DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Training of Spiking Neural Networks with Reinforcement Learning

Benedikt S. Vogler

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Training of Spiking Neural Networks with Reinforcement Learning

Trainieren von gepulsten neuronalen Netzen mit bestärkendem Lernen

Author:	Benedikt S. Vogler
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Mahmoud Akl
Submission Date:	14th July 2020

I confirm that this master's thesis in Robotics, Cognition, Intelligence is my own work and I have documented all sources and material used.

Munich, 14th July 2020 Benedikt S. Vogler

Acknowledgments

I thank Mahmoud Akl for his helpful input to this work.

Thanks to my family, especially my parents, for supporting me to develop my potential.

Thanks to Nadia Singer for her support and making me believe in my skills, when I did not.

I thank Daphne Cornelisse for helpful discussions.

I want to thank everyone, who gave proof-reading feedback.

Thanks to the NEST community for providing the software and support.

I thank the Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Sciences and Humanities (BAdW) for the provisioning and support of Cloud computing infrastructure essential to this publication.

A special mention of Alexandra Elbakyan for her efforts in providing millions of people, including me, with access to knowledge.

Abstract

Spiking Neural Networks (SNN) implemented on neuromorphic hardware offer great potential in computation speed and energy efficiency and thus are of interest for real-time applications like robotics. However, SNN are currently not trainable as the more abstracted Artificial Neural Networks (ANNs). Biologically inspired reinforcement learning algorithms are an option to train SNN. Neuromodulation influenced spike-timing-dependent-plasticity (STDP) has been identified as a learning mechanism in the brain. It can be combined with an actor-critic framework using the reinforcement signal (R-STDP).

In this work, the requirements for the design of the spiking net are evaluated, resulting in a generalized framework to construct and train neuronal motorcontrollers. Central to this generalizability is the usage of the critic extending the use of R-STDP to new problem classes like delayed rewards. We show that solvable problem classes can be further extended to problems with high-dimensional input with the application of vector quantization on receptive fields.

We derive a rate coded abstraction from the used mechanisms in the spiking net. The framework is tested on trajectory-following and pole-balancing problems. The rate coded abstraction is able to solve the pole-balancing task. The fully spiking net struggles with falling weights probably caused by higher sensitivity to biases in the neuromodulatory signal.

Contents

1 Introduction

1.1 Motivation

The dream of constructing thinking machines is quite old. The industrialization caused rise to mechanical machines, which were then used to mechanize computations, which previously could only be performed by thinking in humans. The history of computing is the history of artificial intelligence. Ada Lovelace and Charles Babbage started with work on the Analytical Engine. Modern computing began with Konrad Zuse's machines to automate manual computations. Improving the possibilities of AI is one major driving force of improving the possibilities of computing machines. Neural Networks are the foundation of thinking in animals. Neuromorphic computing offers the accelerated hardware implementation of biologically plausible neural networks.

The modern computer uses the von Neumann architecture and is constructed using metal-oxide-semiconductor field-effect transistors (MOSFET). MOSFET chip design is limited by thermal constraints and physically constrained limits on the fabrication process. It is expected that further advances in MOSFET chips production slow down, making a halt to *Moore's Law*[1]. Some computing problems cannot be addressed with current and expected CPUs. Therefore, new computing paradigms need to be developed: neuromorphic chips and quantum computing.

Neuromorphic hardware is inspired by the workings of neural systems. Biological neural systems utilize events, which are transported with waves of propagating electrochemical potentials. Mathematical models, which utilize spike events can be constructed. The processing of the spike events happens in Spiking Neural Networks (SNN). It is possible to model biological neural networks using neuromorphic hardware but "with a speedup factor of 10^3 up to 10^5 compared to biological real-time" [Dav+19]. The speedup by using general-purpose computers is currently limited to a tenth of real-time [ZG14]. Like von Neumann computers, neuromorphic systems can be found at the levels of (super) computers with many connected MOSFET integrated circuits (chips) or in the form of single Systems-on-chip (SoC). A single chip is very light in weight and hence easily installable in robots, wearables, or cybernetic enhancements. Neuromorphic systems can be faster than von Neumann architectures because they do

[1]Gordon Moore observed that the number of transistors in integrated circuits doubles every two years due to improvements in MOSFET fabrication.

not store the memory separately at a central place but store it in the network where the computations happen. The local information storage can be implemented with low-power, variable-state, nanoscale memory. Developments are tried with memristors or phase-change materials. Another approach is to use massive parallelization of von Neumann cores (e.g., devices like SpiNNaker, Mythic intelligence processing unit).

The twilight of von Neumann computing as the established computing paradigm is not the only driving force in the development of SNN implementations. To understand brain functions the field of neuroscience is limited in its object of examination on living material. Simulations of neural nets give scientists a new tool to understand neural systems by looking at artificially created systems.

Combining insights of neuroscience and making neural networks available in hardware allows the construction of improved artificial intelligence.

Before a new paradigm is utilized outside research facilities, it must have benefits comparing to current technology. We identify three areas where SNN are superior to current technology.

1.1.1 Energy Efficiency and Performance

Using neuromorphic chips training is possible with similar efficiency as in the brain, magnitudes better (in terms of energy consumption and speed) than simulating neural nets on conventional computers. To substantially increase the speed of neural simulations, neuromorphic hardware must be used [ZG14].

Traditional computers based on the von Neumann architecture use complementary metal-oxide-semiconductor (CMOS) chips. CMOS chips dissipate power in each switching. The number of switches on von Neumann computers is based on the clock frequency. The clock frequency typically reaches orders of 10^6 to 10^9 per second. Thermal design points (TDP), the maximum heat dissipation value, in high-end CPUs often reaches 130 W. Modern chips self-overclock in short bursts to reach higher clock frequencies till limiting temperatures or power consumptions are reached. This can create a TDP of 250 W. Not only is the energy consumption of these chips high, but an increased TDP sets the requirements for the needed cooling.

The potential of SNN becomes evident when looking at the brain. The brain consumes roughly 20 W of power [Lin01] while doing computations which are currently not even possible to simulate on von Neumann based supercomputers.

Information transmission and computation in biological brains happens when an event is transmitted from one neuron to another. In artificial synapses, the gates are only switched on incoming events and thus only consume power when events are incoming. Spiking chips could be used as co-processors like general-purpose graphics processors (GPGPU, including tensor and raytracing cores) and cryptographic accelerators. In the area of mobile computing, SNN chips must compare against low-power chips typically used in mobile devices. These chips can currently be found with up to eight asymmetric cores, with some core having a clock speed of up to 2,49 GHz (Apple A12Z) while still not requiring active cooling.

A proof-of-concept by Wunderlich et al. consumed 57mW using the BrainScaleS 2 architecture [Wun+19]. They compared the speed and the energy efficiency and found both to be better on the neuromorphic hardware by order of three magnitudes. Compared to a CPU, Intel's neuromorphic chip "Loihi" demonstrated a 23 times lower energy consumption per inference in a deep learning benchmark[Blo+18].

1.1.2 Event-based Dynamic Vision Sensors

Dynamic vision sensors (DVS) are novel image sensors, which pick up changes in pixel contrast. When observing for motion in the optical input, they have one benefit over conventional sensors. Instead of transmitting a frame each update, they only transport changes, resulting in a total of less data while still transmitting the same or larger amount of information. Current systems are available with a time resolution of tens of microseconds [Pro20].

Spiking actors promise the fast, low energy processing of DVS data without translating to a symbolic representation first [Mes17]. DVS's transmit only the changes in the input as events with a temporal resolution of microseconds. Using DVS sensors and neuromorphic systems a reaction time of motion as low as 3 milliseconds is possible [Liu+15][p. 377], [Con+09].

1.1.3 Neuron-Computer Interfaces

Silicon based event processing is connected to biological neurons to extend their computation or communication capabilities. This is done with neuroprosthetics, e.g., cochlea implants.

It is possible to construct biochips where semi-conductor technology is coupled with biotechnology, e.g., to construct organs-on-a-chip[Huh+10]. Using actual neurons instead of using silicone neurons has become a viable option, allowing novel sensors. The California company Koniku develops novel sensors to detect chemicals by smell. It cites applications in surveillance, detecting explosives for the military and sensors in farming robotics. Another Australian start-up called "Cortical Labs" is working on utilizing biological neurons with an electrode interface. It should be noted that the neural network processing speed on organ-on-chip design cannot be controlled. Their field of application is therefore limited to processing of real-time sensor data

1.2 Neuron Model

Neurons are the building blocks of the brain. They transmit information via spike events. Different concentrations of ions form electrochemical fields across the neuron's membrane. When the electric potential across the membrane produced by ion distribution is big enough, a chain reaction is caused. More ion channels open and causing an increase in the potential. Starting from the soma a wave of the potential travels along the axon. This wave is called a spike. At the end is the axon terminal. Neurotransmitters cross the synaptic cleft to another neuron and transmit information by changing ion concentrations. This contact point forms the synapse. The synapse is often connected to a dendrite, but it can also be connected directly to the soma. The information is transmitted via the effect it has on the potential. The size of the effect a synapse has on the connected system is the synaptic efficacy. This efficacy can change, which is called synaptic plasticity. An increase is called long term potentiation (LTP); the decrease is called long term depression (LDP).

The aspects of neural computation in the brain can be modeled at different stages of complexity and biological abstractions [Her+06]. Probabilistic theories can model a lot of the aspects of a biological system. In SNNs, we abstract ion distributions. Hodgkin and Huxley formed a mechanistic model by using a single equation to describe the diffusion of Cl^-, Na^+, K^+ ions in measurements [HH52]. Because it is easier to compute, the resulting current can be abstracted.

Spiking models have a time component. A spike has a position changing over time as the spike moves like a wave along the membrane of the neuron. A compartment model describes the position of spikes on the neuron over time. We assume that the information on the voltage potential via the description of a single compartment.

By putting the above description into a more formalized model, we get a spiking neural model. In SNNs, the level of abstraction is set so that single spikes can be tracked.

From the perspective of neural computing, the motivation is to use neurons to perform computations. Widely used are the abstracted model of Warren McCulloch and Walter Pitts [MP43], considered the first generation of neural networks. The output of that neuron is binary.

By assuming an abstracted rate code, as the spikes can be integrated, and adding a nonlinear activation function, we obtain the second generation neural network. This model is called *artificial neuron*. An application with the most straightforward artificial neural network (ANN) in a single layer is called the *perceptron*. By putting an analog signal to an ANN, an analog circuit is obtained. During a frame, the activity (number of spikes) of a neuron can be integrated. A frame is based on one set of constant input for a short time. In ANN, the calculations are continuous for one frame.

The so-called Leaky-Integrate-and-Fire (LIF) neuron model assumes that the timing of spikes is of importance. The electrical potential is created by incoming spikes and integrated. It leaks the potential over time. When the integrand hits a threshold, a spike is emitted and the integration term is reset. By adding the temporal information, we arrive at the spiking models, the “third generation” of neural networks.

This binary threshold function used in the McCulloch-Pitts and the LIF neuron to elicit a spike is a simplification. In biological neurons, there is not a hard limit at which transmembrane current creates a spike. The response changes quickly from 'no reaction' to 'full spike' with reactions in between.[GK02][Chapter 2.2]

So far, the spiking neural networks (SNN) are not yet widely established as a tool to solve problems in AI tasks and construct neuro-controllers.

A single compartment simulating the nucleus, and a simple description of a synapse is enough for our concerns. Multi-compartment models split a neuron into more parts.

1.3 Challenges in the Construction of Spiking Neural Nets

Neural Networks cannot be programmed in an imperative fashion using programming languages, thus causing a paradigm change in applied computer science.

A neural network resembles a complex function where inputs are fed into the network, and at the other end, the outputs can be read. Information flows through the net. The flow is defined by the connectivity and the synaptic weights. This information flow is trained by feeding data into the net and using the net's output to tweak the network's parameter forming a cybernetic loop. Thus, a neural network is constructed in a declarative fashion. Inputs and output are specified and a description of the problem is given.

While ANNs can be trained for general-purpose problems, there is currently no established known general-purpose algorithm for SNN [Bin+18]. The lack of an established training algorithm currently makes the appliance of neuromorphic systems challenging.

1.3.1 Focus of This Work

A working architecture covers three areas, which is depicted in a framework in Figure 1.1. The input encoding, the connectivity and the output encoding. The learning mechanism must be developed together with the critic, a function that judges rewards.

In the following, we will look at the topic of implementation and scaling of learning algorithms for spiking neural networks in a reinforcement learning environment.

The benefit of abstraction in models is that computation complexity can be reduced but keeping the phenomenology close to the biological reality. The number of param-

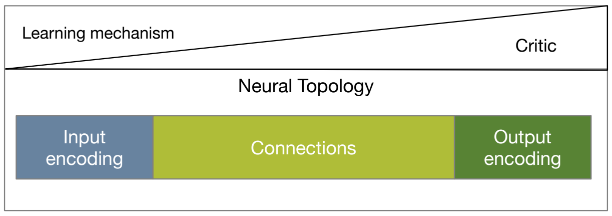

Figure 1.1: The areas of training.

eters for a single neuron is huge. It can be reduced by using plausible distributions [GSD12]. Therefore the choice of parameters should be backed up by neurophysiological findings. From an engineering perspective, we may derivate from biological plausibility if modifications serve the intended goal.

1.4 Learning Methods

How should SNN be constructed so that they copy the computational functionality of brains?

Alan Turing proposed learning to develop an adult brain: "Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain."[Tur50] It is possible to simulate physics and graphics in a simulated environment covering a large number of experiences of a young child's life. However, essential aspects like communication with the mother are missing.

Some models of mechanisms in the brain created by researchers have been shown to explain some phenomena, yet the process of learning is mostly still unclear. It is an open question how to make a system learn a variety of tasks.

Learning is reward based. A reward is created by the brain. W. Schultz presents two mechanisms of reward based learning in animals [Sch15]: Pavlovian learning shows that learning can happen without actions, besides being attentive and just based on stimuli. In Pavlov's experiment a dog learned to salivate following the sound of a bell. First, food was presented with the bell. Later, only the bell rang, however the dog was now conditioned to salivate in anticipation of the food. Thorndike's cat learned that pressing a lever will result in obtaining food. It shows that learning can also be connected to its own actions. Schultz goes on: "The two learning mechanisms can be distinguished schematically but occur frequently together and constitute the building blocks for behavioral reactions to rewards."

The idea of describing formal systems in a feedback loop coupled with the environment was developed by Norbert Wiener and popularized by W. Ross Ashby's work "An Introduction to Cybernetics" (1956). Learning systems are in a cybernetic loop. Therefore cybernetics are the formal foundation for training algorithms.

1.4.1 Evolution

Learning is a mechanism to optimize a given physiognomy to the environment. The foundation of learning is evolution. Evolutionary algorithms optimize parameters with regard to the objective function (here: reward). It has been shown that this is feasible with ANNs [Suc+17]. Evolution favors systems which can learn during their lifetime. Selective breeding could create a body allowing it to learn and with that, getting better fitness. It is better to learn how to learn. Instead of using a fixed model, Jordan et al. used cartesian genetic programming to evolve SNN plasticity rules [Jor+20]. The resulting rules are similar to known models.

1.4.2 Supervised Learning

A supervised learning problem is a problem where a set of input and output pairs is given, and the best weights are to be found minimizing the error. When the output of the network to an input signal is already known, a connectome can be found replicating this output. The output of a controller is given when it is feasible to design one manually, or it has already been developed. Examples include [Cha+12] and [BS10]. The ReSuMe is a supervised method to obtain spike patterns for given sequences [PK10]. The spiking behavior of a neuron is observed. If a spike happens too late, the weight is increased. It is decreased when a spike should happen earlier.

The tempotron is another method with similarities to synaptic plasticity rules in the brain [GS06]. A cost function is constructed, which is then optimized with gradient descent based supervised learning rule.

1.4.3 Reinforcement Learning

Reinforcement learning is similar to supervised learning. The difference is that no data set is required. The data is generated online. Therefore, it is usually used in robotics, where robots continuously learn through experience. Like in supervised learning, every input-output pair is rated by a reward function. This is beneficial over supervised learning because we do not need to construct a working system first.

1.4.4 Inverse Reinforcement Learning

By using expert demonstration, a reward function can be learned [Sin+19]. However, this only works when there is an expert available or another working actor for this problem. So this has similar requirements like supervised learning approaches. Another method mentioned by Singh et al. is “Classifier training.” Here a classifier is trained on recorded data, which is used for reinforcement learning. Thus, it is a combination of supervised learning and reinforcement learning.

1.4.5 Copying Neurophysiological Data

Data from actual living or dead brains can be recorded and loaded in a simulation. It can then be used in a robot. However, obtaining this data is very hard, and a robot has different physiognomy than the biological organism.

1.4.6 Translating Weights from ANN to SNN

An ANN utilizing rectified linear units (ReLU) activation functions can be trained using backpropagation methods and then translating the ANN to a SNN [HE16].

1.5 Training Environment

The training can be either performed in the real world or a model. Constructing a model to a real-world situation can be hard as finding the reward function.

What are the reasons to train first in a simulated environment? There might already be a controller that should be replaced. Another reason is that training on the simulation is faster and less expensive as faults do no harm in a simulated environment. Because of reduced costs in simulations, higher variance in states can be explored.

2 On Reinforcement Learning

Us humans perceive high-level alignment mechanisms in the form of goal planning or motivation. An agent, i.e., a human, has subconscious alignment mechanisms that manifest in behavioral preferences, which we then call character traits. What are the mechanisms to transform motivation to internalized processes, and how are they using neural systems? How do systems learn, and how do we teach?

We will generalize the concept of a teacher teaching humans or animals, to include artificial agents. Instead of a teacher, we will use the term *system designer*. The meaning of a computation arises out of its predictive power unless a system designer imposes meaning. The system designer has some prescriptive believes and seeks to evoke a matching value system in the manipulated system by applying extrinsic motivation. Depending on the form of the applied extrinsic motivation, it evokes behavior leading to a desired state (described by an objective function) or creates the desired behavioral preferences, e.g., seeking novel states (novelty metric) [LS08]. When a system has internalized, i.e., learned, the imposed motivation, it has developed (machine) "values" similar to how a human has "human values". Extrinsic motivation is passed to the learning system in the form of a reward. A reward is a signal rating a current state with regards to a favorable state. The reward is the result of cognitive processes. This cognitive process can be external or embedded internally in the learning system. In order to make use of the resulting learning signal, it must have meaning for the system. This meaning emerges from the design of the learning algorithm.

Intrinsic motivation is the result of evolution. Behavior which improves the chance of reproduction is favored, so organisms under the effect of evolution will be motivated to perform this behavior. Applying artificial evolution mechanisms to a system can be used to evoke intrinsic motivation. The choice of the fitness function used for deciding which agent will reproduce is also external pressure by the system designer.

A system can have multiple goals or constraints, e.g., reaching a goal while minimizing resource consumption; thus, the reward becomes multi-dimensional. Animals are bound to their bodies and one position in space. Therefore they tend to advance to a single point at a time to perform a task with the highest priority (e.g., foraging, mating). Attention mechanisms enforce a dimensionality reduction in the input.

2.1 Reinforcement Learning Framework

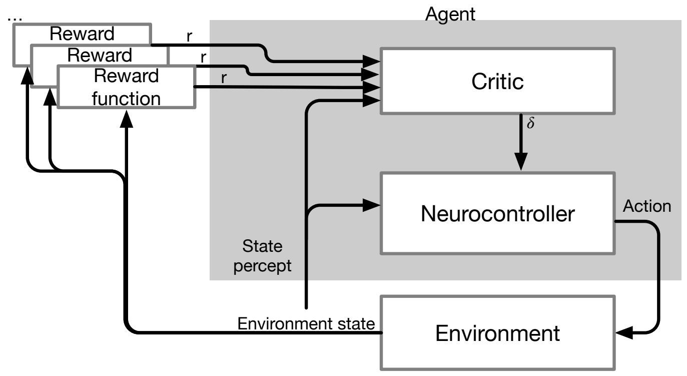

Figure 2.1: A framework of reinforcement learning.

We define a framework of reinforcement learning, depicted in Figure 2.1, by expanding on the neural actor-critic framework by R. S. Sutton and Barto [SB18][p. 396]. The lower part of the framework shows a loop of perception, cognition, and action (PCA-loop), which is examined extensively in the field of cybernetics. In a reinforcement learning context, every loop contains a reward signal; thus, it is also called Perception-Action-Reward Cycle (PARC).

A real-world reinforcement learning system is continuous, but in the RL literature, the model is often discretized, which allows the examination as a Markov decision process (MDP). Digital systems as MOSFET image sensors or calculations in von Neumann computers are also frame-based. An analysis of the continuous form is given by [Doy00], offering improved results when not discretizing.

The behavior of a system is called *policy* and denoted by π_i. It is implemented by the *controller* (in the RL literature often called *actor*). Because the policy changes during the training of the controller over time, it is indexed.

The *reward function* and *critic* judge the actions. They are components of the prediction problem: How does an observed state predict its value[1] [Sut88]? The reward function maps percepts (states) to rewards. The system designer designs the reward function and thereby takes a part of the prediction problem out of the agent. It becomes an element of the environment. From the perspective of the agent, the reward function's output becomes another percept. The critic must account that this percept is not regular.

[1]Sutton and Barto use the term *value* while in their popular work Russel and Norvig[RN16] use the term *utility*. Although the term utility does not suffer from its generic meaning, we will use 'value' to lean on the original publications.

It is a numerical value that has a meaning; it is to be increased. The return of an episode (a trial from start till a terminating condition) is the sum of the returned rewards in each cycle during the episode.

Constructing a reward function is called reward engineering. It is very critical to the success of the RL application. However, this process can be a difficult task as it requires domain knowledge. A common mistake is to generate a reward signal based on the designer's expectation on the agent's correct actions. Sutton and Barto argue that an RL application's success is based on the quality of measuring progress towards the given goal. [SB18][p. 469]

The reward function varies in quality. A reward can be binary to indicate failure or winning states, but a better version would utilize real numbers values for a more nuanced metric. Given multiple constraints, the rewards can become multi-dimensional.

An example of a simple reward function is the reward function used in the inverted pole-balancing problem included in the *gym* framework by OpenAI; a reward of one is returned for every step unless the failure state is reached and the episode is stopped. The success of the run is then given by the *return*, in this case, equivalent to the length of the run. Problems, where a reward is only given some time, are called *sparse reward problems* [SB18][Chapter 17.4]. By adding a penalty to the failure state and the state-value function, it transforms into a delayed reward problem. The penalty must have a lower value than the regular state associated reward.

The designer may confuse a good state with a good way to reach the wanted states and rewards states on the path to the expected goal. We wish that the designer can be without any detailed knowledge of the environment, and the best policy is found intelligently, therefore allowing reward functions of low quality. An *internal evaluation function* should be used instead of the reward function. It is the *adaptive critic* that learns to evaluate the reward signal to form an internal reward signal [Arb03]. The critic estimates total future reward based on the current state and reward. The agent enriches the state and the reward signals by using memory. It learns to predict.

The *Bellman equation*

$$U^{\pi}_{Bell}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

[RN16][Eq. 21.1] expresses the idea that the value of a state is the reward of that state plus future diminished rewards following the current policy π.

An optimal policy π^* is chooses the action with maximum next value.

$$\pi^*(s) = \text{argmax}_{a \in A(s)} \sum s' P(s'|s,a) U(s'). \tag{2.1}$$

[ibid. Eq. 17.4] where P is the probability to change from state s to a new state s' when action a of all available actions in a state $A(s)$ is performed. In the RL literature P is called the model.

Therefore the *Bellman optimality equation* gives the utility for an optimal policy:

$$U^{\pi^*}_{Bell}(s) = R(s) + \max_{a \in A(s)} \sum_{s'} P(s'|s,a) R(s')$$

The discount factor γ weights rewards that arrive sooner as more important than ones that are delayed.

To find the values the Bellman equation is repeatedly called until convergence, which is called the value iteration algorithm [ibid. Eq. 17.6]:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s') \tag{2.2}$$

After a while, this value iteration algorithm converges to the same results as the state-value obtained by using the Bellman equation. The problem with this formulation is that it is required to know all transition probabilities P (the model). By exploring the environment and experiencing the transitions, the model is obtained little by little. The updates can happen incrementally with each new observation. The value based on the transition from s to s' is calculated and then compared to the current value. The difference is partially added to the current value resulting in an approximation of the Bellman equation:

$$U_A(s) \leftarrow U(s) + \underbrace{\alpha(R(s) + \gamma U^{\pi}(s') - U(s))}_{\delta^{TD}} \tag{2.3}$$

[RN16][Eq. 21.3]. Each update step returns a δ^{TD} describing the error between the predicted value and the updated value. It reflects a reward-prediction error (RPE). This procedure is called *temporal-difference* (TD) learning and is performed in the critic module in the presented RL framework. The transition probabilities P are not needed because they are implicitly used. Rare transitions happen less frequently and, accordingly, have a stochastically proportional influence on the state-value. The model is not explicitly stored; therefore, this method is called model-free.

Considering a delayed reward problem again, this algorithm offers a solution to transform delayed reward problems into a non-delay form. The reward function in the example of the pole balancing problem, which always returns one, can now be improved using value iteration. When the pole reaches the losing state, a lower number is returned for this state (penalty). Using the value iteration algorithm until convergence on all the recorded states, we obtain state-value mappings for each recorded state.

After convergence, the quality of the state-value function can be quantified by subtracting the rewards along an optimal policy trajectory from the predicted value.

The higher the quality of the reward function, the lower the difference the approximation has to overcome. This difference is reflected in the TD error δ^{TD}. In a deterministic environment with a fixed policy, the TD error is zero after convergence of the value iteration algorithm. Therefore, the δ^{TD} as a reward signal only reflects changes in the model $P(s'|s,a)$.

In non-neural implementations the policy can be derived from the state-value function. It can be viewed as a potential function. The optimal policy is obtained by taking the maximum of the gradient on this potential field, an optimization method called hill-climbing or gradient ascent.

The distinction in two separate systems shows that representations are learned twice. The critic learns the state-value function, and the actor learns a state-action function.

Action-Value Function If the given reward function is only rating states and not transitions the state-value function will also only rate states. In a deterministic simulation, where the transition probability is $T(s'|s,a) = T(s'|s) = 1$, the action can be derived of the tuple (s,s'). Therefore, the tuple is enough to obtain the *action-value* function $q_\pi : S \times A \to \mathbb{R}$ by using $\Delta u = u(s) - u(s')$.

2.1.1 State-Value Function Implementation in the Critic

Each episode is a collection of tuples containing states, actions, and rewards ([DMH19])

$$D_\pi = [\{s_0, a_0, r_0, ..., s_T, a_T, r_T\}_{1 \leq i \leq n}].$$

By just taking the ordered subset of states $[s_0 ... s_T]$, we get a nonlinear trajectory in state space. The state space is hence a phase space.

With each episode, a new trajectory is obtained. Multiple trajectories can be used for calculating the state-values, building on the idea to view trajectories as samples, where the reward values of the states are then averaged to converge to a final value. This method is called “direct utility estimation” [RN16][Chapter 21.2.1].

The values of the critic are used to train the actor. Therefore we want the critic to learn the state-value function as soon as possible. This is called online learning, as the new data is learned as it arrives. TD-learning just updates the state-value using the following state. Therefore it is called TD(0). The convergence via error propagation in an episode is slow because only one state is updated with each new tuple. By taking a step and updating the last state’s value with the obtained reward, this takes a backward view. Considering the case that subsequent episodes cover the same n states, we need n episodes to propagate the change of the last state to the first state. This observation was also made by Frémaux et al. in their TD-actor and TD-critic architecture [FSG13].

If we get a final reward at the end, this reward can be mapped to the subspace of the trajectory using the Bellman equation and obtaining the utilities for the states in this trajectory. The slow convergence of TD-learning can be improved vastly by updating the states from last to first. This causes faster convergence of the value iteration algorithm. This one instance of n-step TD-learning where $n = \infty$. A terminating state is reached at the end of every episode. This n-step learning described here is therefore optimized for short delayed reward problems with failure states.

From the agent's perspective, it does not matter if this is the end of the episode. The only relevant information is the amount of reward.

Covering the Whole State Space In general, RL problems have a huge state space, so that the training can cover only a fraction. The problem of inducing a state-value function approximation is a supervised learning problem. Known algorithms can solve this inductive learning problem. States in D_π already contain mappings to reward values. States which have not been sampled so far must be inferred using the generalizing of the approximating function utilizing D_π.

We use k-nearest-neighbor (KNN) regression with a weighted distance for the inference.[SB18] The weighted distance is of importance so that the utility slightly differs for similar states. A KNN regression without distance weighting causes a retraction. This leads to cycles in subsequent episodes to be mapped to the same utility values, thus $u(s) = u(s') \Rightarrow \Delta u = 0$. A smooth interpolation would likely return better results, e.g., by natural neighbor interpolation (Sibson interpolation) [Par+06].

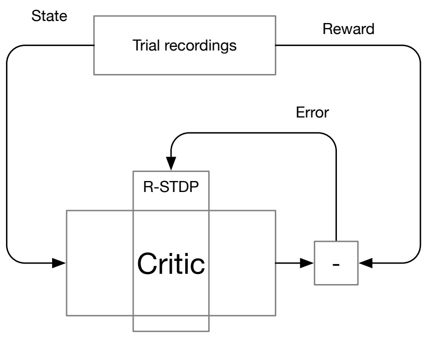

Figure 2.2: A method to train the critic with R-STDP

Other Inference Methods A state value function can be interpolated by training a neural net with supervised learning. We could also train an SNN critic by applying recordings of $s_t ... s_{t+n}$ and outputting the reward after translating spikes to an analog signal (Figure 2.2). The training would be by the same mechanism we propose for the controller. The reward for R-STDP would form a control loop by feeding the difference between the recorded and the output reward back. A unified mechanism would be more biologically plausible. An implementation of a spiking critic approach is found in [FSG13].

2.1.2 Details in the Critic Implementation

The critic is performing state-value function approximating. The observations are continuous, thus creating the need to reduce the state space to a fixed size. We use equidistant tiling (bucketing) as the "coarse coding" strategy. The resolution of the critic is another hyperparameter. The critic resolution has to be higher than the discretization in the actor to learn different stimuli.

It is best to store the observed rewards and their associated values in a hash table, so that reading and inserting is performed in $O(1)$ while having a small memory footprint. The index can be computed from hashing the discretized states.

2.1.3 Neurobiological and Neurophysiological Learning Models

How does this RL model compare to biological learning systems? Sterling and Laughlin discuss the involvement of two systems in motor learning [SL15][p. 87]. One system compares the intended output with the actual pattern (intention learning), while the other system compares the expected payoff with the actual real payoff (reward-prediction learning). Reward-prediction learning enables the Pavlovian conditioning of percepts with a reward [Sch15]. Reward-perception learning is achieved by correlating the event time of the percept with the event time of the obtained reward. Sterling and Laughlin locate the reward-prediction mechanism in the striatum, part of the basal ganglia, and intention learning in the cerebellum. The comparison of the intended with the actual output is implemented using backpropagation. The output of these error calculating brain regions is sent to other brain areas with high spike rates. The biological plausibility of the presented RL framework is confirmed as the model of temporal difference learning matches empirical data in rats [FMD00].

It is believed that the outputs of the error signal systems use neuromodulators. The neuromodulators are distributed via neuromodulatory pathways.

3 Solving the Credit Assignment Problem

The main problem of reinforcement learning with a neural network is solving the spatial credit assignment problem: The question which synapses caused the output. When accounting the time difference between reward and causal neural activity, it is also called the distal reward problem in the behavioral literature[Hul43]. The spatial component asks the question: How can the reward be connected to the activity of neurons in some hidden layers?

Modeling of Learning First, we have a system which returns a reward or rating for some weight vector $\boldsymbol{w}$ describing the current synapses:

$$L(\boldsymbol{w}) : \mathbb{R}^n \to \mathbb{R}$$

Without loss of generality, we restrict us here to a scalar reward. For some problems, an error value is formulated instead, but the error is just the negative reward. In the supervised learning context, this reward function is called a *loss function*. In the RL context, the reward is based on a state. The state is the result of running an actor based on those weights, i.e., synapse parameters.

Training a network means maximizing the return. This can be done with various methods. One method is taking steps in the direction of the gradient. After a while, the weights will settle in a local minimum. This method is called hill-climbing or gradient descent. Formalized:

$$\Delta w = -\eta \nabla_w L \tag{3.1}$$

To use gradient descent means finding a function $f(w) \approx \nabla_w L$.

3.1 Synaptic Plasticity by Backpropagation

Indexing Before we examine multilayer networks, a notation has to be established. We have used indexed notation for the weights with *i* meaning the "from"-neuron, where *j* means the "to"-neuron. In the following the indexing *i,j,k* will be used for the layers of a two-layer feed-forward network.

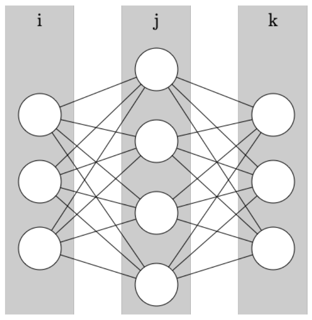

Figure 3.1: Indexing of the layers

Backpropagation The synaptic weights need to be changed (synaptic plasticity) to bring a neural network to a state where the network elicits the wanted behavior. The synapse weight Δw_{ik} can be directly calculated with the backpropagation (BP) algorithm based on the loss. BP utilizes downstream feedback to change upstream information flow.

By applying the chain rule on Equation 3.1 we obtain

$$\Delta w_{ij} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w} \tag{3.2}$$

We define the error signal

$$\delta_j := \frac{\partial L}{\partial z_j}. \tag{3.3}$$

The postsynaptic local reward signal is calculated by forming the sum every weighted connected error signal

$$\delta_j = \sum_k w_{jk} f'(net_k) \tag{3.4}$$

and the neuronal input $net_j = \sum w_{ij} z_i$. This is the backpropagated signal.

In ANNs a neuron's activation is defined as

$$z_j = f(\sum_i z_i w_{ij}) \tag{3.5}$$

in a forward fashion. Therefore

$$\frac{\partial z_j}{\partial w_{ij}} = z_i. \tag{3.6}$$

Combining this with Equation 3.3 we obtain

$$\Delta w_{ij} = z_i \cdot \delta_j \tag{3.7}$$

with the output of a presynaptic neuron z_i.

When looking at SNN, the temporal information in the spike sequences must be addressed. The meaning of a signal can only be found after the whole sequence has been parsed. When a sequence begins and ends is difficult to determine. One strategy is to distribute reward values for a sequence in fixed timesteps.

To backpropagate the error signal, or with an alternative name, the reward per neuron, we need the derivative of the neuron's activation function (c.w. Equation 3.4). The activation function f in traditional McCulloch-Pitts-neurons is set to be a continuously differentiable function. Spiking neurons do not have a differentiable activation function because the derivative is undefined at spike times. Therefore we need other ways to solve the distal reward problem without using the derivative of the activation function.

The brain uses recurrent connections forming recurrent neural nets (RNN). Recurrent nets can also include autapses, where neurons are connected directly to themselves. RNNs can be trained with a variant of the BP algorithm, called backpropagation through time (BPTT) [Bel+19].

Previously algorithms were not biological plausible. E-prop is a biological plausible algorithm on SNN with comparable performance as ANN.[Bel+19] In e-prop

$$\delta_j = \sum_k w_{jk}(y_i - y_i^*)$$

where $(y_i - y_i^*)$ is the deviaton of the output at time t.

Using Firing Rate Instead of Weights When an internal rate code is assumed, the feedback signal is directly connected to an optional activity in a time window.

$$\Delta z_j^* := z_{j,target} - z_{j,current}$$

$$\Delta z_j^* := \delta_j \tag{3.8}$$

Therefore the change in the firing rate to have the optimal value can be determined using the error signal. Should the focus of our works be on modifying the firing rate rather than directly setting the weights? By this method, we ignore high-resolution temporal information. What does it mean to ignore temporal information? Spikes arriving in rapid succession have a bigger effect on the postsynaptic neuron than one spike arriving at the beginning of the cycle and another spike arriving at the end of the cycle. This is a powerful feature of SNN, which is not available in ANN. It is the delays that can be tweaked, instead of synaptic and structural plasticity [DSG17]. The concurrent arrival of spikes at a single neuron can also happen with inhomogeneous spiking when spike transmission delays are different per synapse.

Reinforcing Changes In the training of SNN, our goal is to push the activity to the activity level, which best solves the problem. One may think after changing some activity z_j, the resulting change in the reward signal can be traced and reinforced:

$$\frac{\partial z_j^*}{\partial t_m} := \frac{\partial z_j}{\partial t_{m-1}} \frac{\partial \delta_j}{\partial t_{m-1}} \eta \tag{3.9}$$

where $\eta \in \mathbb{R}^+$ is a scaling factor (learning rate) and t_m a point in time.

The problem with this approach is that a single cause for the activity change can not be determined; it can be caused by a change of weight (Δw) or a change in the state (Δs).

3.1.1 Reward Signal for Deep Networks

The reward signal presented in Equation 3.3 is backpropagated so that the reward signal is calculated per synapse.

One solution for calculation of the reward signal per synapse is offered by Bing et al. [Bin+19a][Eq. 14]. Here, the reward of every dendritic synapse is backpropagated as the reward of the whole downstream neuron by using the average weighted reward per axonic synapse written as

$$r_{i,j} = r_j = \frac{\sum_k |w_{j,k}| r_k}{\sum_k |w_{j,k}|}. \tag{3.10}$$

This downstreamed reward is the same as the reward signal, so $\delta_j := r_j$. The weight change can be calculated with the value of δ_j via Equation 3.7.

The problem with Equation 3.10 is that it implicitly assumes a linear transfer function and, accordingly, a constant derivative when, in reality, it is not linear.

3.1.2 Eligibility Trace by Weight

In addition to considering the reward per synapse, Bing et al. also suggest multiplying a non-temporal "eligibility trace" to each weight change first proposed and defined as

$$g[w_{ij}(t)] = 1 - c_1 e^{-c2|w_{ij}(t)|/w_{max}} \tag{3.11}$$

by Foderaro et al. [FHF10][eq. 7], where c_1 and c_2 are scaling factors. It is inspired by biology so that synapses with higher efficacies produce greater weight changes. Bing et al. formulated it as [Bin+19a][Equation 8]

$$g_{ij}(w) = 1 - c_1 \times w_{ij} \times \exp(-c_2 \times \text{abs}(w_{ij})/w_{max}). \tag{3.12}$$

However, this equation (Equation 3.12) does not quite fulfill its promise: by multiplying with $g_{ij}(w)$, positive weights cause less changes (cf. Figure 3.2). Therefore we argue that the original formulation should be used. To fix this we suggest the formulation

$$g_{ij}(w) = \text{abs}(c_1 w_{ij} \exp(-c_2 \,\text{abs}(w_{ij})/w_{max})). \tag{3.13}$$

An overview by the plots is in Figure 3.2.

The eligibility trace by weight favors some value range of the weights because it makes some ranges attractive by decreasing the weight changes in this range while other ranges are repulsive. Weight changes near a weight of zero get smaller updates so that weight changes may vanish. The offset from the weight axis may be chosen differently.

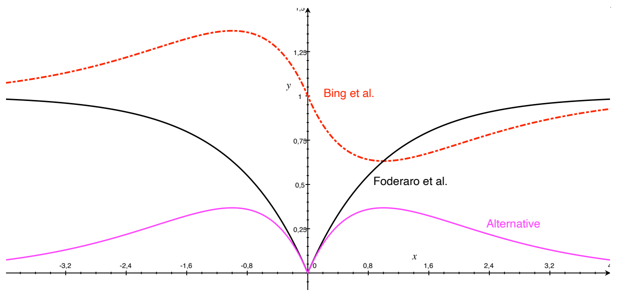

Figure 3.2: Eligibility trace factor. The abscissa shows the weight; the ordinate shows the eligibility factor.

3.2 Using Neuromodulators (R-STDP)

3.2.1 Phenomena in the Brain

It seems that the mammalian brain uses various techniques for solving the issue of credit assignment. The reward is created at a different place and a different time than the time and location of the synaptic firing event that caused the action. Top-down signals through the neurotransmitters dopamine and acetylcholine [Bel+19] and the phenomenon of event-related negativity [PRR13] have been found. Similarly, noradrenaline and serotonin act in a permissive and facilitatory role for the induction of plasticity [FG16].

The feedback mechanisms allow us to partially solve the spatial and temporal credit assignment problem.

3.2.2 R-STDP

A variety of mechanisms are found within different areas of the brain. Hence it is not possible to develop a single unified model. Instead, a mathematical framework can be developed and then tweaked to model different phenomena by changing parameters [FG16].

One basis for this framework could be Hebbian learning. Depending on the correlation of post- and presynaptic firing events synaptic efficacies increase or decrease, commonly referred to as long term potentiation (LTP) resp. long term depression LTD. Formally this two-factor rule is $\langle \dot{w} \rangle \approx H(pre, post)$ where $\langle \cdot \rangle$ denotes expected value[1]. We use the dot notation to designate the derivative of a term with respect to time through this work.

Donald O. Hebb [Heb49] first postulated this two-factor learning rule. One biological implementation was found in spike-timing-depend plasticity, short STDP [BP98]. STDP forms a curve observed in empirical data (Figure 3.3). Buchanan and Mellor found

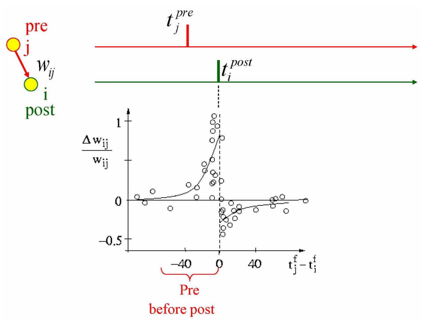

Figure 3.3: From Jesper Sjöström and Wulfram Gerstner (2009), Scholarpedia `http://scholarpedia.org/article/File:STDP-Fig1.JPEG`

a greater variety of STDP curves [BM10]. From the perspective of machine learning, STDP is an unsupervised learning mechanism. Hence, it can only solve a limited class of learning problems.

Frémaux and Gerstner [FG16] suggest a model for tuning STDP into a supervised learning task. The three-factor learning rule is a post-Hebbian theory of learning of the form

$$\langle \dot{w} \rangle = M \times H(pre, post) \tag{3.14}$$

H is an eligibility trace based on the neuron's state in the pre- and postsynaptic neurons. H can be implemented by STDP ($H = \mathbb{E}[\text{STDP}(pre, post)]$). Synapses have a

[1]The origins oft this notation lie in statistical mechanics.

transient memory, called “tags” (experimental literature) or “eligibility” trace (theoretical literature). M is a factor which is implemented in vertebrates in neuromodulators [FG16]. M can be a signal of novelty or success.

The idea of a three-factor learning is slightly older. Dayan and Abbot named this the “three-term covariance rule” in 2001 [DA01][p. 351].

The three factor rule (Equation 3.14) captures the covariance between the three factors.

$$\dot{w} = Cov(H(pre, post), M(a)). \tag{3.15}$$

In the following, we take a closer look at applying this framework for motor learning tasks. The learning methods are still generalizable enough to be used for other tasks.

3.2.3 Neuromodulators for Learning in Vertebrates

A neuron exists in a context that modulates its firing. Neuromodulators are chemicals that affect parts of this context, e.g., the synapses. The amount of the distributed modulator contains temporal information and can thus be described as a signal in theoretical modeling. In models, such as the one presented in this work, the neuromodulatory signal is usually a global signal which is attributed evenly to all neurons. This global reward distribution is also called volume transmission.

Neurotransmitters may be created at different places, e.g., shipped via blood or created at the synapses (e.g., 5-HT is synthesized to tryptophan at the axon terminal) [Har94][p. 168].

Dopamine is a neuromodulator where an effect on LTD and LTP has been observed. The effect of a neuromodulator not only depends on the chemical properties of the neuromodulator but on properties of the target structure [SB18]. There are five known dopamine receptors (D1 - D5). D1 is excitatory, while D2-D5 (put together as D2) are inhibitory. Izhikevich argues that as dopamine via D1 receptors influences LTD and LTP, it should also happen with the LTD and LTP caused by STDP [Izh07]. Therefore, dopamine can be seen as a plasticity modulating signal (M).

The reward value is determined by internal brain activity. In the macaque, dopamine response occurs at the moments of rewards and after stimuli that are predictive of reward [Sch98]. This finding enabled the theory that dopamine encodes a reward-prediction-error.

Reward signals are limited to some brain structures, including midbrain dopamine neurons, striatum, amygdala, and the orbitofrontal cortex [Sch15]. Dopamine is synthesized in dopaminergic cell groups. It originates from the Substantia niagra pars compacta (part of the midbrain) onto the synapses within the dorsal striatum, a nucleus in the basal ganglia [DA01][p. 351]. This dopamine pathway is called the Nigrostriatal pathway. There are hints that the brain uses distal apical dendrites in pyramidal cells located in the neocortex and hippocampus for credit assignment [RL19].

Volume transmission might not be the only mechanism in the brain. Compared to the power of backpropagation, we see that a local distribution component might be missing. If distributed selectively, it contains spatial information. Another place of error signals in the brain is the cerebellum, where fine motor control and motor learning is regulated. Here, the output layer consists of Purkinje cells. Evidence shows that axonal neuronal projections called climbing fibers transport error signals downstream onto Purkinje cells in a one-to-one correspondence [RL19]. The credit assignment onto Purkinje cells is straightforward because the signal is not backpropagated.

Dabney et al. found asymmetric responses for positive and negative reward-prediction-errors in dopamine neurons in mice (Fig. 4 [Dab+20]). Therefore, it is reasonable to scale negative neuromodulatory signals with a different factor than the positive response. They further found diversity in the dopamine cell's reward-prediction responses correlated to the encoded probability of these cells. Thus, using two fixed factors depending on the polarity of the single-dimensional neuromodulatory signal is a simplification of the biological reality.

Concluding, it can be said that neuromodulators are probably used for learning motor planning and motor control.

Multi-Dimensional Critic in Biological Systems So far, the internal evaluation functions have been presented as a scalar value. Most learning methods use a scalar value for the internal critic's signal. However, there is no reason to limit the neuromodulator tensor to the order of zero (i.e., a scalar). Any modification of the neuromodulatory signal must be accompanied by a mechanism that makes use of this signal. Biological systems show evidence that the corresponding mechanism uses multi-dimensional signals by using multiple neuromodulators, e.g., acetylcholine. Acetylcholine is associated with novelty and dopamine with reward; however, the association of transmitter and the meaning can be a nonlinear mixture.

The concept of the dopaminergic reward signal being a single quantity is being questioned as there is evidence that reward signals in the brain are encoding a probability distribution [Dab+20]. In RL experiments, it has been shown that using a probability distribution can boost the agent's performance [BDM17]. These findings hint at a future biologically plausible implementation.

3.2.4 Modeling Neuromodulators

Training establishes the state-value function in the critic. An active learning agent chooses the best policy as specified by Equation 2.1. This results in a greedy behavior.

When the state value function is not yet fully learned, there might be better policies, but they will not be discovered. In contrast to a symbolic hill-climbing controller, a neurocontroller does not always choose the optimal policy. The policy is based on the current weights.

The arrival of a reward in the form of dopamine can be modeled as spiking events or an analog signal. Current R-STDP theory requires M to be a real numbered signal, that is sometimes positive and sometimes negative (e.g., [Mes17][Fig. 4.15]) to strengthen desired weight changes and invert the weights on wrong changes. As we have established in chapter 2, it is more suitable to use an internal reward signal than the direct appliance of the reward.

The use of δ^{TD} (Equation 2.3) is biologically plausible; however, we established that the convergence of the value iteration algorithm is slow and can be improved. By speeding up the learning of the state-value function, the error signal paradoxically vanishes quickly, thus dramatically slowing down the learning of the policy in the controller. The agent's state should have changed after an action has been taken. We can rate this action by comparing it with the optimal policy: We can use the difference between the value of two states as the neuromodulatory signal $\nabla U = \Delta u = U(s) - U(s')$.

Some problems can be solved without constructing the state-value function; these are problems in which the error can be calculated at any time, and a taken action directly influences the error. Examples are found in [Mes17], [Tie+19]. In these problems, the policy derived from the state-value function is already given by the policy based on the reward function.

The hill-climbing strategy is greedy and shows an example of the *exploitation/exploration conflict*. It possible to add a countermeasure to nudge the behavior towards a direction. In brains, the balancing between these poles is influenced by mood experienced in feelings like motivation, curiosity, depression, courage, and anxiety, largely influenced by hormones. The probability distribution for a reward can be calculated based on the distributional reward prediction error theory ([Dab+20]), given the recorded states. By multiplying with the inverse of the probability ($k = 1 - p$) we can get a novelty based reward signal. How such a novelty signal may be used in the neural actor remains open.

3.3 Bias in R-STDP

When using rate code for the output, the activity is proportional to the preceding layer's activity:

$$\mathbb{E}[z_k] = \sum_j \sum_k w_{jk} \mathbb{E}[z_j]$$

where k is an output neuron index and n_j are upstream neurons.

In a feed-forward net STDP is mostly positive (pre before post), thus causing more LTP than LTD, meaning

$$\mathbb{E}[H] > 0. \tag{3.16}$$

Once the agent reaches a terminating state, the episode will be over. In problems, like in this case, the final state means that the agent failed; thus, it must be rated worse than the starting state. This is the failure condition.

$$u(s_0) > u(s_e). \tag{3.17}$$

When the error signal is $\delta = \dot{u}$ the accumulated error signal in a episode till the end of the episode e is

$$\mathbb{E}[\delta] = \int_0^e \dot{u}(s_t)dt = [u(s_t)]_0^e = u(s_e) - u(s_0). \tag{3.18}$$

We also know that a failure state must be penalizing, thus $u(s_e) < 0$. Therefore it follows that taking Equation 3.17 into account, $\mathbb{E}[M] < 0$.

The problem is similar to a Lyapunov function. Unless it stays in a stable spiral, every trajectory will end in the failure state after some time. Therefore, in the limit, the utility will be zero or negative:

$$\lim_{t\to\infty} u(s_t) \leq 0.$$

The expected change on the positive STDP (Equation 3.16) is inverted by the negative biased third factor as $\mathbb{E}[\dot{w}] = \mathbb{E}[H]\mathbb{E}[M] < 0$, thus resulting in LTD until the net dies. In the case of $\delta = \dot{u}$, falling weights are expected behavior.

The Influence of the Starting Position The starting position of the agent defines the bias by defining $u(s_0)$. Adding a penalty for terminating states increases the bias by decreasing $u(s_e)$.

Generally speaking, $u(s_0)$, and hence the bias, is different in each episode because the learning process results in changes of $u(s_0)$. This temporal aspect is omitted in the notation above.

Experimental evidence from simulation runs confirms the hypothesis and shows slight variations in the bias across episodes because of changed starting value. The Figure 3.4 depicts that after the first episode the starting value has changed. It also shows that for subsequent episodes the state-value function returns higher values in the starting position than the failure condition.

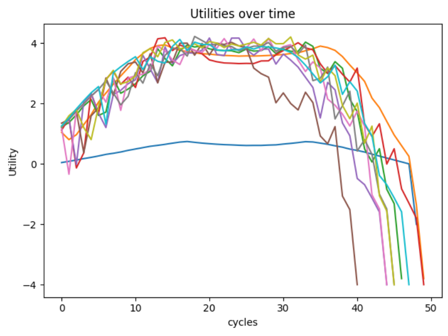

Figure 3.4: Utilities of many subsequent episodes plotted over time.

3.3.1 Centering the Neuromodulatory Signal

Ideally we want to use a neuromodulatory signal to be centered (i.e., $\mathbb{E}[M] := 0$). From the Equation 3.18 it follows

$$\Rightarrow u(s_e) - u(s_0) = 0 \Leftrightarrow u(s_e) = u(s_0) \tag{3.19}$$

The centered equation Equation 3.19 violates the failure condition (Equation 3.17). Thus, this implementation of the neuromodulatory signal will always add a negative bias.

The option we have is to add a bias term to M: $\mathbb{E}[M] = u(s_e) - u(s_0) + b$. The bias is known after the environment initialization and $u(s_0)$ is obtained. The designer of the environment usually knowns $u(s_e) = r(s_e)$.

We suggest three possible countermeasures to center the signal:

- Add a baseline to push the default reward.
- Accumulate rewards and clamp. At the limit, only allow rewards in the other direction. Use value contingent across episodes.
- Use different scaling depending on the polarity of the neuromodulatory signal (not covered in this work).

Static Baseline A bias term on the neuromodulatory signal pushes the learning into unsupervised learning by using STDP to learn correlations between neural activity and reward. The learning is shifted towards unsupervised learning by adding a bias as $\mathrm{Cov}(w, H)$ is increased. [FG16]

Dynamic Baseline Wunderlich et al. [Wun+19] trained an SNN controller using the deviance of the current reward from the task-specific expected reward. A task, indexed by k, is here defined as agent's states in which a different action has to be taken, which is usually every state. States are surjectively mapped to place cells. The task-specific expected reward is updated as an exponentially weighted moving average:

$$\overline{R}_k \leftarrow \overline{R}_k + \gamma(R - \overline{R}_k) \tag{3.20}$$

This approximation gives a baseline to compare against. The neuromodulatory signal is then

$$M = R - \overline{R}_k \tag{3.21}$$

$\overline{R}_k$ must only contain the reward of a single stimulus/response association. Each task has a different mean reward, which will be integrated into the baseline. If conditioned on multiple stimuli/response associations, it will contain a bias. [FSG10]

When the agent reaches a state that it has not seen before, $\overline{R}_k$ is uninitialized. $\overline{R}_k \leftarrow R$ can be used for initial values. This has the effect that the first time the critic sees the state, the error signal is zero and ignored. In order to learn, the states must be visited multiple times. When the resolution of the critic is high, fewer collisions happen. It hence takes a longer time to learn. Therefore, it is beneficial to initialize a value using some inference method like multivariate natural neighbors or k-nearest-neighbors.

By using a baseline method the neuromodulatory signal will be a lot smaller. Combined in one update cycle at time t Equation 3.20 is inserted in Equation 3.21 results in

$$(1 - \gamma)(R_t - \overline{R}_{k,t}).$$

Another question is whether the baseline should be updated before setting M or afterward. The difference in the time of the update lies only in the scale of the signal.

Theorem 1 The method of a dynamic baseline requires the reward function r to be dependent on the tuple of (s, a) and cannot be based solely on s.

Proof. Let B be a set of states reduced to a shared bucketed state. Depending on B the shifted signal M is limited in value:

$$\sup M_B = \max_{s \in B}(R(s) - \mathbb{E}[R_B])$$

By increasing the resolution, the $\sup M_B$ goes to 0 as the size of the set B shrinks. With that, the range of possible reward for one set B also shrinks: $\lim_{res \to \infty} R_B - \mathbb{E}[R_B] \to 0$.

Higher variance than by using R is obtained by using $q : S \times A \to \mathbb{R}$ of Equation 3.21

$$M := q(s, a) - \mathbb{E}[\{\forall a \in A : q(s, a)\}] \tag{3.22}$$

By considering the action, the supremum gets a lower limit. Reducing the bucket size still keeps the actions.

$$\lim_{res \to \infty} \sup M_B \nrightarrow 0 \tag{3.23}$$

$\square$

This exposes a significant problem. The dynamic baseline can not resolve the bias, as it requires multiple actions per bucket, which violates the requirement of one task per bucket.

Implementation As established in subsection 3.2.4, we do not want to use the reward for the neuromodulatory signal but use the state-value function. As just shown, it is better to use the state-action function for the centering. The state-value function can be obtained by using Δu_k. Thus, we replace R with Δu in Equation 3.21 resulting in

$$M = \Delta u - \overline{\Delta u_k}.$$

3.4 Temporal Credit Assignment Problem

So far, we have addressed the spatial part of the credit assignment issue. Now, we take a look at the temporal credit assignment.

A large delay can occur between the neural activity at the point of decision and the subsequent reward. The mammalian brain learns better when a reward follows the action closely [Hul43]. The problem of temporal credit assignment can only be "solved" in a stochastic sense. A neuromodulatory signal is periodically given and thereby reducing the time between the firing and the feedback. The reward signal's timing compared to LTP is relevant and could explain the temporal eligibility trace. Dopamine agonists have only an effect on LTP in a short time window [Izh07].This requires that the problem is turned to a non-delayed form by utilizing a critic.

Eligibility traces fulfill an important role in connecting the reward with the causing neural activity. STDP increases the eligibility trace, while the trace is continuously lowered over time. An exception is found in the works of Wunderlich et al.[Wun+19] They use non-decaying eligibility traces.

4 Architectural Considerations and the Role of Neural Topology

Not only the synaptic weight shapes the computation of a net, but how the information flows through the net. Which neurons are connected to which? How is information encoded? How can the net change adapt to its required structure?

4.1 Effects of Synaptic Structure

Learning Difficulties with R-STDP and Symmetry R-STDP is limited in cases of excitatory symmetric wiring. There are subgraphs where symmetrical weights from two different neurons cannot be transformed into an asymmetrical graph by just relying on R-STDP. Our performed experiments show that the weights go up and down together, not breaking symmetry. Examples of these graphs are displayed in Figure 4.1.

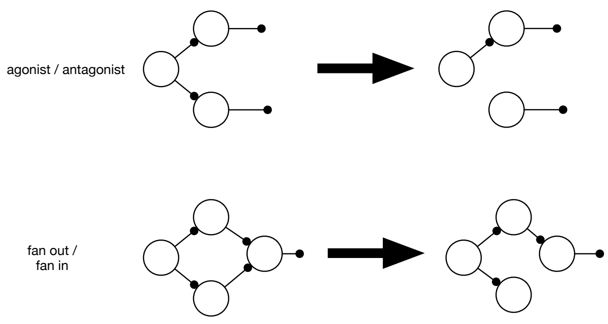

Figure 4.1: Cases in which symmetry breaking is not possible with plain R-STDP

By definition, an agent can only perform one action at a time. Therefore the action is based on the maximum activity, and the agent ignores every other action calculation. The chosen action causes a neuromodulatory feedback signal M. The three-factor rule can now help to explain this behavior. The neuromodulatory signal, one of the three factors, is the same for every synapse. There is neural activity; therefore, H, comprised of the other two factors, has the same sign. This results in the same direction of the weight change for every synapse in this subgraph:

$$\forall i,j \in \mathbb{N} : sgn(Cov(H_i, M)) = sgn(Cov(H_j, M))$$

with indexed H per synapse.

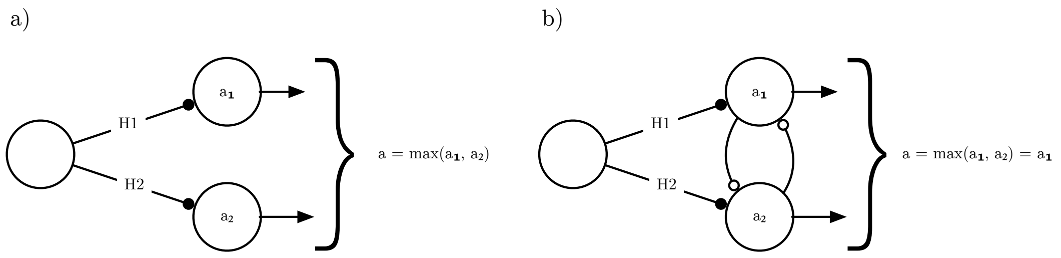

Figure 4.2: **A** H1 and H2 stay symmetric. **B** Lateral inhibition (empty arrows) can be used to break the symmetry in learning.

Example Take, for example, the case of symmetric connections, as pictured in Figure 4.2 a), M is derived from the reward of all the output neurons $R(a)$ and it can be split into two parts with $M_1 \propto R(a_1)$ and $M_2 \propto R(a_2)$:

$$M = M_1 + M_2$$

When action one is chosen over action two because of the higher firing rate, the neuromodulator split is $M = M_1$ and $M_2 = 0$. Therefore $\dot{w}_1 = M_1 H_1$ and $\dot{w}_2 = M_1 H_2$. In this configuration, both Hs are positive, so cause the same sign in the direction of the change in weight. Although the exact value of a_2 had no effect on the result and $Cov(H_2, R(a_1)) = 0$ the weight w_2 gets changed, caused by another neurons activity.

What we see here is a neural level example of (post-)Hebbian learning that causes the fallacy of post hoc ergo propter hoc[1]. R-STDP encodes the covariance of neural activity and the action and not the true causal relation. Although a cell is not connected to the reward, firing events of a cell may return a covariance, which is then enforced.

4.1.1 Symmetry Breaking with Lateral Inhibition

In a comparable topology, Fremaux et al. add lateral inhibition connections in the output layer [FSG13] and note that it is needed for training success (depicted in Figure 4.2 b)).

[1]'Post hoc ergo propter hoc' is a logical fallacy where causality is assumed because two events happen after each other.

The effect of this inhibition is that it suppresses weaker postsynaptic neuron's activity, and with that, the future influence on the reward. The strength in the weight change across synapses is different and should be maximized.

The lateral inhibition layer should have constant synapses so that the lateral inhibition effect can not disappear during training.

Theorem 2 Lateral connections break the symmetry of R-STDP learning.

Proof. Parallel wired lateral inhibition has the effect of a softmax-layer (winner-takes-all behavior) and reduces the STDP on the synapses with the smaller effect on the output ($H_2 \to 0$). The inhibition hence reduces the impact of one factor of the three-factor learning rule. $\square$

Symmetry Breaking with Structural Plasticity One way of preventing the symmetry lock is by adding structural plasticity, e.g., by a random reformation of synapses. Reformations in connections introduce nonlinearities in weight updates. During training, both connections might get removed, but by chance, only one connection gets reinstated. Then, the symmetry is broken, opening the way to move out of this local minimum.

Spüler et al. [SNR15] successfully utilized synaptic plasticity for network training by reconnecting synapses with a weight factor below 0.2 (initial 1, maximum 5) to a new neuron. Following, we suggested a method where the removal of synapses can happen at higher weight values.

4.1.2 Changing the Activity with Structural Plasticity

A method of utilizing structural plasticity was presented by Butz et al. [BO13]. Homeostasis is used to keep the neural activity at a set-point. This idea was continued by Diaz-Pier et al. [Dia+16]. They showed that it is possible to modify structural plasticity to obtain a target activity. Structural plasticity is used to find a connectome that elicits a given (in vivo recorded) spiking signal. The idea is based on the set-point hypothesis, where the system will settle in homeostasis at a target firing-rate. The algorithm was implemented using the NEST library.

To our knowledge, this method has not been tested on dynamic input, for instance, with a supervised learning environment. Here, we want to adapt this method to work with reinforcement learning.

The current activity z should be shifted in the direction of the target. The activity z is not a parameter, so the algorithm cannot set it directly. One option is to change the strength of existing weights, as already discussed. Another option is to indirectly set

z by adding or removing synapses during simulation time, called structural synaptic plasticity.

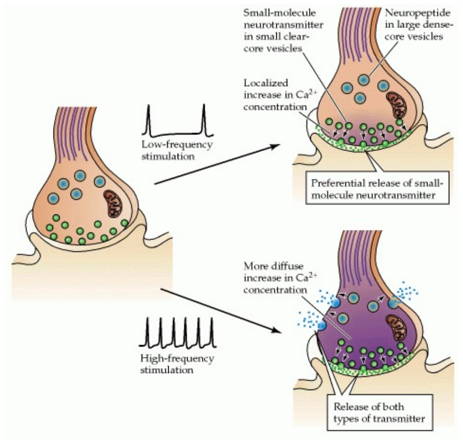
Small-molecule neurotransmitter in small clear-core vesicles
Neuropeptide in large dense-core vesicles
Localized increase in Ca2+ concentration
Low-frequency stimulation
Preferential release of small-molecule neurotransmitter
More diffuse increase in Ca2+ concentration
High-frequency stimulation
Release of both types of transmitter

Figure 4.3: Taken from: Neurons Often Release More Than One Transmitter Neuroscience. 2nd edition. Purves D, Augustine GJ, Fitzpatrick D, et al., editors. Sunderland (MA): Sinauer Associates; 2001.

Higher frequency firing increases the intracellular calcium (Ca^{2+}) concentration (Figure 4.3), an indicator of average activity. This calcium concentration is then linked to the number of synaptic elements (dendritic spines and axonal boutons, i.e., connections), meaning higher concentrations allow more synaptic elements. The number of the connections again changes the activity and, as a result, the future calcium concentration because more excitatory connections mean higher activity. Activity over a threshold will decrease the calcium concentration. After a while, the system's activity level settles in an equilibrium $z = \epsilon$. The system behavior can be described in a simplified form when the calcium concentration is replaced with the activity variable z. The phase portrait of the resulting growth curve is chosen to match the one qualitatively shown below in Figure 4.4. The system has two equilibria: one unstable lower one, η, and the stable ϵ. Activity under the threshold activity of the repeller η causes the neuron to die.

Setting ϵ to the target activity in a supervised learning setting makes it possible to let the system settle into a connectivity with the target activity level; therefore, a desired stable z^* results from the modification of the curve by moving the equilibrium point ϵ with the target activity z^* as $\frac{d\epsilon}{dt} = \frac{dz^*}{dt} = \Delta z^*$ (Equation 3.8). Because the activity is set per neuron, it also requires a local error signal. Thus, it only works in shallow networks or requires the implementation of some form of backpropagation.

While this method should manage the structural plasticity, the synaptic plasticity can

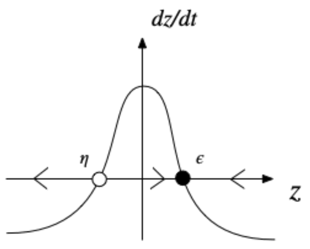

Figure 4.4: Sketch of phase portrait showing changes in the activity by using structural plasticity.

be added by combining it with R-STDP.

Because this proposed method implements structural plasticity based on a reward signal, we call this *reward based structural plasticity*.

Discretizing Network Connectivity Updates So far, we have described dz/dt as a continuous value; however, synaptic elements are discrete. A single synapse can have a big impact on the firing rate.

While changes caused by R-STDP are applied every time a neuron fires, the changes in connectivity happen only some time because of the computational cost and biological realistic growth rate (one element per second according to Diaz-Pier et al. [Dia+16]). The value dz/dt is obtained by the presented curve based on the current activity z. The result is added each update of the spiking network (here every ms) to the number of synaptic elements. How the current activity of a neuron z is obtained is examined in subsection 4.3.2. The number of elements is rounded and discretized for the connectivity update. The axons' assignment to dendrites is made randomly, ignoring the biological characteristics of axonal spines and dendritic boutons.

Choosing Synapses When one synapse should be removed or added, how is it decided which one out of many to choose? One method is the completely random choice. A more sophisticated approach is to consider the spatial arrangement of cells when forming new connections. Using closer neighbors incites the formation of local clusters of information processing. Butz and van Ooyen [BO13] use a Gaussian kernel to determine the likelihood. The benefit of using spatial data is that this could be extended to refrain from using a global algorithm. Using local rules can benefit the use of locality in neuromorphic implementations.

Possible Downsides The suggested method of structural plasticity as well as R-STDP both act on z, so that the combined changes might be too big in the moment of rewiring.

4.1.3 Structural Plasticity for Topology Discovery

Some problems require a distinct topology to solve a problem. Yet, at the beginning of training, there is no knowledge available about the required topology. Therefore, at a starting point, a fully forward connected structure (feed-forward) can be assumed. This fully connected structure must be able to change to any connectome. This is only possible by removing and adding synapses. An example is given in Figure 4.5.

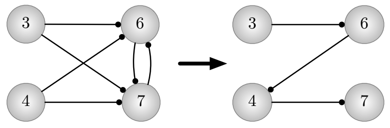

Figure 4.5: The final algorithm must support changes in topology from fully connected to reduced connectomes.

4.1.4 Trimming for Faster Computation

It is possible to just rely on synaptic plasticity while calculating the weights with R-STDP. When a weight of a synapse is zero, no electric potential is propagated, and the effect is the same as the removal of the synapse. When weights can be set to zero and are not removed, the graph stays fully connected, which results in high computational costs. It is consequently better to trim low weights. Once a weight reaches a lower threshold, it causes a neuron to die. The synapse cannot grow again because growing with R-STDP needs neuronal activity.

4.2 Weight Initialization

In ANN training weights are usually chosen uniformly from positive and negative numbers.

R-STDP only modulates the synaptic efficacies when neural spiking occurs. Therefore, the simulation must be initialized with high weights, such that the motor neurons fire in the initial setup. No postsynaptic firing occurs when many weights are too low, which can cause a net to die. This can happen in simulations when the weight changes are too large, i.e., the modulatory signal is too big. The introduction of inhibitory neurons increases the likelihood that this happens because the introduction of inhibitory spikes can cause sudden changes to end the downstream activity.

This initialization strategy matches observations that more excitatory neurons than inhibitory neurons are found in the mammalian brain. 84-92% of the connected neurons along pyramidal dendrites are excitatory [VN16].

In ANN, the excitatory or inhibitory character is determined at the synapse by the sign of the weight; in contrast, it depends on the whole neuron in the mammalian brain. Neurons generally release multiple neurotransmitters within the same presynaptic terminals. They are typically either excitatory by releasing glutamate or inhibitory by releasing GABA [VN16]. However, neurons in the retina represent one known exception [Mil88].

4.3 Encoding of Inputs and Outputs

While the environment and the critic use real numbered value signals, a neural network uses spikes. Therefore the signals have to be transcoded. Building on the RL framework presented in Figure 2.1, we obtain some building blocks, as displayed in Figure 4.6.

The observation must be encoded in a neural representation using spikes to make use of a spiking neural actor. In biological systems, the encoding of both inputs can happen via *firing rate* (rate code) or using *place fields* (labeled-line code in the cochlea). A place field is connected to a corresponding *place cell*, which fires when a position in space is reached. Those encoding mechanisms are copied for simulated networks.

Information via Firing Rate When the information is transmitted via the firing rate, it must happen in the range between no firing and the maximum firing rate. A negative firing rate is not possible. When utilizing rate code and the values to be encoded can be positive and negative, there are two ways to encode this information. Two neurons per input dimension are needed to either set the 'positive' or' negative' neuron to fire, depending on the sign of the encoded value. Another option is to add a bias value in the transcoding process so that less firing than a rate is interpreted as a negative value. The translation of an analog signal to a spiking firing rate happens via spike generators. Generators can generate sinusoidal spikes or Poisson distributed. Sinusoidal generators may create phase-locked signal propagation, while Poisson distributed spikes even repeated correlations out.

Information via Place Fields The connection between place fields and hippocampal place neurons is learned. In this simulation, the value ranges of the state space are known beforehand, so the fields for the cells are established before training.

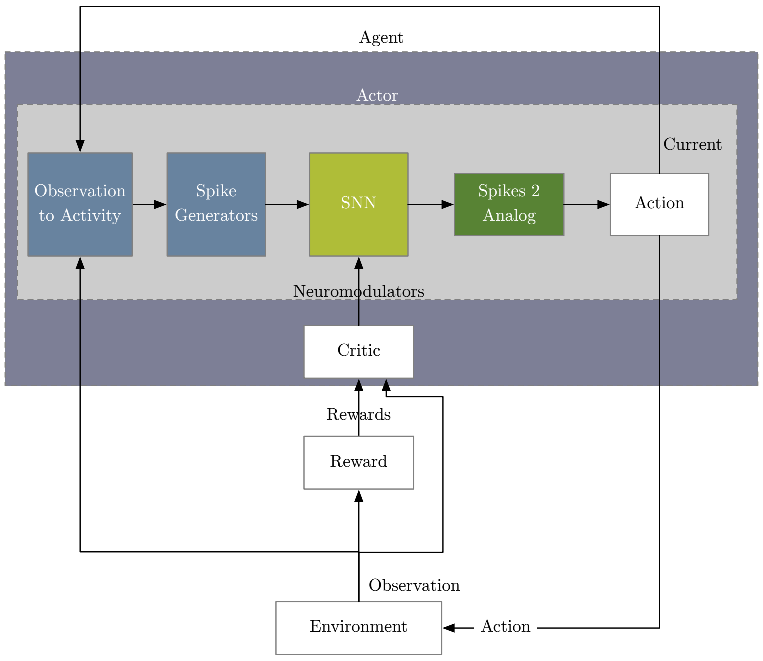

Figure 4.6: Reinforcement learning encoding and decoding pipeline. Each box represents a function. First, the activation per neuron is calculated. It is then encoded in a spike signal. The spiking signal is then for readout translated back to an analog signal. The activity modifies the internal representation, which can be used again as an observation.

Most related works cover the input space in all dimensions equidistantly, thus resemble the functionality of biological *grid cells* found in the dorsocaudal medial entorhinal cortex (dMEC). Natural grids cells cover space in hexagonal shapes (equilateral triangles). In this simulation, the cells are located in the input layer, where analog signals are translated into their spiking neuronal representations. The comparison with receptive fields is suitable. Each neuron (or population) corresponds to a discrete receptive field. Hafting et al. found that the size of the place fields in the dMEC remains the same when the surrounding space is increased. The spacing and field size increases from dorsal to ventral dMEC. A place neuron can not only respond to a single place field but have multiple place fields. [Haf+05]

Additional cells with specialized functions, which are currently not used in the simulation, are border cells and head direction cells[Luo15][p. 444ff]. In the engineered system, the head direction is typically encoded in the output layer because it guides locomotion.

A continuous mapping from receptive fields to stimulus is found in the cochlea. The place principle of hearing states of a spatial position of neuronal activity in the cochlea to map to a frequency. Neurons form continuous neuronal fields along an axis[GK02][p. 242ff]. In their model, the receptive field is determined by the activity of the neurons in an interval along an axis.

Input Space to Place Cell Activation The place fields activation is a coarse coding strategy similar to the bucketing in the critic. We utilize hard limits for the critic's bucketing, so each input has only one corresponding bucket. However, the place fields are smooth and allow activations of multiple cells at the same time. The activations have a center with maximum activation with decreasing activations from this center. Broad and fuzzy fields allow generalization across local neighborhoods. Widely separated points in input space remain independent.

The equation giving the activation for a place cell are based on [FSG13][eq. 22] is

$$f_i(\boldsymbol{x}(t)) := \exp\left(-\frac{\|\boldsymbol{x}(t)-\boldsymbol{x}_i\|^2}{\sigma_{PC}^2}\right) \tag{4.1}$$

where $\boldsymbol{x}_i$ are the center positions of the indexed place cells and σ_{PC} the distance between the cells.

When configuring the place cells, a place field's center position for each cell must be given. Therefore the borders of the state space have to be known for configuration. The discrete place cell encoding reserves a neuron for a part of the input space across all dimensions. The number of receptive fields decides the resolution of the encoding. Without some dimensionality reduction method, the place cell encoding suffers from the *curse of dimensionality*: The volume grows exponentially with the number of dimensions. We take a look at one approach to better cover the state space in section 4.5

When the distances per dimension are different, the scaling must be performed first before taking the norm, thus

$$f_i(\boldsymbol{x}(t)) := \exp\left(-\left\|\frac{\boldsymbol{x}(t)-\boldsymbol{x}_i}{\boldsymbol{\sigma}_{PC}}\right\|^2\right). \tag{4.2}$$

In this case $\boldsymbol{\sigma}_{PC}$ is a vector.

When the place cells are spread with inhomogeneous distances, the size of the receptive fields is unknown. The position of neighbors determine should determine the size: Sparsely covered areas should have bigger, while dense clusters should cause smaller receptive fields.

The place field size can be adaptively determined using the average distance of the k-nearest neighbors. $\sigma_{j,PC} := \frac{n}{\sum_j \|x(t)-x_j\|}$. By running experiments, we found this to have no beneficial influence on the training performance.

4.3.1 Output Encoding

The output neurons encode the actions which should be taken next. Locomotion always happens relative to the current position of the body in space; in contrast, absolute encoding is conceptually closer to higher-level planning. The action can either be relative to the current state or absolute by representing the goal state.

A neuron population is a group of neurons sharing functionality. A population vector was identified in primates, which encodes the directions in different populations where the influence of each direction is determined by the firing rate of the corresponding population [Luo15][Chapter 8]. A one-dimensional joint movement can be implemented with two populations (flexor and extensor). As with the input encoding, a labeled line code (place cells) can encode the absolute target position – Each neuron represents each target position [Wun+19]. A controller put after the network output then translates this absolute position into the relative action by utilizing knowledge about the current location. One can imagine that this translation can also happen with neural circuits.

4.3.2 From Spikes to Actions

The information in the spikes of the last layer is translated into an action. Each neuron's activity converts to an analog signal. The trivial approach for spikes-to-analog transcoding is to use the number of spikes in each cycle, thus assuming a rate code and weighting each spike evenly. The activity is then calculated as

$$z_n = \sum_{t_s \in \mathcal{F}} \int_{t_c}^{t_e} \delta_D(t_s - t)dt$$

where δ_D is the Dirac delta function, $\mathcal{F}$ the set of spike times, and t_c, t_e cycle start and end times. The number of spikes then translates linearly to the activity. Later in this chapter, we will look at a more sophisticated approach of converting spikes to an analog signal by filtering but first will take a look at the translation to an action.

What is the correspondence between an action signal and the impact it has on the actor? Some problems are binary, where the maximum action is performed, while others are weighted by the activity. The action signal consists of multiple weighted actions. The influence on the action of each action signal is $\boldsymbol{\theta}$. It is another set of hyperparameters that needs to be specified by trial and error by the system designer, i.e., with a parameter sweep for each scaling dependent action. For a rate coded signal, the maximum firing rate of a neuron must be tweaked to match the desired target signal amplitude. However, the firing rate of a single neuron is limited. When using rate coding, higher firing rates can be achieved by observing a whole population instead of a single neuron (Figure 4.7). Thus, decoded as an analog signal, a wider numerical range is possible. Increasing the firing rates with a population, the influence of the hyperparameter $\boldsymbol{\theta}$ vanishes, as the system can include the amplification before transcoding in the training.

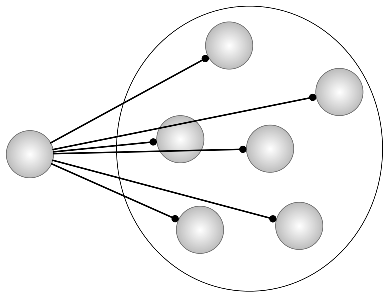
Figure 4.7: A single neuron is connected to a population resulting in amplification.

Using a population can have another beneficial effect. When a single neuron encodes the information of the extensor of flexor, the significance of a single spike is too big: the difference between two or three spikes in a cycle leaves not much room for fine-tuning. Because the relationship $\inf \Delta z_t = \frac{z_{max}}{\Delta t}$ determines the smallest step size in activity levels, an increased maximum firing rate (z_{max}) allows decoding of more possible steps per time window ($\inf \Delta z_t$) or a shorter time window (Δt). When viewing the activity as probabilistic events adding more neurons to a population decreases the variance of the spikes.

Filtering By counting the number of equally weighted spikes per cycle, only one value is obtainer per cycle. A continuous signal has the benefit of enabling the sampling at higher speed and at any point in time; therefore, it is possible to use different asynchronous cycle times in writing and reading. For instance, a slow sensor might be limited at a sampling rate of 10 Hz while the actor could be updated at 100 Hz.

Fremaux et al. suggests the use of filtering by convolution with a kernel ($\kappa(t)$) to translate a spike signal $f(t)$ to a continuous signal [FSG13]:

$$f(t) * \kappa(t) = \int f(s)\kappa(t-s)ds$$

The kernel is defined as

$$\kappa(t) = \begin{cases} \frac{\exp(\frac{-t}{\tau}) - \exp(\frac{-t}{\upsilon})}{\tau - \upsilon} & t \geq 0 \\ 0 & t < 0 \end{cases}$$

To make the kernel causal ,i.e., "$\kappa(s) \sim 0, \forall s < 0$", clamping of negative values is added, as suggested by Fremaux et al.

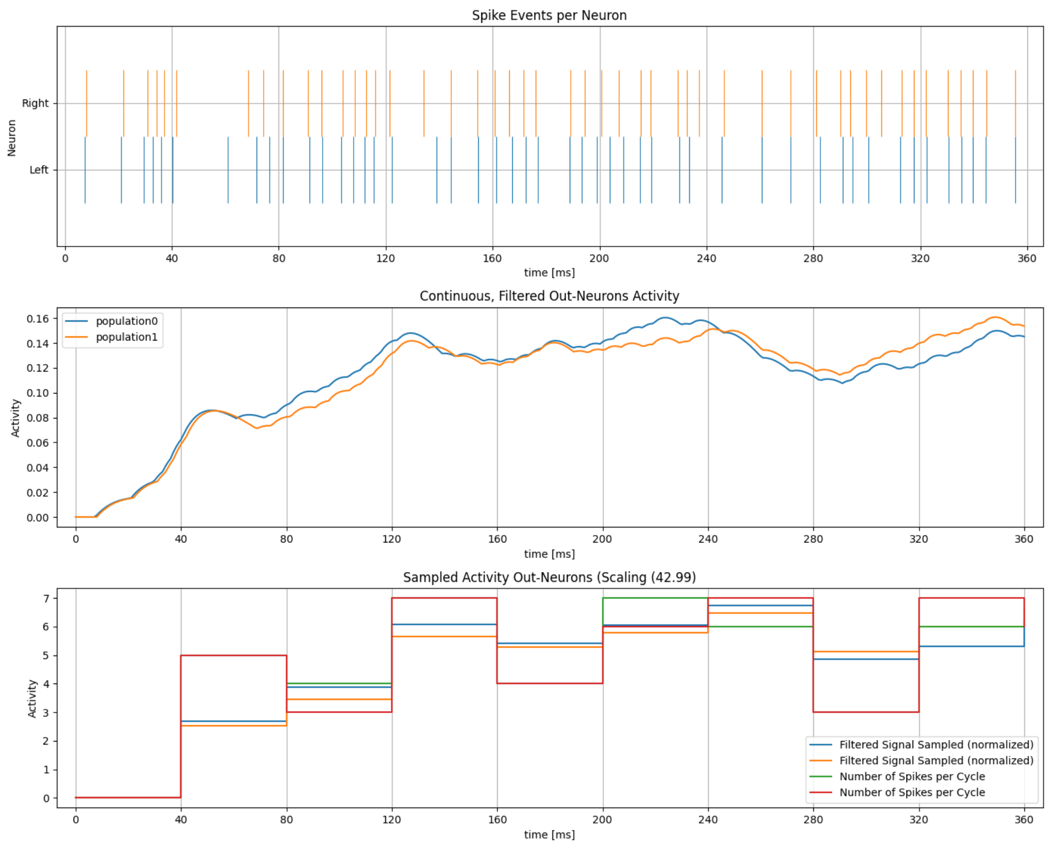

Figure 4.8: **A** Many neurons spike, but only distinct neurons are read, here showing extensor/flexor neurons. **B** The filtered signal is continuous and could be read at any time; however, we only do this at the end of every cycle. **C** The rate coded signal can be read out by integrating the spikes or sampling the filtering signal. When scaled using the least sum of squares, differences in the signals are visible.

The scale of a_e and a_f is much smaller when the signal f is filtered compared to the method of integrating the number of spikes in a cycle; thus, the sensitivity θ must be chosen differently. This makes comparing training runs with the unfiltered signal with the filtered signal difficult. We used the sum of least squares to find the best scale for equalizing the signal's amplitude. This reveals slight differences (Figure 4.8).

Filtering has a disadvantage compared to simply counting spikes. The simulation runs slower when the filtering is applied as every spike is exponentiated.

Now we know that the input and output encoding from an analog signal need signal transformation and meaningful neuron choice. But what happens in between?

4.3.3 The Framework as an Embedded Device

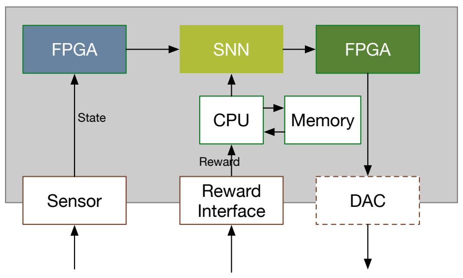

Figure 4.9: Agent brain on an embedded device. The FPGAs calculate functions which are currently not designed using spikes. The DAC is an optional digital-to-analog converter to actuate motors via voltage.

The agent of this framework can be constructed as an embedded device. A model shows the structure in Figure 4.9. Because the rate of spikes flowing though the system might be too high, Field-Programmable Gate Arrays (FPGAs) might be used instead of CPUs. Because the critic is learning over time, it needs reading and writing access to memory.

4.4 Analysis of the Place Cell Architecture

By using place cell encoding with two lateral inhibitory connected output neurons/populations, all components for an feed-forward information processing net are given. The net is depicted in Figure 4.10. Generators create the spikes, which are repeated by parrot neurons. The distinction in generators and neurons comes from used library NEST.

Extensions Beyond Feed-Forward Extension to the computing capabilities to extend beyond feed-forward capabilities. If connections to upstream layers are allowed, it

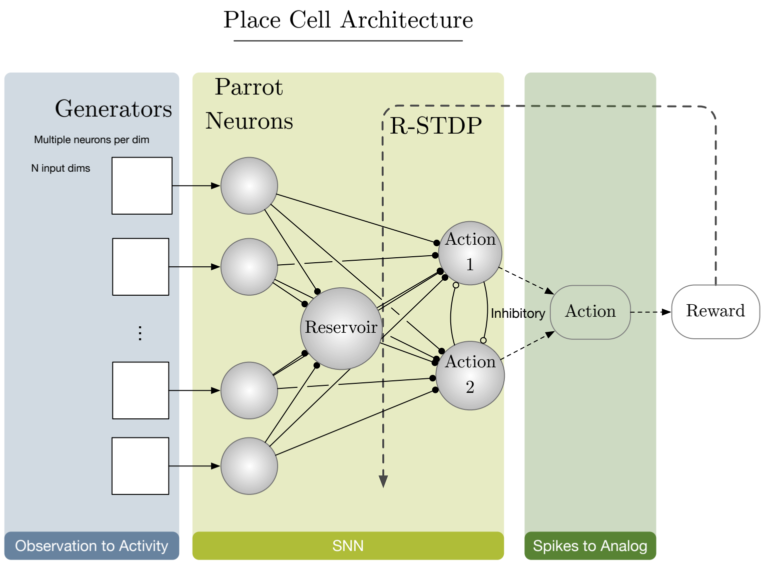

Figure 4.10: The resulting place cell architecture. Multiple free neurons can be added, thus enabling recurrence.

is called "recurrent". Recurrence can be added by adding a reservoir of fully and recurrently connected neurons. Recurrent connections inside a layer allow some form of short term memory [WB19]. Using recurrence allows new forms of computing as "reservoir computing". Here a reservoir is stimulated, and then read out [MNM02]. This has found commercial applications when implemented on neuromorphic hardware [QIE18].

The place cell architecture relies on rate codes and can be modeled without spikes as matrix multiplication. The policy $\pi : S \to A^n$ to obtain an action vector created by the neural network could consequently be completely modeled with a symbolic calculation:

$$\pi(\boldsymbol{o}_t) := f(\boldsymbol{o}_t^T W), \tag{4.3}$$

where $\boldsymbol{o}$ is the rate encoded observed state, f a transfer function, and W are the weights to be trained by R-STDP in matrix form. Matrix W is also called "action matrix" [DA01][p. 351]. The transfer function is approximately linear when using rate cote, as shown by simulated data in Figure 4.11. Therefore, it can be removed from the equation.

Related implementations of this architecture are found are found in the works of [Bin+19b], [Mes17], [Tie+19].

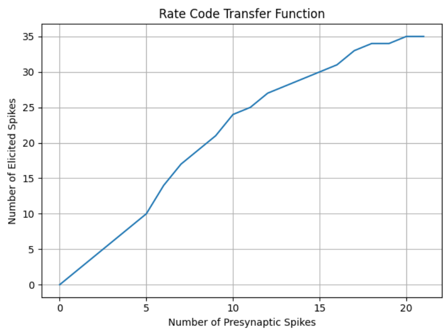

Figure 4.11: The number of incoming spikes is scaled linearly.

Learning Rule Reduction Hypothesis Learning in place cell input encoded net with lateral inhibition in the last layer can be simplified to the learning of the most active neuron by the learning rule $\dot{w}^\star \propto M$.

Explanation During one input frame, the input $\boldsymbol{o}_t$ is constant. The place cell encoding causes one neuron in the input layer N to be significantly activated more than others. This neuron is noted with a star. The cell is connected to an output neuron. The output neuron will spike proportional to the activity in the input layer. Although a postsynaptic spike can be elicited by any neuron, in this frame, most postsynaptic spikes are caused by the most activated neuron. Therefore, the eligibility trace for this neuron is $\mathbb{E}[H^\star] > 0$. This guarantee is not given for the less active neurons, however their influence is proportional smaller $|E[H^{N \setminus \star}]| \ll |E[H^\star]|$. For these reasons, H can be removed from the learning equation, and the algorithm collapses to $\dot{w}^\star \propto M$.

Lookup Table Implementation When only two opposing actions are available, the Markov-decision process Equation 4.3 can be modeled with a lookup table or an array. The array replaces the spiking-encoded action with a rate-based abstracted real-valued table entry, combining the opposing actions.

Following the Learning Rule Reduction Hypothesis, the training happens by modifying the tabular entries after obtaining the error signal using an exponentially weighted moving average:

$$z_i \leftarrow z_i + \underbrace{\operatorname{sgn}(z_i)\delta_i\mu}_{\Delta z^*} \tag{4.4}$$

with learning rate μ. The sign function corresponds to the maximum operation on the two output neurons.

Although using absolute encoding in three-layer SNN, we also found this learning strategy applied in the work of Clawson et al. [Cla+16]. An agent was trained by setting $\Delta w = \mu \cdot r(s)$.

When applying this learning rule the combined algorithm results:

```
s ← inital state;
W ← inital weight;
while training do
    (index, action) ← $getAction(s)$;
    s ← $act(s, action)$;
    $\delta$ ← $getErrorSignal(s)$;
    W[index] ← W[index]+ sgn($a$)$\delta$·learningrate;
end
```

This algorithm allows problems to be examined at a corresponding architecture with multiple orders of magnitude faster training time.

Topological Constraint Typically a feed-forward architecture has no topological constraint on the mapping from the observed state to the considered input cells or the corresponding weights. The place cell encoding requires a topological constraint on the mapping: For two states, if the distance in state space is big, the overlap in neighbored cell activation should be minimal, and if the distance is small, the overlap should be bigger. This additional constraint on the perceptron is implemented in the "Cerebellar Model Articulation Controller" (CMAC)[Alb75]. The strategy of partitioning state in receptive fields is also called "tile coding" or "coarse coding" [SB18][Chapter 9.5.4].

4.5 Vector Quantization for Input Size Reduction

The state space can become really huge because of the curse of dimensionality. The utilization of state space is non-uniform, with some areas with frequently observed states and with other regions that are rarely observed or impossible to reach. Some areas share some similarities (a mode) [Kin+20]: E.g., a robotic arm gripping an object at different contact positions is holding the object and moves it connected to the arm. This forms the same mode, where all policies should be the same. The place fields encoding can help to utilize this non-uniform distribution better and put place fields in areas where it matters. Hence, we can obtain the principal components. Local differences in density are not considered by using a uniform grid aligned to the principal components. The objective is to optimize the placement of place cells to cover observed states in state space better, thus minimizing the sum of squared distances

$$\arg\min \sum_{i=1}^{k} \sum_{\mathbf{x}\in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 .$$

Another problem description is the optimization of a k-means clustering where the incoming observed states are assigned to one of the k place cells with the nearest mean. An online-learning algorithm is required to solve this problem, e.g., by using vector quantization. This method is utilizing the principle of competitive learning, where information is shared across topological neighbors. Each observation has a position in the state space. The place cell selected is the place cell nearest to the input state of the agent[2]. Every place cell's center position is shifted into the direction of this observed state.

The feedback signal modifies the action of a cell without affecting the other cells. The action could be included in the place cells and also shifted in the vector quantization (VQ) steps. Although it might be beneficial for the training, it breaks the correspondence of the symbolic computation with the SNN. Like the training of SNN, the weights must be only modified by means of the feedback signal. When the correspondence is not given, insights on this implementation cannot be transferred to SNN.

The cell's centers p_i are shifted in the direction of the sample s with

$$p_{i,t} \leftarrow \lambda \exp(\left\| \vec{sp}_{i,t-1} \right\| / a) \cdot \vec{sp}_{i,t-1}$$

where λ is a scalar factor and a is an exponentially decreasing each cycle starting at $a \leftarrow 1$. The effect of these parameters depends on the progress on the training. Changes of the other aspects of the training with an influence on training progress thus need a recalibration of the decaying speed da/dt and effect size λ.

Comparing non-VQ runs with VQ-runs show that it benefits learning performance. The problem is defined as that an average return of 195 means that it is solved. The average return after 300 episodes and a critic resolution of 400 (more is better):

number of cells	no vector quantization
$3 \times 3 \times 5 \times 5$	142
$5 \times 5 \times 7 \times 7$	232
$7 \times 7 \times 15 \times 15$	297

When activating VQ with different parameters, the following results are obtained: With $3 \times 3 \times 5 \times 5$ (225 neurons) and VQ:

[2]The assignment of areas in state space to single points is a Voronoi partitioning.

	$\lambda = 0.0001$	$\lambda = 0.001$	$\lambda = 0.01$
da/dt=-0.0001	218	191	153
da/dt=-0.001	150	**337**	150
da/dt=-0.01	84	280	234

With $5 \times 5 \times 7 \times 7$ (1.225 neurons) and VQ:

	$\lambda = 0.0001$	$\lambda = 0.001$	$\lambda = 0.01$
da/dt=-0.0001	219	195	230
da/dt=-0.001	231	336	**386**
da/dt=-0.01	199	137	266

With $7 \times 7 \times 15 \times 15$ (11.025 neurons) and VQ:

	$\lambda = 0.0001$	$\lambda = 0.001$	$\lambda = 0.01$
da/dt=-0.0001	406	271	434
da/dt=-0.001	313	**432**	385
da/dt=-0.01	163	346	386

A similar performance of an average return of 297 with the high-resolution grid can be achieved with only 2% of the place cells ($3 \times 3 \times 5 \times 5$ grid) with VQ (average return of 337).

Size of the Receptive Fields The receptive fields can overlap, which causes an input to activate multiple neighboring place cells. When the state is gradually changed, this overlap creates smooth transitions of the activation between cells. When a homogeneously covered space is connected to a laterally inhibitory layer, a winner-gets-all-behavior (cf. subsection 4.1.1) causes a Voronoi partitioning with only the largest activated output to be of significance. Therefore, the receptive field's size or scaling is irrelevant as long as the order in the activation levels is kept.

4.6 A Minimal Problem: Error Minimization (Line Following)

A target reaching problem is a problem where the state must be changed to a desired state. It is one of the problems with the least complexity, yet it can teach us a lot about SNN. The problem is defined by a reward function, which in the target reaching problem, is represented by a norm $r(s) = \|s - x\|$. Thus, by utilizing the L^2-norm, the problem is of quadratic, convex form. Convex functions have a global minimum and no local minima. A state value function is depicted in Figure 4.12.

There are two ways to feed the information into the network in an error minimization problem. The information about the target position and the actual position can be either

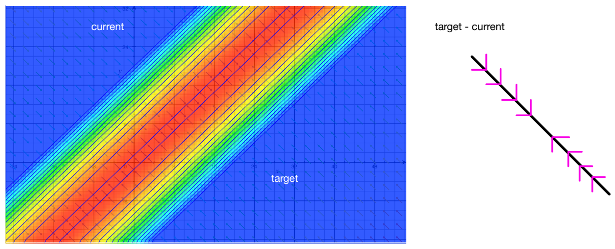

Figure 4.12: The value is color-coded. The optimal action is pointing towards the center of the strip.

given as two values, where the network needs to add a subtract block in the controller to obtain the error value or by directly giving the relative difference (Figure 4.13). Compared to the relative input, the system has additional information when using these absolute values. It could exhibit different behavior based on the value.

The problem can be solved with a neuro-actor implementation of Equation 4.3 and the relative input.

The action of the agent is implemented by using two neurons. The spikes are translated to an activity level using rate code. One output neuron's activity level represents the speed of going to the left, while the other represents the activity to go the right $a = (z_e - z_f)\theta$ with sensitivity parameter θ. This is an example of the extensor and flexor pattern.

Inhibitory Neurons Problems where the optimal states follow along a positive (non-zero, non-infinite) slope in state space (like in Figure 4.12) cannot be solved just by using excitatory neurons. The dimensionality is reduced to one by subtracting target and current position. This is equivalent to an orthogonal projection. Inhibitory neurons are needed to implement the subtraction operation. When the absolute input encoding is used, the net is required to learn how to subtract those values. A possible network is shown in Figure 4.13 c). Connecting an excitatory neuron to an inhibitory neuron inverts the signal's polarity on one synapse. When different synapse polarities on one neuron are possible, this signal inversion can be integrated directly into the synapse.

We can use some domain knowledge to narrow the architecture's capabilities with the effect on improving its performance: We expect that the system should exhibit the same behavior in any target or actual range; only the error is of importance. Therefore, we can only feed the error into the network. By obtaining the error by symbolic subtracting before feeding the input to the SNN, calculations are moved from the SNN to a CPU or FPGA. Now, no SNN subtraction is needed in the controller; thus, it can be constructed

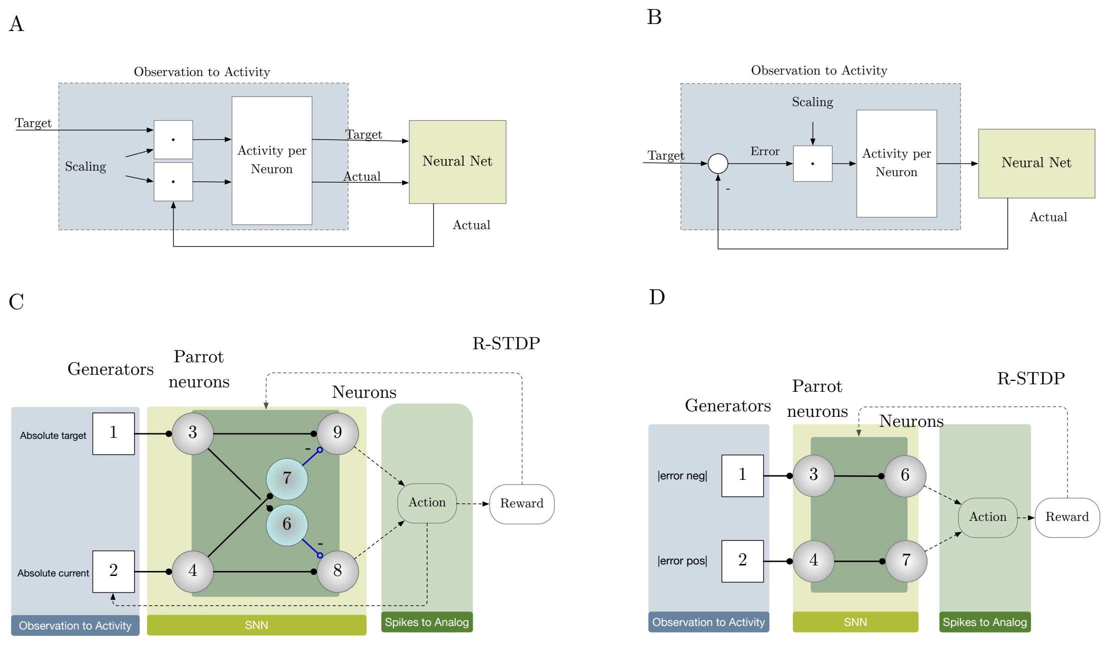

Figure 4.13: Details in neural actor input implementation with **A** absolute input or **B** relative input to the neural net. The many boxes after obtaining the activity per neuron are abstracted to the "neural net". **C** Solving neural net configuration for A, **D** 'minimal net' solving neural network configuration for B

without inhibitory neurons. This reduces the solution space for the required topology, as the subtraction operation does not have to be found by the net. Another effect is the dimensionality reduction of the input space, reducing the numbers of neurons and synapses and, with that, increasing the speed of learning progress. An implementation of this strategy was used by Spüler and Rosenstiel [SNR15] by manually designing the output pattern of the difference neuron. A solution network with the pre-computed error is displayed in Figure 4.13 D). An alternative is to use a subtractor microcircuit block of the "STICK" framework developed by Lagorce and Benosman[LB15], a more complex but spike-timing synchronized solution.

When inhibitory neurons are included in the training, one must consider that inhibitory activity correlates negatively with the postsynaptic firing; hence the curve H must be inverted for the learning factor.

More Difficult Problems This problem can be made more difficult to bridge the gap of real-world applications. If the environment contains obstacles, where the agent must move away from the goal for a time, the reward function becomes non-convex. A line-following or trajectory-following problem is a target-reaching problem, where the target changes its position during an episode.

5 Methodology and Results

5.1 Experiments

We use pole balancing and line-following.

5.1.1 Line-Following Environment Implementation

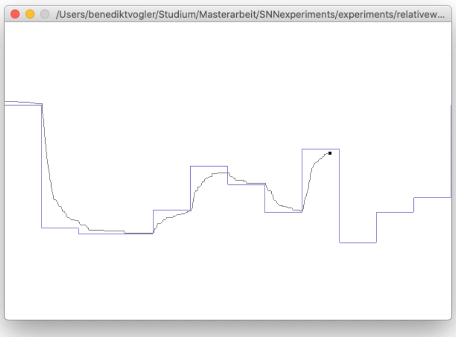
/Users/benediktvogler/Studium/Masterarbeit/SNNexperiments/experiments/relativew...

Figure 5.1: Graphical representation of an agent with a minimal net connectome following the line.

We found that a minimal goal-reaching problem was not available in the OpenAI gym framework. Therefore, we implemented a line-following problem in the framework. The gym framework allows easily exchangeable environments, which makes comparison easy. It also supports the rendering of environments. (Figure 5.1). The simulated pole-balancing environment is deterministic, so we do not need to store the transition probabilities.

The job of the agent is to move to a presented goal via locomotion. This goal state changes to a random position after a while resulting in a path to follow. The agent has a position along the line and obtains the next position. Our first attempt in constructing a line-following environment with a fixed sinusoidal path resulted in overfitting and safe strategies showing an instance of the exploration-exploitation balancing problem.

In our second attempt at designing the line following environment, a dominant, save strategy for the agent is just going straight. This happens when the net dies. This can be very confusing because a dying net is in most cases showing return-over-time curves to have learned to run on this environment.

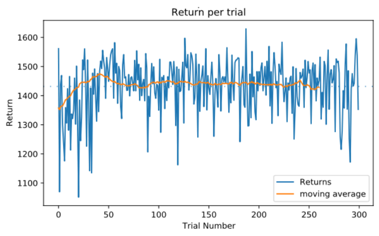

Figure 5.2: Progress is indicated, although the net learned to just go straight

The reward function is already optimal so that the optimal policy based on the state-value function is the same as one based on the reward (cf. Figure 5.3). A state may be reached that is outside the specified range. Usually, these states are failure states. They are considered as a regular state in the critic but are not displayed. The visualization shows that some regions are rarely covered in the simulation.

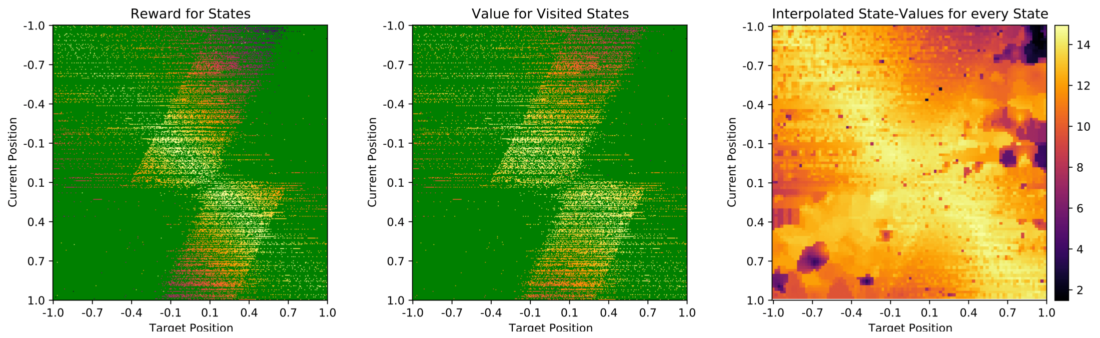

Figure 5.3: Visualization of critic's state maps for the line-following problem. The green color indicates missing data. The shape of the reward and computed state-values match. The diagram to the right shows the state value function when interpolating with a two-nearest-neighbor regression. The convexity of the problem becomes visible in all plots.

Using a population in the output layer should not improve the result because a single neuron with the fitting weight does not scale the firing rate, and no scaling is required to solve this problem.

5.1.2 Pole Balancing

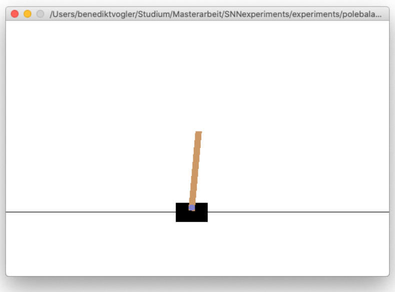
/Users/benediktvogler/Studium/Masterarbeit/SNNexperiments/experiments/polebala...

Figure 5.4: OpenAI gym graphical representation of the pole balancing problem.

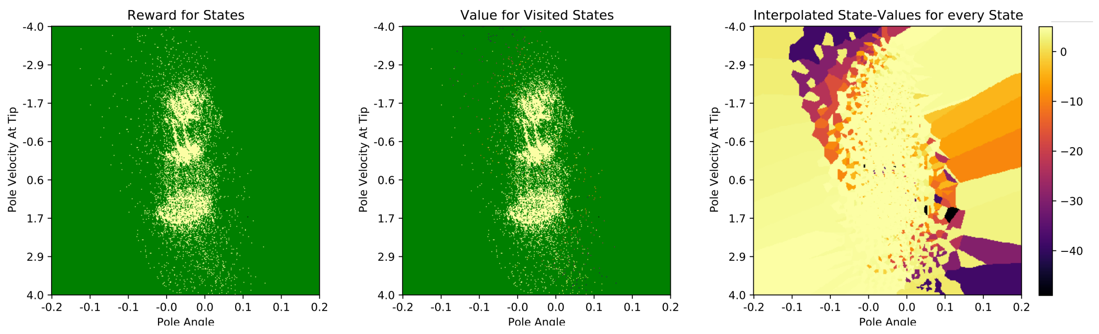

Figure 5.5: The state values (middle) are calculated from the reward (left). The green color indicates missing data. Here a projection to two dimensions is shown. On the right, the displayed interpolation is calculated by interpolating the 2d-projected values shown in the middle plot.

CartPole, or pole balancing or inverted pendulum, is a problem where a pole connected to a cart should be balanced upright [BSA83]. We use the implementation in the OpenAI gym (Figure 5.4). The observations cover four dimensions (car position x,$\dot{x}$, pole angle θ, $\dot{\theta}$). The action space is a binary choice of the direction of a force acting on the cart. This is different from the line-following problem, as here, no sensitivity to the output activity is of no relevance. The failure states are when x and θ exceed some limit. The state-value space can not be described by a simple distance function and is therefore non-quadratic. As defined in OpenAI, the reward is always one. We add a penalty when the pole exceeds the safe limits and terminates. Without adding a penalty, the reward would be one at all times resulting in a reward of 1 for critic derived states. Accumulating the reward (Sutton and Barto call this the *return*) is only useful if only the integrated reward per episode is used for training (e.g., with evolutionary learning)

For R-STDP, the reward must be returned quickly after the action was taken, therefore, requiring an adaptive critic. The pole balancing implementation can be transformed into an instant reward problem by approximating the Bellman equation: instead of returning a reward of one, an instant reward of $r = 50cos(\theta)$ [FSG13] is calculated. We showed that pole balancing can be solved using value iteration on the returned reward and by adding a penalty for failing without modifying the reward function.

Experiments with the abstracted non-event-based model (section 4.4) show that in the pole-balancing experiments a grid of $7 \times 7 \times 15 \times 15$ place cells as used by [FSG13] is overparameterized. Reducing the number to a fraction of 2% cells shows a better performance.

The plot of the projected state value map (Figure 5.5) confirms the non-linear problem definition. It also shows that the state space is covered in clusters.

5.2 Simulations

5.2.1 Existing SNN Software

There are different open source libraries allowing simulations of neural networks. Although outdated, for a more detailed overview of existing simulator software, we refer to [Bre+07].

PyNN	A simulator independent front-end. It supports the back-ends NEST, NEURON, and Brian. PyNN scripts can run on the Neuromorphic Computing Platform developed in the Human Brain Project (HBP).
NEST	The Neural Simulation Tool (NEST) supported by the NEST Initiative can model various spiking models [GD07]. It offers a Python interface to a C++ programmed back-end and is supported by PyNN. The initial version is from 2007. It is used in the HBP neurorobotics platform and the HBP Brain Simulation Platform.
SPORE	The SPORE module is a framework for developing online reinforcement learning algorithms with synaptic plasticity using the NEST library. A rich documentation is lacking and only contains one example. It does not support the latest versions of NEST. `https://github.com/IGITUGraz/spore-nest-module`
Rockpool	A rather new (2019) library developed by SynSense with support to run simulations of the companies hardware. `https://gitlab.com/aiCTX/rockpool`
Norse	A new library with a focus on machine learning. It is compatible with the machine learning library PyTorch. `https://github.com/norse/norse`
Brian	The development of this library started more than ten years ago. It is supported by PyNN. Equations can be translated into neuron models. `https://briansimulator.org`
GeNN	A SNN simulator with code generation for Nvidia GPUs (CUDA) [YTN16]. An automatic translation from Brian to the GeNN library is possible [SGN20]. `https://genn-team.github.io/genn/`
NEURON	Supported by PyNN. The focus lies on smaller scale like microcircuits and cells. `https://neuron.yale.edu/neuron/`

Our Choice The ecosystem of SNN software is largely coming from the field of computational neuroscience. Interest from the field of machine learning is growing producing developments like the Norse library. We chose NEST because it has been used with R-STDP before and has an active community.

5.2.2 Resetting with NEST

Version 2 of NEST includes a reset method. This method has been marked as deprecated in NEST 2 and is has been removed in the preview version of NEST 3. The reason for this removal was the unclear definition to which state the net should be reset. The removal results in much code that needs to be written, as the network connectome has to be read at the end of each episode, and it has to be manually be reloaded before setting up a new episode. This functionality was implemented by storing all connectome information in redundancy to upon reset pass to the NEST back-end. Partial resetting happens in each episode in RL training. Non-RL experiments where repeated experiments are performed on the same connectome suffer from the same issue. Upon this lack in NEST 3, Daphne Cornelisse and us proposed to reintroduce this method to NEST 3 [VC20]. It is unclear to which state the reset call should lead, so we favor a solution where only spike events are reset. An alternative is the implementation of methods to extract and load the connectome.

After an episode, before resetting, the simulation needs to be run without any input for some time so that the effect of the STDP in the last cycle can have an effect.

5.2.3 Parallelizing Training

Each training episode happens incrementally. How can parallelism be used for a training instance? When running SNN simulations on von Neumann computers, the performance can be improved by running several simulations at once. The weight changes of the parallel instances are then averaged. This is comparable to a batched stochastic gradient descent (SGD). When training ANN with SGD, each training pair has different data. Likewise, it is of importance that each training has slightly different starting conditions. The implementation in the NEST library of synaptic plasticity does not allow threaded simulations.

5.2.4 Deployment on Neuromorphic Hardware

Our proposed SoC design can be first tested by using multiple connected devices. The actor should run on neuromorphic hardware. Currently, there are two European research devices. The SpiNNaker system is a multicore architecture based on ARM cores allowing more than a million cores connected in one machine. The other system is BrainScaleS 2 that physically emulated neurons operating at 10,000 times of real-time. To run the spiking actor on a SpiNNaker system, the library PyNN must be used. In this library, the support of neuromodulated synapses seems to be not yet existing. The PyNN simulator back-end for the SpiNNaker is called "sPyNNaker"[Rho+18]. Mikaitis et al. modified sPyNNaker to support dopaminergic connections by providing the

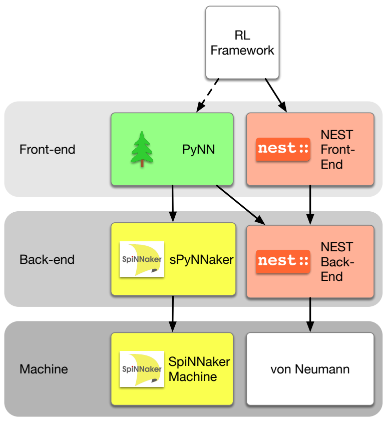

Figure 5.6: Different libraries and their relationships. A solid arrow indicates usage, while the dotted arrow indicates future development.

neuromodulatory information via special types of synapses [Mik+18]. It is unclear whether these changes have been integrated into the regular available PyNN and sPyNNaker libraries.

A chip that does not support reinforcement learning can be loaded with weights that were obtained by reinforcement learning training on a simulation.

5.2.5 Simulating on a GPU

When simulations are scaled, neuroscientists use cloud computing or supercomputers. For the use of robotics, the communication overhead and possible package-loss require the use of embedded devices or edge computing. Computation speed may be improved by harnessing the power of GPUs. GPUs originated as graphics accelerators for video games but are used increasingly in various general-purpose computing tasks. Now, edge computing devices are sold with similar power consumption to neuromorphic hardware. Mikaitis et al. [Mik+18] implemented R-STDP on an Nvidia Jetson TX1. The implementation's computing time scales linearly with the number of synapses. The usage of these systems is therefore limited to small networks. The SpiNNaker system outperformed the GPU at around 6.5 million synapses. The usage of more powerful GPUs with higher power consumption and bigger form factor would push this limit higher, thus being of interest for research. *NeuronGPU* is a novel library, developed by Bruno Golosio, to simulate SNN on the GPU [Gol20]. It is planned to integrate this to NEST by adding communication neurons, which communicate incoming and outgoing spikes to and from the GPU. This allows a hybrid simulation in which parts of the network are simulated on the CPU while other parts are computed on the GPU. The algorithm currently does not support synaptic plasticity.

5.3 Implementation Details

5.3.1 Our Implementation

We implemented the framework using Python 3.8 and the latest NEST 3 preview version. The NEST 3 version offers an improved Python interface. However, further developments for more straightforward support of reinforcement learning problems are needed. Some discoveries about the shortcomings of NEST 3 were provided as feedback and already got integrated.

The used neuron model ist the default integrate and fire 'iaf_psc_alpha.'. Neuromodulated synapses have been modeled by Potjans et al. as an extension to the NEST software [PMD10]. They found that the distribution of the neuromodulatory signal makes distributed simulation difficult. Their implementation approach uses volume transmitters, which can be used for multiple synapses. The dopamine neuron is connected to the STDP synapses using the *volume_transmitter* interface. The amount of spikes is set in the parameter n. By setting the spike rate via the programming interface, fractional values can be used.

We ran simulations on multiple devices, including an Intel i9600K 6-core machine and the LRZ compute cloud, both running Ubuntu 20, and an Intel-based Mac running macOS 10.15. Results were continuously collected in an externally hosted database. This adds network traffic and makes the simulation slower but allows easy training progress observation and data analysis.

The software allows the modification of many configuration settings. Some predefined experiments are included.

We included some features in the software, but did not use them in the final experiments:

- Support of populations in the output layers.
- Support of a reservoir.
- Inhibitory neurons.
- Automatic synaptic pruning.
- Random structural plasticity based on synaptic set-point.
- Filtering of the output signal.

The benefit of the chosen methods was not evident in the examined problems or beyond this study's scope. Under these considerations not included in the software were:

- Stochastic parallel training.
- Structural plasticity as an additional mechanism for firing rate adjustments.

5.3.2 Hyperparameters

Hyperparameters are parameters of the training that are not trained by the core learning algorithm. However, this distinction in regular and hyperparameter is arbitrary decided by the inclusion of the parameters in the core algorithm or some meta-algorithm. The best hyperparameters can be found by running a grid-search: running a training for every combination of the hyperparameters. With a resolution of only five steps per hyperparameter, n hyperparameters result in 5^n training runs, thus requiring massive parallelization. We hence checked parameters, if applicable, in isolation. If we suspected correlated effects, we tested them together in a grid search. Some parameters were found by simulation on the faster abstracted table-based simulation. In the following, we list every relevant parameter and the evaluated range.

Parameter Name	**Grid Search Range**	**Optimal Value by Grid Search on Abstraction**
num neurons per dim	$5 \times 5 \times 7 \times 7$	-
cycle length[1]	40	-
pole balancing penalty	-50	-
observation scaling	400	-
resolution of the critic	50 - 400	400
size of the receptive field[2]	0.3 - 5	1
reward signal factor	0.0001 - 0.8	-
limit on the reward signal	1 - 8	7
vector quantization convergence speed	10^{-4} - 10^{-2}	10^{-3}
vector quantization scaling	10^{-4} - 10^{-2}	10^{-3}
lateral inhibition weight	100 - 400	-
sensitivity out scaling[3]	? - ?	-
baseline learning rate	0.9	-
Fremaux filtering τ	40	-
Fremaux filtering v	10	-

[1]: The cycle length was chosen based on previous comparable work. The cycle time is predefined on real-world applications with frame-based sensors and a symmetrically timed read and write. As described in section 4.3.2, the input and output cycle time can be chosen independently.

[2]: Assuming equidistant tiling, the receptive field can be scaled to a multiple of this.

[3]: This parameter is only required if the environment makes use of a continuous action. The impact of this parameter can be reduced by using populations (subsection 4.3.2)

Some parameters can be analyzed without cross-checking with other parameters, as we expect them to be independent. We found the maximum M to be optimal in the range of 5-7.

Grid Search The grid search parameter range can be specified in a JSON file. Every globally accessible configuration variable can be overwritten by using the variables name. Combinations of network architectures and solving algorithms are configured in Python experiment files. Parallelization was implemented by spawning multiple processes via the multiprocessing module of the standard Python library. Although

NEST is designed to support parallelization, NEST's current version does not support it when synaptic plasticity is enabled. A grid search has no data exchange across the training instances. Such problems are called "embarrassingly parallel". By running several instances in parallel, the training time gains a very high speedup.

When running on a server without a graphics output (headless), a display driver is not configured. This causes problems with the OpenGL rendering of the OpenAI gym. To support headless execution, we added a headless mode, where rendering is not performed in the report generation.

The latest build of NEST can be obtained and run via Docker, allowing easy setup. After installing the Python dependencies via pip and the requirements.txt file, experiments can be run. The supported Python version of our software is Python 3.7 and 3.8.

Unfortunately, the software contains a memory leak. After some time, more than 15 GB of RAM (of 18 GB available) is filled when performing a grid search. When the memory is almost full, further training freezes. The source of the memory leak is not known. We suspect the critic accumulates many Python objects which are not released when an instance completes.

5.4 Results

5.4.1 Results Regarding the Neuromodulatory Signal

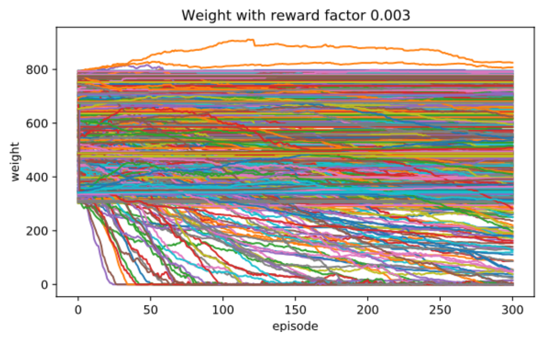

Figure 5.7: Without any countermeasure the weights of active neurons decrease till the net dies.

We observe that in the default case, the weights of the excitatory neurons continuously fall (depicted in Figure 5.7) until the weights are so low that no firing occurs anymore, and the system dies.

Results with Constant Baseline We tried shifting the expected reward by adding a bias each cycle. We ran a parameter sweep to test the effect of different values for this value. For each bias value, ranging from no bias to a high value, a training run for hundreds of episodes was held. Until a certain bias value, a decreasing weight trend in training was visible. At some point, the dependency on this bias became chaotic, with high sensitivity to the bias value – slight variations in the bias resulting in different training outcomes. Further increases in the bias resulted in training runs where the weights mainly grew. Just adding a constant bias was not enough for training success.

We also looked at using the expected bias and distribute it evenly as $b = \frac{u(s_e)-u(s_0)}{\mathbb{E}[n]}$ where n is the number of cycles in an episode resulted. This approach resulted in too much growth.

Results on Clamping The second suggested approach is to clamp the accumulated reward signals so that the integrated signal stays in a range. Unfortunately, it does not work as the learning stops (Figure 5.8). Our explanation for this is that the following problem is encountered (Figure 5.9). Some episodes have more negative utilities (as shown in section 3.3), moving the accumulated reward to the lower limit. Now at the limit, mostly negative reward signals are blocked. Further changes to the network do not happen. Therefore, the next episode is the same. The training is now stuck.

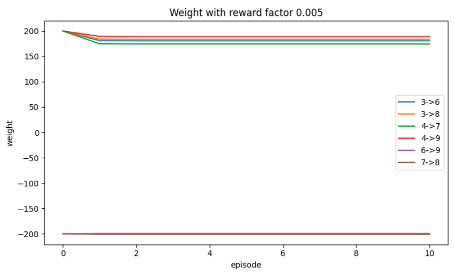

Figure 5.8: Learning stops, as the delta in state-value is only negative.

The Influence of the Penalty. Experiments showed that the choice of the penalty value highly influences training progress. We found no obvious correlation. The variation in training runs might be explained by noise. Differences in penalty affect the δu by scaling the values, which is similar to the effect of a different learning rate.

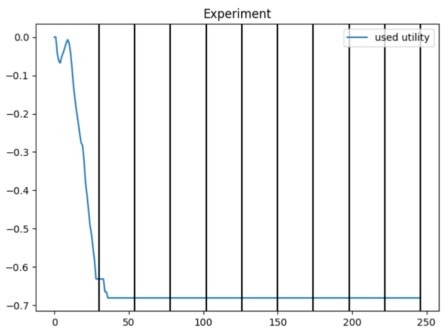

Figure 5.9: There are no upwards curves when the allowed utility bias reaches the limit.

5.4.2 Results Regarding Structural Plasticity

One easy way to add synaptic plasticity is to add synaptic pruning where synapses with a low weight (e.g., a tenth) of the initial weight are pruned. By setting a number of synapses, the connectome should have synaptic plasticity that can be used to establish new connections. Reconnecting every couple of cycles results in periodic disturbances with no beneficial effect. If a network has developed sufficient topology but has free connections, redundant connections are established.

5.4.3 Results on Life Following

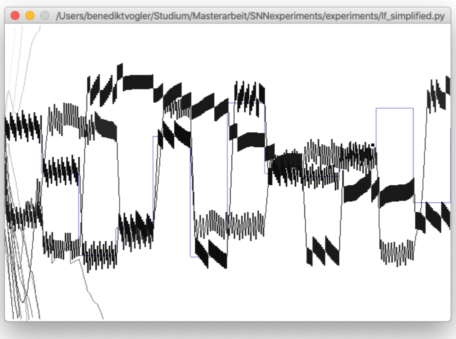
/Users/benediktvogler/Studium/Masterarbeit/SNNexperiments/experiments/lf_simplified.py

Figure 5.10: Jiggling around the target path caused by overshooting. Multiple trajectories are visible of subsequent episodes.

The implemented neural input can be relative or absolute, as presented in Figure 4.13. We compare the place cell architecture with the 'minimal net' (Figure 4.13, D). The minimal net is a minimal connectome and returns a good and smooth solution. When training using place cell encodings, some jiggling can appear (Figure 5.10). The agent overshoots because the activity in the action neurons is too big.

The simplified actor learns when no dynamic baseline is used. We were not able to reproduce the learning with a baseline. When using spiking neurons and no baseline is given, the net dies, resulting in a sub-optimal but high return.

5.4.4 Results on Pole-Balancing

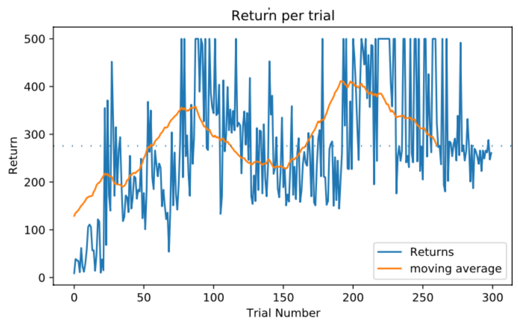

Figure 5.11: The pole-balancing problem is learned after 100 episodes with the non-spiking but equivalent table based implementation and vector quantization. After 40 episodes the problem is solved.

The table-based algorithm indicated that pole-balancing is solvable with a low number of parameters. We implemented this actor in our framework and tested it on the pole-balancing problem. The agent quickly learns to solve the pole-balancing problem (Figure 5.11).

Spiking Actor Implementation We performed a grid-search on the spiking net with a low number of receptor neurons. All trials suffered from an up and down curve of the performance.

When the baseline is initialized with the first return, only the second visit of a state has an effect. Therefore, it needs many episodes before learning happens (Figure 5.12 A). In comparison, when the baseline initialization happens via KNN regression, learning starts earlier (Figure 5.12 B). Although a dynamic baseline was used, the weights were overall falling (Figure 5.13). Interestingly the maximum learned performance is better when using no baseline (Figure 5.14).

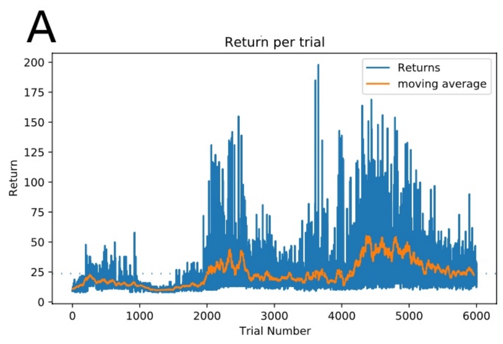

Figure 5.12: Return over time on the pole-balancing problem. **A** Baseline initialization with Δu_0. **B** Baseline initialization with KNN regression.

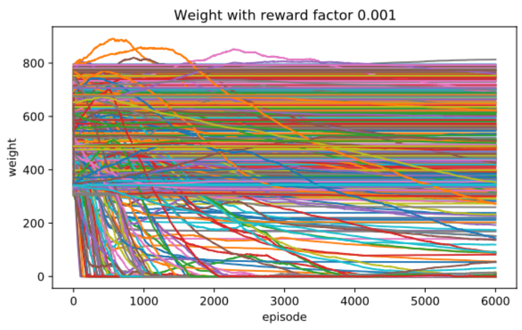

Figure 5.13: The weights are falling on the pole-balancing problem, although a dynamic baseline was used.

Vector quantization helps with the training. Grid-search on the vector quantization parameters for $5 \times 5 \times 7 \times 7$ grid returned the best average result with $\lambda = 0.001$ and $a = 0.001$. When vector quantization is disabled, the training peaks once but then does not improve upon this point (Figure 5.14).

Training Computing Performance An analysis with the Python included cProfile Profiler showed that around 10% of the run time is spent on setting the neurotransmitters in each cycle.

We noted the time of the trainings. On the LRZ compute cloud server, the line-following problem's average performance was 4.4 ms/cycle. On the 1k neurons pole balancing, the performance was measured with 232.2 ms/cycle, thus slower than real-time. The pole-balancing problem runs at 50 frames per second, i.e., 20 ms update

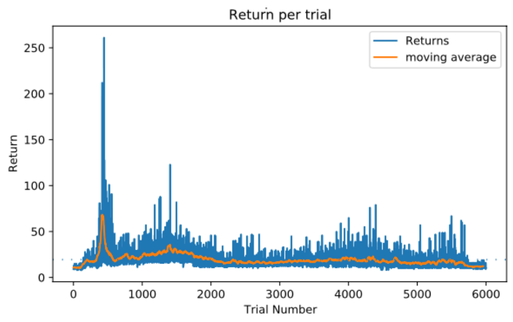

Figure 5.14: Return on the pole-balancing problem when vector quantization is disabled.

time. With 225 neurons 160,86 ms/cycle (although it had frequent database writes). On the Intel computer pole-balancing with 225 neurons: 40,88 ms/cycle, which is almost real-time. The wall-clock training time is dependent on the number of episodes the training is run. An SNN with 11.025 neurons (22.050 parameters) required a wall-clock time of around 5h of training time to make statements about the training result.

6 Discussion

6.1 Discussion of the Results

Event-based SNN controllers need a training algorithm. We derived a framework to construct such controller, but found the learning in the spiking implementation to be limited.

6.1.1 Analysis of Baseline Limitations

We elaborated on the requirements to use R-STDP. An important factor is the centering of the neuromodulatory signal. We found a static baseline to be an inadequate centering method to make the system learn. Only the static baseline of the evenly distributed bias eliminates the bias when integrating over the episode. However, in many problems, the length of an episode is unknown beforehand; thus, the even distribution can not be guaranteed. It is aggravated by the fact that during training, the episode length changes toward longer episodes. Even when the baseline is calculated on the basis of a known episode length a bias remains.

The dynamic error signal baseline is another approach to solve uncentered problems. We found that this implementation of a spiking actor training on the pole-balancing problem was only working to a limited extent, and line following was not solvable. A dynamic baseline for each task, which is defined input/output pair, has been suggested. We defined a task based on the bucket used by the critic for the input. Falling weights indicate that the dynamic baseline is not working as expected.

It can happen that states in one bucket can require different actions. Therefore, the correction of the neuromodulators still contains a task based bias.

A high number of buckets reduces the bias inducing effect of multiple actions per bucket. However, this introduces a trade-off, as a higher resolution leads to fewer repeated bucket lookups of the same buckets. With fewer lookup collisions, many buckets are visited only once or a small number of times. These states can thus not develop an averaged baseline.

Surprisingly the table-based implementation is able to solve the pole balancing task, while the spiking implementation is unable to. We suspect that the stated equivalence is in fact not quite given. One difference is in the pace of the learning. Compared more training episodes are required because the spiking implementation includes the noisy eligibility trace, while the assignment to the correct cell in the table is made directly. This might increase problems with biases. Firing events in the non-major active cell can cause anti-correlated STDP. This role may be bigger than expected.

6.1.2 Failure of trajectory-following

That the dominant strategy fails, altough is explainable

6.1.3 Limits of Rate Code

The place cell microcircuit is based on the firing rate code. When a neuron creates a spike train, the downstream neurons connected with a high enough synaptic weight responds with a spike train with a proportionally scaled firing rate. The rate code loses the nonlinearities because amplification of the firing rate in the signal flow is only linear. When the activation of a postsynaptic neuron is typically under the threshold to elicit a spike, the relative timing of the incoming spikes matter: When arriving closely together, they can cause spike; Arriving distributed over a long time, no spike is created. Thus, when the activity level is low, the relative firing times of many downstream neurons are of importance to make use of the nonlinear capabilities of neurons. This highlights the role of adjustments in the delay times from spike emittance till arrival at the postsynaptic neuron.

6.1.4 Dimensionality Reduction with Vector Quantization

The vector quantization algorithm works as intended. It is able to move the place cells for better utilization of the state space, improving the performance on the problem, and reducing training time. The calculation about the activation must be performed on a CPU. The naive approach calculates the activation for every neuron, which can become a slow operation. Optimization for picking a reduced set to compute the activations is easier when the positions are regular and fixed.

6.1.5 Simulations Beyond the CPU

Neuromorphic Deployment We showed that the proposed framework can be implemented with Python and the spiking actor using NEST, simulating on the CPU. As pointed out in this paper, some improvements on NEST would still be beneficial for faster execution and programming with less code. However, the goal is to bring this framework to neuromorphic hardware.

While simulations running on the simulator independent library PyNN can run on the SpiNNaker chip, the PyNN software is currently not versatile enough to work on R-STDP implementations. The ability to run NEST directly on neuromorphic hardware would be beneficial for fast exploration of algorithms.

Training Time and Development Cycles SNN training on CPUs can become very slow, as we showed in this work. A slow training time results in slow feedback loops and is a major hurdle in the research and development incorporating SNN. For development, training time is comparable to compilation time in programming, although the number of parameters has a stronger influence on the training time than code length on compilation time [Tha84]. Training ANNs can take up to months of training time and cost millions of dollars. The success of GPU computing for ANN underlines the market's desire need for short training cycles. Our own experience showed the negative psychological impact of the delays it has for developing. The typical development cycle involves that many hypotheses are pursued and a change to the code or configuration is performed to examine this hypothesis. The training is then started. A big delay until the training result is obtained requires the researcher/developer to perform mental context switching. Later, when the results have been computed, to analyze them, the hypothesis again must be made aware.

Many insights on the application of SNN are to learn from toy examples, but for the widespread adoption of commercial SNN applications, the nets have to be scaled further. Hence, we expect that advances in neuromorphic hardware will create a demand for more software solutions. For more complex task the Neurorobotics Platform, part of the Human Brain Project, could be used for providing the software infrastrucutre.

A development cycle is not only dependent on the training time but also on the available supporting tools like visualization software, access to hardware and dataset. A lot is to learn from the ANN ecosystem, which includes mature ANN learning libraries used commercially worldwide. In the year 2006, Gregory V. Wilson noted, "Increasingly, the real limit on what computational scientists can accomplish is how quickly and reliably they can translate their ideas into working code."[WD06] This is still true today.

6.1.6 A Modular View

It might be beneficial to view different neural architectural strategies as modules with different functions. When a camera is connected in a shallow network, it can map each pixel to a neuron in parallel [Mes17]. When a problem has a high dimensional space, i.e., for video input, a receptive field encoding will create too many neurons. An absolute encoding may connect each pixel with a single neuron, thus requiring only a neuron count at the number of pixels. This layer might then be mapped to the place cells. This would map the raw video input to a latent state in latent space. By extending the shallow network to connect it to a deep network, it allows the emergence of more complex behavior and the usage of temporal information in activations in latent space.

When used for motor tasks, the comparison with biological systems comes to mind. We speculated that using different layers might be beneficial. A similar distinction of functionality in different modules can be observed in the brain. "At a conceptual level, the vertebrate propulsive locomotor system can be divided into four components, one for each selection, initiation, maintenance, and execution of the motor program."[Liu+15][Chapter 5.1.2, citing [Gri03] and [Gri+00]]. The mesencephalic locomotor region (MLR) initiates and maintains locomotion (ibid.). Central pattern generators (CPG), specialized circuits in the spinal cord, are activated by the MLR. Our neuronal model is thus comparable with the MLR, while the CPG could be compared to the translation of the actor signal to the electric motor controller. Remarkably, the wiring of the CPG shows parallels to the place cell architecture with flexor and extensor neurons connected with inhibitory synapses [Luo15][Chapter 8.6].

By adding more layers or recurrently connected reservoirs, the solution space of mechanisms using precise timing grows. A Poisson-distributed encoding of the input signal contains no precise timing information in the single input spikes. Thus no computational benefit in the usage of spike timing can be used and solutions jave to rely on the linear processing of the firing rate.

6.2 Chances and Risks of Applications

Scientific research at (technical) universities is a construct of instrumental reason [Hor47] to develop technology. The main goal is to secure the power and prosperity of a nation. Research is not political and ethically neutral but rather it can be seen as a means to these ends.. Therefore, we want to point out a few key chances and risks of the possible appliance of SNN technology. Ethical judgment is left to the reader.

6.2.1 Mobile and Wearable Technology

A significant benefit of reduced power usage in computations can be gained in the field of mobile and wearable technology. They are required to be small and light and are packed with sensors. Because of the limitations in size and weight, they can only run with a limited energy supply. Wearables need all-day battery life for widespread adoption. Progress would thus lead to more powerful neuroprosthesis and wearables like earbuds or smart glasses. Low power consumption in specialized AI chips enables always-on processing of audio and video or other sensor data like LiDAR. Sensors in devices are typically used for surveillance or human-computer-interaction.

Currently, human augmentation by prosthesis is only performed where the abilities are below the norm, like in cochlea implants. This might change in the future as SNN enable the bridge of computers with neural tissue paving the way for transhumanistic ideals. They open the door to new cyborg techniques like advanced wearables and neuroprosthesis. Humans living the "civilized" western way of living are alienated from their nature. Effects are visible in diseases of affluence, e.g., depression or obesity. The "second nature" is an environment created by humans using technology and cultivation [Geh97]. The transformation of this cultivated encountered second nature can be viewed as something where the body should follow and cope to. This idea extends even to allow living in new environments like space [CK60].

In contrast to this technological transhumanistic view exists the traditional humanistic view. This view holds that the design of the human environment should be changed using the means of law, architecture, and cultural rules instead of transforming humans to match the environment [Wei76].

6.2.2 Neurorobotics

Robotic systems can be applied for more tasks when they are able to work autonomously in changing environments, thus have intelligence. Neuromorphic systems can implement this intelligence. When artificial brains become more like the biological model, it makes sense to combine it with anthropomimetic robots, robots that work and feel like humans [Ric+16]. It is expected that spiking will lead to more natural-appearing robots. Robots use feedback-loops to manipulate joints like vertebrate animals. The first step for anthropomimetic brains is to build an artificial cerebellum to make motor control systems using neuromorphic hardware and spiking networks. This allows the control of artificial limbs under human guidance.

Reduced energy consumption makes robots lighter by reducing the need for bigger batteries or increases the runtime. It enables the development of small robotics inspired by animals like birds, insects, and fish [Cla+16].

The fast reaction time utilizing SNN and DVS allows faster industrial robots or autonomously moving machines like autonomous driving.

A smaller-scale deployment sees potential extended applications in dealing with disasters like viruses, earthquakes, and nuclear accidents, or rescue operations. The massive deployment of robots and improved machines in our society will have big implications.

6.2.3 Modern Warfare and Geopolitics

Knowledge about neural chips is a special form of knowledge as it is a form of metacognition, knowledge about obtaining knowledge. Access to knowledge and information processing transforms highly-developed societies. The postmodernist philosopher Jean-François Lyotard [Lyo84] argues that "In the computer age, the question of knowledge is now more than ever a question of government." The knowledge about cognitive systems, which includes SNN, is thus also a question of nationalistic interests.

Historically modern warfare has been closely linked with computer technology. It utilizes autonomous drones, robotics, missiles, and defense intercepting installations. The theory of cybernetics has its roots in the work of anti-aircraft guns. Today, computer vision and AI are increasingly essential parts used in the military. Computers enabled the upwards spiraling arms race with nuclear missiles starting with the SAGE-Rocket system[1] [Wei76].

Autonomous systems rely on artificial intelligence and are increasingly being used in the military. Autonomy offers stealth features because no detectable wireless communication is needed. It also prevents from enemy sabotage by signal jamming. Hypersonic missiles rely on autonomous flight to evade defense mechanism. Autonomous drones and missiles can have better target accuracy and extended range by having faster or more energy-efficient computing. Faster and accurate computer vision increases the automatic target-locking of guns, high-powered lasers, and missiles (e.g., the infrared homing missile "PARS 3 LR" by Diehl Defence). Other applications of autonomous systems are quadruped robots, that can navigate autonomously (e.g., "Vision" by Ghost Robotics). Improvements in battery technology created a new autonomous weapon: Large swarms of autonomously navigating and armed multicopter (commonly armed with explosives).

Access to better cyber-physical systems changes the geopolitical landscape by making conventional strategies obsolete. Therefore, access to advanced very large-scale integrated chips is a geopolitical strategic advance. It is estimated that chip foundry company TSMC will spend $12 billion on building a semiconductor fabrication plant on US soil [TSM20], as trade wars and US-Chinese military involvement in the pacific show Taiwan's unclear role in providing the western world with advanced chips.

6.2.4 Advancing the Field of AI

Advanced AI reduces the need to use human capabilities as a tool to control nature since artificial cognitive systems can replace humans. Flexible, stronger AI makes it cheaper to develop and deploy robots. This may create the effect that cheaper, low

[1]American radar station network "Semi-Automatic Ground Environment."

qualification jobs are more replaced by machines, thus, increasing the required skills participate in the job market increases.

Finding new methods to build learning systems can advance the research in building artificial general intelligence (AGI). The constructing of an AGI and reaching a technological singularity[2] is a transhumanistic metanarrative found through the history of computing. Building an AGI bears the risks that the system is not aligned with human values and cause existential risk, a problem known as the AI alignment problem or AI control problem [Bos14]. For human-like thinking, machines need human senses: "Since the human mind has applied itself to, for example, problems of aesthetics involving touch, taste, vision, and hearing, AI will have to build machines that can feel, taste, see, and hear." [Wei76][Chapter 5, p. 139] Biologic inspired spiking neural systems close the mechanistic gap between robots and humans. By using methods of human learning the concept of "Predictability through inheritance" can be applied [Bos14][p. 108]: "If a digital intelligence is created directly from a human template (as would be the case in a high-fidelity whole brain emulation), then the digital intelligence might inherit the motivations of the human template."

[2]A hypothetical point in the future where technological growth becomes uncontrollable. The often claimed source is based on an intelligence explosion by a superintelligence.

7 Conclusion and Outlook

7.1 Conclusion

This work derived a general-purpose reinforcement learning framework utilizing R-STDP. It was laid out how it could be possible to construct a domain-independent low-power learning embedded device by assigning the different functions of the RL framework to different hardware modules. The encoding of states with place fields forms a popular microcircuit. We showed how R-STDP needs a symmetry breaking mechanism for this neural topology, e.g, by using lateral inhibition with inhibitory synapses. We explained its effectiveness by comparing it with the architecture of a CMAC. A table-based implementation can be used for a fast analysis of modification of components in this framework. The present study suggests the usage of movable receptor fields by the method of vector quantization. In an instance of using the table-based implementation for analysis of this microcircuit, we tested the impact of vector quantization. Vector quantization turned out to better utilize the state space and reduce the impact of the curse of dimensionality allowing the application on high-dimensional problem classes. It also showed that solving the famous pole balancing problem should require a lot fewer neurons than in previous R-STDP attempts.

We also showed the need in constructing the neuromodulatory signal using the state-value function calculated by the critic instead of using the obtained reward directly. It improves the training performance or even enables the training of problems with a delayed reward. In deterministic problems, the action-value function can be used. We encountered a number of issues with a bias in the neuromodulatory signal. The reward distribution in the RL framework was identified as a source. A task-based bias remains in the counter-measure of using multiple bucketed based dynamic baselines.

7.2 Outlook

Further work is needed on the critic and the design of the neuromodulatory signal. The coarse-graining of the critic is a crucial part. Most real-world applications work in the continuous domain, so critic implementations should also adopt this. While we found an equation producing a neuromodulator signal for any problem, equations based on the TD error may perform better because of parallels to biology. Recent findings on the complex nature of dopamine indicate that design rules for RPE slopes might be an approach for neuromodulatory reward centering. Other neuron context-modificating neuromodulators like acetylcholine might be used for feedback signals.

Some of our experiments with inhibitory neurons indicated that they might require special learning rules. Therefore, they were excluded from our experiments. In this work, it was shown that for symmetry breaking inhibitory neurons, respectively synapses, are required. This observation and the application in the brain show that inhibitory connections play a crucial part, and its role in microcircuits should be further examined.

In this work, we evaluated the usage of spiking shallow feed-forward networks with R-STDP. Networks with several layers, called deep networks, can solve more problems, i.e., by performing dimensionality reduction. R-STDP can solve the distal reward problem with an eligibility trace. Because of the global reward signal, it should also solve the distal reward problem in deep networks. Scaled networks could enable neuromorphic chips to solve complex problems based on high dimensional input, e.g., camera input. For a real-world application, scaling to more complex topologies is a requirement. More complex topologies are more challenging to train. In this work, the importance of structural plasticity to solve some problems was pointed out. We suspect a growing role of structural plasticity with increased complexity of topologies.

The dimensionality reduction by vector quantization is only one suggested approach for dealing with high dimensionality. A biologically plausible algorithm might be found by comparing designs with neurophysiological data.

With scaled networks the problem of long development cycles becomes significant. A speedup may be obtained by stochastic parallel training, or an R-STDP implementation running on GPUs. Better software support for neuromorphic deployment is also needed to bring SNN from simulation to reality.

Bibliography

[Alb75] J. S. Albus. "A New Approach to Manipulator Control: the Cerebellar Model Articulation Controller (CMAC)." In: *Transactions of the ASME, Series G. Journal of Dynamic Systems, Measurement and Control* 97 (1975), pp. 220–233.

[Arb03] M. A. Arbib, ed. *The Handbook Of Brain Theory And Neural Networks*. 2nd ed. MIT Press, 2003.

[BDM17] M. G. Bellemare, W. Dabney, and R. Munos. *A Distributional Perspective on Reinforcement Learning*. July 2017. arXiv: `1707.06887 [cs.LG]`.

[Bel+19] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass. "A solution to the learning dilemma for recurrent networks of spiking neurons." In: *bioRxiv* (Aug. 2019). DOI: `10.1101/738385`. eprint: `https://www.biorxiv.org/content/early/2019/12/09/738385.full.pdf`. URL: `https://www.biorxiv.org/content/early/2019/12/09/738385`.

[Bin+18] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll. "A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks." In: *Frontiers in Neurorobotics* 12 (July 2018). ISSN: 1662-5218. DOI: `10.3389/fnbot.2018.00035`. URL: `http://dx.doi.org/10.3389/fnbot.2018.00035`.

[Bin+19a] Z. Bing, I. Baumann, Z. Jiang, K. Huang, C. Cai, and A. Knoll. "Supervised Learning in SNN via Reward-Modulated Spike-Timing-Dependent Plasticity for a Target Reaching Vehicle." In: *Frontiers in Neurorobotics* 13 (May 2019). ISSN: 1662-5218. DOI: `10.3389/fnbot.2019.00018`. URL: `http://dx.doi.org/10.3389/fnbot.2019.00018`.

[Bin+19b] Z. Bing, Z. Jiang, L. Cheng, C. Cai, K. Huang, Maier, and A. Knoll. "End to End Learning of a Multi-layered SNN Based on R-STDP for a Target Tracking Snake-like Robot, to be appear." In: *2019 IEEE International Conference on Robotics and Automation(ICRA)*. May 2019.

[Blo+18] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith. *Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware*. 2018. arXiv: `1812.01739 [cs.LG]`.

[BM10] K. A. Buchanan and J. R. Mellor. "The activity requirements for spike timing-dependent plasticity in the hippocampus." eng. In: *Frontiers in synaptic neuroscience* 2 (2010), p. 11. ISSN: 1663-3563 (Electronic); 1663-3563 (Linking). DOI: `10.3389/fnsyn.2010.00011`.

[BO13] M. Butz and A. van Ooyen. "A Simple Rule for Dendritic Spine and Axonal Bouton Formation Can Account for Cortical Reorganization after Focal Retinal Lesions." In: *PLoS Computational Biology* 9.10 (Oct. 2013). Ed. by L. J. Graham, e1003259. ISSN: 1553-7358. DOI: `10.1371/journal.pcbi.1003259`. URL: `http://dx.doi.org/10.1371/journal.pcbi.1003259`.

[Bos14] N. Bostrom. *Superintelligence: Paths, Dangers, Strategies*. Kindle Edition. OUP Oxford, 2014.

[BP98] G.-q. Bi and M.-m. Poo. "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type." In: *The Journal of Neuroscience* (Dec. 1998).

[Bre+07] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, J. Harris Frederick C, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. P. Davison, S. El Boustani, and A. Destexhe. "Simulation of networks of spiking neurons: a review of tools and strategies." In: *Journal of computational neuroscience* 23.3 (Dec. 2007), pp. 349–398. DOI: `10.1007/s10827-007-0038-6`. URL: `https://pubmed.ncbi.nlm.nih.gov/17629781`.

[BS10] A. Bouganis and M. Shanahan. "Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity." In: *The 2010 International Joint Conference on Neural Networks (IJCNN)* (2010), pp. 1–8.

[BSA83] A. Barto, R. Sutton, and C. Anderson. "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem." In: *IEEE Transactions on Systems, Man, and Cybernetics* (1983).

[Cha+12] G. L. Chadderdon, S. A. Neymotin, C. C. Kerr, and W. W. Lytton. "Reinforcement learning of targeted movement in a spiking neuronal model of motor cortex." In: (2012).

[CK60] M. Clynes and N. Kline. "Cyborgs and Space." In: *Astronautics* (Sept. 1960), p. 26.

[Cla+16] T. S. Clawson, S. Ferrari, S. B. Fuller, and R. J. Wood. "Spiking neural network (SNN) control of a flapping insect-scale robot." In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. Dec. 2016, pp. 3381–3388. DOI: 10.1109/CDC.2016.7798778.

[Con+09] J. Conradt, M. Cook, R. Berner, P. Lichtsteiner, R. Douglas, and T. Delbruck. "A pencil balancing robot using a pair of AER dynamic vision sensors." In: *in Circuits and Systems, 2009. IEEE International Symposium on, 2009*. 2009, pp. 781–784.

[DA01] P. Dayan and L. F. Abbott. *Theoretical Neuroscience - Computational and Mathematical Modelling of Neural Systems*. MIT Press, 2001.

[Dab+20] W. Dabney, Z. Kurth-Nelson, N. Uchida, C. K. Starkweather, D. Hassabis, R. Munos, and M. Botvinick. "A distributional code for value in dopamine-based reinforcement learning." In: *Nature* (2020). DOI: 10.1038/s41586-019-1924-6. URL: https://doi.org/10.1038/s41586-019-1924-6.

[Dav+19] A. P. Davison, E. Müller, S. Schmitt, B. V. D. Lester, and T. Pfeil. *HBP Neuromorphic Computing Platform Guidebook: About the BrainScaleS hardware*. Nov. 2019. URL: https://electronicvisions.github.io/hbp-sp9-guidebook/pm/pm_hardware_configuration.html#f1.

[Dia+16] S. Diaz-Pier, M. Naveau, M. Butz-Ostendorf, and A. Morrison. *Automatic Generation of Connectivity for Large-Scale Neuronal Network Models through Structural Plasticity*. Tech. rep. May 2016.

[DMH19] G. Dulac-Arnold, D. Mankowitz, and T. Hester. *Challenges of Real-World Reinforcement Learning*. 2019. arXiv: 1904.12901 [cs.LG].

[Doy00] K. Doya. "Reinforcement Learning in Continuous Time and Space." In: *Neural Computation* 12.1 (2000), pp. 219–245. DOI: 10.1162/089976600300015961. eprint: https://doi.org/10.1162/089976600300015961. URL: https://doi.org/10.1162/089976600300015961.

[DSG17] M. Deger, A. Seeholzer, and W. Gerstner. "Multicontact Co-operativity in Spike-Timing–Dependent Structural Plasticity Stabilizes Networks." In: *Cerebral Cortex* 28.4 (Dec. 2017), pp. 1396–1415. ISSN: 1460-2199. DOI: 10.1093/cercor/bhx339. URL: http://dx.doi.org/10.1093/cercor/bhx339.

[FG16] N. Frémaux and W. Gerstner. "Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules." In: *Frontiers in Neural Circuits* 9 (Jan. 2016). ISSN: 1662-5110. DOI: 10.3389/fncir.2015.00085. URL: http://dx.doi.org/10.3389/fncir.2015.00085.

[FHF10] G. Foderaro, C. Henriquez, and S. Ferrari. "Indirect Training of a Spiking Neural Network for Flight Control via Spike-Timing-Dependent Synaptic Plasticity." In: (2010).

[FMD00] D. Foster, R. Morris, and P. Dayan. "A Model of Hippocampally Dependent Navigation, Using the Temporal Difference Learning Rule." In: *HIPPOCAMPUS* 10.1-16 (2000).

[FSG10] N. Frémaux, H. Sprekeler, and W. Gerstner. "Functional Requirements for Reward-Modulated Spike-Timing-Dependent Plasticity." In: *The Journal of Neuroscience* 30.40 (Oct. 2010), p. 13326. DOI: `10.1523/JNEUROSCI.6249-09.2010`. URL: `http://www.jneurosci.org/content/30/40/13326.abstract`.

[FSG13] N. Frémaux, H. Sprekeler, and W. Gerstner. "Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons." In: *PLoS Computational Biology* 9.4 (Apr. 2013). Ed. by L. J. Graham, e1003024. ISSN: 1553-7358. DOI: `10.1371/journal.pcbi.1003024`. URL: `http://dx.doi.org/10.1371/journal.pcbi.1003024`.

[GD07] M. Gewaltig and M. Diesmann. "NEST (NEural Simulation Tool)." In: *Scholarpedia* 2.4 (2007). revision #130182, p. 1430. DOI: `10.4249/scholarpedia.1430`.

[Geh97] A. Gehlen. *Der Mensch. Seine Natur und seine Stellung in der Welt*. 13th. Wiesbaden: AULA, 1997.

[GK02] W. Gerstner and W. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[Gol20] B. Golosio. *NeuronGPU*. 2020. URL: `https://github.com/golosio/NeuronGPU`.

[Gri+00] S. Grillner, L. Cangiano, G.-Y. Hu, R. Thompson, R. Hill, and P. Wallén. "The intrinsic function of a motor system - From ion channels to networks and behavior." In: *Brain research* 886 (Jan. 2000), pp. 224–236. DOI: `10.1016/S0006-8993(00)03088-2`.

[Gri03] S. Grillner. "The motor infrastructure: from ion channels to neuronal networks." In: *Nature Reviews Neuroscience* 4.7 (2003), pp. 573–586. DOI: `10.1038/nrn1137`. URL: `https://doi.org/10.1038/nrn1137`.

[GS06] R. Gütig and H. Sompolinsky. "The tempotron: a neuron that learns spike timing–based decisions." In: *Nature Neuroscience* 9.3 (Feb. 2006), pp. 420–428. ISSN: 1546-1726. DOI: `10.1038/nn1643`. URL: `http://dx.doi.org/10.1038/nn1643`.

[GSD12] W. Gerstner, H. Sprekeler, and G. Deco. "Theory and Simulation in Neuroscience." In: *Science* 338.6103 (2012), pp. 60–65. ISSN: 0036-8075. DOI: 10.1126/science.1227356. eprint: https://science.sciencemag.org/content/338/6103/60.full.pdf. URL: https://science.sciencemag.org/content/338/6103/60.

[Haf+05] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser. "Microstructure of a spatial map in the entorhinal cortex." In: *Nature* 436.7052 (2005), pp. 801–806. DOI: 10.1038/nature03721. URL: https://doi.org/10.1038/nature03721.

[Har94] R. L. Harvey. *Neural Network Principles*. Prentice-Hall International, Inc., 1994.

[HE16] E. Hunsberger and C. Eliasmith. "Training Spiking Deep Networks for Neuromorphic Hardware." In: *CoRR* abs/1611.05141 (2016). arXiv: 1611.05141. URL: http://arxiv.org/abs/1611.05141.

[Heb49] D. Hebb. *The Organization of Behavior*. Wiley: New York, 1949.

[Her+06] A. V. M. Herz, T. Gollisch, C. K. Machens, and D. Jaeger. "Modeling Single-Neuron Dynamics and Computations: A Balance of Detail and Abstraction." In: *Science* 314.5796 (2006), pp. 80–85. ISSN: 0036-8075. DOI: 10.1126/science.1127240. eprint: https://science.sciencemag.org/content/314/5796/80.full.pdf. URL: https://science.sciencemag.org/content/314/5796/80.

[HH52] A. L. Hodgkin and A. F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve." In: *The Journal of Physiology* 117.4 (1952), pp. 500–544. DOI: 10.1113/jphysiol.1952.sp004764. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1952.sp004764. URL: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764.

[Hor47] M. Horkheimer. *Eclipse of Reason*. Oxford University Press, 1947.

[Huh+10] D. Huh, B. D. Matthews, A. Mammoto, M. Montoya-Zavala, H. Y. Hsin, and D. E. Ingber. "Reconstituting Organ-Level Lung Functions on a Chip." In: *Science* 328.5986 (2010), pp. 1662–1668. ISSN: 0036-8075. DOI: 10.1126/science.1188302. eprint: https://science.sciencemag.org/content/328/5986/1662.full.pdf. URL: https://science.sciencemag.org/content/328/5986/1662.

[Hul43] C. L. Hull. *Principles of Behavor: An Introduction to Behavior Theory*. New York: Apleton-Centruy-Crofts, Inc., 1943.

[Izh07] E. M. Izhikevich. “Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling.” In: *Cerebral Cortex* 17.10 (Jan. 2007), pp. 2443–2452. ISSN: 1047-3211. DOI: 10.1093/cercor/bhl152. eprint: http://oup.prod.sis.lan/cercor/article-pdf/17/10/2443/894946/bhl152.pdf. URL: https://doi.org/10.1093/cercor/bhl152.

[Jor+20] J. Jordan, M. Schmidt, W. Senn, and M. A. Petrovici. *Evolving to learn: discovering interpretable plasticity rules for spiking networks*. 2020. arXiv: 2005.14149 [q-bio.NC].

[Kin+20] Z. Kingston, A. M. Well, M. Moll, and L. E. Kavraki. “Informing Multi-Modal Planning with Synergistic Discrete Leads.” In: *IEEE Intl. Conf. on Robotics and Automation* (2020).

[LB15] X. Lagorce and R. Benosman. “STICK: Spike Time Interval Computational Kernel, a Framework for General Purpose Computation Using Neurons, Precise Timing, Delays, and Synchrony.” In: *Neural Computation* 27.11 (2015). PMID: 26378879, pp. 2261–2317. DOI: 10.1162/NECO_a_00783. eprint: https://doi.org/10.1162/NECO_a_00783. URL: https://doi.org/10.1162/NECO_a_00783.

[Lin01] J. Ling. *Power of a Human Brain*. 2001. URL: https://hypertextbook.com/facts/2001/JacquelineLing.shtml.

[Liu+15] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, eds. *Event-based neuromorphic systems*. John Wiley & Sons, 2015.

[LS08] J. Lehman and K. O. Stanley. “Exploiting open-endedness to solve problems through the search for novelty.” In: *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI*. MIT Press, 2008.

[Luo15] L. Luo. *Principles of Neurobiology*. New York: Taylor & Francis Group, 2015.

[Lyo84] J.-F. Lyotard. *The Postmodern Condition: A Report on Knowledge*. 111 Third Avenue South, Suite 290, Minneapolis, MN 55401-2520: University of Minnesota Press, 1984.

[Mes17] C. Meschede. “Training Neural Networks for Event-Based End-to-End Robot Control.” MA thesis. Technical University Munich, 2017.

[Mik+18] M. Mikaitis, G. Pineda García, J. C. Knight, and S. B. Furber. “Neuromodulated Synaptic Plasticity on the SpiNNaker Neuromorphic System.” In: *Frontiers in Neuroscience* 12 (Feb. 2018). ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00105. URL: http://dx.doi.org/10.3389/fnins.2018.00105.

[Mil88] R. F. Miller. "Are single retinal neurons both excitatory and inhibitory?" In: *Nature* 336.6199 (1988), pp. 517–518. DOI: 10.1038/336517a0. URL: https://doi.org/10.1038/336517a0.

[MNM02] W. Maass, T. Natschläger, and H. Markram. "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations." In: *Neural Computation* 14.11 (2002), pp. 2531–2560. DOI: 10.1162/089976602760407955. eprint: https://doi.org/10.1162/089976602760407955. URL: https://doi.org/10.1162/089976602760407955.

[MP43] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[Par+06] S. W. Park, L. Linsen, O. Kreylos, J. D. Owens, and B. Hamann. "Discrete Sibson interpolation." In: *IEEE Transactions on Visualization and Computer Graphics* 12.2 (2006), pp. 243–253.

[PK10] F. Ponulak and A. Kasiński. "Supervised Learning in Spiking Neural Networks with ReSuMe: Sequence Learning, Classification, and Spike Shifting." In: *Neural Computation* 22.2 (Feb. 2010), pp. 467–510. ISSN: 1530-888X. DOI: 10.1162/neco.2009.11-08-901. URL: http://dx.doi.org/10.1162/neco.2009.11-08-901.

[PMD10] W. Potjans, A. Morrison, and M. Diesmann. "Enabling functional neural circuit simulations with distributed computing of neuromodulated plasticity." eng. In: *Front Comput Neurosci* 4 (2010), p. 141. ISSN: 1662-5188 (Electronic); 1662-5188 (Linking). DOI: 10.3389/fncom.2010.00141.

[Pro20] Prophesee. *Imago "VisionCam" Sensor powered by Prophesee*. 2020. URL: https://www.prophesee.ai/2019/09/19/high-speed-counting-event-based-vision/.

[PRR13] M. Posner, M. Rothbart, and M. Rueda. "Chapter 22 - Developing Attention and Self-Regulation in Infancy and Childhood." In: *Neural Circuit Development and Function in the Brain*. Ed. by J. L. Rubenstein and P. Rakic. Oxford: Academic Press, 2013, pp. 395–411. ISBN: 978-0-12-397267-5. DOI: 10.1016/B978-0-12-397267-5.00059-5. URL: http://www.sciencedirect.com/science/article/pii/B9780123972675000595.

[QIE18] N. Qiao, G. Indiveri, and K. Eng. *Ultra-low-power auditory processing in real-time*. Tech. rep. aiCTX AG, Nov. 2018.

[Rho+18] O. Rhodes, P. A. Bogdan, C. Brenninkmeijer, S. Davidson, D. Fellows, A. Gait, D. R. Lester, M. Mikaitis, L. A. Plana, A. G. D. Rowley, A. B. Stokes, and S. B. Furber. "sPyNNaker: A Software Package for Running PyNN Simulations on SpiNNaker." In: *Frontiers in Neuroscience* 12 (2018), p. 816. ISSN: 1662-453X. DOI: `10.3389/fnins.2018.00816`. URL: `https://www.frontiersin.org/article/10.3389/fnins.2018.00816`.

[Ric+16] C. Richter, S. Jentzsch, R. Hostettler, J. A. Garrido, E. Ros, A. Knoll, F. Röhrbein, P. van der Smagt, and J. Conradt. "Scalability in Neural Control of Musculoskeletal Robots." In: (2016).

[RL19] B. A. Richards and T. P. Lillicrap. "Dendritic solutions to the credit assignment problem." In: *Current Opinion in Neurobiology* 54 (2019). Neurobiology of Learning and Plasticity, pp. 28–36. ISSN: 0959-4388. DOI: `10.1016/j.conb.2018.08.003`. URL: `http://www.sciencedirect.com/science/article/pii/S0959438818300485`.

[RN16] S. J. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Pearson Education Ltd., 2016.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. second edition. MIT Press, 2018.

[Sch15] W. Schultz. "Neuronal Reward and Decision Signals: From Theories to Data." In: *Physiological Reviews* 95.3 (July 2015), pp. 853–951. ISSN: 1522-1210. DOI: `10.1152/physrev.00023.2014`. URL: `http://dx.doi.org/10.1152/physrev.00023.2014`.

[Sch98] W. Schultz. "Predictive Reward Signal of Dopamine Neurons." In: *Journal of Neurophysiology* 80.1 (1998). PMID: 9658025, pp. 1–27. DOI: `10.1152/jn.1998.80.1.1`. eprint: `https://doi.org/10.1152/jn.1998.80.1.1`. URL: `https://doi.org/10.1152/jn.1998.80.1.1`.

[SGN20] M. Stimberg, D. F. M. Goodman, and T. Nowotny. "Brian2GeNN: accelerating spiking neural network simulations with graphics hardware." In: *Scientific Reports* 10.1 (2020), p. 410. DOI: `10.1038/s41598-019-54957-7`. URL: `https://doi.org/10.1038/s41598-019-54957-7`.

[Sin+19] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. "End-to-End Robotic Reinforcement Learning without Reward Engineering." In: *CoRR* abs/1904.07854 (2019). arXiv: `1904.07854`. URL: `http://arxiv.org/abs/1904.07854`.

[SL15] P. Sterling and S. Lauglin. *Principles of Neural Design*. MIT Press, 2015.

[SNR15] M. Spüler, S. Nagel, and W. Rosenstiel. "A Spiking Neuronal Model Learning a Motor Control Task by Reinforcement Learning and Structural Synaptic Plasticity." In: *International Joint Conference on Neural Networks (IJCNN)* (2015).

[Suc+17] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning." In: *CoRR* abs/1712.06567 (2017). arXiv: 1712.06567. URL: http://arxiv.org/abs/1712.06567.

[Sut88] R. S. Sutton. "Learning to predict by the methods of temporal differences." In: *Machine Learning* 3.1 (1988), pp. 9–44. DOI: 10.1007/BF00115009. URL: https://doi.org/10.1007/BF00115009.

[Tha84] A. J. Thadhani. "Factors affecting programmer productivity during application development." In: *IBM Systems Journal* 23.1 (1984), pp. 19–35. ISSN: 0018-8670. DOI: 10.1147/sj.231.0019.

[Tie+19] J. C. V. Tieck, P. Becker, I. Peric, J. Kaiser, M. Akl, D. Reichard, A. Roennau, and R. Dillmann. "Learning target reaching motions with a robotic arm using dopamine modulated STDP." In: *18th IEEE International Conference on Cognitive Informatics and Cognitive Computing*. 2019.

[TSM20] TSMC. *TSMC Announces Intention to Build and Operate an Advanced Semiconductor Fab in the United States*. Hsinchu, Taiwan, May 2020. URL: https://www.tsmc.com/tsmcdotcom/PRListingNewsArchivesAction.do?action=detail&newsid=THGOANPGTH&language=E.

[Tur50] A. M. Turing. "Computing Machinery and Intelligence." In: *Mind* 49 (May 1950), pp. 433–460.

[VC20] B. S. Vogler and D. Cornelisse. *Issue on Reset Network on the nest-simulator GitHub repository*. May 2020. URL: https://github.com/nest/nest-simulator/issues/1618.

[VN16] K. L. Villa and E. Nedivi. "Excitatory and Inhibitory Synaptic Placement and Functional Implications." In: *Dendrites: Development and Disease*. Ed. by K. Emoto, R. Wong, E. Huang, and C. Hoogenraad. Tokyo: Springer Japan, 2016, pp. 467–487. ISBN: 978-4-431-56050-0. DOI: 10.1007/978-4-431-56050-0_18. URL: https://doi.org/10.1007/978-4-431-56050-0_18.

[WB19] J. C. R. Whittington and R. Bogacz. "Theories of Error Back-Propagation in the Brain." In: *Trends in Cognitive Sciences*. 2019.

[WD06] G. V. Wilson and T. Dunne. "Macroscope: Where's the Real Bottleneck in Scientific Computing?" In: *American Scientist* 94.1 (2006), pp. 5–6. ISSN: 00030996. URL: `http://www.jstor.org/stable/27858697`.

[Wei76] J. Weizenbaum. *Computer Power and Human Reason: From Judgement to Calculation*. New York: W. H. Freeman and Company, 1976.

[Wun+19] T. Wunderlich, A. F. Kungl, E. Müller, A. Hartel, Y. Stradmann, S. A. Aamir, A. Grübl, A. Heimbrecht, K. Schreiber, D. Stöckel, C. Pehle, S. Billaudelle, G. Kiene, C. Mauch, J. Schemmel, K. Meier, and M. A. Petrovici. "Demonstrating Advantages of Neuromorphic Computation: A Pilot Study." In: *Frontiers in Neuroscience* 13 (2019), p. 260. ISSN: 1662-453X. DOI: `10.3389/fnins.2019.00260`. URL: `https://www.frontiersin.org/article/10.3389/fnins.2019.00260`.

[YTN16] E. Yavuz, J. Turner, and T. Nowotny. "GeNN: a code generation framework for accelerated brain simulations." In: *Scientific Reports* 6.1 (2016), p. 18854. DOI: `10.1038/srep18854`. URL: `https://doi.org/10.1038/srep18854`.

[ZG14] F. Zenke and W. Gerstner. "Limits to high-speed simulations of spiking neural networks using general-purpose computers." In: *Frontiers in Neuroinformatics* 8 (2014), p. 76. ISSN: 1662-5196. DOI: `10.3389/fninf.2014.00076`. URL: `https://www.frontiersin.org/article/10.3389/fninf.2014.00076`.