reinforcement learning agents are trained in simulation before deployed in real world [95, 158]. Such simulations provide safe and efficient environments in which the agent can explore actions freely without the risk of causing real damage to itself, humans or other parts of the environment. However, even

advanced simulators are rarely able to capture all physical properties of the real world. In addition, real-world scenarios are often subject to uncertainties, perturbations and changing environments. This *sim-to-real* gap leads to a significant degradation of performance when the agent is deployed in a real-world setting [129, 158].

To account for uncertainties, perturbations and non-stationary environments, various methods have been developed that improve the robustness of reinforcement learning agents. We follow the approach of Moos et al. [95] and categorize robust RL methods into four different categories: (i) *Transition and reward robust designs* that modify the transition probability function $\mathcal{P}_{ss'}$ during training to account for uncertainties in the system dynamics; (ii) *Disturbance robust designs* that describe modeling errors and parameter changes as disturbance forces acting on the agent; (iii) *Action robust designs* that promote robustness by distorting the agent's actions; and (iv) *Observation robust designs* that present modified observations of the actual state to the RL agent to enhance the system's robustness towards noise and measurement errors. In the following, we briefly discuss each of these approaches.

(i) Transition and reward robust designs. The basic idea behind transition and reward robust designs is to account for possible uncertainties in the system dynamics by modifying the state transition probability matrix $\mathcal{P}_{ss'}$ of an MDP. Most works within this area assume an uncertainty set $\mathcal{U}_{\mathcal{P}}$ of possible transition matrices from which the transition matrix is sampled, aiming at finding a policy that performs well on all system dynamics within the set [7, 149, 152]. This formulation is also known as robust MDP and gives rise to a mini-max problem, where a policy is sought that maximizes the return for the worst possible transition matrix $\mathcal{P}$:

$$\max_{\pi \in \Pi} \min_{\mathcal{P} \in \mathcal{U}_{\mathcal{P}}} \mathbb{E}_{\mathcal{P},\pi} \left[G_0 \right] \tag{3.1}$$

where the return G_0 is defined as in equation 2.3 and $\pi \in \Pi$ is a policy from the set of possible policies Π. While this formulation guarantees robustness towards uncertain system dynamics that might lead to worst-case scenarios, Bagnell et al. [7] have shown that finding a stationary policy that solves the above optimization problem is NP-hard. Hence, more restricted uncertainty sets are considered in practice. Bagnell et al. [7] propose a stochastic dynamic two-player zero-sum game, where an adversary operates on a convex uncertainty set to find the worst possible transition matrix. Furthermore, the authors prove the existence of optimal policies if the uncertainty set is a

compact and convex polytope. However, such polytopes result in solutions with high computational costs. To address this issue, Nilim and Ghaoui [102] describe the uncertainty set in terms of likelihood functions. Nilim and Ghaoui show that such an uncertainty model is statistically accurate and increases the computational costs only slightly compared to the classical MDP formulation. Iyengar [62] further proves that if an uncertainty set upholds the (s, a)-rectangularity property, i.e. the transition probabilities $\mathcal{P}_{ss'}$ within the uncertainty set $\mathcal{U}_{\mathcal{P}}$ are unrelated for each state-action pair, an optimal stationary deterministic policy to equation 3.1 exists. Wiesenmann et al. [150] extend this condition to the less restrictive (s)-rectangularity property, where independence is only required with respect to different states.

One major drawback of a worst-case formulation such as the robust MDP is that respective solutions tend to produce overly conservative policies. This results in poor performances, in particular if the sim-to-real gap is small [81]. To address this problem, various variants of the robust MDP formulation have been proposed, many trying to incorporate additional information about the parameter distributions [125, 154]. Other approaches try to overcome conservatism by loosening the rectangularity assumption. Mannor et al. [91] introduce k-rectangular uncertainty sets, a subclass of coupled uncertainty sets flexible enough to produce non-conservative solutions. Goyal and Grand-Clément [48] consider a factor matrix uncertainty set, where the transition probability is a linear function of a respective factor matrix.

Most of the approaches discussed so far solve the robust MDP setting for the tabular case using some form of dynamic programming approach [102, 152]. However, dynamic programming suffers from the curse of dimensionality and is, hence, not suitable for large or continuous state and action spaces. Shashua and Mannor [118] introduce a robust version of the DQN algorithm that solves large scale robust MDP settings by exchanging the nominal Bellman TD error in the loss function with a robust version accounting for targets that are sampled from an uncertainty set. Other works propose robust policy gradient methods that solve the robust MDP for continuous action spaces [88, 89].

(ii) Disturbance robust designs. From robust control, it is known that modeling errors and parameter changes can be represented by external disturbances to the system [9, 159]. The central idea behind disturbance robust designs is to apply such disturbances to the agent during training to promote robustness. Based on the theory of H_∞-control, Morimoto and Doya [96] present a two-player differential game in which a disturbing agent

aims at applying worst-case disturbances to the system, while the opposing agent tries to control the system. The authors can show that their framework coincides with the analytic solution of a respective linear H_∞-controller. Furthermore, experiments indicate that the system exhibits superior robustness properties on nonlinear control tasks when compared to conventional RL methods. Pinto et al. [113] extend the work by Morimoto and Doya to deep reinforcement learning by using neural networks as function approximators. The authors propose a similar game setup as in [96], where an agent is trained jointly with an adversary that applies destabilizing forces to the system. Instead of modeling these disturbances explicitly and sampling from a huge set of possible trajectories, the adversary is reinforced to minimize the protagonist's reward by applying forces on pre-defined locations, generating trajectories that are in worst α-percentile. Experimental results show that the adversary can effectively sample trajectories with worst rewards leading to a control strategy robust to training initializations, disturbances and changes in the test environment. Even in the absence of parameter changes, the respective algorithm, denoted as robust adversarial reinforcement learning (RARL), exhibits superior performance on a variety of continuous control tasks. However, no theoretical guarantees for the superior performance or convergence properties have been provided by the authors.

(iii) Action robust designs. Another approach to promote robustness within RL is to distort the agent's output during training. Tessler et al. [137] argue that perturbations in the action space can model sudden or constant interrupting forces acting on the agent. The authors introduce two related, but different frameworks to account for perturbations in the action space: the *probabilistic action robust MDP* (PR-MDP) and *noisy action robust MDP* (NR-MDP). While in the former framework, an adversary governs the decision making for short periods of time, the latter framework is characterized by an adversary that applies bounded perturbations to the agent's actions. Both frameworks exhibit robustness improvements compared to conventional RL methods. However, this approach does not handle worst-case perturbations. Pan et al. [106] propose a risk-averse version of RARL [113], denoted as risk-averse robust adversarial reinforcement learning algorithm (RARARL). The authors propose a two-player zero-sum game with an asymmetric reward function that emphasizes aversion to catastrophic events by assigning strong negative rewards to such states. Following the PR-MDP setup, the protagonist and the adversary take turns in taking control over the action space. Similar to [133], Pan et al. model risk as the variance of value functions using

an ensemble of Q-value networks. The algorithm is designed such that the protagonist is encouraged to seek lower variance actions, while the adversary pursues high variance actions. Experiments show that RARARL experiences fewer catastrophic events during testing as compared to agents without an risk-averse adversary.

(iv) Observation robust designs. In real-world scenarios, sensor signals are often affected by noise and small measurement errors. This distorts the agent's perception of the true state of the environment. To promote robustness towards sensor noise and sensory errors, observation robust designs present perturbed observations to the RL agent during training. In most cases, this is done by either adding noise to state observations or by using adversarial attacks on observations to force the agent into unwanted and undesired states [110]. Zhang et al. [157] propose a theoretical framework for modeling state observations within an MDP that have been modified by an adversarial attack. Based on this state-adversarial MDP, the authors develop a policy regularization scheme that exploits convex relations within neural networks. The authors perform a series of experiments that show superior robustness under a variety of different adversarial attacks. Sun et al. [126] extend the state-adversarial MDP framework to the more general state-noisy MDP (SN-MDP) that accounts for both random and adversarial state observation noises. Furthermore, the authors explain the vulnerability of the least square loss in conventional RL towards observation noises based on gradient norms, and show the superior robustness of distributional reinforcement learning methods towards noisy state observations.

3.2 Spiking Reinforcement Learning

Due to their energy efficiency on dedicated neuromorphic hardware, spiking neural networks are promising candidates for solving control tasks under low-power conditions when combined with deep reinforcement learning. However, because of challenges in training and limited amount of neuromorphic datasets, there exists only few SNN-based reinforcement learning algorithms yet [134]. Based on the way SNNs are trained (see section 2.2.3), spiking RL methods can be classified into three different approaches: (i) conversion of RL methods into the spiking domain; (ii) spiking RL methods trained via supervised learning techniques; and (iii) spiking RL approaches exploiting local learning rules.

Patel et al. [109] convert a deep Q-network [93] trained on the Atari game

of Breakout [12] into spiking domain by matching the firing rates of the SNN with the activation of the ANN. The authors report only small degradation of performance for the converted SNN. Furthermore, Patel et al. [109] show that the SNN exhibits superior robustness towards input errors and noise such as occlusion attacks, compared to the initial DQN. The authors hypothesize the binary nature of SNNs to be responsible for the robustness properties, although they do not give theoretical justifications for this assumption. Tan et al. [134] follow a similar approach, improving earlier conversion methods by defining firing rates more accurately. The authors show that the converted network achieves comparable performances on 17 different Atari games. However, the SNN agent is observed to perform poorly on a few games. A drawback of conversion methods is that the resulting networks are constraint to perform equally well or worse than their respective ANN counterparts. In addition, not all types of layers can be converted [112]. Tang et al. [135] propose a different approach where a spiking actor network (SAN) is trained jointly with a deep critic network in order to learn control policies for mapless navigation. The main motivation of this spiking DDPG network (SDDPG) is to combine the energy efficiency of SNNs with the optimally of modern deep RL methods. To train the two networks in conjunction with each other, an extension of spatiotemporal backpropagation (STBP) is used. The hybrid approach could show superior power performance and a higher rate of successful navigation compared to a map-based navigation scheme, the conventional DDPG network and a DNN to SNN conversion. Chen et al. [28] develop a deep spiking Q-network (DSQN) that uses a layer of leaky-integrate (LI) neurons to decode the discrete spike-train output of a SNN into a continuous signal representing the Q-value function. The resulting RL agent is trained using spike-based backpropagation based on a surrogate gradient method. In order to predict Q-values, optimal statistics over the LI cells' membrane potential are collected, where the maximum mebrane voltage indicates most promising results. Experiments show that the performance of DSQN is superior to that of DQN in a range of exemplary Atari games. Furthermore, DSQN exhibits superior learning stability and robustness to adversarial attacks such as the fast gradient sign method (FGSM). Based on reward-based e-prop, Bellec et al. [11] propose a local learning scheme that allows an SNN agent to successfully play a set of Atari games, using a schedule of increasing episode lengths. The authors show that the online scheme reaches similar performance on the Fishing Derby game as respective offline algorithms applied to LSTM networks.

Overall, we see that there exist various concepts and approaches that address robustness within reinforcement learning. While transition and reward robust designs actively modify the underlying dynamics of the system, disturbance, action and observation robust designs implicitly affect the system by perturbing the agent or elements of the environment. However, all methods aim at increasing the agent's robustness by disturbing the underlying MDP. In this thesis, we look at a different approach. Spiking neural networks have been reported to be inherently robust towards sensor noise and measurement errors [109]. We further investigate this promising property by evaluating the performance of a spiking actor-critic network under noisy state observations and varying operating conditions. For the design of the spiking actor-critic network, we draw inspirations from the work done by Tang et al. [135] and Chen et al. [28]. A detailed description of the respective method can be found in the next chapter.

Chapter 4

Method

In this chapter, we present details of the fully spiking DDPG algorithm (FS-DDPG), a spiking actor-critic network intended to solve continuous control tasks with high-dimensional state spaces. We begin this chapter by providing a general overview of the network structure. Then, we look at individual components in more detail. We further derive updates rules to train the FS-DDPG network. Finally, we elaborate on the coding scheme used in this thesis to encode continuous input data into discrete spike patterns.

4.1 Fully Spiking DDPG Network

The FS-DDPG algorithm is a spiking actor-critic network that aims at learning optimal control policies for continuous control tasks by maximizing the expected return in a Markov decision process. It consists of a spiking actor network (SAN) and a spiking critic network (SCN). A schematic overview of the network structure can be found in figure 4.1. At each time step t of a sequence of agent-environment interactions, the RL agent receives a continuous representation of the environment's state $S_t \in \mathcal{S}$. To account for the spiking nature of the actor-critic network, this state representation is first encoded into a discrete spike pattern $\hat{S}_t^{T_{sim}}$ of length T_{sim}, where T_{sim} denotes the number of simulation time steps. A detailed description of the encoding scheme can be found in section 4.2.1. The encoded state observation $\hat{S}_t^{T_{sim}}$ is then passed to the spiking actor network where a corresponding action a_t is predicted. Together with the encoded state observation, information about the action is forwarded to the spiking critic network, where a respective action-value Q_t is predicted. Finally, the action is applied and the environment transits into a new state $S_{t+1} \in \mathcal{S}$. This process is repeated until the episode

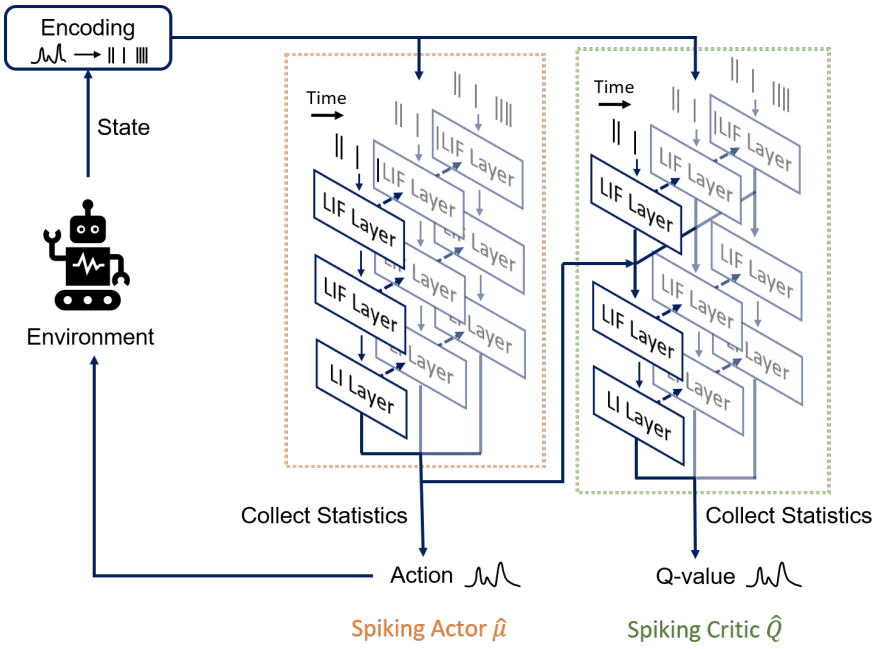


Figure 4.1: Schematic overview of the fully spiking DDPG network. The FS-DDPG comprises a spiking actor network and a spiking critic network. The spiking actor network predicts an action a_t given a discrete spike representation of the state observation $\hat{S}_t^{T_{sim}}$. Based on information about the action predicted and the encoded state observation, the spiking critic network estimates a respective action-value Q_t. The action is finally applied and the environment transits into a new state S_{t+1}.

ends, i.e. a terminal state has been reached. In the following, we discuss the spiking actor network and spiking critic network in more detail.

4.1.1 Spiking Actor Network

The spiking actor network $\hat{\mu}(s|\phi^{\hat{\mu}})$, with parameters $\phi^{\hat{\mu}} \in \mathbb{R}^{d_{\hat{\mu}}}$, specifies the current policy $\pi_{\phi^{\hat{\mu}}}$ by mapping states of the environment to actions to be taken within those states. Similar to the work done by Tang et al. [135], the SAN resembles a fully connected spiking neural network consisting of a series of LIF layers. Each LIF layer constitutes a linear layer followed by a set of leaky integrate-and-fire (LIF) neurons (see equation 2.26 and 2.27). However, in contrast to [135], we do not use a spike-count-based interpolation scheme to obtain the final action command, as this restricts the number of possible actions. Rather, we use a final layer of non-spiking leaky-integrate (LI) neurons to represent continuous-valued actions. Leaky-integrate neurons can be considered as a special case of LIF neurons, where the threshold

voltage in equation 2.27 is set to infinity. In this regard, LI neurons do not fire or reset their membrane potential, but maintain a continuous membrane voltage that follows the dynamics defined in equation 2.26. To solve for the membrane voltage over time, we use the one-step Euler method as defined in equation 2.39. This results in a temporal sequence of membrane potentials predicted by the spiking actor network. As the sequence contains information about the respective action to take, we denote it as *temporal action content* $A_t^{T_{sim}} := \{V_1, V_2, \ldots, V_{T_{sim}}\}$. Together with the encoded state observation, the *temporal action content* is further passed to the spiking critic network, where a respective action-value is predicted. To obtain the final action a_t, we follow the approach of Chen et al. [28] and collect temporal statistics for the sequence of membrane voltages $A_t^{T_{sim}}$. In particular, we compute the action a_t to be the mean *temporal action content*, i.e. the mean membrane voltage of the SAN's last LI layer:

$$a_t = \overline{A}_t^{T_{sim}} = \frac{1}{T_{sim}} \sum_{t=1}^{T_{sim}} V_t^{SAN} \tag{4.1}$$

where V_t^{SAN} represents the membrane voltage of the SAN's leaky-integrate neurons at simulation time step t.

Training SAN via SuperSpike

The aim of the FS-DDPG algorithm is to find an optimal policy $\pi_{\phi^{\hat{\mu}}*}$ that maximizes the expected return from the start distribution $J(\phi^{\hat{\mu}}) = \mathbb{E}_{r_i, s_i \sim \mathcal{P}_{\pi_\phi}, a_i \sim \pi_\phi}[G_0]$. For non-spiking actor-critic methods, Silver et al. [121] have shown that to find such a policy, the non-spiking actor $\mu(s|\theta^\mu)$ can be updated through the deterministic policy gradient algorithm:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\beta(s)} \left[\nabla_{\theta_\mu} \mu_\theta(s|\theta^\mu) \nabla_a Q(s, a|\theta^Q)|_{a=\mu_\theta(s)} \right] \tag{4.2}$$

where $\rho^\beta(s)$ represents the state distribution under the behavior policy β and $Q(s, a|\theta^Q)$ denotes the non-spiking critic network with parameters θ^Q (see section 2.1.3.2 for more details). Note that this formulation of the *policy gradient* is equivalent to finding a deterministic policy $\mu(s|\theta^\mu)$ that maximizes the expected return defined by the action-value function:

$$J(\theta) = \mathbb{E}_{s \sim \rho^\beta(s)} \left[Q(s, a|\theta^Q)|_{a=\mu_\theta(s)} \right] \tag{4.3}$$

where the action-value function $Q(s, a|\theta^Q)$ is assumed to be differentiable with respect to the actions a. We can use this formulation to analogously define

the loss function for the spiking actor network $\hat{\mu}(s|\phi^{\hat{\mu}})$:

$$L_{\hat{\mu}}(\phi^{\hat{\mu}}) = -J(\phi^{\hat{\mu}}) = -\mathbb{E}_{s\sim\rho^{\hat{\beta}}(s)}\left[\hat{Q}(s, A^{T_{sim}}|\phi^{\hat{Q}})|_{A^{T_{sim}}=\hat{\mu}_{\phi(s)}}\right] \tag{4.4}$$

where $\rho^{\hat{\beta}}(s)$ denotes the state distribution under the behavior policy $\hat{\beta}$ and $\hat{Q}(s, A^{T_{sim}}|\phi^{\hat{Q}})$ is the action-value function estimated by the spiking critic network.

As mentioned in section 2.2.3, spiking neural networks propagate non-differentiable action potentials. Therefore, the gradient of the loss function $\nabla_{\phi^{\hat{\mu}}} L_{\hat{\mu}}$ with respect to the SAN's network parameters is not defined. However, it is possible to compute an approximation $\tilde{\nabla}_{\phi^{\hat{\mu}}} L_{\hat{\mu}}$ of the gradient that can be used to minimize the loss by performing gradient descent. In this thesis, we use SuperSpike [155] to approximate the gradient of the loss function and to update the spiking actor network, respectively. As described in section 2.2.3, SuperSpike computes $\tilde{\nabla}_{\phi^{\hat{\mu}}} L_{\hat{\mu}}$ based on surrogate gradients. The weights of the SAN can then be updated following equation 2.37, where the error signal is defined according to the loss function $L_{\hat{\mu}}$.

4.1.2 Spiking Critic Network

The spiking critic network $\hat{Q}(s, A^{T_{sim}}|\phi^{\hat{Q}})$, with parameters $\phi^{\hat{Q}} \in \mathbb{R}^{d_{\hat{Q}}}$, estimates the action-value function to indicate whether a respective action is desirable in the current state. Similar to the spiking actor network, the SCN resembles a fully connected SNN with multiple LIF layers, and a final LI layer to represent continuous values. As depicted in figure 4.1, it first processes the encoded state observation $\hat{S}_t^{T_{sim}}$ and the *temporal action content* $A_t^{T_{sim}}$ by passing them separately through a LIF layer. The output is concatenated and further processed by the network. Similar to the spiking actor network, the final output of the SCN is a sequence of continuous membrane potentials, which we denote as *temporal action-value content* $Q_t^{T_{sim}} := \{V_1, V_2, \ldots, V_{T_{sim}}\}$. To finally obtain an action-value Q_t, temporal statistics for the sequence of membrane voltages $Q_t^{T_{sim}}$ are collected. In particular, we compute the action-value to be the mean *temporal action-value content*, i.e. the mean membrane voltage of the SCN's last LI layer:

$$Q_t = \overline{Q}_t^{T_{sim}} = \frac{1}{T_{sim}} \sum_{t=1}^{T_{sim}} V_t^{SCN} \tag{4.5}$$

where V_t^{SCN} represents the membrane voltage of the SCN's LI layer at simulation time step t.

Training SCN via SuperSpike

In Q-learning [146], the main idea is to learn the action-value function by approximating the recursive Bellman equation 2.11 using temporal difference learning (see section 2.13 for more details). However, while Q-learning is restricted to small finite state and action spaces, we consider function approximators that can handle high-dimensional state and continuous action spaces. Similar to the DDPG algorithm [80], the spiking critic network is therefore trained by minimizing the loss function:

$$L_{\hat{Q}}(\phi^{\hat{Q}}) = \mathbb{E}_{s\sim\rho^{\hat{\beta}}(s),A^{T_{sim}}\sim\hat{\beta}}\left[\left(\hat{Q}(s, A^{T_{sim}}|\phi^{\hat{Q}}) - y\right)^2\right] \tag{4.6}$$

where $\rho^{\hat{\beta}}(s)$ denotes the state distribution under the behavior policy $\hat{\beta}$ and y represents the target value:

$$y = r + \gamma\, \hat{Q}(s', \hat{\mu}(s')|\phi^{\hat{Q}}) \tag{4.7}$$

where s' is the subsequent state of the environment. Similar as before, the gradient of the loss function $\nabla_{\phi^{\hat{Q}}} L_{\hat{Q}}$ is not defined due to the non-differentiable nature of the spiking neurons' output. As for the spiking actor network, we compute an approximation $\tilde{\nabla}_{\phi^{\hat{Q}}} L_{\hat{Q}}$ of the gradient and update the SCN's parameters using the SuperSpike algorithm described in section 2.2.3.

4.1.3 Addressing Stability

The spiking actor network and spiking critic network can be considered as parameterized nonlinear function approximators to estimate an action a_t or an action-value Q_t, respectively. As mentioned in section 2.1.2.2, nonlinear function approximators are known to exhibit stability issues when used within reinforcement learning [143]. For instance, Q-learning is no longer guaranteed to converge when nonlinear function approximators are applied to predict the Q-value function [129]. The DDPG algorithm [80] addresses such convergence issues by introducing a variety of techniques that promote stability (see section 2.1.4.1). Inspired by the success of the DDPG algorithm on continuous control tasks, we apply these techniques also to the FS-DDPG algorithm. In this regard, the FS-DDPG can be considered as a spiking version of the DDPG network. Subsequently, we will briefly discuss these techniques.

Replay Buffer

A major challenge when using neural networks as function approximators within RL are correlations between data samples [93]. For instance, as the agent interacts with its environment, each state observation depends on the previous state. However, most optimization algorithms for neural networks assume that data samples are independently and identically distributed, which is not a fair assumption in reinforcement learning. As in DDPG [80], FS-DDPG uses a replay buffer to address this issue. The main idea is that at each time step t, the agent's experience $e_t = (s_t, A_t^{T_{sim}}, r_t, s_{t+1})$ is stored in a finite-size buffer $D_t = \{e_1, e_2, \ldots, e_t\}$. During learning, the network is updated by sampling a mini-batch uniformly from the replay buffer $(s, A^{T_{sim}}, r, s') \sim U(D)$. This reduces correlations in the sequence of state observations and smooths over changes in the data distribution.

Target Networks

When using nonlinear function approximators with TD-methods such as Q-learning, another source of instability are correlations between the action-values Q and target values y [80]. In particular, the Q-update is inclined to diverge when the function approximator that is updated is being used to compute the target value. To address this issue, Mnih et al. [93] suggest target networks that compute target values. These networks are only periodically updated, wherefore correlations between the action-values and the target are reduced. Lillicrap et al. [80] slightly modify this approach. Rather than updating the target networks by copying the weights of the learned networks every C iterations, the authors introduce a *soft* update scheme where the weights of the target networks slowly track the learned network. We follow the approach of Lillicrap et al. and use a copy of the spiking actor network $\hat{\mu}'(s|\phi^{\hat{\mu}'})$ and spiking critic network $\hat{Q}'(s, A_t^{T_{sim}}|\phi^{\hat{Q}'})$ to compute the target value:

$$y = r + \gamma\, \hat{Q}'(s', \hat{\mu}'(s'|\phi^{\hat{\mu}'})|\phi^{\hat{Q}'}) \tag{4.8}$$

where s' denotes the subsequent state observation. As in DDPG [80], the weights of the target networks are updated by the soft Polyak update rule:

$$\phi' = \tau\phi + (1-\tau)\,\phi' \qquad \text{with} \quad \tau \ll 1 \tag{4.9}$$

where the target network ϕ' slowly tracks the learned network ϕ.

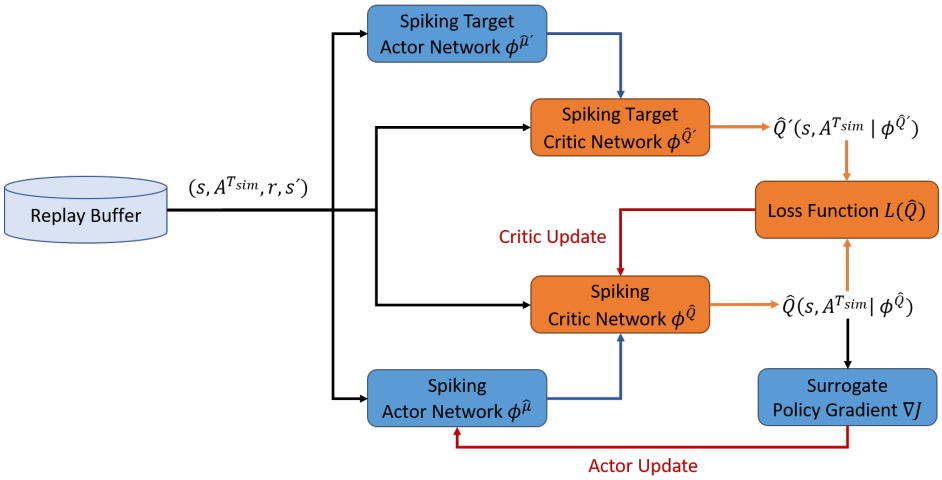


Figure 4.2: Schematic overview of the training framework for the FS-DDPG algorithm. Experience is first stored and then sampled uniformly from a replay buffer. Furthermore, spiking target networks are used to compute the target y for the update of the spiking critic network. Finally, the SAN is updated based on a surrogate policy gradient.

Action Noise

Similar to the DDPG algorithm, the FS-DDPG represents an off-policy method where the exploration can be considered separately from the learning algorithm. To encourage exploration of the FS-DDPG, we follow the approach of Lillicrap et al. [80] and add noise sampled from a noise process $\mathcal{N}$ to the actor policy:

$$\hat{\mu}''(s) = \hat{\mu}(s|\phi^{\hat{\mu}}) + \mathcal{N} \tag{4.10}$$

where the noise process $\mathcal{N}$ can be chosen depending on the task. In this thesis, we look at action noises sampled from normal distributions and Ornstein-Uhlenbeck processes [144].

The complete training framework of the FS-DDPG algorithm is depicted in figure 4.2. At each training step, a mini-batch of the agent's experience is sampled uniformly from the replay buffer $(s, A^{T_{sim}}, r, s') \sim U(D)$ and passed to the spiking actor network $\hat{\mu}(s|\phi^{\hat{\mu}})$ and its respective target network $\hat{\mu}'(s|\phi^{\hat{\mu}'})$. Based on the encoded state observation, the spiking actor networks predict the temporal action contents $A^{T_{sim}}$ and $A^{T'_{sim}}$, respectively, which are passed to the corresponding spiking critic networks $\hat{Q}(s, A^{T_{sim}}|\phi^{\hat{Q}})$ and $\hat{Q}'(s, A^{T'_{sim}}|\phi^{\hat{Q}'})$. Subsequently, the action-value Q and the target value y are

predicted. Finally, the SAN is updated by minimizing the loss function defined in equation 4.4, using gradient descent and SuperSpike [155] as described in section 2.2.3. Similarly, the SCN is updated based on the loss function defined in equation 4.6, where the target is computed according to equation 4.8. The target networks are slowly tracking the learned networks as specified in equation 4.9.

4.2 Input Encoding

The FS-DDPG algorithm is a spiking actor-critic network that communicates via discrete action potentials. However, as further discussed in section 5, the FS-DDPG is used to solve continuous control tasks in environments with continuous state observations $S_t \in \mathcal{S}$. In this regard, information about the continuous input signals has to be encoded into discrete spike patterns $\hat{S}^{T_{sim}}$ before the FS-DDPG network can process them. In section 2.2.2, we have discussed several techniques to encode information into spike trains. In this thesis, we use an encoding scheme that treats continuous input signals as direct currents to a LIF cell, assuming constant stimulus over a number of time steps T_{sim}.

4.2.1 Min-Max Current Coding

Leaky integrate-and-fire neurons emit a spike once the membrane voltage $v(t)$ exceeds a certain threshold v_{th}. When using an encoding scheme that treats continuous input signals as direct currents to LIF neurons, it is important to note that low input currents might result in silent neurons where no spikes are emitted, while high currents might result in over-excited neurons that emit a spike every simulation time step (see figure 4.3). More importantly, silent neurons stay silent for even lower input signals, while over-excited neurons do not differentiate between yet higher input currents. This results in a tremendous loss of information. In this regard, it is important to scale the continuous input signal appropriately before applying it to the LIF neuron. In this thesis, we use a min-max scaling scheme where the range of possible state observations $[S_{min}, S_{max}]$ is rescaled to the interval $[a, b]$:

$$\tilde{S}_t = a + \frac{(S_t - S_{max})(b - a)}{S_{max} - S_{min}} \tag{4.11}$$

with S_{max} denoting the upper bound and S_{min} representing the lower bound of possible state observations. By choosing appropriate boundaries for

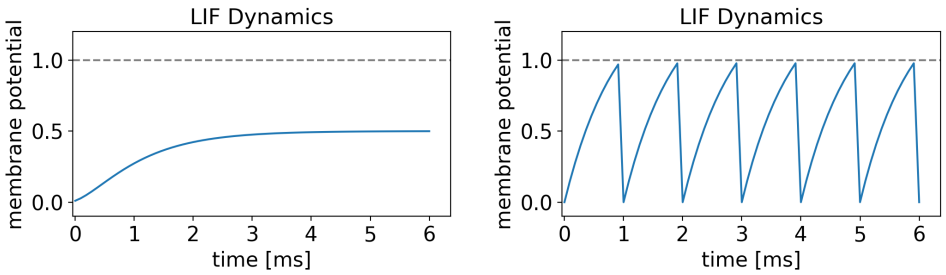


Figure 4.3: Membrane potential of a leaky integrate-and-fire neuron for different direct input currents. In the left graph, the direct current input is so low that the neuron's membrane potential does not exceed the threshold voltage v_{th} indicated by the grey dashed line. In this regard, the neuron remains silent. On the right hand side, a neuron is over-excited by a very high direct input current. Hence, the neuron spikes at every time step.

the interval $[a, b]$, we can avoid neurons that are silent or over-excited for a wide range of values. Note however that for some environments, state observations might be unbounded and respective values for S_{min} and S_{max} are not available. Even more so, traditional reinforcement learning is a paradigm for online learning, where the agent does not learn from a fixed batch of data, but from data collected by a series of agent-environment interactions [40]. Therefore, values for S_{min} and S_{max} can not simply be deduced from minimizing or maximizing over a given dataset. We address this issue by collecting data about possible state observations during a series of interactions between the non-spiking DDPG network and a respective environment E. In particular, given a set of state observations D_S^E encountered during a set of interactions between the DDPG network and the environment E, we define the minimum and maximum state observation for E as:

$$S_{min}^E = \min_{S \in D_S^E}(S) \qquad S_{max}^E = \max_{S \in D_S^E}(S) \tag{4.12}$$

Once S_{min}^E and S_{max}^E are computed, the signal can be rescaled using the formula in equation 4.11. To properly model continuous state observations as direct currents flowing into the LIF neuron, we need to ensure that the input current is positive. Furthermore, we want to treat high negative values similarly to high positive values, as physical quantities such as positions or velocities with similar magnitude often constitute similar meaning (while indicating opposing directions). To achieve this, we duplicate the array of state observations and reverse its sign $S'_t = -S_t$. We further append the two

arrays S_t and S'_t and apply a rectified linear unit (ReLU) to the concatenated signal. This results in a in an array of positive state observations where initially positive state values are followed by originally negative states observations.

In this thesis, we use what we denote as *min-max current coding scheme* to encode continuous input signals into discrete spike patterns. A schematic overview of the complete coding scheme can be found in figure 4.4. First, the empiric bounds of the state observations S^E_{min} and S^E_{max} are computed based on a series of agent-environment interactions between the non-spiking DDPG network and the environment E (see equation 4.12). State observations S_t that exceed the empiric bounds are clipped to these bounds according to the following equation:

$$S_t = \min(\max(S_t, S^E_{min}), S^E_{max}) \tag{4.13}$$

Subsequently, the negative state duplicate $S'_t = -S_t$ is defined, concatenated with S_t and the array is applied to a rectified linear unit. The result is then scaled to the interval $[a, b]$ using the min-max scaling scheme defined in equation 4.11. Finally, the transformed signals are applied as direct currents to a layer of LIF neurons, where a respective spike pattern $\hat{S}^{T_{sim}}$ of length T_{sim} is obtained.

Let $\mathcal{F}_\theta : \mathcal{S} \rightarrow \hat{\mathcal{S}}^{T_{sim}}$ denote the final min-max current coding scheme, parameterized by the interval boundaries $[a, b]$ and dynamics of the LIF neurons, i.e. $\theta = [a, b, \tau_m, \tau_s, v_{rest}, v_{th}]$. To find appropriate parameters θ, we are interested in minimizing the information loss that arises during the encoding process. In particular, we aim at finding a set of parameters θ that results in diverse spike patterns for different input values. While an injective mapping between input values and spike patterns would be ideal in this regard, such a mapping is infeasible given the limited number of simulation time steps T_{sim} and dynamics of LIF neurons. Instead, we aim at finding the set of parameters θ that maximizes the entropy for a sequence of spike patterns obtained from $\mathcal{F}_\theta$. In particular, let $\boldsymbol{S} \in \mathbb{R}^N$ be an array of N equidistant input values ranging from S^E_{min} to S^E_{max}. The mapping $\hat{\boldsymbol{S}}^{T_{sim}} = \mathcal{F}_\theta(\boldsymbol{S})$ results in a binary matrix of size $N \times T_{sim}$, where the n-th row represents the encoded spike pattern of the n-th element of $\boldsymbol{S}$. By computing the entropy of this matrix, we obtain a quantitative measure for the diversity of spike patterns obtained from $\mathcal{F}_\theta$. However, instead of computing a single entropy value for the matrix $\hat{\boldsymbol{S}}^{T_{sim}}$, we look at a profile of entropy values obtained from various local scales. For this, we define a two-dimensional filter F_k of size $k \times k$ that, similar to an average pooling layer, slides over the binary matrix $\hat{\boldsymbol{S}}^{T_{sim}}$ and computes the average activation within the neighborhood

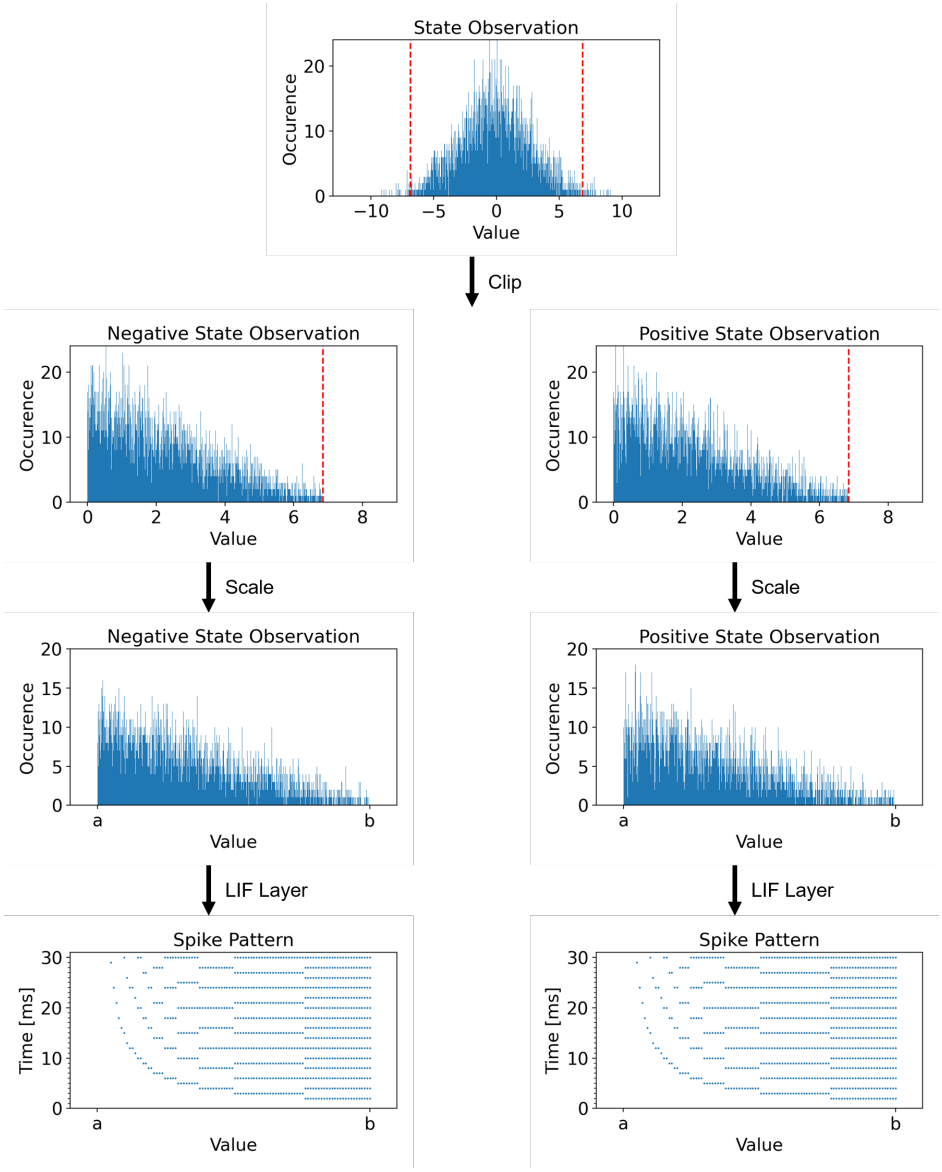


Figure 4.4: Schematic overview of the min-max current coding scheme. First, state observations S_t are clipped to the empiric boundaries S^E_{min} and S^E_{max}. Then, observations are duplicated and negated $S'_t = -S_t$. The resulting array is concatenated with the original observations and applied to a rectified linear unit. Subsequently, the state observations are scaled to the interval $[a, b]$ by applying the min-max scaling defined in equation 4.11. Finally, the transformed signals are applied as direct current to a layer of LIF neurons. This yields the discrete spike pattern $\hat{S}^{T_{sim}}$.

k. For $k = 1, 2, \ldots, \min(N, T_{sim})$, we obtain $K = \min(N, T_{sim})$ different representations of the binary matrix $\hat{\boldsymbol{S}}^{T_{sim}}$, expressing information on various

scales [46]. In particular, the representation $\hat{\boldsymbol{S}}_k^{T_{sim}}$ can be computed using:

$$\hat{\boldsymbol{S}}_k^{T_{sim}}(i,j) = \frac{1}{k^2} \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \hat{\boldsymbol{S}}^{T_{sim}}(i+m, j+n) \tag{4.14}$$

where $\hat{\boldsymbol{S}}_k^{T_{sim}}(i,j)$ represents the value of $\hat{\boldsymbol{S}}_k^{T_{sim}}$ for row i and column j. Note that during the filter operation, the dimensions of matrix $\hat{\boldsymbol{S}}_k^{T_{sim}}$ are reduced to $(N-k+1) \times (T_{sim}-k+1)$. For each matrix $\hat{\boldsymbol{S}}_k^{T_{sim}}$, we compute its Shannon's entropy according to:

$$H(\hat{\boldsymbol{S}}_k^{T_{sim}}) = -\sum_i p_i \, log_2(p_i) \tag{4.15}$$

where p_i denotes the probability of the i-th activation value of matrix $\hat{\boldsymbol{S}}_k^{T_{sim}}$. This results in an entropy profile as depicted in figure 4.5. By performing a grid search over possible parameter configurations θ, we choose the set of parameters for which the entropy profile is highest. The final parameters of our min-max current coding scheme are summarized in table 4.1.

In the next chapter, we present a series of experiments designed to evaluate the robustness of the FS-DDPG algorithm towards sensor noise and varying operating conditions. With this, we aim to answer the research questions introduced in section 1.1.

Table 4.1: Parameters θ for the min-max current coding scheme $\mathcal{F}_\theta$. The parameters are based on an entropy analysis of resulting spike patterns.

Parameter	
a	0.5
b	13.1
Synaptic Time Constant τ_s	0.02
Membrane Time Constant τ_m	0.01
Threshold Voltage v_{th}	1.0
Resting Voltage value v_{rest}	0.3
Time Step Size Δt	0.001
Simulation time T_{sim}	30

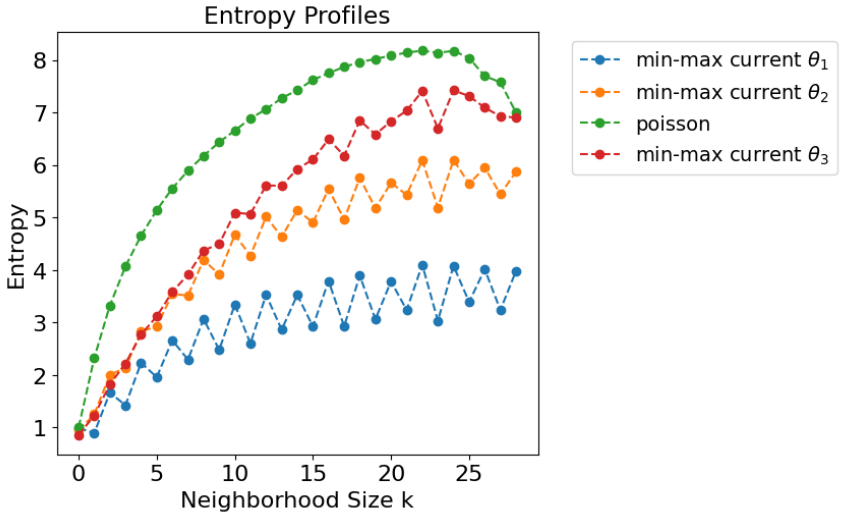


Figure 4.5: Entropy profiles for different parameter configurations θ. As can be seen, the choice of parameters highly influences the entropy profile of the coding scheme. For a Poisson coding, where the input signal is translated into a spike train drawn from a random Poisson process with probability proportional to the input value, the entropy profile is highest. However, even the deterministic min-max current coding scheme can achieve a high entropy profile when tuned thoroughly.

Chapter 5

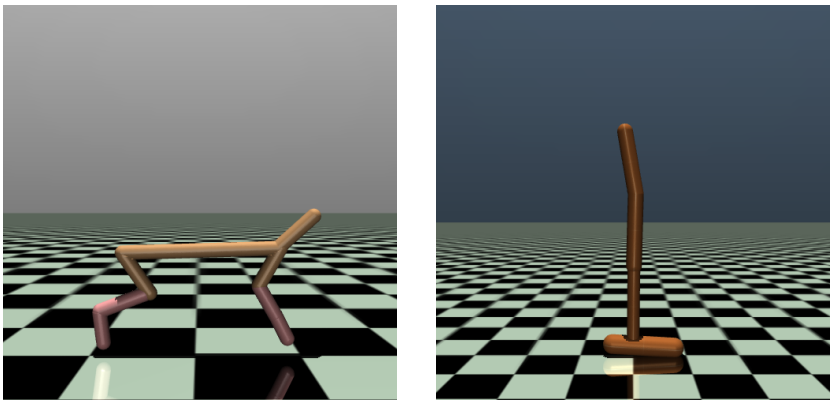
Experimental Setup

To test our hypotheses and answer the research questions defined in section 1.1, we conduct a series of experiments that study the robustness of the FS-DDPG and non-spiking DDPG algorithm towards sensor noise and perturbations of the environment. In particular, we train and evaluate both algorithms on two different locomotion problems defined within the MuJoCo simulator [140]. After successful training of the FS-DDPG and non-spiking DDPG algorithm on the respective tasks, we evaluate the algorithms' robustness towards sensor noise by gradually introducing noise to the agents' state observations. In addition, we vary parameters such as the agents' mass or joint stiffness to analyze the agents' response to perturbations of the environment. Finally, we compare the performance of the two algorithms in the presence of the disturbances mentioned.

In this chapter, we outline the details of these experiments and elaborate on respective design choices, for instance regarding the type of noise introduced. We begin this chapter by providing an overview of the MuJoCo simulation environment and different control tasks on which we train and evaluate our RL agents. Subsequently, we present details regarding the training process in section 5.2. We motivate our choice of codebase and describe how we choose network parameters. In section 5.3, we elaborate on the robustness evaluation. In particular, we describe how we introduce noise to the agents' state observations and which parameters of the environment we perturb.

5.1 Simulation Environment

We conduct experiments on two different continuous control tasks defined within the MuJoCo [140] simulation environment. The MuJoCo simulator



(a) HalfCheetah-v3 (b) Hopper-v3

Figure 5.1: Renderings of different continuous control tasks defined within the MuJoCo simulator. Each task represents a locomotion problem in which a multi-joint robot is controlled to move forwards. The tasks vary in their complexity as they comprise different numbers of states and control variables (see table 5.1 for more details).

is a physics engine that models optimization-based contact dynamics, while representing multi-joint dynamics in efficient generalized coordinates. This allows for a fast and accurate simulation of embodied multi-joint structures interacting with their environment [140]. The simulator is used for model-based control in various research areas such as robotics, biomechanics and animation.

We interface MuJoCo through OpenAI Gym [23], an open-source library for developing and evaluating reinforcement learning algorithms. The two control tasks used in this thesis represent robotic locomotion problems of different complexity. Renderings of the simulated robots and their respective environments can be found in figure 5.1. Each task involves the control of complex multi-joint dynamics based on high-dimensional state observations. For better reproducibility and a fair comparison with related work, we use the original control tasks defined in [23] with their respective default parameters and do not modify the reward function or environment during training. In the following, we describe each locomotion problem in more detail.

HalfCheetah

The *HalfCheetah* environment is based on the work done by Wawrzyński [148]. It represents a locomotion problem in which a 2-dimensional cheetah robot is controlled to run forward as fast as possible (see figure 5.1a). The

biped robot consists of 9 rigid links connected via 8 joints. While the robot's head and torso are fixed, torques can be applied on the remaining 6 actuated joints that connect the robot's front and back thighs, shins and feet. In this regard, the robot's action space is 6-dimensional $\mathcal{A} \in \mathbb{R}^6$. The environment consists of a 17-dimensional observable state space $\mathcal{S} \in \mathbb{R}^{17}$ that comprises continuous information about the joints' positions and velocities. For a detailed overview of the state space's quantities, we refer to [23].

The goal of the cheetah is to move forward, i.e. to its right, as fast as possible. The reward function is composed of two parts: a positive reward for moving forward and a negative reward for extreme actions. The full reward function can be formalized as:

$$R_{t+1} = \underbrace{\frac{x_{t+1} - x_t}{dt}}_{forward\ reward} - \underbrace{c \left\| a_t \right\|^2}_{action\ reward} \tag{5.1}$$

where x_t and x_{t+1} denote the cheetah's position in horizontal direction before and after the action $a_t \in \mathbb{R}^6$ is applied, respectively. The parameter dt represents the time between two actions and is set to its default value of 10^{-2}. The coefficient c is a constant with default value 10^{-1}. An episode of the halfcheetah environment terminates when the episode length exceeds 1000 time steps.

Hopper

This task includes the control of a 2-dimensional one-legged robot that moves forward via a series of jumps. The environment is based on the locomotion problem introduced by Erez et al. [34]. The *hopper* robot consists of four rigid links - a torso at the top, a thigh in the middle and a shin followed by a foot in the bottom (see figure 5.1b). Its links are connected via three actuated joints. In this regard, the robot's action space is 3-dimensional $\mathcal{A} \in \mathbb{R}^3$. The observable state space is 11-dimensional and comprises information about the positions and velocities of the robot's links. A summary of the environment's characteristics can be found in table 5.1.

The aim of the hopper is to move forward in the direction to its right as fast as possible, while performing controlled movements. This includes that the hopper does not fall down or jump too high, in which case the episode would terminate early. Its reward function extends the reward function of the previous task by a term that encourages long episodes. In this regard, the reward function comprises three terms and can be defined as:

$$R_{t+1} = \underbrace{\frac{x_{t+1} - x_t}{dt}}_{forward\ reward} - \underbrace{c\,\|a_t\|^2}_{action\ reward} + \underbrace{1}_{alive\ reward} \tag{5.2}$$

where dt is set to a default value of 0.002 and the coefficient c is set to 10^{-3}. An episode of the hopper environment ends when its length exceeds 1000 time steps or the robot is in an unhealthy state, i.e. it is considered to perform uncontrolled actions. An unhealthy state is defined to occur if one of the following conditions holds true:

- The value of a state variable is no longer finite
- An observable state variable has an absolute value greater than 100
- The robot's position above the ground has exceeded 0.7 meters (the robot has jumped too high)
- The angle of the joint connecting the robot's torso and thigh is less than 0.2 radians (the hopper has fallen down)

The two control tasks vary in complexity as they comprise different numbers of state and control variables. In particular, the halfcheetah environment involves higher state and action dimensions than the hopper environment (see table 5.1). Furthermore, the tasks comprise different reward functions. While the halfcheetah environments focuses on maximizing the robot's forward velocity, the hopper environment also accounts for the quality of movements, as undesired motions lead to an early termination of the episode, which results in a lower reward.

To test our hypotheses and to answer the research questions defined in section 1.1, we train and evaluate the FS-DDPG and non-spiking DDPG algorithm on each of the two continuous control tasks. In the following, we elaborate on the training process and outline details regarding the choice of parameters. In section 5.3, we describe a series of experiments specifically designed to evaluate the algorithms' robustness towards sensor noise and perturbations of the environments.

Table 5.1: Overview of the dimensions and boundaries of the state and action space for each control task.

Control Task	**dim**$(\mathcal{S})$	**Boundaries**$(\mathcal{S})$	**dim**$(\mathcal{A})$	**Boundaries**$(\mathcal{A})$
HalfCheetah	17	$[-\infty, \infty]$	6	$[-1, 1]$
Hoppper	11	$[-\infty, \infty]$	3	$[-1, 1]$

5.2 Implementation Details

Recent work on reproducibility within deep reinforcement learning has highlighted difficulties in reproducing results of state-of-the-art deep RL algorithms [56]. In particular, both intrinsic sources of non-determinism such as random network initialization or stochastic environments, and extrinsic sources such as the choice of hyperparameters or codebases have been found to contribute to the difficulties reported. To circumvent these challenges and strive towards reproducibility, we run our experiments for various random seeds, i.e. different random initializations, report all hyperparameters and rely on codebases that are well established within the community. Furthermore, we publish our code[1] for reproducing the results presented in this thesis.

While spiking neural networks are characterized by short inference times when applied on dedicated neuromorphic hardware [112], their training on conventional hardware can be rather time consuming and computationally demanding. However, to obtain a robust measure of an algorithm's performance and to overcome the reproducibility crisis reported in [56], a high number of training iterations on different random seeds is desired. This results in a trade-off between obtaining results in a reasonable amount of time and ensuring statistical significance. Due to time constraints of this degree project, we train our RL algorithms for 1 million time steps across 5 different random seeds on each control task. We further evaluate the performance of each algorithm every 10^4 time steps on a separate evaluation environment with no exploration noise by averaging over 5 episodes.

For the implementation of the FS-DDPG algorithm, we draw inspirations from the work done by Tang et al. [135] and Chen et al. [28]. Similar to Tang et al. [135], we define a spiking actor network (SAN) with two hidden LIF layers that comprise 256 LIF neurons each. However, in contrast to [135], we do not interpolate the final action based on the spike-count of the SAN's final LIF layer. Instead, we follow the approach of Chen et al. [28] and use an output layer of non-spiking leaky-integrate neurons to represent continuous-valued actions (see section 4.1). Finally, we apply a hyperbolic tangent (tanh) function to bound the actions. Similarly, the spiking critic network consists of two hidden LIF layers with 512 LIF neurons each and a final LI output layer. As described in section 4.1, the SCN first passes the encoded state observation $\hat{S}_t^{T_{sim}}$ and temporal action content $A_t^{T_{sim}}$ through separate LIF layers and then concatenates the processed information. A schematic overview of the FS-DDPG's final network architecture can be found in figure B.1.

[1] https://github.com/PMMon/SpikingRL

For an accelerated training of the FS-DDPG network on conventional hardware (GPUs), we make use of PyTorch's *CUDA Graphs* feature [100]. *CUDA Graphs* is a mechanism to reduce performance overheads induced by multiple individual launches of GPU operations through the CPU. In particular, it defines multiple CUDA kernels as a single graph of operations instead of launching a sequence of operations individually. While the feature can result in a significant speedup in training time, it does yet not support capture-safe[2] optimizations based on Adam [68]. Hence, network parameters are updated using RMSProp [49, 57]. As described in section 4.1.3, the networks are trained based on mini-batches sampled uniformly from a replay buffer. Following the approach of Fujimoto et al. [41], we use a strictly exploratory policy for the first N time steps to alleviate the impact of the policy's initial parameters. After N steps, exploration is encouraged by adding noise to the actor policy as described in equation 4.10.

For a fair comparison between the FS-DDPG and the non-spiking DDPG algorithm, we use a similar network size for both algorithms. In particular, the non-spiking actor network comprises two feedforward layers of 256 hidden nodes followed by rectified linear units between each layer, and a tanh unit following the final output layer. Similarly, the non-spiking critic network contains two hidden layers of size 512, followed by ReLUs. A schematic overview of the DDPG's network architecture can be found in figure B.2. As for the FS-DDPG algorithm, we optimize the actor and critic network of the DDPG based on RMSProp.

In their work, Henderson et al. [56] highlight the importance of proper hyperparameter selection. The authors' show that the choice of hyperparameters can have a significant effect on the algorithm's performance across different environments. For proper hyperparameter selection, we perform an automated hyperparameter search based on Optuna [6], an open-source framework for hyperparameter optimization. To sample from the space of hyperparameters, we use an independent sampling scheme based on the tree-structured Parzen estimator (TPE) algorithm. The sampler chooses the set of hyperparameters x which maximizes the ratio $\frac{l(x)}{g(x)}$, where $l(x)$ is a Gaussian Mixture Model (GMM) fitted to the set of parameters that is associated with the best objective values, and $g(x)$ is a GMM fitted to the remaining hyperparameters. To find appropriate hyperparameters, we use a budget of 10 trials with a maximum of $2 \cdot 10^5$ time steps for each algorithm and environment. A detailed overview of the final hyperparameters for each algorithm and environment can be found in appendix B.

[2] For more details, see this issue report.

Finally, Henderson et al. [56] show that differences in implementation can have a significant influence on the performance of deep RL algorithms. Hence, the choice of codebase plays an important role when training and evaluating reinforcement learning methods. To account for this issue, we base our implementation on the well established *Stable-Baselines3* [115] framework. *Stable-Baselines3* implements a variety of state-of-the-art deep RL algorithms, which have been tested and benchmarked extensively against reference codebases [115]. In particular, we use the framework's implementation of the non-spiking DDPG algorithm [80] and extend the code to implement our FS-DDPG network. For training and evaluating our algorithms, we build upon *RL Baselines3 Zoo* [114], a training framework for RL algorithms that is based on *Stable-Baselines3*. As mentioned in section 2.2.4, we use the spiking neural network simulator `Norse` [111] to simulate the information flow and dynamics within the FS-DDPG. `Norse` builds upon PyTorch and allows to develop, train and evaluate networks of spiking neurons. For more details about `Norse`, we refer to section 2.2.4.

5.3 Robustness Evaluation

Based on the control tasks described in section 5.1, we conduct a series of experiments that explore the RL agents' robustness towards sensor noise and perturbations of the environment. Unlike most approaches within robust reinforcement learning that disturb the underlying MDP during *training* to increase robustness (see section 3.1), we aim at evaluating the agents' inherent robustness towards such perturbations. In this regard, we conduct experiments on agents that are *trained on undisturbed environments*. We evaluate the algorithms' robustness towards sensor noise by introducing noise to the agents' observations. Furthermore, we vary parameters such as the agents' mass or joint stiffness to analyze their response to perturbations of the environment. In the following, we present details of these experiments and elaborate on the respective design process.

5.3.1 Sensor Noise and Measurement Errors

In real-world scenarios, sensors and other measurement devices are often subject to errors, noise and inaccuracies. Such errors might perturb the state observations of an RL agent and can lead to a distorted perception of the environment [126, 157]. The resulting mismatch between the agent's perception and the true state of the environment can have a severe impact

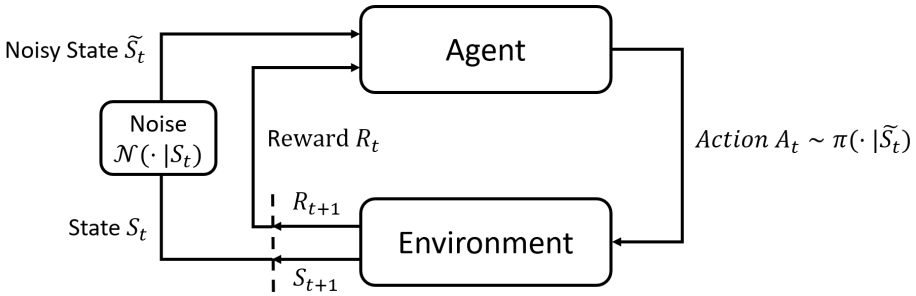


Figure 5.2: Schematic overview of the state-noisy Markov Decision Process [126]. The state-noisy MDP extends the conventional MDP by a noise process $\mathcal{N}$ that perturbs the state observation S_t by mapping it to the noisy state observation $\tilde{S}_t$. The agent predicts an action $A_t \sim \pi(\cdot|\tilde{S}_t)$ based on the perturbed state observation $\tilde{S}_t$, while the true state of the environment S_t remains unchanged.

on the agent's performance. Reinforcement learning algorithms that are not robust to such distortions are likely to predict suboptimal actions that might lead to catastrophic events [157]. For a successful application in real-world scenarios, it is therefore crucial that respective RL algorithms exhibit a certain degree of robustness towards sensor noise and measurement errors.

To formally describe disturbances such as sensor noise and measurement errors within reinforcement learning problems, we adapt the *state-noisy Markov Decision Process (SN-MDP)* introduced by Sun et al. [126]. A schematic overview of the SN-MDP can be found in figure 5.2. It extends the classical MDP by a noise process $\mathcal{N}(\cdot|S_t)$ that maps state observations S_t to noisy state observations $\tilde{S}_t$. Note however, that only the agent's state observations are perturbed, while the true state of the environment remains unchanged. In this regard, the action is predicted based on the perturbed state observation $A_t \sim \pi(\cdot|\tilde{S}_t)$, while the environment transits from the true state S_t to a new state S_{t+1}, following the transition probability function $\mathcal{P}_{SS'}$. In particular, as the true state S_t might be different from the noisy state observation $\tilde{S}_t$, the agent might choose a suboptimal action $A_t \sim \pi(\cdot|\tilde{S}_t)$ that leads to poor behavior.

Unlike Sun et al. [126], we focus on random state noise that models measurement errors and sensor noise, and do not consider adversarial state noise as this would exceed the scope of this thesis. In particular, we look at two different noise types: *additive white Gaussian noise (AWGN)* and an adapted version of *impulse noise*.

Additive White Gaussian Noise

Due to its mathematical tractability, additive white Gaussian noise is commonly used to model noise stemming from a variety of natural noise sources. For instance, thermal noise generated by the thermal motion of charge carriers in electrical circuits can be modeled as AWGN [8]. As thermal noise is present in all electrical circuits, it is a major source of sensor noise. We apply additive white Gaussian noise to the agent's state observation S_t by adding noise sampled from a zero-mean normal distribution $\mathcal{N}(0, \sigma^2)$ with variance σ^2 to the array of state observation. In particular, the i-th element of the array of noisy state observations $\tilde{S}_t$ at time t is obtained by:

$$\tilde{S}_t^i = S_t^i + Z_t^i \quad \text{with} \quad Z_t^i \sim \mathcal{N}(0, \sigma_i^2) \tag{5.3}$$

where S_t^i denotes the i-th element of the array of unperturbed state observations and Z_t^i represents a sample of additive white Gaussian noise. For our experiments, we look at the impact of different state variables S_t^i affected by AWGN. In particular, we evaluate the agents' performance for individual state variables affected by additive white Gaussian noise, as well as for multiple state variables perturbed simultaneously. When applying AWGN to state observations, we need to make sure that the signal-to-noise ratio stays within reasonable bounds. In particular, as each state variable lies within different value ranges, we cannot use the same variance σ_i^2 for all elements. Otherwise, certain state variables might be highly affected by noise, while others are not. Let m^i be the mean absolute value of state observation i, computed from a series of unperturbed agent-environment interactions. We define the magnitude M^i of state observation i by rounding the mean absolute value m^i down to its first non-zero decimal. In this regard, a state observation with mean absolute value of $m^i = 0.0974$ would exhibit a magnitude of $M^i = 0.09$. We set the final variance of the AWGN to be a fraction c of the state observations' magnitude $\sigma_i^2 = c\, M^i$. To account for various noise levels, we evaluate the agents' performance for different fractions $c = [10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}]$.

Impulse Noise

Impulse noise is characterized by sharp and sudden disturbances in the input signal. It usually occurs within acoustic signals or images. However, one could imagine situations in which sudden disturbances also occur for other signals, for instance due to erroneous sensors. We model impulse noise by defining

the probability $p_i \in [0, 1]$ for which a sudden disturbance d_i is applied to state observation S_t^i. In particular, we obtain the noisy state observation as:

$$\tilde{S}_t^i = \begin{cases} \text{sign}\,(S_t^i)\, d^i & \text{if} \quad p_i \geq X \sim \mathcal{U}(0,1) \\ S_t^i & \text{otherwise} \end{cases} \tag{5.4}$$

where $d^i = c\, M^i$ is a fraction of the magnitude M^i of state observation i, as defined in the previous section. For our experiments, we evaluate the agents' performance for individual state variables affected by impulse noise, as well as for multiple state variables disturbed simultaneously. To account for different magnitudes of disturbances, we evaluate the agents' performance for different fractions $c = [10^0, 10^1, 10^2, 10^3]$. In particular, we consider impulse noise characterized by high amplitudes. Furthermore, we look at different values of p_i, i.e. the probability that a disturbance occurs.

5.3.2 Perturbations of the Environment

Besides sensor noise and measurement errors, RL agents applied in real-world scenarios are often affected by perturbations of the environment [33]. For instance, when a robot navigates through unknown terrain, the friction coefficient between the floor and the robot's legs might change abruptly as the floor conditions alter. Furthermore, properties such as the robot's joint stiffness or mass might vary over time as the robot's mechanical parts deteriorate [83]. Reinforcement learning algorithms trained on certain conditions of the environment often struggle with such perturbations [33]. However, to successfully apply RL agents in real-world scenarios, it is important that these algorithms are able to cope with such changes.

To analyze the agents' robustness towards perturbations of the environment, we evaluate each agent on test environments with varying initial conditions. In particular, we vary the agents' torso mass and joint stiffness. Let $\boldsymbol{K}_e^{train}$ be the joint stiffness matrix for training environment e. The new joint stiffness matrix for the respective evaluation environment is then defined as:

$$\boldsymbol{K}_e^{test} = c\, \boldsymbol{K}_e^{train} \tag{5.5}$$

where $c \in \mathbb{R}$ is a constant. To account for different values of the agents' joint stiffness, we evaluate the agents' performance for various fractions within the range $c \in [10^{-4}, 10^1]$.

Similarly, we define m_e^{train} to be the agents' torso mass for training

environment e. The new torso mass for the respective test evaluation environment is defined as:

$$m_e^{test} = c\, m_e^{train} \tag{5.6}$$

To account for different values of the agents' torso mass, we evaluate the agents' performance for various fractions within the range $c \in [10^{-4}, 10^{1}]$. In the next chapter, we present results of the experiments introduced in this chapter.

Chapter 6

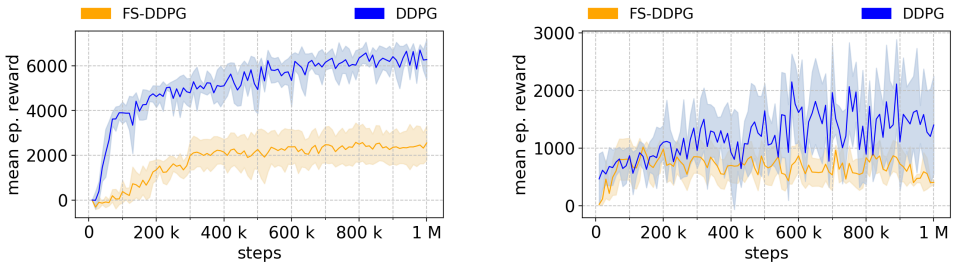
Results

In the previous chapter we have introduced a series of experiments that study the robustness of the FS-DDPG and non-spiking DDPG algorithm towards sensor noise and perturbations of the environment. In the chapter at hand, we present the results of these experiments.

We begin by looking at the algorithms' performance on the undisturbed control tasks described in section 5.1. In particular, we display learning curves and report best performance values for each algorithm and environment. Subsequently, we present results of the robustness evaluation introduced in section 5.3. First, we report the algorithms' performance in the presence of sensor noise and measurement errors. Then, we look at the algorithms' robustness towards perturbations of the environment. Finally, we relate the results to our initial hypotheses and answer the research questions defined in section 1.1. In the next chapter, we discuss respective findings in a broader context.

6.1 Training Results

We train and evaluate the FS-DDPG and non-spiking DDPG algorithm on the two different continuous control tasks described in section 5.1. As mentioned in section 5.2, each algorithm is trained for 1 million time steps across 5 different random seeds on each locomotion problem. Note that in order to study the algorithms' inherent robustness, we train both algorithms on undisturbed environments. Figure 6.1 depicts the algorithms' performance for each control task, evaluated every 10^4 time steps on a test environment with no exploration noise. As can be seen, both algorithms converge to a stable solution after a finite number of time steps. However, compared to the non-



(a) HalfCheetah-v3 (b) Hopper-v3

Figure 6.1: Learning curves of the FS-DDPG and non-spiking DDPG algorithm for the different control tasks. Each algorithm is trained for 1 million time steps across 5 different random seeds and evaluated on a test environment with no exploration noise. The curves denote the mean return averaged over 5 episodes. Shaded regions represent standard deviations of the mean return over the different random seeds.

spiking DDPG algorithm, the FS-DDPG exhibits a lower final performance for both control tasks. Furthermore, the DDPG algorithm displays a higher learning speed for the halfcheetah environment. During training, we save the agents' state, i.e. their learnable parameters, for the best performing runs. After training, we evaluate these *best models* on a test environment for each control task. Respective results are presented in table 6.1. The results are consistent with the results depicted in figure 6.1 and underline the superior performance of the non-spiking DDPG algorithm on the two undisturbed control tasks.

A detailed discussion about the FS-DDPG's inferior performance on the undisturbed control tasks can be found in section 7.1. In the next section, we evaluate the algorithms' response to sensor noise and perturbations of the environment.

Table 6.1: Average return $\pm$ standard deviation of the algorithms' *best models*. Each model is evaluated for 15 episodes on a test environment with no exploration noise. As each algorithm is trained across 5 different random seeds, 5 different *best models* are evaluated for each algorithm. The return is averaged over all episodes and models.

Control Task	**FS-DDPG**	**DDPG**
HalfCheetah	2188.18 ± 1158.06	$\mathbf{6599.48 \pm 1007.26}$
Hoppper	871.27 ± 278.11	$\mathbf{2910.85 \pm 456.38}$

6.2 Robustness Evaluation

Once we have successfully trained the FS-DDPG and non-spiking DDPG algorithm on the different locomotion problems, we study their robustness towards sensor noise and perturbations of the environment. As described in section 5.3, we evaluate the algorithms' robustness towards sensor noise by gradually introducing noise to the agent's state observations. Furthermore, we vary parameters such as the agents' mass or joint stiffness to analyze their response to perturbations of the environment. We begin by presenting the algorithms' performance in the presence of sensor noise and measurements errors. Subsequently, we look at the algorithms' response to perturbations of the environment. Finally, we discuss the results in the light of our initial hypotheses and answer the research questions defined in section 1.1.

6.2.1 Sensor Noise and Measurement Errors

As described in section 5.3.1, we evaluate the algorithms' robustness towards sensor noise and measurement errors by introducing two different types of noise to the agents' state observations: *additive white Gaussian noise* and an adapted version of *impulse noise*.

We apply AWGN to the agents' state observations S_t by adding noise sampled from a zero-mean normal distribution $\mathcal{N}(0, \sigma_i^2)$ with varying variance $\sigma_i^2 = c\,M^i$ to the array of state observations (see equation 5.3), where c is a fraction of the state observation's magnitude M^i. In particular, to account for various noise levels, we evaluate the agents' performance for different fractions $c = [10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}]$. Similarly, we apply our version of impulse noise to the agents' state observations S_t by introducing sharp, sudden disturbances $d^i = c\,M^i$ with probability $p_i \in [0, 1]$ to the array of state observations (see equation 5.4). In particular, we look at impulse noise that is characterized by high amplitudes. To account for various noise levels, we consider different fractions $c = [10^{0}, 10^{1}, 10^{2}, 10^{3}]$.

For each control task, we evaluate the algorithms' performance on a test environment perturbed as described above. Note that we evaluate the algorithms' *best models* acquired during the training process, as described in section 6.1. For our experiments, we look at the impact of different state variables perturbed by AWGN or impulse noise. In the following, we present respective results for the halfcheetah environment. Similar results have been obtained for the hopper environment, which are presented in appendix A.1.2.

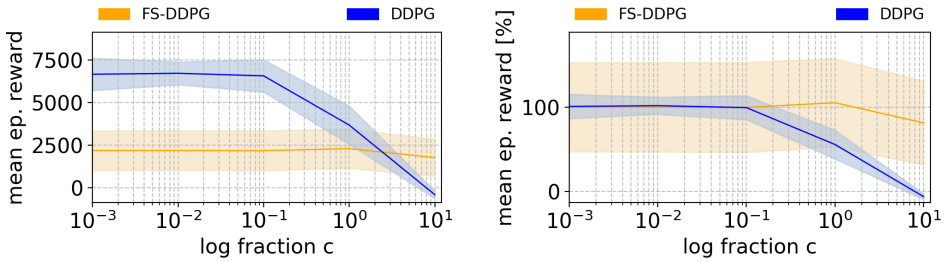


Figure 6.2: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a test environment in which the agents' state observations are distorted by additive white Gaussian noise. In particular, the observation of the agents' *front tip angle* is perturbed by AWGN. To account for different noise magnitudes, various values for the fraction c are evaluated (see section 5.3.1). On the right, the average return is reported relatively to the algorithms' performance on an undisturbed environment (as reported in table 6.1). Shaded regions represent standard deviations of the average return over the different random seeds.

HalfCheetah

As mentioned in section 5.1, the *halfcheetah* environment is characterized by a high-dimensional state space with 17 different state observations that comprise information about the positions and velocities of the robot's links. Furthermore, the robot is controlled by 6 actuated joints that connect its front and back thighs, shins and feet.

We begin by analyzing the impact of additive white Gaussian noise applied to the agents' state observations. In particular, we introduce AWGN to the state observation S_t^1 that records the *angle of the robot's front tip*. Figure 6.2 depicts the algorithms' performance on a test environment in which the state observation is perturbed respectively. The figure shows that both algorithms exhibit a drop in performance as the noise level, i.e. the variance $\sigma_1^2 = c\,M^1$ of the Gaussian noise, increases. However, we can observe that the performance of the non-spiking DDPG algorithm starts to decrease for much lower noise levels than for the FS-DDPG algorithm. In particular, while the non-spiking DDPG algorithm deteriorates for fractions higher than $c \geq 10^{-1}$, the performance of the FS-DDPG algorithm stays almost constant until it slightly decreases for fractions higher than $c \geq 10^{0}$. Moreover, the average return of the non-spiking DDPG algorithm drops below the average return of the FS-DDPG algorithm for fractions higher than $c \geq 2 \cdot 10^{0}$. This behavior becomes even more evident when looking at the algorithms' performance relative to the performance on the undisturbed environment,

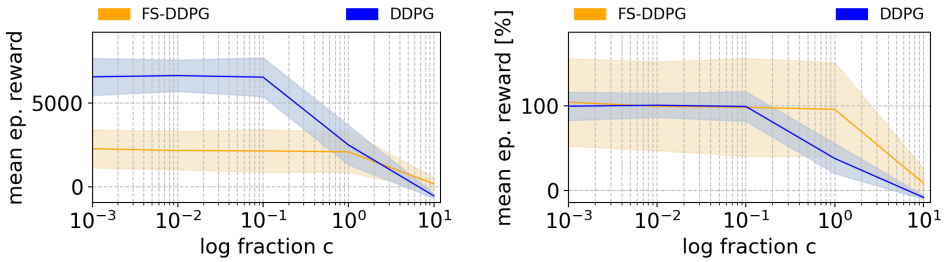


Figure 6.3: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a test environment in which the agents' state observations are distorted by additive white Gaussian noise. In particular, observations of the agents' *front tip's angular velocity and velocity in x and y- direction* are perturbed by AWGN.

depicted in the right graph of figure 6.2. While the FS-DDPG exhibits a nearly constant performance for all fractions of c, seeming to be almost agnostic to the perturbations, the performance of the non-spiking DDPG degrades significantly for noise levels with fraction higher than $c \geq 10^{-1}$.

Similar observations can be made when perturbing multiple state observations at once. In particular, we introduce AWGN to the robot's state observations S_t^8, S_t^9 and S_t^{10}, which represent the robot's *front tip velocity in x and y-direction, and its angular velocity*, respectively. Figure 6.3 illustrates the corresponding performance of the two algorithms. Similar to previous results, the performance of the FS-DDPG algorithm remains almost constant for a wide range of noise levels. Only for noise levels with fractions higher than $c \geq 10^{0}$, the performance of the FS-DDPG starts to decrease. In contrast, the non-spiking DDPG algorithm deteriorates already for low noise levels with fractions higher than $c \geq 10^{-1}$. Notably, the average return of the non-spiking DDPG algorithm drops below the the average return of the FS-DDPG algorithm for fractions higher than $c \geq 2 \cdot 10^{0}$. Similar results are obtained for other state observations perturbed by additive white Gaussian noise. For a detailed overview of these results, we refer to appendix A.1.1.

Subsequently, we evaluate the impact of impulse noise applied to the agents' state observations. In particular, we introduce impulse noise as defined in equation 5.4 to the state observations S_t^2 and S_t^3, which measure the *angle of the agents' back thighs and shins*. Figure 6.4 depicts the algorithms' performance for a disturbance probability of $p = 0.2$. Similar as for AWGN, we observe a decrease in performance for both algorithms as the noise level, i.e. the magnitude of the disturbance, increases. However, while the non-spiking

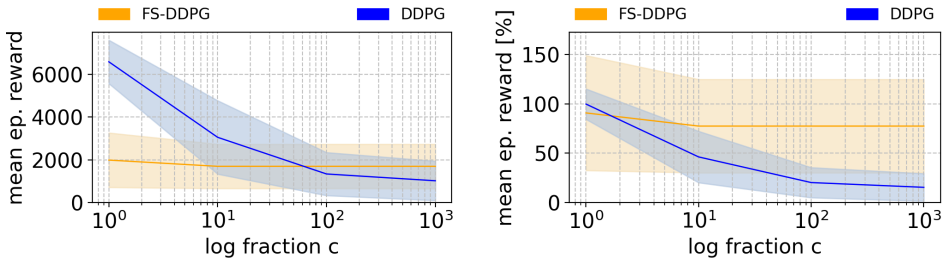


Figure 6.4: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a test environment in which the agents' state observations are distorted by impulse noise. In particular, observations regarding the *angle of the agents' back thighs and shins* are perturbed by impulse noise with a probability of $p = 0.2$. To account for different noise magnitudes, various values for the fraction c are evaluated (see section 5.3.1). Shaded regions represent standard deviations of the average return over the different random seeds.

DDPG algorithm exhibits a significant performance drop by almost 100% relative to the unperturbed case, the FS-DDPG deteriorates only slightly. In particular, we observe that the performance of the FS-DDPG algorithm remains constant for fractions higher than $c \geq 10^1$, while the DDPG's performance decreases continuously for increasing noise levels. This behavior becomes even more evident when increasing the probability that a state observation is affected by impulse noise. Figure 6.5 shows the algorithms' performance for a disturbance probability of $p = 0.4$. As can be seen, both algorithms degrade as the noise level increases. However, while the DDPG

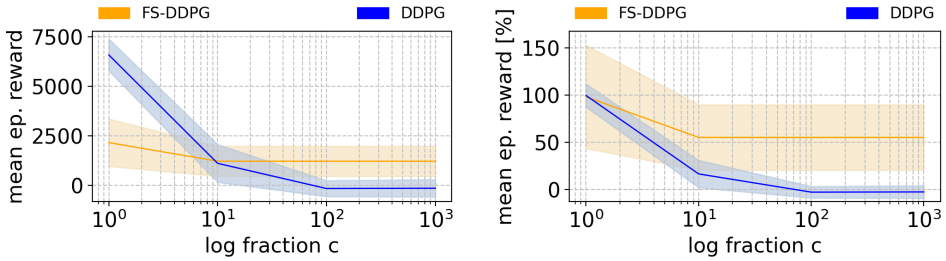


Figure 6.5: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a test environment in which the agents' state observations are distorted by impulse noise. In particular, observations regarding the *angle of the agents' back thighs and shins* are perturbed by impulse noise with a probability of $p = 0.4$.

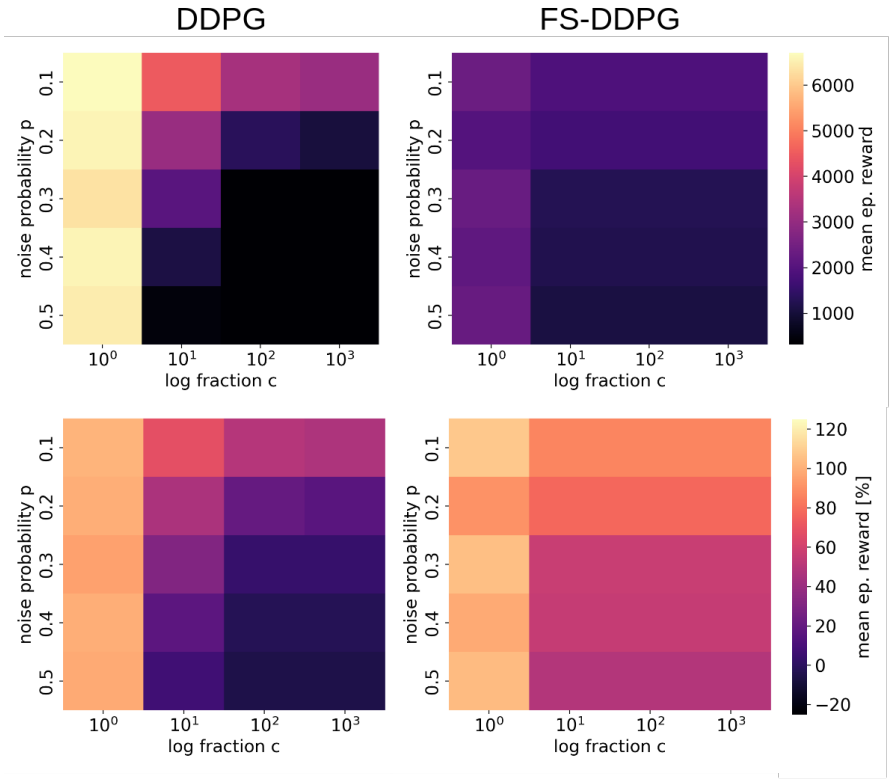


Figure 6.6: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a test environment in which the agents' state observations are distorted. In particular, state observations regarding the *angle of the agents' back thighs and shins* are perturbed by impulse noise. The average return is reported for various values of the fraction c and disturbance probability p. Furthermore, the average return is illustrated relatively to the algorithms' performance on an undisturbed environment (see lower row). For a detailed overview of the exact values for each combination of c and p and respective standard deviations, see table A.1 and A.2.

algorithm exhibits a significant drop in performance with increasing noise magnitude, the FS-DDPG deteriorates only moderately and shows constant performance for fractions greater or equal to $c \geq 10^1$. Moreover, the FS-DDPG algorithm outperforms the DDPG algorithm for fractions higher than $c \geq 10^1$, as the average return of the non-spiking DDPG algorithm falls below that one of the FS-DDPG algorithm.

To better see the algorithms' behavior for various probability values p and fractions c, we visualize their performance in a heatmap, as depicted in figure 6.6. The figure depicts the algorithms' mean return for different combinations of p and c. As illustrated, the DDPG algorithm performs well for low

noise levels characterized by small fractions c and disturbance probabilities p. However, as the noise magnitude and disturbance probability increases, the algorithm highly degrades, exhibiting low average return values compared to the initial performance. In contrast, the FS-DDPG algorithm shows a much more robust behavior towards impulse noise. While the average return for small values of c and p is considerably lower than for the DDPG algorithm, the performance of the FS-DDPG algorithms drops only slightly for increasing noise magnitudes and disturbance probabilities. This behavior becomes even more evident when looking at the algorithms' average return relative to the performance on the undisturbed environment, as depicted in the lower row of figure 6.6.

Similar results are obtained for other state observations perturbed by impulse noise. For a detailed overview of respective results, we refer to appendix A.1.1. In the next section, we look at the algorithms' response to perturbations of the environment.

6.2.2 Perturbations of the Environment

To evaluate the agents' robustness towards perturbations of the environment, we evaluate each algorithm on test environments with varying initial conditions. In particular, we vary the agents' torso mass and joint stiffness. As described in section 5.3.2, we define the new torso mass m_e^{test} and joint stiffness matrix $\boldsymbol{K}_e^{test}$ as fractions c of the initial parameters for which the agents' are trained on (see equation 5.5 and 5.6). To account for different test conditions, we consider various fractions within the range $c \in [10^{-4}, 10^{1}]$. In the following, we present results for each control task. Similar to before, we evaluate the algorithms' *best models* acquired during the training process.

HalfCheetah

We begin by evaluating the FS-DDPG and non-spiking DDPG algorithm on test environments with varying torso masses. Figure 6.7 depicts the algorithms' performance for different test conditions. As can be seen, both algorithms deteriorate on test environments for which the agents' torso mass m_{hc}^{test} differs from the initial mass m_{hc}^{train}, i.e. for fractions $c > 1$ and $c < 1$. In particular, the agents exhibit a significant drop in performance for large torso masses. This is partly due to the fact that the heavy torso pushes down on the cheetah robot and restricts it to slow forward movements. For test conditions in which the torso weights less than during training, both algorithms exhibit a moderate drop in performance of around 50% compared to the

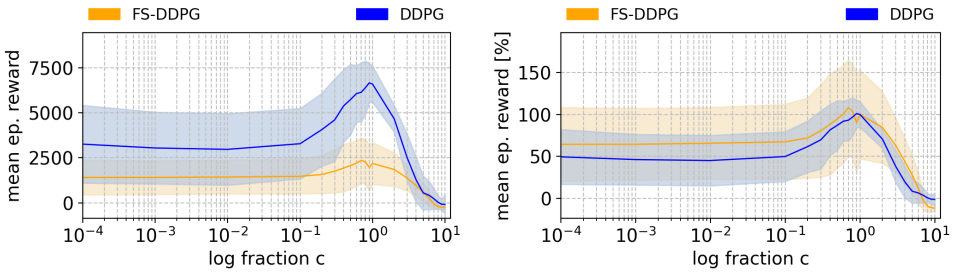


Figure 6.7: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on test environments with different torso masses. Shaded regions represent standard deviations of the average return over the different random seeds.

unperturbed case. In general, both algorithms seem to be equally affected by the perturbations introduced. This becomes even more evident when looking at the algorithms' average return relative to the performance on the unperturbed environment. While the relative performance of the FS-DDPG algorithm is slightly higher than of the DDPG algorithm, both algorithms seem to respond comparably to varying test conditions.

Similar observations can be made when varying the algorithms' joint stiffness matrix $\boldsymbol{K}_{hc}^{test}$. Figure 6.8 shows the performance of the FS-DDPG and DDPG algorithm for different joint stiffness matrices $\boldsymbol{K}_{hc}^{test}$. Similar as before, both algorithms exhibit a drop in performance for test environments in which the agents' joint stiffness varies from the training environment. In contrast to the aforementioned results, the relative performance of the FS-DDPG algorithm is slightly lower than the relative performance of the non-spiking

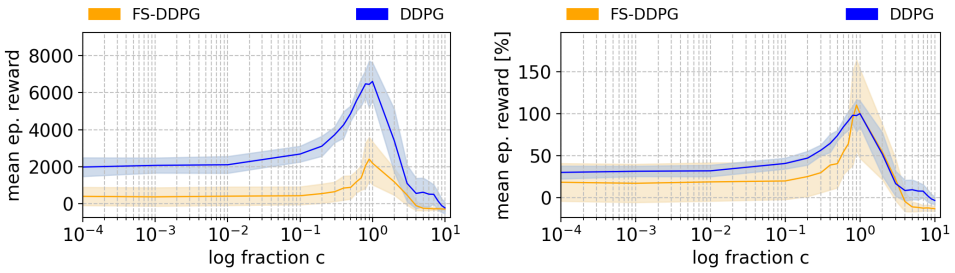


Figure 6.8: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on test environments with different joint stiffness matrices. Shaded regions represent standard deviations of the average return over the different random seeds.

DDPG algorithm. However, both algorithms seem to respond similarly when affected by changes in the joint stiffness matrix.

In summary, we have shown that both the FS-DDPG and non-spiking DDPG algorithm exhibit a drop in performance when exposed to sensor noise or perturbations of the environment. This aligns with our initial hypothesis H1 defined in section 1.1. However, results presented in section 6.2.1 indicate that the FS-DDPG algorithm is more robust to sensor noise and measurement errors than the non-spiking DDPG algorithm. In particular, we have seen that while the DDPG algorithm is highly sensible to state observations perturbed by AWGN or impulse noise, the FS-DDPG algorithm is only slightly affected by respective distortions. In section 6.2.2, we have presented the algorithms' response to perturbations of the environment by varying the agents' torso mass and joint stiffness. In particular, both algorithms demonstrate a similar decrease in performance when evaluated for varying test conditions. In this regard, hypothesis H2 does only partly align with the results presented in this chapter. While the FS-DDPG algorithm appears to be more robust to sensor noise and measurement errors than the non-spiking DDPG algorithm, both algorithms seem to respond similarly to perturbations of the environment. In the next chapter, we discuss the results presented in this chapter in a broader context.

Chapter 7

Discussion

In this chapter, we elaborate on the results presented in chapter 6 and discuss key findings in more detail. We begin by elaborating on the performance of the FS-DDPG and non-spiking DDPG algorithm on the unperturbed control tasks presented in section 6.1. In particular, we discuss difficulties in training spiking neural networks and relate them to the results presented. Subsequently, we look at key findings of the robustness evaluation presented in section 6.2. We relate the results to previous works and discuss possible reasons for the superior robustness of the FS-DDPG algorithm towards sensor noise and measurement errors. Finally, we point out limitations of this degree project and discuss possible directions for future studies.

7.1 Training Performance

Spiking neural networks bridge the gap between neuroscience and machine learning by employing biological neuron models as their computational units [84]. Unlike conventional ANNs, SNNs communicate via sparse, event-driven sequences of electrical impulses [112]. Due to their favorable properties on neuromorphic hardware such as high energy-efficiency and low latency, spiking neural networks are considered promising candidates for power-efficient computing [98, 136, 153].

In this thesis, we have introduced the FS-DDPG algorithm, a spiking actor-critic network intended to solve continuous control tasks characterized by high-dimensional state and action spaces. In section 6.1, we have seen that the FS-DDPG algorithm successfully learns control policies for different locomotion problems, each involving the control of complex multi-joint dynamics.

However, when comparing the algorithm's performance with the performance of the DDPG algorithm [80], a non-spiking actor-critic network with comparable network architecture (see sections 2.1.4.1 and 4.1.3), we observe the non-spiking DDPG algorithm to outperform the FS-DDPG algorithm on all control problems. Such performance gaps between deep SNNs and conventional deep neural networks have been observed for various tasks and domains [97, 132, 138], and are yet considered a major drawback of deep spiking neural networks [112]. In the following, we want to discuss two main challenges associated with spiking neural networks that affect the networks' performance: (i) *the need for neural encoding and decoding schemes* to convert conventional data formats into spike trains and vice versa, and (ii) *suboptimal training algorithms* that can yet not compete with gradient-based backpropagation techniques applied within deep neural networks.

Lossy Coding Schemes

Due to the lack of large and expressive neuromorphic datasets, most spiking neural networks are still evaluated on conventional AI benchmarks [132]. While these benchmarks resemble data formats that can be directly processed by deep neural networks, some form of conversion into discrete spike trains is required before the data can be further used by SNNs. Although there exists a variety of methods to encode input data into discrete spike patterns (see section 2.2.2), such conversion schemes almost always entail some loss of information [112]. This puts SNNs at disadvantage.

In this thesis, we treat continuous input values as direct currents to LIF neurons. Assuming constant stimulus over T_{sim} time steps, continuous data is encoded into spike trains with a spike rate proportional to its input values. Although we scale input values and choose parameters governing the LIF dynamics such that the encoded data demonstrates diverse spike patterns (see section 4.2.1), this encoding scheme results in a significant loss of information. In particular, due to the substantial amount of training time needed for rate-based SNNs on conventional hardware, the number of simulation time steps T_{sim} for which training can be achieved within reasonable duration is limited. Encoding continuous signals that cover a wide range of values into short spike trains of around 30 to 40 time steps without losing a significant amount of information is extremely challenging. Moreover, as mentioned in section 4.2.1, classical reinforcement learning problems do not adhere to a fixed dataset [40]. Instead, the distribution of data observed by the agent strongly depends on the agent's behavior and might change over the course of the agent-

environment interaction. For instance, minor differences in input values might be important during the first few time steps, while they can be neglected during later time steps (or vice versa). An efficient rate-based encoding scheme that captures such a variety of scales, given the limited number of simulation time steps, has to the best of our knowledge yet not been introduced.

To circumvent these problems, alternative coding schemes could be applied (see section 2.2.2). One approach would be to rely on spike coding schemes. While rate-based SNNs have successfully been applied in practice [36], they neglect important temporal information about the timing of individual spikes. Behavioral studies indicate that the reaction times of humans and other animals are too short to collect the statistics required by rate coding schemes [17, 139]. As mentioned in section 2.2.2, spike coding schemes make efficient use of spatiotemporal information within spike patterns by encoding information in the exact timing of spikes. Furthermore, they promote sparse and fast information processing [97]. However, most SNNs based on spike coding schemes have yet not been able to match the accuracy of ANNs or rate-based SNNs [112]. In particular, designing learning algorithms for SNNs that convey information through the timing of individual spikes has shown to be a challenging task [132]. This leads us to the second major challenge associated with spiking neural networks.

Suboptimal Training Algorithms

While conventional deep neural networks have shown great success in using gradient-based backpropagation for training [74], there is yet a unifying learning algorithm missing that overcomes the challenges associated with training deep spiking neural networks. One major difficulty relates to the neurons' non-differentiable transmission of action potentials [99]. In section 2.2.3, we have presented a series of learning algorithms that address this issue. In this thesis, we train SNNs via SuperSpike [155], a biological plausible error backpropagation scheme based on surrogate gradients. As described in section 2.2.3, SuperSpike estimates the gradient of the output error with respect to the network's weights by approximating the partial derivative of hidden neurons as the product of a filtered version of the presynaptic spike train and a surrogate function, i.e. a differentiable nonlinear function of the postsynaptic membrane potential that models the neuron's firing behavior [155]. While this estimation allows for training spiking neural networks with multiple hidden layers of deterministic LIF neurons, the approximations introduced result in a discrepancy between the true error gradient and the gradient obtained. This

discrepancy can have a considerable impact on the performance of deep SNNs. Furthermore, SuperSpike has been observed to scale poorly for large multilayer networks [99]. Other approaches such as local learning rules can overcome these issues by introducing local errors that operate on the immediate output of each layer [66]. While such learning rules would allow fast and efficient ways of training that are compatible with neuromorphic hardware, local learning schemes generally exhibit major difficulties in training lower layers of the network [112]. In this regard, they rarely reach the accuracy of deep neural networks trained with gradient-based backpropagation.

As of now, no unifying learning algorithm for spiking neural networks exists that exploits the spatiotemporal properties of spiking neurons and overcomes the current challenges related to training SNNs [112]. However, research on efficient local learning rules and training methods that consider the temporal information within spike patterns has gained increasing attention [132]. Overall, finding a successful learning rule that is compatible with an efficient coding scheme would not only help to close the performance gap between SNNs and conventional deep neural networks, it could also lead to new discoveries about how the brain learns.

7.2 Spiking Robustness

This thesis sat out to investigate the robustness properties of a spiking actor-critic network towards sensor noise and perturbations of the environment. For this, we have introduced the FS-DDPG algorithm, a spiking actor-critic network for continuous control capable of processing high-dimensional state observations. Furthermore, we have presented the DDPG algorithm [80], an existing non-spiking actor-critic network with comparable network architecture. We have trained both algorithms on different control tasks defined within the MuJoCo simulator [140] and designed a series of experiments to evaluate the networks' robustness (see section 5.3). In particular, we have analyzed the algorithms' robustness towards sensor noise by introducing additive white Gaussian noise and a version of impulse noise to the agents' state observations. Furthermore, we have varied parameters such as the agents' torso mass or joint stiffness to study the algorithms' response to perturbations of the environment. In section 6.2, we have presented the results of these experiments. While both the FS-DDPG and non-spiking DDPG algorithm seem to respond similarly to perturbations of the environment, results indicate that the FS-DDPG algorithm exhibits a superior robustness towards sensor noise and measurement errors compared to the non-spiking

actor-critic network. In particular, we have seen that while the DDPG algorithm is highly sensitive to state observations perturbed by AWGN and impulse noise, the FS-DDPG algorithm is only slightly affected by respective distortions. In the following, we elaborate on these findings and discuss possible reasons for this behavior.

Robustness towards AWGN

To study the robustness of the FS-DDPG algorithm towards sensor noise and measurement errors, we have perturbed the agent's perception of the environment by additive white Gaussian noise. In particular, we have added noise sampled from a zero-mean normal distribution $\mathcal{N}(0, \sigma^2)$ to the agent's state observations S_t (see equation 5.3). During the encoding process, the resulting array of noisy state observation $\tilde{S}_t$ is treated as constant current to a layer of LIF neurons (see section 4.2). The noisiness of this current can be modeled by splitting the input into a deterministic and a stochastic part:

$$I(t) = I^{det}(t) + I^{noise}(t) \tag{7.1}$$

where the stochastic term $I^{noise}(t)$ represents the impact of additive white Gaussian noise. Eventually, this leads to the noisy input model described in section 2.2.1. In particular, as we add white noise to the agent's state observations, the dynamics of the LIF neurons affected by AWGN can be described by equation 2.30 with $\xi(t) \equiv Z_t \sim \mathcal{N}(0, \sigma^2)$. Following a similar analysis on noisy input neurons as in [44], we can derive the membrane voltage of LIF neurons affected by AWGN. Let us denote t_0 as the moment when the membrane potential starts to rise from its resting potential $v(t_0) = v_{rest} = 0$. The solution to the membrane potential affected by AWGN for $t > t_0$ is:

$$v(t) = \frac{R}{\tau_m} \int_0^{t-t_0} \exp^{-\frac{s}{\tau_m}} \; I^{det}(t-s)\, ds + \frac{R}{\tau_m} \int_0^{t-t_0} \exp^{-\frac{s}{\tau_m}} \; Z_t \, ds \tag{7.2}$$

with $Z_t \sim \mathcal{N}(0, \sigma^2)$. For constant input current $I^{det}(t) \equiv I_0$, we obtain:

$$\begin{aligned} v(t) &= \frac{R}{\tau_m} (I_0 + Z_t) \int_0^{t-t_0} \exp^{-\frac{s}{\tau_m}} \; ds \\ v(t) &= R\,(I_0 + Z_t) \left[1 - \exp^{-\frac{(t-t_0)}{\tau_m}}\right] \end{aligned} \tag{7.3}$$

Hence, as long as the noisy membrane voltage stays far below the voltage threshold v_{th}, its distribution is Gaussian with mean:

$$\bar{v}(t) = R\, I_0 \left[1 - \exp^{-\frac{(t-t_0)}{\tau_m}}\right] \tag{7.4}$$

Note that the expected membrane potential of a LIF neuron with noisy input is that of a noiseless LIF neuron. Furthermore, its variance can be computed as:

$$\begin{aligned}
\text{var}(v(t)) &= \mathbb{E}\left[(v(t) - \bar{v}(t))^2\right] = \mathbb{E}\left[\left(R\, Z_t \left[1 - \exp^{-\frac{(t-t_0)}{\tau_m}}\right]\right)^2\right] \\
\text{var}(v(t)) &= \mathbb{E}\left[(Z_t - 0)^2\right]\; R^2 \left[1 - \exp^{-\frac{(t-t_0)}{\tau_m}}\right]^2 \\
\text{var}(v(t)) &= \sigma^2\, R^2 \left[1 - \exp^{-\frac{(t-t_0)}{\tau_m}}\right]^2
\end{aligned} \tag{7.5}$$

where σ denotes the amplitude of the noise. Assuming that no spike will occur, the noise causes the membrane voltage to drift away from the noiseless case. In the limit, the distance between the noisy membrane voltage and an unaffected membrane potential approaches:

$$\sqrt{\text{var}(v_\infty)} = R\,\sigma \tag{7.6}$$

As the membrane potential approaches the threshold v_{th}, a spike occurs for $v(t_s) > v_{th}$. Due to the stochastic nature of the noisy membrane voltage, the exact timing of the spike t_s becomes non-deterministic. Gerstner et al. [44] show that for small noise amplitudes $0 < \sigma \ll R\, I^{det} - v_{th}$ and superthreshold stimulus I^{det}, i.e. a deterministic current $R\, I^{det} > v_{th}$ that would excite the LIF neuron to fire regularly even without noise impact, the timing of the spike t_s evolves around a Gaussian distribution centered around the deterministic spike time t_s^{det} (see figure 7.1). In particular, the distribution of spike times within which a spike occurs can be described by:

$$P_0(t_s|0) = \frac{1}{\sqrt{2\,\pi}}\, \frac{\bar{v}'}{R\,\sigma}\, \exp\left[-\frac{1}{2}\, \frac{(\bar{v}')^2\, (t_s - t_s^{det})^2}{R^2\, \sigma^2}\right] \tag{7.7}$$

where $\bar{v}' = \frac{\partial \bar{v}}{\partial t}|_{t=t_s^{det}}$ is the slope of the mean membrane potential evaluated at the deterministic spike time t_s^{det}. Note that the width of the distribution $P_0(t_s)$ is proportional to the noise magnitude σ. Hence, as the noise level increases, the output spike pattern starts to deviate from the noiseless spike output. However, Gerstner et al. [44] argue that for superthreshold stimulation, white noise only slightly affects the behavior of LIF neurons. While it might broaden the interspike interval distribution, i.e. the distribution

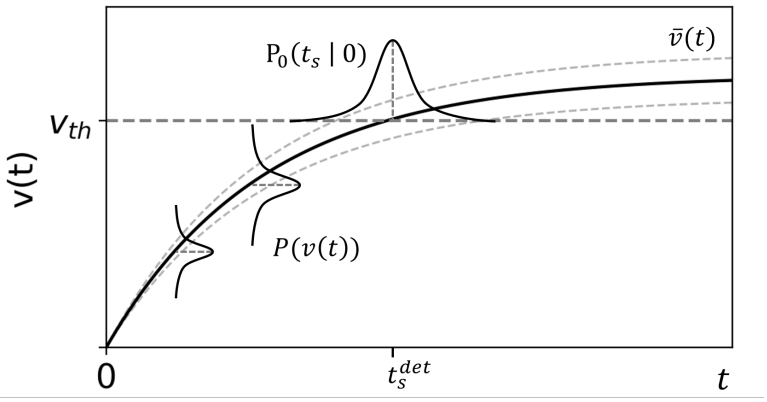


Figure 7.1: Membrane voltage $v(t)$ depicted over time of a LIF neuron affected by noisy input current. Within the subthreshold regime, the membrane voltage evolves around a Gaussian distribution centered at the membrane potential of a respective noiseless LIF neuron. Due to the stochastic nature of the noisy membrane voltage, the time of the membrane voltage crossing the threshold level becomes non-deterministic. In particular, the spike time t_s can be described by a Gaussian distribution centered around the deterministic spike time t_s^{det} [44].

of time durations between spikes, it has only little influence on the resulting spike train and its respective firing rate.

In contrast to the FS-DDPG algorithm, the non-spiking DDPG algorithm lacks temporal dynamic behavior. During inference, the non-spiking DDPG algorithm is presented the array of noisy state observations $\tilde{S}_t$ once, effectively offsetting the array of true state observations S_t by random values. Respectively, a high noise value might cause the output of the network's linear layer to change dramatically. This can cause the inference accuracy of the ANN to drop considerably. Spiking neurons evolve over time. Instead of passing the processed input immediately to subsequent layers, spiking neural networks are simulated for T_{sim} time steps. By integrating input over time, spiking neurons smooth the information contained in the noisy state observations $\tilde{S}_t$. Hence, noise does not propagate through SNNs as easily as through deep neural networks.

This aligns with the results presented in section 6.2.1. While the performance of the non-spiking DDPG algorithm is highly affected by AWGN, the FS-DDPG algorithm seems almost agnostic to the perturbations introduced. Only for noise with very high magnitudes σ, respective spike patterns seem to deviate notably and the performance of the FS-DDPG algorithm decreases.

Robustness towards Impulse Noise

Besides the impact of additive white Gaussian noise on the algorithms' performance, we have studied the algorithms' response to sharp and sudden disturbances in their input signals. For this, we have introduced a version of impulse noise that replaces state observations S_t^i by erroneous measurements of high magnitude (see equation 5.4). The results presented in section 6.2.1 show that the non-spiking DDPG algorithm is highly sensitive to impulse noise. In particular, as the noise magnitude and disturbance probability increases, the algorithm highly degrades, exhibiting low average return values for all control tasks. In contrast, the FS-DDPG algorithm shows a much more robust behavior towards impulse noise. In particular, the algorithm deteriorates only slightly in the presence of impulse noise. In addition, results show that the performance of the FS-DDPG algorithm remains constant as the magnitude of the noise exceeds a certain value. This behavior can be explained by looking at the way the FS-DDPG algorithm encodes continuous state observations S_t into discrete spike patterns $\hat{S}_t^{T_{sim}}$. As described in section 4.2, the FS-DDPG algorithm rescales continuous input values to the interval $[a, b]$ to avoid silent or overly excited neurons (see equation 4.11). For this, it defines minimum and maximum state observations S_{min}^E, S_{max}^E for each environment E. During the encoding process, state observations S_t that exceed these boundaries are clipped respectively (see equation 4.13). In this regard, the encoding scheme introduces an upper bound to noise values. This promotes the algorithm's robustness towards impulse noise as it limits the noise level to S_{min}^E or S_{max}^E, respectively. In contrast, the DDPG algorithm is affected by unbounded noise. While this allows the DDPG algorithm to react to extreme values that it has not encountered so far, it might also lead to the drastic decrease in performance observed.

The superior robustness of the FS-DDPG algorithm towards sensor noise and measurement errors aligns with findings of previous works. Patel et al. [109] show that a deep Q-network converted to spiking domain is less susceptible to occlusion attacks than a respective non-spiking DQN. Li et al. [77] demonstrate that SNNs are more robust to Gaussian noise in synaptic weights than comparable ANNs.

Research indicates that noise plays an important part in biological neural networks [26]. For instance, cortical neurons show noisy behavior when exposed to slowly changing external stimulus [25]. Whether the presence of noise is a result of thermal noise, microscopic chaos, or the essence of an

efficient way of coding is still unclear [44]. Regardless, the fact that biological neurons efficiently operate in the presence of noise suggests that they must be inherently robust to random fluctuations. The results presented in this thesis demonstrate that similar to their biological counterpart, spiking neural networks exhibit a certain degree of robustness towards sensor noise and measurement errors. Besides their high energy-efficiency and fast inference on neuromorphic hardware, this could be a substantial advantage of spiking neural networks compared to conventional ANNs.

In the following, we discuss limitations of our work and suggest possible directions for future studies.

7.3 Limitations and Future Work

To study the robustness of the FS-DDPG algorithm towards sensor noise and perturbations of the environment, we have conducted a series of experiments that evaluate the algorithm's response to noisy state observations and varying test conditions. We compared the algorithm's performance to the DDPG algorithm [80], a non-spiking actor-critic network with comparable network architecture, and showed that the FS-DDPG exhibits superior robustness towards sensor noise and measurement errors. In section 7.2, we have discussed possible reasons for this behavior and pointed out that the temporal dynamics within spiking neural networks might play an important role in the network's robustness.

While temporal information processing is inherent to SNNs, the DDPG algorithm lacks temporal dynamic behavior. Despite their architectural similarities, this strongly distinguishes the two algorithms and limits their comparability. To further investigate the impact of temporal dynamics on the networks' robustness and to promote comparability, we suggest future research to compare the FS-DDPG algorithm to a non-spiking actor-critic that exhibits temporal dynamic behavior. For instance, Neftci et al. [99] highlight the structural equivalence of SNNs and recurrent neural networks (RNNs). In this regard, we suggest to incorporate temporal dynamics by considering non-spiking actor-critic networks based on RNNs.

A major limitation of the spiking actor-critic network introduced in this thesis is the high computational cost when trained on conventional hardware. The amount of time needed for training the FS-DDPG algorithm limits the number of random seeds and possible experiments conducted in this degree project. To reduce training time and therefore make the application of SNNs more practical, further advances within on-chip learning are required that

not only promote energy-efficient and fast inference, but also training. For this, compatible learning rules have to be developed that can compete with training algorithms applied within conventional deep neural networks. Once developed, we suggest future research to exploit these methods and train the FS-DDPG algorithm for an increasing number of random seeds.

As described in section 7.1, the performance of the FS-DDPG algorithm is limited by its lossy coding scheme and suboptimal training algorithm. For future studies, it might be interesting to look at different coding schemes and learning algorithms that better exploit the spatiotemporal properties of SNNs. In particular, we suggest to look at schemes that utilize information encoded in the timing of spikes as these methods additionally reduce firing rates and promote energy-efficiency on neuromorphic hardware [112].

To circumvent the problem of lossy data encoding and to further close the performance gap between SNNs and conventional ANNs, we suggest to perform a similar analysis as done in this thesis on neuromorphic data. While there exists only few neuromorphic datasets within reinforcement learning yet, the advantages of such data formats are plentiful. Not only would such benchmarks alleviate the need for lossy encoding schemes, they would also better capture temporal dynamics of the real world compared to frame-based approaches and other conventional data formats [112].

Chapter 8

Conclusion

This thesis sat out to study the robustness of a spiking actor-critic network towards sensor noise and perturbations of the environment. For this, we introduced the *fully spiking deep deterministic policy gradient (FS-DDPG)* algorithm, a spiking actor-critic network for continuous control that can handle high-dimensional state and action spaces. We successfully trained the FS-DDPG algorithm on different locomotion problems, each involving the control of complex multi-joint dynamics. Furthermore, we conducted a series of experiment to analyze the algorithm's robustness. In particular, we evaluated the algorithm's response to sensor noise and measurement errors by gradually introducing noise to the agent's state observations. In addition, we varied parameters such as the agent's torso mass or joint stiffness to analyze its robustness towards perturbations of the environment. We further compared the FS-DDPG algorithm to the DDPG method [80], a non-spiking actor-critic network with comparable network architecture. The experimental results show that both algorithms degrade under the influence of sensor noise and perturbations of the environment. However, we could observe the FS-DDPG algorithm to be significantly more robust to sensor noise and measurement errors than the non-spiking DDPG algorithm. In particular, while the DDPG algorithm exhibits a considerable drop in performance when exposed to state observations perturbed by AWGN or impulse noise, the FS-DDPG algorithm is only slightly affected by respective distortions. In contrast, both algorithms seem to respond similarly to perturbations of the environment.

These results support our initial hypothesis H1 (see section 1.1), stating that both the FS-DDPG and non-spiking DDPG algorithm deteriorate in the presence of sensor noise and perturbations of the environment. Furthermore, we partly affirm hypothesis H2. While we find the FS-DDPG algorithm to

be more robust to sensor noise and measurement errors than the non-spiking DDPG algorithm, we can not conclude similar robustness benefits towards perturbations of the environment.

We discussed possible reasons for the behavior observed and identified three important aspects that contribute to the superior robustness of the FS-DDPG algorithm towards sensor noise and measurement errors. First, AWGN causes the exact timing of spikes to evolve around a Gaussian distribution that is centered at the deterministic spike time and has a width proportional to the magnitude of the noise. While the neuron's spike pattern thus starts to deviate from the unperturbed output as the noise level increases, the respective firing rate remains almost constant for moderate noise magnitudes. Secondly, the non-spiking DDPG algorithm lacks temporal dynamics. Hence, noise added to the agent's state observations is immediately processed and passed to subsequent layers. This might cause the network's output to change dramatically, which can result in a significant drop in accuracy. In contrast, spiking neural networks smooth noise contained in the agent's state observations by integrating input over time. Finally, the FS-DDPG's min-max current coding scheme introduces an upper bound to state observations. This limits noise introduced to the agent's state observations and thus promotes the algorithm's robustness towards disturbances of high magnitude.

While the FS-DDPG algorithm can yet not match the performance of the DDPG algorithm on the unperturbed control tasks, its superior robustness towards sensor noise and measurement errors is an important advantage for a successful deployment in complex real-world scenarios. To further close the performance gap between the FS-DDPG algorithm and other non-spiking actor-critic networks, we suggest future research to look at different coding schemes that make efficient use of the spatiotemporal properties of spiking neural networks. Furthermore, we encourage subsequent work to perform a similar analysis as presented in this thesis on neuromorphic datasets. While there exists only few neuromorphic benchmarks within reinforcement learning yet, we can see a positive trend, especially in the collection of dynamic vision sensor (DVS) datasets [42].

Overall, deep spiking reinforcement learning is still at an early stage with only few spiking RL algorithms existing yet [28]. In this thesis, we show that deep spiking RL agents do not only promise low latency and high energy-efficiency when applied on neuromorphic hardware, they also demonstrate desirable robustness properties towards sensor noise and measurement errors. These properties become increasingly important as the demand for RL algorithms deployed in complex real-world scenarios grows.

Appendices

Appendix A

Robustness Evaluation

A.1 Sensor Noise and Measurement Errors

In this section, we report additional results of the experiments conducted to evaluate the algorithms' robustness towards sensor noise and measurement errors.

A.1.1 HalfCheetah Environment

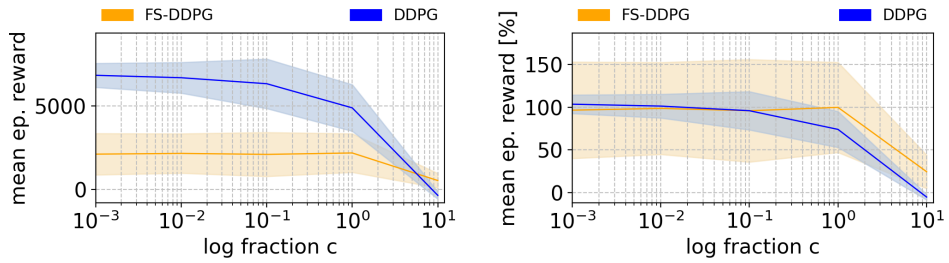


Figure A.1: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a halfcheetah test environment in which the agents' state observations of the *rotor's angular velocity at the agents' back thighs, shins and feed* are perturbed by AWGN.

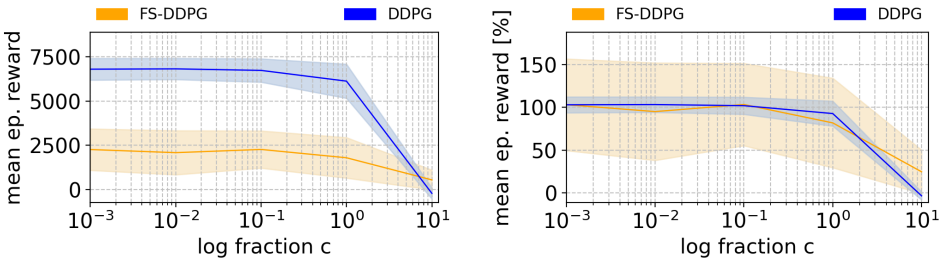


Figure A.2: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a halfcheetah test environment in which the agents' state observations regarding the *angle of the agents' back thighs and shins* are perturbed by AWGN.

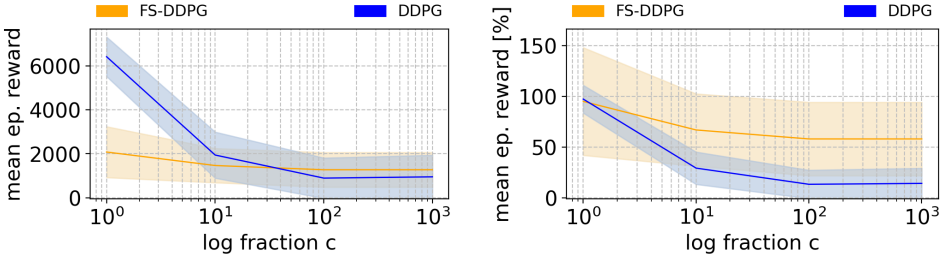


Figure A.3: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a halfcheetah test environment in which the agents' state observations regarding the *angular velocity of the agents' front thighs, shins and feet* are perturbed by impulse noise with a probability of $p = 0.2$.

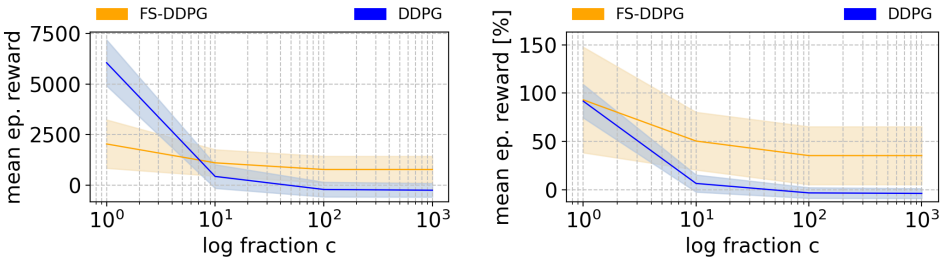


Figure A.4: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a halfcheetah test environment in which the agents' state observations regarding the *angular velocity of the agents' front thighs, shins and feet* are perturbed by impulse noise with a probability of $p = 0.4$.

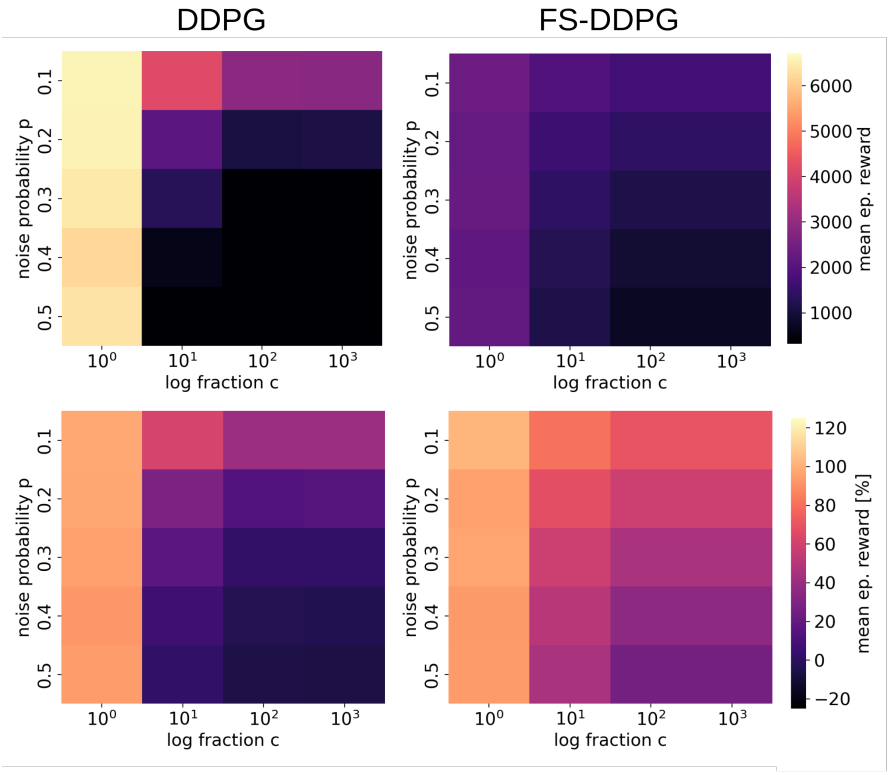


Figure A.5: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a halfcheetah test environment in which the agents' state observations regarding the *angular velocity of the agents' front thighs, shins and feet* are perturbed by impulse noise. The average return is reported for various values of the fraction c and disturbance probability p. Furthermore, the average return is illustrated relatively to the algorithms' performance on an undisturbed environment (see lower row). For a detailed overview of the exact values for each combination of c and p and respective standard deviations, see table A.3 and A.4

Table A.1: Average return of the DDPG algorithm evaluated for 15 episodes on a halfcheetah test environment in which the agents' state observations regarding the *angle of the agent's back thigh and shin* are perturbed by impulse noise. The average return is reported for various values of the fraction c and disturbance probability p. Standard deviations are computed over the different random seeds.

DDPG				
p	**c**			
	10^0	10^1	10^2	10^3
0.1	6666.50 ± 882.43	4481.72 ± 1671.96	3277.59 ± 1387.23	3069.17 ± 1350.03
0.2	6581.92 ± 1015.80	3050.90 ± 1716.85	1337.52 ± 1006.37	1019.57 ± 920.09
0.3	6313.22 ± 1366.93	2060.61 ± 1336.43	236.06 ± 644.02	233.50 ± 640.18
0.4	6575.01 ± 806.96	1103.41 ± 953.17	-167.21 ± 399.36	-152.22 ± 438.01
0.5	6476.40 ± 974.90	456.05 ± 745.47	-373.49 ± 301.84	-342.92 ± 336.21

Table A.2: Average return of the FS-DDPG algorithm evaluated for 15 episodes on a halfcheetah test environment in which the agents' state observations regarding the *angle of the agent's back thigh and shin* are perturbed by impulse noise. The average return is reported for various values of the fraction c and disturbance probability p. Standard deviations are computed over the different random seeds.

FS-DDPG				
p	**c**			
	10^0	10^1	10^2	10^3
0.1	2354.72 ± 1107.05	1890.15 ± 1143.07	1890.15 ± 1143.07	1890.15 ± 1143.07
0.2	1986.43 ± 1273.24	1695.23 ± 1035.47	1695.23 ± 1035.47	1695.23 ± 1035.47
0.3	2293.46 ± 1201.00	1246.13 ± 881.88	1246.13 ± 881.88	1246.13 ± 881.88
0.4	2149.59 ± 1194.61	1210.83 ± 756.49	1210.83 ± 756.49	1210.83 ± 756.49
0.5	2273.23 ± 1160.70	1073.28 ± 874.74	1073.28 ± 874.74	1073.28 ± 874.74

Table A.3: Average return of the DDPG algorithm evaluated for 15 episodes on a halfcheetah test environment in which the agents' state observations regarding the *angular velocity of the agents' front thighs, shins and feet* are perturbed by impulse noise. The average return is reported for various values of the fraction c and disturbance probability p. Standard deviations are computed over the different random seeds.

DDPG				
p	**c**			
	10^0	10^1	10^2	10^3
0.1	6445.16 ± 1057.41	4045.78 ± 1129.09	2706.63 ± 1206.44	2675.75 ± 1186.67
0.2	6429.17 ± 900.67	1944.18 ± 1050.69	892.80 ± 919.67	950.65 ± 988.12
0.3	6295.20 ± 1059.23	1134.07 ± 941.08	130.29 ± 578.27	156.44 ± 620.75
0.4	6051.98 ± 1137.87	430.23 ± 585.16	-214.84 ± 364.09	-251.55 ± 338.27
0.5	6214.64 ± 659.94	111.77 ± 398.30	-376.70 ± 220.76	-431.17 ± 198.41

Table A.4: Average return of the FS-DDPG algorithm evaluated for 15 episodes on a halfcheetah test environment in which the agents' state observations regarding the *angular velocity of the agents' front thighs, shins and feet* are perturbed by impulse noise. The average return is reported for various values of the fraction c and disturbance probability p. Standard deviations are computed over the different random seeds.

FS-DDPG				
p	**c**			
	10^0	10^1	10^2	10^3
0.1	2233.74 ± 1071.99	1766.19 ± 848.31	1541.83 ± 955.69	1541.83 ± 955.69
0.2	2081.72 ± 1160.15	1466.93 ± 779.90	1273.40 ± 791.24	1273.40 ± 791.24
0.3	2118.57 ± 1168.70	1271.42 ± 753.38	995.68 ± 665.48	995.68 ± 665.48
0.4	2035.49 ± 1196.52	1103.14 ± 653.30	773.19 ± 655.57	773.19 ± 655.57
0.5	2043.26 ± 1065.19	1003.10 ± 655.63	559.81 ± 654.39	559.81 ± 654.39

A.1.2 Hopper Environment

In the following, we present results of the robustness evaluation (see section 5.3) performed on the hopper environment.

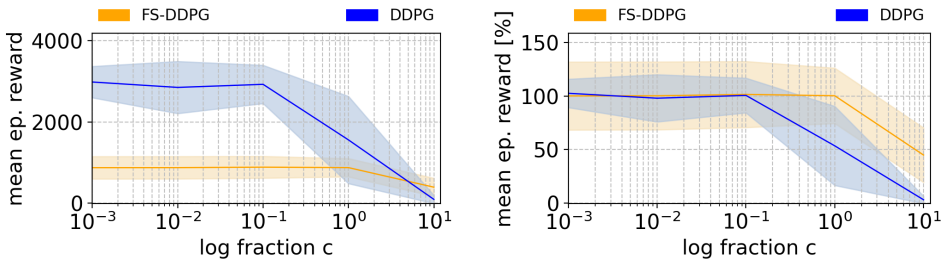


Figure A.6: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observations regarding the *angular velocity of the robot's top* is perturbed by AWGN.

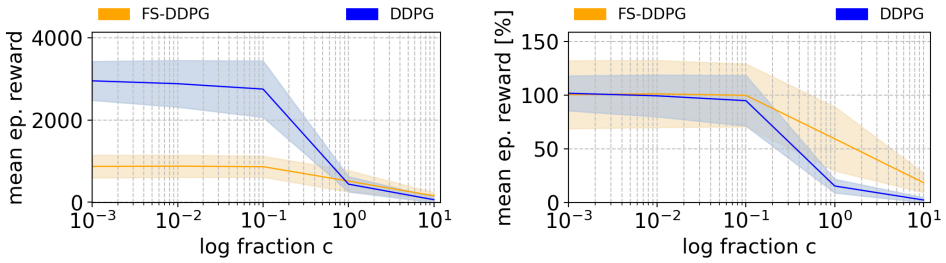


Figure A.7: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observations of the *robot's height and the angle of its top* are perturbed by AWGN.

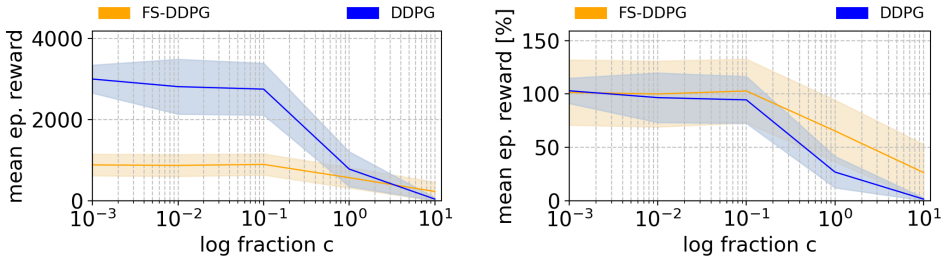


Figure A.8: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observations regarding the *angle of the robot's thigh joint, leg joint and foot joint* are perturbed by AWGN.

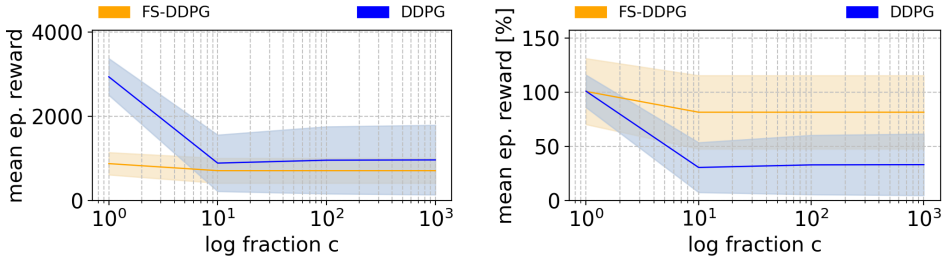


Figure A.9: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observation of the *robot's height* is perturbed by impulse noise with a probability of $p = 0.1$.

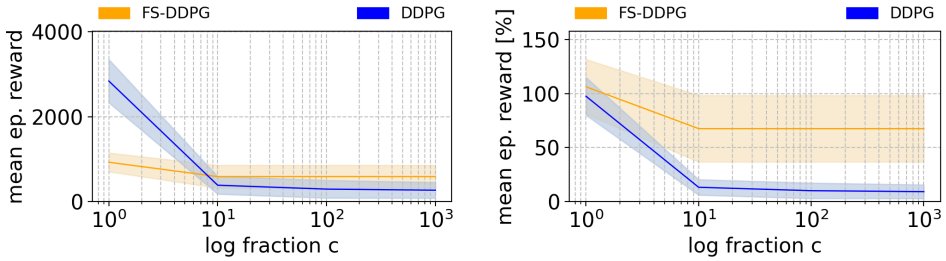


Figure A.10: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observation of the *robot's height* is perturbed by impulse noise with a probability of $p = 0.2$.

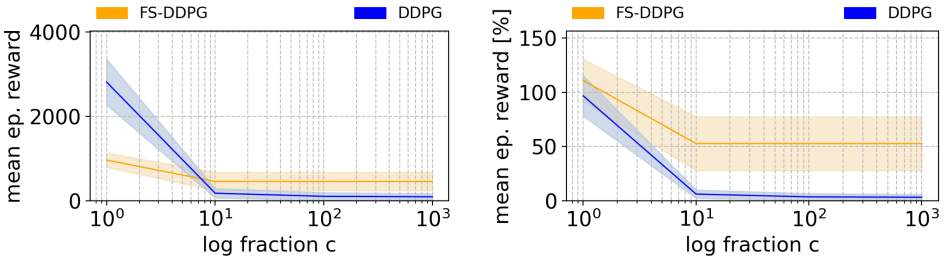


Figure A.11: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observation of the *robot's height* is perturbed by impulse noise with a probability of $p = 0.4$.

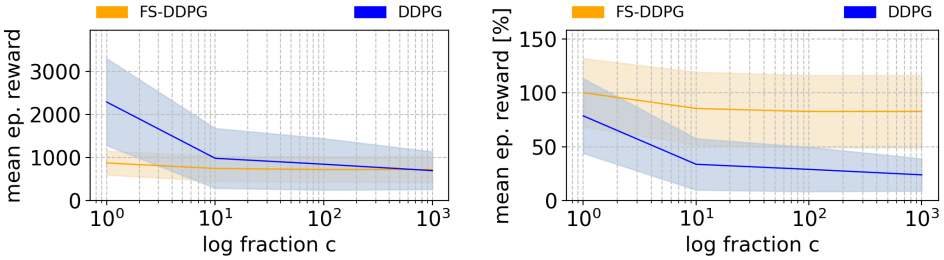


Figure A.12: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observation regarding the *angular velocity of the robot's top* is perturbed by impulse noise with a probability of $p = 0.2$.

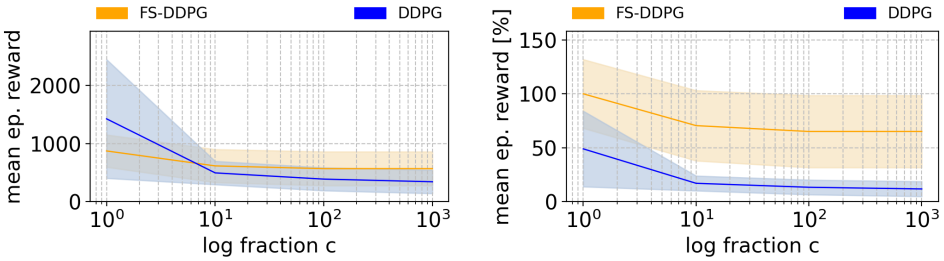


Figure A.13: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observation regarding the *angular velocity of the robot's top* is perturbed by impulse noise with a probability of $p = 0.4$.

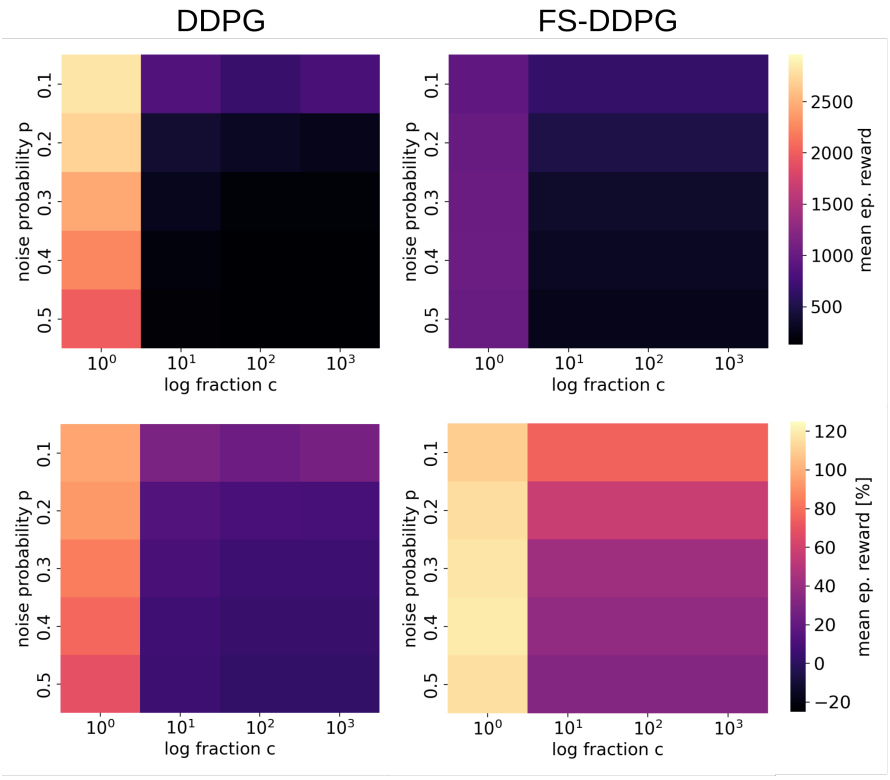


Figure A.14: Average return of the FS-DDPG and non-spiking DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observations of the *robot's height and the angle of its top* are perturbed by impulse noise. The average return is reported for various values of the fraction c and disturbance probability p. Furthermore, the average return is illustrated relatively to the algorithms' performance on an undisturbed environment (see lower row). For a detailed overview of the exact values for each combination of c and p and respective standard deviations, see table A.5 and A.6

Table A.5: Average return of the DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observations of the *robot's height and the angle of its top* are perturbed by impulse noise. The average return is reported for various values of the fraction c and disturbance probability p. Standard deviations are computed over the different random seeds.

DDPG				
p	**c**			
	10^0	10^1	10^2	10^3
0.1	2807.80 ± 660.64	840.92 ± 572.09	684.58 ± 429.31	785.34± 565.75
0.2	2696.77 ± 674.41	393.10 ± 215.31	312.02 ± 174.21	264.93 ± 133.27
0.3	2445.65 ± 804.39	278.86 ± 143.33	175.80 ± 114.43	175.11 ± 108.62
0.4	2246.69 ± 966.72	200.21 ± 131.94	117.76 ± 92.27	111.71 ± 85.01
0.5	1998.57 ± 1019.30	163.19 ± 105.23	73.27 ± 63.78	66.41 ± 58.93

Table A.6: Average return of the FS-DDPG algorithm evaluated for 15 episodes on a hopper test environment in which the agents' state observations of the *robot's height and the angle of its top* are perturbed by impulse noise. The average return is reported for various values of the fraction c and disturbance probability p. Standard deviations are computed over the different random seeds.

FS-DDPG				
p	**c**			
	10^0	10^1	10^2	10^3
0.1	952.33 ± 204.99	659.53 ± 298.66	659.53 ± 298.66	659.53 ± 298.66
0.2	1000.75 ± 123.84	492.92 ± 220.29	492.92 ± 220.29	492.92 ± 220.29
0.3	1019.65 ± 74.52	361.10 ± 57.58	361.10 ± 57.58	361.10 ± 57.58
0.4	1034.54 ± 18.36	319.45 ± 59.79	319.45 ± 59.79	319.45 ± 59.79
0.5	1005.16 ± 128.59	275.12 ± 57.48	275.12 ± 57.48	275.12 ± 57.48

Appendix B

Hyperparameters

In this chapter, we report hyperparameters of the FS-DDPG and non-spiking DDPG algorithm. As mentioned in section 5.2, hyperparameters are tuned based on Optuna [6]. For each control task, we present parameters regarding the networks' architecture, training process and optimization procedure. Furthermore, we report parameters of the FS-DDPG's neuron dynamics in table B.1 and B.2, which are similar for both control tasks.

Table B.1: Hyperparameters for the LIF neurons of the FS-DDPG algorithm as described in equations 2.39 and 2.40.

Parameter	FS-DDPG
Inverse Synaptic Time Constant $\frac{1}{\tau_s}$	200
Inverse Membrane Time Constant $\frac{1}{\tau_m}$	100
Threshold Voltage v_{th}	1.0
Resting Voltage value v_{rest}	0.0
Time Step Size Δt	0.001
Gradient Smoothness β	100

Table B.2: Hyperparameters for the LI neurons of the FS-DDPG algorithm.

Parameter	FS-DDPG
Inverse Synaptic Time Constant $\frac{1}{\tau_s}$	200
Inverse Membrane Time Constant $\frac{1}{\tau_m}$	100
Resting Voltage value v_{rest}	0.0
Time Step Size Δt	0.001

B.1 HalfCheetah Environment

This section gives an overview of the final hyperparameters for the halfcheetah environment. Table B.3 presents parameters regarding the networks' architecture, training process and optimization procedure.

Table B.3: Hyperparameters of the FS-DDPG and non-spiking DDPG algorithm for the halfcheetah environment. While the FS-DDPG algorithm uses action noise sampled from an Ornstein-Uhlenbeck process (OU), the non-spiking DDPG algorithm samples from a normal distribution $\mathcal{N}(0.0, 0.1)$ with zero mean and standard deviation of 0.1. Note that the DDPG's critic network receives both state observation and action as input to the first layer, while the spiking critic network concatenates the encoded state observation and temporal action content after the first LIF layer.

Parameter	**FS-DDPG**	**DDPG**
Actor Architecture	$[34, 256, 256, 6]$	$[17, 256, 256, 6]$
Critic Architecture	$[34, 512 + 6, 512, 1]$	$[17 + 6, 512, 512, 1]$
Batch Size	100	100
Discount Factor γ	0.99	0.99
Training Starts after N Steps	10^4	10^4
Optimizer	RMSProp	RMSProp
Learning Rate	10^{-3}	10^{-3}
Momentum μ	0.0	0.0
Smoothing Constant α	0.99	0.99
Numerical Stability Constant ϵ	10^{-8}	10^{-8}
Replay Memory Size	10^6	10^6
Target Update Rate τ	0.005	0.005
Exploration Policy	OU: $[\theta = 0.15, \mu = 0.0, \sigma = 0.2]$	$\mathcal{N}(0.0, 0.1)$
Simulation Time Steps T_{sim}	30	-

B.2 Hopper Environment

This section gives an overview of the final hyperparameters for the hopper environment. Table B.4 presents parameters regarding the networks' architecture, training process and optimization procedure.

Table B.4: Hyperparameters of the FS-DDPG and non-spiking DDPG algorithm for the hopper environment. While the FS-DDPG algorithm uses action noise sampled from an Ornstein-Uhlenbeck process (OU), the non-spiking DDPG algorithm samples from a normal distribution $\mathcal{N}(0.0, 0.1729)$ with zero mean and standard deviation of 0.1729. Note that the DDPG's critic network receives both state observation and action as input to the first layer, while the spiking critic network concatenates the encoded state observation and temporal action content after the first LIF layer.

Parameter	**FS-DDPG**	**DDPG**
Actor Architecture	$[22, 256, 256, 3]$	$[11, 256, 256, 3]$
Critic Architecture	$[22, 512 + 3, 512, 1]$	$[11 + 3, 512, 512, 1]$
Batch Size	100	64
Discount Factor γ	0.995	0.995
Training Starts after N Steps	10^3	10^3
Optimizer	RMSProp	RMSProp
Learning Rate	$4.6 \cdot 10^{-4}$	$4.6 \cdot 10^{-4}$
Momentum μ	0.0	0.0
Smoothing Constant α	0.99	0.99
Numerical Stability Constant ϵ	10^{-8}	10^{-8}
Replay Memory Size	$2 \cdot 10^6$	$2 \cdot 10^6$
Target Update Rate τ	0.008	0.008
Exploration Policy	OU: $[\theta = 0.15, \mu = 0.0, \sigma = 0.2]$	$\mathcal{N}(0.0, 0.1729)$
Simulation Time Steps T_{sim}	30	-

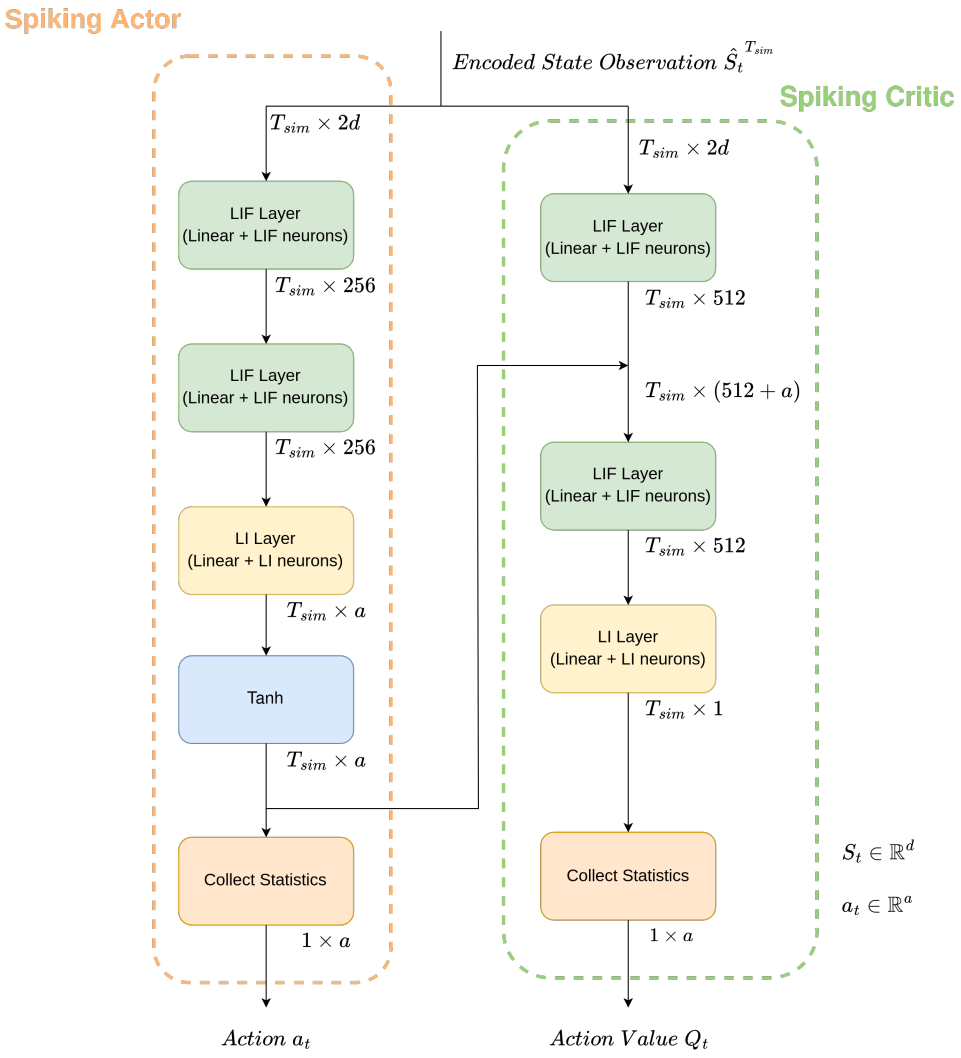


Figure B.1: Schematic overview of the FS-DDPG's final network architecture. The dimension of the input are reported for each layer. The encoded state observation $\hat{S}_t^{T_{sim}}$ of length T_{sim} is obtained from the min-max current coding scheme introduced in section 4.2. Finally, the action a_t and action-value Q_t are computed by collecting temporal statistics over the networks' output.

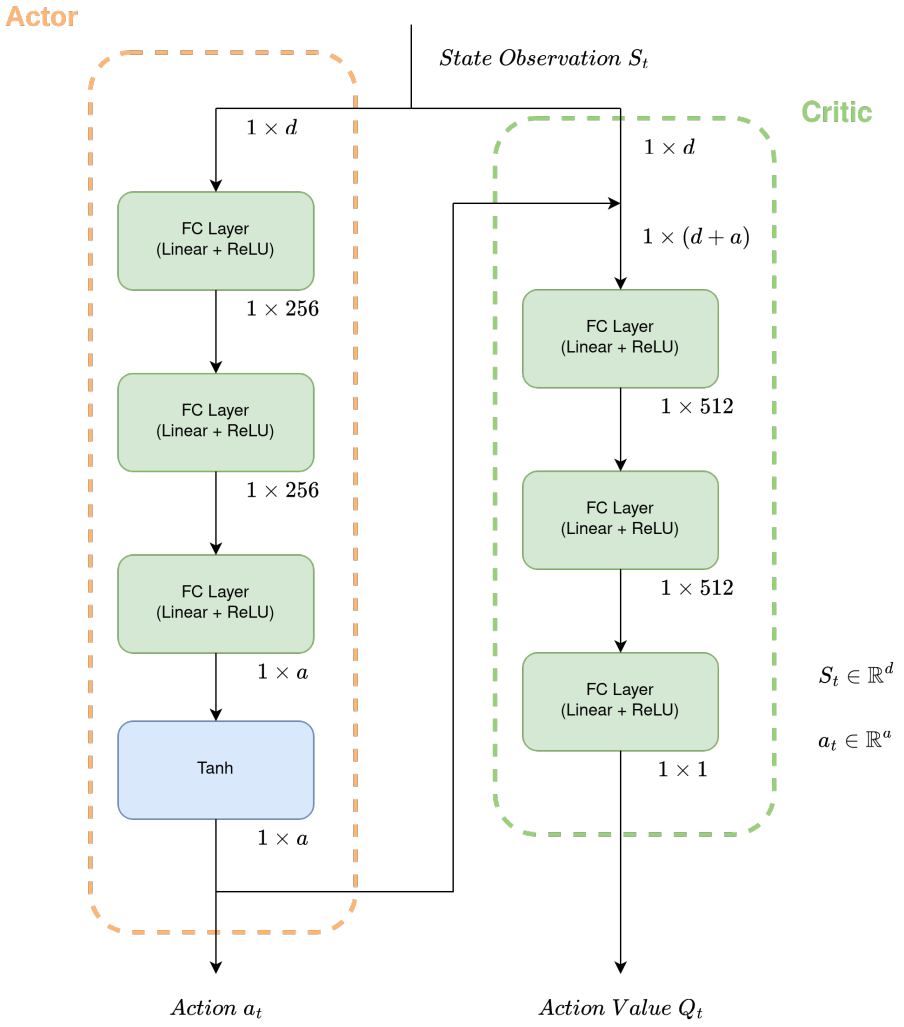


Figure B.2: Schematic overview of the DDPG's final network architecture. The architecture aligns with the implementation of the DDPG algorithm provided by *Stable-Baselines3* [115]. In particular, the critic receives both state observation $S_t \in \mathbb{R}^d$ and action $a_t \in \mathbb{R}^a$ as input to the first layer.

References

[1] L.F Abbott. “Lapicque’s introduction of the integrate-and-fire model neuron (1907).” In: *Brain Research Bulletin* 50.5 (1999), pp. 303–304. ISSN: 0361-9230. DOI: https://doi.org/10.1016/S0361-9230(99)00161-6.

[2] M. Abeles. “Firing Rates and Weil-Timed Events in the Cerebral Cortex.” In: *Models of Neural Networks: Temporal Aspects of Coding and Information Processing in Biological Systems*. Ed. by Eytan Domany, J. Leo van Hemmen, and Klaus Schulten. New York, NY: Springer New York, 1994, pp. 121–140. ISBN: 978-1-4612-4320-5. DOI: 10.1007/978-1-4612-4320-5_3.

[3] Karen E. Adolph, Bennett I. Bertenthal, Steven M. Boker, Eugene C. Goldfield, and Eleanor J. Gibson. “Learning in the Development of Infant Locomotion.” In: *Monographs of the Society for Research in Child Development* 62.3 (1997), pp. i–162. ISSN: 0037976X, 15405834. URL: http://www.jstor.org/stable/1166199 (visited on 06/23/2022).

[4] E. D. Adrian. “The impulses produced by sensory nerve endings.” In: *The Journal of Physiology* 61.1 (1926), pp. 49–72. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1926.sp002273. DOI: https://doi.org/10.1113/jphysiol.1926.sp002273.

[5] E. D. Adrian and Yngve Zotterman. “The impulses produced by sensory nerve-endings.” In: *The Journal of Physiology* 61.2 (1926), pp. 151–171. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1926.sp002281. DOI: https://doi.org/10.1113/jphysiol.1926.sp002281.

[6] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. “Optuna: A Next-generation Hyperparameter Optimization Framework.” In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[7] J. Andrew Bagnell, Andrew Y. Ng, and Jeff G. Schneider. *Solving Uncertain Markov Decision Processes*. Tech. rep. Carnegie Mellon University, 2001.

[8] John R. Barry, Edward A. Lee, and David G. Messerschmitt. *Digital Communication*. Springer New York, 2004. DOI: 10.1007/978-1-4615-0227-2.

[9] Tamer Başar and Pierre Bernhard. *H∞-Optimal Control and Related Minimax Design Problems. A Dynamic Game Approach*. New York: Birkhäuser Boston, MA, 2008. ISBN: 978-0-8176-4756-8. DOI: https://doi.org/10.1007/978-0-8176-4757-5.

[10] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. "Long short-term memory and Learning-to-learn in networks of spiking neurons." In: *CoRR* abs/1803.09574 (2018). URL: http://arxiv.org/abs/1803.09574. arXiv: 1803.09574.

[11] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. "A solution to the learning dilemma for recurrent networks of spiking neurons." en. In: *Nature Communications* 11.1 (Dec. 2020), p. 3625. ISSN: 2041-1723. DOI: 10.1038/s41467-020-17236-y.

[12] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. "The Arcade Learning Environment: An Evaluation Platform for General Agents." In: *CoRR* abs/1207.4708 (2012). URL: http://arxiv.org/abs/1207.4708. arXiv: 1207.4708.

[13] Richard Bellman. "A Markovian Decision Process." In: *Indiana Univ. Math. J.* 6 (4 1957), pp. 679–684. ISSN: 0022-2518. DOI: 10.1512/IUMJ.1957.6.56038.

[14] Richard Ernest Bellman. *The Theory of Dynamic Programming*. en. Tech. rep. RAND Corporation, Jan. 1954. URL: https://www.rand.org/pubs/papers/P550.html (visited on 05/02/2022).

[15] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. "Dota 2 with Large Scale Deep Reinforcement Learning." In: *CoRR* abs/1912.06680 (2019). URL: http://arxiv.org/abs/1912.06680. arXiv: 1912.06680.

[16] Guo-qiang Bi and Mu-ming Poo. “Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type.” In: *Journal of Neuroscience* 18.24 (1998), pp. 10464–10472. ISSN: 0270-6474. eprint: https://www.jneurosci.org/content/18/24/10464.full.pdf. DOI: 10.1523/JNEUROSCI.18-24-10464.1998.

[17] William Bialek, Rob de Ruyter van Steveninck, Fred Rieke, and David Warland. *Spikes - exploring the neural code*. MIT Press, 1996. DOI: 10.1017/CBO9781107447615.

[18] Enrico Bibbona, Gianna Panfilo, and Patrizia Tavella. “The Ornstein–Uhlenbeck process as a model of a low pass filtered white noise.” In: *Metrologia* 45 (Dec. 2008), S117. DOI: 10.1088/0026-1394/45/6/S17.

[19] Sander Bohte, Joost Kok, and Han Poutré. “Error-Backpropagation in Temporally Encoded Networks of Spiking Neurons.” In: *Neurocomputing* 48 (Feb. 2001), pp. 17–37. DOI: 10.1016/S0925-2312(01)00658-0.

[20] Olaf Booij and Hieu Nguyen. “A gradient descent rule for spiking neurons emitting multiple spikes.” In: *Information Processing Letters* 95 (Sept. 2005), pp. 552–558. DOI: 10.1016/j.ipl.2005.05.023.

[21] Matthew Botvinick, Sam Ritter, Jane X. Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. “Reinforcement Learning, Fast and Slow.” In: *Trends in Cognitive Sciences* 23.5 (2019), pp. 408–422. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2019.02.006.

[22] Steven J. Bradtke. “Reinforcement Learning Applied to Linear Quadratic Regulation.” In: *Advances in Neural Information Processing Systems 5, [NIPS Conference]*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 295–302. ISBN: 1558602747.

[23] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[24] Nicolas Brunel and Mark van Rossum. “Quantitative investigations of electrical nerve excitation treated as polarization: Louis Lapicque 1907 · Translated by:” in: *Biological Cybernetics* 97 (Dec. 2007), pp. 341–349. DOI: 10.1007/s00422-007-0189-6.

[25] H. L. Bryant and J. P. Segundo. “Spike initiation by transmembrane current: a white-noise analysis.” In: *The Journal of physiology* 260(2) (1976), pp. 279–314. DOI: 10.1113/jphysiol.1976.sp011516.

[26] Lars Buesing, Johannes Bill, Bernhard Nessler, and Wolfgang Maass. "Neural Dynamics as Sampling: A Model for Stochastic Computation in Recurrent Networks of Spiking Neurons." In: *PLOS Computational Biology* 7.11 (Nov. 2011), pp. 1–22. DOI: 10.1371/journal.pcbi.1002211.

[27] T. Carnevale. "Neuron simulation environment." In: *Scholarpedia* 2.6 (2007). revision #154131, p. 1378. DOI: 10.4249/scholarpedia.1378.

[28] Ding Chen, Peixi Peng, Tiejun Huang, and Yonghong Tian. "Deep Reinforcement Learning with Spiking Q-learning." In: *arXiv:2201.09754 [cs]* (Jan. 2022). arXiv: 2201.09754. URL: http://arxiv.org/abs/2201.09754 (visited on 02/28/2022).

[29] Andrew P. Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. "PyNN: A Common Interface for Neuronal Network Simulators." eng. In: *Frontiers in neuroinformatics* 2 (Jan. 2009). PMC2634533[pmcid], pp. 11–11. ISSN: 1662-5196. DOI: 10.3389/neuro.11.011.2008.

[30] Thomas Degris, Patrick M. Pilarski, and Richard S. Sutton. "Model-Free reinforcement learning with continuous action in practice." In: *2012 American Control Conference (ACC)*. 2012, pp. 2177–2182. DOI: 10.1109/ACC.2012.6315022.

[31] Esther Derman, Daniel J. Mankowitz, Timothy A. Mann, and Shie Mannor. *Soft-Robust Actor-Critic Policy-Gradient*. 2018. arXiv: 1803.04848 [cs.LG].

[32] Peter U. Diehl, Guido Zarrella, Andrew S. Cassidy, Bruno U. Pedroni, and Emre Neftci. "Conversion of Artificial Recurrent Neural Networks to Spiking Neural Networks for Low-power Neuromorphic Hardware." In: *CoRR* abs/1601.04187 (2016). URL: http://arxiv.org/abs/1601.04187. arXiv: 1601.04187.

[33] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. *Challenges of Real-World Reinforcement Learning*. 2019. arXiv: 1904.12901 [cs.LG].

[34] Hugh Durrant-Whyte, Nicholas Roy, and Pieter Abbeel. "Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts." In: *Robotics: Science and Systems VII*. 2012, pp. 73–80.

[35] Chris Eliasmith and Charles H. Anderson. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. en. Ed. by Terrence J. Sejnowski and Tomaso A. Poggio. Computational Neuroscience Series. Cambridge, MA, USA: A Bradford Book, Oct. 2002. ISBN: 978-0-262-05071-5.

[36] Chris Eliasmith and Charles H. Anderson. “Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems.” In: *IEEE Transactions on Neural Networks* 15 (2004), pp. 528–529.

[37] Jason Kamran Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu. “Training Spiking Neural Networks Using Lessons From Deep Learning.” In: *CoRR* abs/2109.12894 (2021). URL: https://arxiv.org/abs/2109.12894. arXiv: 2109.12894.

[38] Michael D. Forrest. “Can the Thermodynamic Hodgkin-Huxley Model of Voltage-Dependent Conductance Extrapolate for Temperature?” In: *Computation* 2.2 (2014), pp. 47–60. ISSN: 2079-3197. DOI: 10.3390/computation2020047.

[39] Nicolas Frémaux, Henning Sprekeler, and Wulfram Gerstner. “Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons.” In: *PLOS Computational Biology* 9 (Apr. 2013), pp. 1–21. DOI: 10.1371/journal.pcbi.1003024.

[40] Scott Fujimoto and Shixiang (Shane) Gu. “A Minimalist Approach to Offline Reinforcement Learning.” In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 20132–20145. URL: https://proceedings.neurips.cc/paper/2021/file/a8166da05c5a094f7dc03724b41886e5-Paper.pdf.

[41] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods.” In: *arXiv:1802.09477 [cs, stat]* (Oct. 2018). arXiv: 1802.09477. URL: http://arxiv.org/abs/1802.09477 (visited on 05/02/2022).

[42] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. “Event-Based Vision: A Survey.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.1 (2022), pp. 154–180. DOI: 10.1109/TPAMI.2020.3008413.

[43] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511815706.

[44] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014. DOI: 10.1017/CBO9781107447615.

[45] Marc-Oliver Gewaltig and Markus Diesmann. “NEST (NEural Simulation Tool).” In: *Scholarpedia* 2.4 (2007), p. 1430.

[46] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[47] D. F. M. Goodman and R. Brette. “Brian simulator.” In: *Scholarpedia* 8.1 (2013). revision #129355, p. 10883. DOI: 10.4249/scholarpedia.10883.

[48] Vineet Goyal and Julien Grand-Clément. *Robust Markov Decision Process: Beyond Rectangularity*. 2018. DOI: 10.48550/ARXIV.1811.00215.

[49] Alex Graves. “Generating Sequences With Recurrent Neural Networks.” In: *CoRR* abs/1308.0850 (2013). URL: http://arxiv.org/abs/1308.0850. arXiv: 1308.0850.

[50] Wenzhe Guo, Mohammed E. Fouda, Ahmed M. Eltawil, and Khaled Nabil Salama. “Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems.” In: *Frontiers in Neuroscience* 15 (2021). ISSN: 1662-453X. DOI: 10.3389/fnins.2021.638474.

[51] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.” In: *CoRR* abs/1801.01290 (2018). URL: http://arxiv.org/abs/1801.01290. arXiv: 1801.01290.

[52] Hado Hasselt. “Double Q-learning.” In: *Advances in Neural Information Processing Systems*. Vol. 23. Curran Associates, Inc., 2010. URL: https://papers.nips.cc/paper/2010/hash/091d584fced301b442654dd8c23b3fc9-Abstract.html (visited on 05/02/2022).

[53] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning.” In: *arXiv:1509.06461 [cs]* (Dec. 2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461 (visited on 05/02/2022).

[54] Hananel Hazan, Daniel J. Saunders, Hassaan Khan, Devdhar Patel, Darpan T. Sanghavi, Hava T. Siegelmann, and Robert Kozma. “BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python.” In: *Frontiers in Neuroinformatics* 12 (2018). ISSN: 1662-5196. DOI: 10.3389/fninf.2018.00089.

[55] “Hebb, D. O. The organization of behavior: A neuropsychological theory. New York: John Wiley and Sons, Inc., 1949. 335 p. $4.00.” In: *Science Education* 34.5 (1950), pp. 336–337. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sce.37303405110. DOI: https://doi.org/10.1002/sce.37303405110.

[56] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. “Deep Reinforcement Learning that Matters.” In: *CoRR* abs/1709.06560 (2017). URL: http://arxiv.org/abs/1709.06560. arXiv: 1709.06560.

[57] Geoffrey Hinton. *Coursera Neural Networks for Machine Learning lecture 6*. 2018.

[58] A. L. HODGKIN and A. F. HUXLEY. “A quantitative description of membrane current and its application to conduction and excitation in nerve.” eng. In: *The Journal of physiology* 117.4 (Aug. 1952). PMC1392413[pmcid], pp. 500–544. ISSN: 0022-3751. DOI: 10.1113/jphysiol.1952.sp004764.

[59] Yangfan Hu, Huajin Tang, Yueming Wang, and Gang Pan. “Spiking Deep Residual Network.” In: *CoRR* abs/1805.01352 (2018). URL: http://arxiv.org/abs/1805.01352. arXiv: 1805.01352.

[60] Oliver C. Ibe. “Chapter 12 - Special Random Processes.” en. In: *Fundamentals of Applied Probability and Random Processes (Second Edition)*. Ed. by Oliver C. Ibe. Boston: Academic Press, Jan. 2014, pp. 369–425. ISBN: 978-0-12-800852-2. DOI: 10.1016/B978-0-12-800852-2.00012-2.

[61] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” In: *CoRR* abs/1502.03167 (2015). URL: http://arxiv.org/abs/1502.03167. arXiv: 1502.03167.

[62] Garud N. Iyengar. “Robust Dynamic Programming.” In: *Mathematics of Operations Research* 30.2 (2005), pp. 257–280. ISSN: 0364765X, 15265471. URL: http://www.jstor.org/stable/25151652 (visited on 05/28/2022).

[63] E.M. Izhikevich. “Simple model of spiking neurons.” In: *IEEE Transactions on Neural Networks* 14.6 (2003), pp. 1569–1572. DOI: 10.1109/TNN.2003.820440.

[64] E.M. Izhikevich. “Which model to use for cortical spiking neurons?” In: *IEEE Transactions on Neural Networks* 15.5 (2004), pp. 1063–1070. DOI: 10.1109/TNN.2004.832719.

[65] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. “Reinforcement Learning: A Survey.” In: *CoRR* cs.AI/9605103 (1996). URL: https://arxiv.org/abs/cs/9605103.

[66] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. “Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE).” In: *Frontiers in Neuroscience* 14 (2020). ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00424.

[67] Christoph Kayser, Marcelo A. Montemurro, Nikos K. Logothetis, and Stefano Panzeri. “Spike-Phase Coding Boosts and Stabilizes Information Carried by Spatial and Temporal Spike Patterns.” In: *Neuron* 61.4 (2009), pp. 597–608. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2009.01.008.

[68] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *International Conference on Learning Representations* (Dec. 2014).

[69] Donald E Kirk. *Optimal control theory: an introduction*. English. OCLC: 101663. 1970. ISBN: 978-0-13-638098-6 978-0-486-43484-1.

[70] C. Koch. *Biophysics of Computation: Information Processing in Single Neurons*. New York: Oxford University Press, 1999.

[71] Christof Koch and Idan Segev. *Methods in neuronal modeling: from ions to networks*. Cambridge University Press, 1998.

[72] Vijay Konda and John Tsitsiklis. “Actor-Critic Algorithms.” In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[73] Intel’s Neuromorphic Computing Lab. *Lava: A Software Framework for Neuromorphic Computing*. 2021. URL: https://lava-nc.org/.

[74] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning.” In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.

[75] Rémy Lestienne. “Determination of the precision of spike timing in the visual cortex of anaesthetised cats.” In: *Biological Cybernetics* 74.1 (Jan. 1996), pp. 55–61. ISSN: 1432-0770. DOI: 10.1007/BF00199137.

[76] Chen Li, Runze Chen, Christoforos Moutafis, and Steve Furber. “Robustness to Noisy Synaptic Weights in Spiking Neural Networks.” In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207019.

[77] Chen Li, Runze Chen, Christoforos Moutafis, and Steve Furber. “Robustness to Noisy Synaptic Weights in Spiking Neural Networks.” In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207019.

[78] Yuxi Li. *Deep Reinforcement Learning: An Overview*. 2017. DOI: 10.48550/ARXIV.1701.07274.

[79] Yuxi Li. "Deep Reinforcement Learning: An Overview." In: *CoRR* abs/1701.07274 (2017). URL: http://arxiv.org/abs/1701.07274. arXiv: 1701.07274.

[80] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." en. In: *arXiv:1509.02971 [cs, stat]* (July 2019). arXiv: 1509.02971. URL: http://arxiv.org/abs/1509.02971 (visited on 02/09/2022).

[81] Shiau Hong Lim, Huan Xu, and Shie Mannor. "Reinforcement Learning in Robust Markov Decision Processes." In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/0deb1c54814305ca9ad266f53bc82511-Paper.pdf.

[82] John Lisman and György Buzsáki. "A Neural Coding Scheme Formed by the Combined Function of Gamma and Theta Oscillations." In: *Schizophrenia Bulletin* 34.5 (June 2008), pp. 974–980. ISSN: 0586-7614. eprint: https://academic.oup.com/schizophreniabulletin/article-pdf/34/5/974/5317373/sbn060.pdf. DOI: 10.1093/schbul/sbn060.

[83] G. Liu. "Control of robot manipulators with consideration of actuator performance degradation and failures." In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 3. 2001, 2566–2571 vol.3. DOI: 10.1109/ROBOT.2001.933009.

[84] Wolfgang Maass. "Networks of spiking neurons: The third generation of neural network models." In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7.

[85] Hamid Reza Maei. "Convergent Actor-Critic Algorithms Under Off-Policy Training and Function Approximation." In: *CoRR* abs/1802.07842 (2018). URL: http://arxiv.org/abs/1802.07842. arXiv: 1802.07842.

[86] Hamid Reza Maei. "Gradient Temporal-Difference Learning Algorithms." PhD thesis. Edmonton, Alberta: University of Alberta, 2011. URL: https://era.library.ualberta.ca/items/fd55edcb-ce47-4f84-84e2-be281d27b16a/view/373459a7-72d1-4de2-bcd5-5f51e2f745e9/Hamid_Maei_PhDThesis.pdf.

[87] Z. F. Mainen and T. J. Sejnowski. "Reliability of spike timing in neocortical neurons." In: *Science (New York, N.Y.)* 268(5216) (1995), pp. 1503–1506. DOI: 10.1126/science.7770778.

[88] Daniel Mankowitz, Timothy Mann, Pierre-Luc Bacon, Doina Precup, and Shie Mannor. "Learning Robust Options." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018). URL: https://ojs.aaai.org/index.php/AAAI/article/view/12115.

[89] Daniel J. Mankowitz, Nir Levine, Rae Jeong, Abbas Abdolmaleki, Jost Tobias Springenberg, Timothy A. Mann, Todd Hester, and Martin A. Riedmiller. "Robust Reinforcement Learning for Continuous Control with Model Misspecification." In: *CoRR* abs/1906.07516 (2019). URL: http://arxiv.org/abs/1906.07516. arXiv: 1906.07516.

[90] Daniel J. Mankowitz, Timothy A. Mann, Pierre-Luc Bacon, Doina Precup, and Shie Mannor. "Learning Robust Options." In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA: AAAI Press, 2018. ISBN: 978-1-57735-800-8.

[91] Shie Mannor, Ofir Mebel, and Huan Xu. "Robust MDPs with k-Rectangular Uncertainty." In: *Mathematics of Operations Research* 41.4 (2016), pp. 1484–1509. ISSN: 0364765X, 15265471. URL: http://www.jstor.org/stable/26179995 (visited on 05/30/2022).

[92] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning." In: (2016). DOI: 10.48550/ARXIV.1602.01783.

[93] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. "Human-level control through deep reinforcement learning." In: *Nature* 518 (Feb. 2015), pp. 529–33. DOI: 10.1038/nature14236.

[94] Janosch Moos, Kay Hansel, Hany Abdulsamad, Svenja Stark, Debora Clever, and Jan Peters. "Robust Reinforcement Learning: A Review of Foundations and Recent Advances." In: *Machine Learning and Knowledge Extraction* 4 (Mar. 2022), pp. 276–315. DOI: 10.3390/make4010013.

[95] Janosch Moos, Kay Hansel, Hany Abdulsamad, Svenja Stark, Debora Clever, and Jan Peters. "Robust Reinforcement Learning: A Review of Foundations and Recent Advances." en. In: *Machine Learning and Knowledge Extraction* 4.1 (Mar. 2022). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, pp. 276–315. ISSN: 2504-4990. DOI: 10.3390/make4010013.

[96] Jun Morimoto and Kenji Doya. "Robust Reinforcement Learning." In: *Neural Comput.* 17.2 (Feb. 2005), pp. 335–359. ISSN: 0899-7667. DOI: 10.1162/0899766053011528.

[97] Hesham Mostafa, Bruno U. Pedroni, Sadique Sheik, and Gert Cauwenberghs. "Fast classification using sparsely active spiking networks." In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2017, pp. 1–4. DOI: 10.1109/ISCAS.2017.8050527.

[98] Robert A. Nawrocki, Richard M. Voyles, and Sean E. Shaheen. "A Mini Review of Neuromorphic Architectures and Implementations." In: *IEEE Transactions on Electron Devices* 63 (2016), pp. 3819–3829.

[99] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks." In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63. DOI: 10.1109/MSP.2019.2931595.

[100] Vinh Nguyen, Michael Carilli, Sukru Burc Enyilmaz, Vartika Singh, Michelle Lin, Natalia Gimelshein, Alban Desmaison, and Edward Yang. *Accelerating PyTorch with CUDA Graphs*. 2021. URL: https://pytorch.org/blog/accelerating-pytorch-with-cuda-graphs/ (visited on 07/05/2022).

[101] Rui Nian, Jinfeng Liu, and Biao Huang. "A review On reinforcement learning: Introduction and applications in industrial process control." In: *Computers & Chemical Engineering* 139 (2020), p. 106886. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2020.106886.

[102] Arnab Nilim and Laurent Ghaoui. "Robustness in Markov Decision Problems with Uncertain Transition Matrices." In: *Advances in Neural Information Processing Systems*. Ed. by S. Thrun, L. Saul, and B. Schölkopf. Vol. 16. MIT Press, 2003. URL: https://proceedings.neurips.cc/paper/2003/file/300891a62162b960cf02ce3827bb363c-Paper.pdf.

[103] OECD. *Understanding the Brain: The Birth of a Learning Science*. 2007, p. 264. DOI: https://doi.org/https://doi.org/10.1787/9789264029132-en.

[104] Sindhu Padakandla. "A Survey of Reinforcement Learning Algorithms for Dynamically Varying Environments." In: *ACM Comput. Surv.* 54.6 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3459991.

[105] K. Pakdaman, M. Thieullen, and G. Wainrib. “Fluid limit theorems for stochastic hybrid systems with application to neuron models.” In: *Advances in Applied Probability* 42.3 (2010), pp. 761–794. DOI: 10.1239/aap/1282924062.

[106] Xinlei Pan, Daniel Seita, Yang Gao, and John Canny. “Risk Averse Robust Adversarial Reinforcement Learning.” In: *arXiv:1904.00511 [cs]* (Mar. 2019). arXiv: 1904.00511. URL: http://arxiv.org/abs/1904.00511 (visited on 02/23/2022).

[107] Zihan Pan, Jibin Wu, Yansong Chua, Malu Zhang, and Haizhou Li. “Neural Population Coding for Effective Temporal Classification.” In: *CoRR* abs/1909.08018 (2019). URL: http://arxiv.org/abs/1909.08018. arXiv: 1909.08018.

[108] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[109] Devdhar Patel, Hananel Hazan, Daniel J. Saunders, Hava T. Siegelmann, and Robert Kozma. “Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to ATARI games.” In: *CoRR* abs/1903.11012 (2019). URL: http://arxiv.org/abs/1903.11012. arXiv: 1903.11012.

[110] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. “Robust Deep Reinforcement Learning with Adversarial Attacks.” In: *CoRR* abs/1712.03632 (2017). URL: http://arxiv.org/abs/1712.03632. arXiv: 1712.03632.

[111] Christian Pehle and Jens Egholm Pedersen. *Norse - A deep learning library for spiking neural networks*. Version 0.0.7. Documentation: https://norse.ai/docs/. Jan. 2021. DOI: 10.5281/zenodo.4422025.

[112] Michael Pfeiffer and Thomas Pfeil. “Deep Learning With Spiking Neurons: Opportunities and Challenges.” In: *Frontiers in Neuroscience* 12 (2018). ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00774.

[113] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. "Robust Adversarial Reinforcement Learning." In: *arXiv:1703.02702 [cs]* (Mar. 2017). arXiv: 1703.02702. URL: http://arxiv.org/abs/1703.02702 (visited on 02/18/2022).

[114] Antonin Raffin. *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo. 2020.

[115] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. "Stable-Baselines3: Reliable Reinforcement Learning Implementations." In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[116] B. J. Richmond and L. M. Optican. "Temporal encoding of two-dimensional patterns by single units in primate inferior temporal cortex. II. Quantification of response waveform." In: *Journal of Neurophysiology* 57.1 (1987). PMID: 3559669, pp. 147–161. eprint: https://doi.org/10.1152/jn.1987.57.1.147. DOI: 10.1152/jn.1987.57.1.147.

[117] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification." In: *Frontiers in Neuroscience* 11 (2017). ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682.

[118] Shirli Di-Castro Shashua and Shie Mannor. "Deep Robust Kalman Filter." In: *CoRR* abs/1703.02310 (2017). URL: http://arxiv.org/abs/1703.02310. arXiv: 1703.02310.

[119] David Silver. *Lectures on Reinforcement Learning*. URL: https://www.davidsilver.uk/teaching/. 2015.

[120] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961.

[121] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic Policy Gradient Algorithms." en. In: *Proceedings of the 31st International Conference on Machine Learning*. ISSN: 1938-7228. PMLR, Jan. 2014, pp. 387–395. URL: https://proceedings.mlr.press/v32/silver14.html (visited on 02/11/2022).

[122] Richard B. Stein. "A Theoretical Analysis of Neuronal Variability." In: *Biophysical Journal* 5.2 (1965), pp. 173–194. ISSN: 0006-3495. DOI: https://doi.org/10.1016/S0006-3495(65)86709-1.

[123] Marcel Stimberg, Romain Brette, and Dan FM Goodman. "Brian 2, an intuitive and efficient neural simulator." In: *eLife* 8 (Aug. 2019). Ed. by Frances K Skinner, e47314. ISSN: 2050-084X. DOI: 10.7554/eLife.47314.

[124] Christoph Stöckl and Wolfgang Maass. "Classifying Images with Few Spikes per Neuron." In: *CoRR* abs/2002.00860 (2020). URL: https://arxiv.org/abs/2002.00860. arXiv: 2002.00860.

[125] Malcolm Strens. "A Bayesian framework for reinforcement learning." In: *In Proceedings of the Seventeenth International Conference on Machine Learning*. ICML, 2000, pp. 943–950.

[126] Ke Sun, Yi Liu, Yingnan Zhao, Hengshuai Yao, Shangling Jui, and Linglong Kong. "Exploring the Robustness of Distributional Reinforcement Learning against Noisy State Observations." In: *CoRR* abs/2109.08776 (2021). URL: https://arxiv.org/abs/2109.08776. arXiv: 2109.08776.

[127] Richard S Sutton. "On the virtues of linear learning and trajectory distributions." In: *Proceedings of the Workshop on Value Function Approximation at The 12th International Conference on Machine Learning*. ICML, 1995. URL: http://incompleteideas.net/papers/sutton-95-VFAabstract.pdf.

[128] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[129] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. en. Ed. by Francis Bach. 2nd ed. Adaptive Computation and Machine Learning series. Cambridge, MA, USA: A Bradford Book, Nov. 2018. ISBN: 978-0-262-03924-6.

[130] Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. "Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation." In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 993–1000. ISBN: 9781605585161. DOI: 10.1145/1553374.1553501.

[131] Richard S. Sutton, Ashique Rupam Mahmood, and Martha White. "An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning." In: *CoRR* abs/1503.04269 (2015). URL: http://arxiv.org/abs/1503.04269. arXiv: 1503.04269.

[132] Aboozar Taherkhani, Ammar Belatreche, Yuhua Li, Georgina Cosma, Liam P. Maguire, and T.M. McGinnity. "A review of learning in biologically plausible spiking neural networks." In: *Neural Networks* 122 (2020), pp. 253–272. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2019.09.036.

[133] Aviv Tamar, Dotan Di Castro, and Shie Mannor. "Learning the Variance of the Reward-To-Go." In: *Journal of Machine Learning Research* 17.13 (2016), pp. 1–36. URL: http://jmlr.org/papers/v17/14-335.html.

[134] Weihao Tan, Devdhar Patel, and Robert Kozma. "Strategy and Benchmark for Converting Deep Q-Networks to Event-Driven Spiking Neural Networks." In: *CoRR* abs/2009.14456 (2020). URL: https://arxiv.org/abs/2009.14456. arXiv: 2009.14456.

[135] Guangzhi Tang, Neelesh Kumar, and Konstantinos P. Michmizos. "Reinforcement co-Learning of Deep and Spiking Neural Networks for Energy-Efficient Mapless Navigation with Neuromorphic Hardware." In: *arXiv:2003.01157 [cs]* (July 2020). arXiv: 2003.01157. URL: http://arxiv.org/abs/2003.01157 (visited on 02/28/2022).

[136] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. "Deep learning in spiking neural networks." In: *Neural Networks* 111 (2019), pp. 47–63. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2018.12.002.

[137] Chen Tessler, Yonathan Efroni, and Shie Mannor. "Action Robust Reinforcement Learning and Applications in Continuous Control." In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 2019, pp. 6215–6224. URL: https://proceedings.mlr.press/v97/tessler19a.html.

[138] Johannes C. Thiele, Olivier Bichler, and Antoine Dupret. "Event-Based, Timescale Invariant Unsupervised Online Deep Learning With STDP." In: *Frontiers in Computational Neuroscience* 12 (2018). ISSN: 1662-5188. DOI: 10.3389/fncom.2018.00046.

[139] Simon Thorpe, Denis Fize, and Catherine Marlot. "Speed of processing in the human visual system." In: *Nature* 381(6582) (1996), pp. 520–522. DOI: 10.1038/381520a0.

[140] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control." In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[141] Joaquin J. Torres and Pablo Varona. "Modeling Biological Neural Networks." In: *Handbook of Natural Computing*. Ed. by Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 533–564. ISBN: 978-3-540-92910-9. DOI: 10.1007/978-3-540-92910-9_17.

[142] Martin Tovée and Edmund Rolls. "Information Encoding in Short Firing Rate Epochs by Single Neurons in the Primate Temporal Visual Cortex." In: *Visual Cognition* 2 (Mar. 1995), pp. 35–58. DOI: 10.1080/13506289508401721.

[143] John N. Tsitsiklis and Benjamin Van Roy. *An analysis of temporal-difference learning with function approximation*. Tech. rep. IEEE Transactions on Automatic Control, 1997.

[144] G. E. Uhlenbeck and L. S. Ornstein. "On the Theory of the Brownian Motion." In: *Phys. Rev.* 36 (5 Sept. 1930), pp. 823–841. DOI: 10.1103/PhysRev.36.823.

[145] Xiangwen Wang, Xianghong Lin, and Xiaochao Dang. "Supervised learning in spiking neural networks: A review of algorithms and evaluations." In: *Neural Networks* 125 (2020), pp. 258–280. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2020.02.011.

[146] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning." en. In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698.

[147] Christopher John Cornish Hellaby Watkins. "Learning from Delayed Rewards." PhD thesis. Cambridge, UK: King's College, May 1989. URL: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[148] Paweł Wawrzyński. "A Cat-Like Robot Real-Time Learning to Run." In: *Adaptive and Natural Computing Algorithms*. Ed. by Mikko Kolehmainen, Pekka Toivanen, and Bartlomiej Beliczynski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 380–390. ISBN: 978-3-642-04921-7.

[149] Chelsea C. White and Hany K. Eldeib. "Markov Decision Processes with Imprecise Transition Probabilities." In: *Operations Research* 42.4 (1994), pp. 739–749. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/171626 (visited on 05/28/2022).

[150] Wolfram Wiesemann, Daniel Kuhn, and Breç Rustem. “Robust Markov Decision Processes.” In: *Mathematics of Operations Research* 38.1 (2013), pp. 153–183. ISSN: 0364765X, 15265471. URL: http://www.jstor.org/stable/23358653 (visited on 05/30/2022).

[151] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.” In: *Machine Learning* 8 (2004), pp. 229–256.

[152] Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. “Robust control of uncertain Markov Decision Processes with temporal logic specifications.” In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. 2012, pp. 3372–3379. DOI: 10.1109/CDC.2012.6426174.

[153] Jibin Wu, Yansong Chua, Malu Zhang, Qu Yang, Guoqi Li, and Haizhou Li. “Deep Spiking Neural Network with Spike Count based Learning Rule.” In: *CoRR* abs/1902.05705 (2019). URL: http://arxiv.org/abs/1902.05705. arXiv: 1902.05705.

[154] Huan Xu and Shie Mannor. “Distributionally Robust Markov Decision Processes.” In: *Mathematics of Operations Research* 37.2 (2012), pp. 288–300. ISSN: 0364765X, 15265471. URL: http://www.jstor.org/stable/23256624 (visited on 05/30/2022).

[155] Friedemann Zenke and Surya Ganguli. “SuperSpike: Supervised learning in multi-layer spiking neural networks.” In: *Neural Computation* 30.6 (June 2018). arXiv: 1705.11146, pp. 1514–1541. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco_a_01086.

[156] Friedemann Zenke and Surya Ganguli. “SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks.” In: *Neural Computation* 30.6 (June 2018), pp. 1514–1541. DOI: 10.1162/neco_a_01086.

[157] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane S. Boning, and Cho-Jui Hsieh. “Robust Deep Reinforcement Learning against Adversarial Perturbations on Observations.” In: *CoRR* abs/2003.08938 (2020). URL: https://arxiv.org/abs/2003.08938. arXiv: 2003.08938.

[158] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey.” In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2020, pp. 737–744. DOI: 10.1109/SSCI47803.2020.9308468.

[159] Kemin Zhou and John Comstock Doyle. *Essentials of robust control*. Vol. 104. Prentice hall Upper Saddle River, NJ, 1998.

TRITA-EECS-EX- 2022:00