

Reinforcement Learning ohne Backpropagation in Neural Regulatory Networks

Eine erste Abschätzung

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Biomedizinische Informatik

eingereicht von

B.Sc. Julian Lemmel

Matrikelnummer 01427667

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu

Wien, 7. Mai 2019

Julian Lemmel

Radu Grosu



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Reinforcement Learning without Backpropagation in Neural Regulatory Networks

A preliminary assessment

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Biomedical Informatics

by

B.Sc. Julian Lemmel

Registration Number 01427667

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu

Vienna, 7th May, 2019

Julian Lemmel

Radu Grosu



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

B.Sc. Julian Lemmel

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. Mai 2019

Julian Lemmel



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Mittels Reinforcement Learning (RL) können Steuerungen für diskrete und kontinuierliche Probleme durch Lernen an Beispielen erschaffen werden. Obwohl RL ursprünglich von der Neurowissenschaft inspiriert wurde, bedienen sich die meisten Ansätze des *backpropagation* Algorithmus, welcher als biologisch unplausibel gilt und somit vermutlich nicht genügt um menschenähnliche Lernfähigkeiten zu erreichen. Einige verschiedene Modelle der *synaptischen Plastizität* wurden in der Neurowissenschaft durch Beobachtung von isolierten Neuronen entwickelt. Solche Modelle könnten als Alternativen zum allgegenwärtigen *backpropagation* Algorithmus dienen, indem sie zur Berechnung von Netzwerkparameteranpassungen verwendet werden. Neural Regulatory Networks (NRNs) sind spezielle RNNs, deren innerer Zustand aufgrund von einem aus der Biologie stammenden Modell berechnet werden. Für diese Diplomarbeit wurde ein Framework nach dem Stand der Technik in RL entwickelt, welches sich solcher NRNs bedient. Das Framework wird getestet, indem es für das Balancieren eines *inversen Pendels* auf einem Wagen trainiert wird. Lernregeln, die auf Modellen der synaptischen Plastizität basieren, werden auf zwei verschiedene Arten integriert: die *custom gradients* Methode zielt darauf ab, die *echten* Gradienten, die mithilfe von *backpropagation* berechnet werden, durch biologisch plausible Berechnungen zu ersetzen; die *plasticity dynamics* Methode erweitert das Modell der NRNs um die synaptische Plastizität, welche somit während der gesamten Laufzeit mitberechnet wird. Beide Methoden wurden mit drei verschiedenen Lernregeln getestet: der *Hebbschen* Regel, *Oja's* Regel und der *bcm* Regel. Die Ergebnisse deuten darauf, dass das Training mithilfe der *BCM* Regel beschleunigt werden kann.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Reinforcement Learning (RL) aims at creating controllers for discrete and continuous problems and was initially inspired by neuroscience. However, the most successful methods are relying on *backpropagation* for calculating the gradients of the loss-function. The *backpropagation* algorithm is considered to be biologically implausible suggesting that it will not suffice when striving for human-like learning abilities. Neuroscience has brought forth different models of *synaptic plasticity* by observing isolated neurons. Such models could serve as alternatives to the ubiquitous *backpropagation* algorithm for calculating changes to network parameters. Neural Regulatory Networks are special RNNs whose inner states are calculated according to dynamics derived from biological observations. In this thesis, a novel framework based on state-of-the-art RL techniques and using NRNs, is introduced and experimented with by applying it to a *cartpole* balancing task. Two different methods of incorporating *learning rules* based on models of synaptic plasticity are investigated: the *custom gradients* method replaces the *real* gradient calculated by *backpropagation* with a biologically plausible synaptic plasticity rule, the *plasticity dynamics* method leaves the gradients unchanged but introduces additional plasticity dynamics that act throughout the entire unrolling of network states. Both methods were tested with three different learning rules: *hebb's* rule, *oja's* rule and the *BCM* rule. The results suggest that training can be accelerated when using the *BCM* rule.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	vii
Abstract	ix
Contents	xi
1 Introduction	1
2 State of the art	3
2.1 Neuroscience	3
2.2 Reinforcement Learning	8
2.3 Neural Regulatory Networks	17
3 Methods	25
3.1 Backpropagation with custom gradients	25
3.2 LTC with Plasticity Dynamics	26
3.3 Learning Rules	26
3.4 RL setting	27
3.5 Implementation	28
4 Results and Discussion	31
4.1 Experiments	31
4.2 Discussion	35
4.3 Outlook	35
List of Figures	37
List of Tables	39
List of Algorithms	39
Bibliography	41



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Introduction

This thesis deals with local *learning rules* derived from models of synaptic plasticity, and their viability when applied to Neural Regulatory Networks. It serves as preliminary work for a subsequent dissertation in the area and sets the stage for ample investigations into the topic.

Over the years, neuroscience has brought forth many different models of synaptic plasticity. They try to explain how synaptic weights are changed over time leading to the formation of new memories. Meanwhile, Deep Reinforcement Learning (DRL) became very successful as agents were trained to excel humans at games like chess and Go. Although Reinforcement Learning (RL) dates back way beyond the advent of DRL, convincing and generalizable results were almost exclusively owed to DRL which is why in this thesis the two abbreviations are used interchangeably. While the algorithms used in DRL were initially inspired by neuroscience, the algorithms mostly depend on *backpropagation* for computing synaptic updates which is biologically implausible. Likewise, contemporary neural network models are a gross oversimplification of biological neural networks and do not entirely capture actual neuronal dynamics, which is addressed by Neural Regulatory Networks (NRNs).

In neuroscience, models of synaptic plasticity ("*learning rules*") are derived from in-vitro or in-vivo (e.g. in the hippocampus) observations of synapses when applying "induction" patterns of activations to afferent neurons. Although such models are often used in *unsupervised* learning, a direct relation between them and learning in RL has not been established yet. What they have in common is the reliance on local information (pre- and post-synaptic activations) only for computing the updates. Hebb's Rule is one example of a simple *rate-based* rule in line with experimental evidence and can be subsumed by the phrase "fire together - wire together". Different *learning rules* and their implications will be introduced later.

A RL framework using NRNs was implemented for this thesis. NRNs are Recurrent Neural Networks (RNNs) whose inner states are computed using neuronal dynamics derived from neuroscience. They are very succinct meaning they need a small amount of neurons while retaining universal approximation capabilities. The framework implements three different *learning rules* that can be applied according to two alternative modes of operation: *custom gradients* and *plasticity dynamics*, which will be introduced in chapter 3. Additionally, it can be trained using regular backpropagation for comparison.

Novel RL approaches are often tested on toy examples that are easily simulated in order to save time when accumulating necessary training samples. Accordingly, a continuous *cartpole* balancing task was used for assessing the framework's performance.

The remainder of this thesis is divided into three chapters: Chapter 2 looks at previous work relevant to this thesis and is further divided into three sections: *Neuroscience*, where we take a look at the mechanisms of learning in biology; *Reinforcement Learning* which offers a short introduction to the history of RL followed by summary of recent publications of interest; and finally *Neural Regulatory Networks* giving a formal definition and elaborating on their relation to contemporary neural network models.

In chapter 3, the framework used for experimentation is introduced and implementation details such as python packages used and hyper-parameter settings are discussed.

Finally, chapter 4 presents the results obtained and explores their implications and possible improvements. All the findings of this thesis are eventually summarized and further possibilities for investigating the novel approach are mentioned.

State of the art

2.1 Neuroscience

2.1.1 Neurons and Synapses

The human nervous system is made up of billions of specialized cells called neurons, that, among other things, allow primates to understand abstract connections in their environment, consider a variety of actions to take and exert fine motor control. Neurons maintain a gradient of ion concentrations (K^+ , Na^+) across their membrane which results in a membrane potential of around $-70mV$ in the resting state (they are "polarized"). They are connected to each other by chemical and electrical synapses. Chemical synapses are located at the intersection between presynaptic (or afferent) axons and postsynaptic (or efferent) dendrites and function through neurotransmitters released into the synaptic cleft that induce changes to the permeability of the postsynaptic membrane, subsequently altering intracellular ion concentration. There are two distinct types of chemical synapses called *excitatory* and *inhibitory*. While the neurotransmitters used by excitatory synapses (e.g. Glutamate) lead to depolarization of the postsynaptic neuron which may result in an Action Potential (AP) that is propagated to downstream synapses, the neurotransmitters of inhibitory synapses (e.g. GABA) cause hyperpolarization which makes the expressions of an AP less likely. Electrical synapses - also called *gap junctions* - function as bidirectional connections between two neurons that allow certain ions to travel between the two. This connection causes the difference in membrane potentials of the two neurons to decrease over time.

Connectionism refers to the theory that attributes the entirety of mental phenomena to neural networks and the synaptic connections therein. Learning is believed to arise through changes in synaptic transmission which can occur both at the synapses, called "*synaptic*" *plasticity*, and the neuron's bodies, referred to as "*intrinsic*" *plasticity*. While the former is the result of various different processes that result in changes to the

synaptic cleft and subsequently in altered signal transmission (or *synaptic strength*), the latter describes changes to the neuronal body including membrane resting potential and capacitance. Together they make up all the mechanisms involved in *neuronal plasticity*. While *synaptic* plasticity has been thoroughly investigated in the past resulting in a wealth of mathematical models of the underlying mechanisms, there is little known about *intrinsic* plasticity. Accordingly, this thesis only deals with *synaptic* plasticity.

2.1.2 Synaptic plasticity

Many different molecular processes are involved in *synaptic plasticity*. Nevertheless, they can be distinguished by how long the effects on signal transmission persist. While short-term plasticity has immediate effects which regress within seconds, long-term plasticity lasts from minutes up to years. [Kan13] Furthermore, the changes can result in either an increase or a decrease in synaptic strength, termed *potentiation* (or enhancement) and *depression* accordingly. Due to their presumed importance for memory and learning, Long-Term-Potentiation (LTP) and Long-Term-Depression (LTD) are being investigated heavily.

Plasticity is activity dependent according to Hebb's theory that was developed by Donald Hebb in the 1940s. Herein he postulated a strength update rule that involved just pre- and postsynaptic activity, i.e. action potential (AP) firing rates. This rule states that synaptic strength will increase whenever pre- and postsynaptic neurons both have high firing rates, and is often rephrased as "fire together, wire together". [Kan13]

However, Hebb's rule is insensitive of time. In the late 1980s a new theory was created refining Hebb's ideas. This new theory was termed Spike-Timing-Dependent-Plasticity (STPD). Accordingly, synaptic updates may depend on the exact timing of pre- and postsynaptic AP spikes. [GKNP14]

Associativity is a term used for strength updates that result in increased transmission as a response to correlated input activity. In other words, synapses will be strengthened whenever their afferent neuron shows high activity coinciding with high activity of neighboring afferent neurons connected to the same efferent neuron. This resembles classical conditioning in a sense.

Finally, synaptic plasticity can be *homo-synaptic* or *hetero-synaptic*. While the induced changes will depend on both the presynaptic and postsynaptic activity in the former case, they only depend on the postsynaptic activity in the latter. Hebbian plasticity for example is *homo-synaptic*. [Kan13]

2.1.3 Cellular mechanisms of synaptic plasticity

Long-Term-Potentiation

The underlying processes of LTP in the CA1 region of the hippocampus in rodents are fairly well understood. Therefore, they will be used as a demonstrative example, as found in [Kan13]. LTP by itself is achieved by a number of different mechanisms that

may act at each synapse. Those mechanisms have certain similarities. In any case the synaptic signal transmission is enhanced by alterations in either the presynaptic or the postsynaptic cell or both. In many cases LTP requires an induction event to occur which triggers a cascade that leads to the expression of alterations. For the CA3-CA1 synapse, these inducing events seem to always be located at the postsynaptic membrane.

LTP Induction Synapses in said region exhibit two types of neurotransmitter receptors, i.e. ligand-gated membrane channels, which both respond to *glutamate: N-methyl-D-aspartate (NMDA)* and *α-amino-β-hydroxy-5-methylisoxazole-4-propionate (AMPA)*. As soon as *glutamate* binds to them, AMPA receptors allow Na^+ and K^+ ions to permeate resulting in a postsynaptic potential. Conversely, NMDA receptors are blocked by Mg^{2+} when the postsynaptic membrane is not excited, i.e. near resting potential. Only upon arrival of a postsynaptic spike the membrane is sufficiently depolarized for the Mg^{2+} to be repelled by electromagnetic force. The NMDA channels will subsequently contribute to the further depolarization of the postsynaptic membrane. Additionally, the NMDA receptors will allow Ca^{2+} to permeate into the cell which induces a cascade eventually leading to LTP expression.

LTP expression In the early stage of LTP expression, the increase in Ca^{2+} at the postsynaptic spine triggers downstream signalling pathways that include *calcium/calmodulin*-dependent protein kinase II (CaMKII), protein kinase C (PKC), and *tyrosine* kinases. This leads to phosphorylation of the AMPA receptors and subsequently to an increase in their permeability. Moreover, endosomal vesicles carrying new AMPA receptors will fuse into the membrane upon their phosphorylation. This ultimately leads to an increased sensitivity to glutamate at the postsynaptic membrane. If the inducing event was strong enough, late LTP will occur which leads to synthesis of new mRNAs and proteins through the cAMP and PKA signaling pathway. This results in the formation of new dendritic spines.

There are also mechanisms acting at the presynaptic cell after induction of LTP. Although it is unclear how the occurrence of induction at the postsynaptic membrane is signaled back to the presynaptic cell, there is certain evidence for the existence of retrograde messenger molecules, such as *endocannabinoids*, that are released into the synaptic cleft. [MI95] LTP happens due to the enhanced transmitter release that is initiated by the retrograde signal.

2.1.4 Models of synaptic plasticity

A computable mathematical expression for synaptic strength updates enables the construction of computer simulations that provide evidence to be compared with experimental data. For reasons of practicality, phenomenological models of synaptic plasticity that treat the underlying biochemical processes as a black-box are preferable to holistic models that try to encapsulate every single molecular interaction, which are far too complicated for efficient computation.

There exist two different approaches to modeling synaptic plasticity: rate based models and spike-time-dependent-plasticity (STDP) models. In the former case, synaptic updates depend on pre- and postsynaptic activities while disregarding detailed timing of spikes. Conversely, spike timing is a relevant factor in the latter case. [Sho07]

Rate Based Models In a general sense, rate based plasticity rules take the following form [GKNP14]:

$$\frac{d w_{ij}}{dt} = F(v_i, v_j, w_{ij}, r) \quad (2.1)$$

where w_{ij} is the synaptic strength of the synapse connecting neurons i and j , v_i and v_j are their respective firing rates, and r can for example denote a numerical reward, which could correspond to dopaminergic excitation. Note that this does not include any information on individual spike timing.

Assuming that F is sufficiently well behaved the function can be expanded in a Taylor series around $v_i = v_j = 0$ as shown below. [GKNP14] All the equations presented in the remainder of this section are special cases of the Taylor series having different constants c_i set to 0.

$$\frac{d w_{ij}}{dt} = c_0(w_{ij}) + c_1^{pre}(w_{ij})v_j + c_1^{post}(w_{ij})v_i + c_2^{pre}(w_{ij})v_j^2 + c_2^{post}(w_{ij})v_i^2 + c_{11}^{corr}(w_{ij})v_i v_j + \mathcal{O}(v^3)$$

Hebb's rule is our first example of a rate-based model of synaptic plasticity which, in its simplest form, looks as follows:

$$\frac{d w_{ij}}{dt} = \gamma v_i v_j \quad (2.2)$$

γ has to be > 0 for the update rule to be Hebbian, If $\gamma < 0$ it is called anti-Hebbian as synaptic strength decreases upon correlated activity. This rule allows for arbitrarily large synaptic strengths which is why an upper limit needs to be enforced by introducing a hard or soft bound. [GKNP14]

Oja's rule is similar to Hebb's rule but introduces a quadratic regularization term that depends on the weights. It is meant to make the afferent synaptic weights of each neuron converge to being normalized to $\sum_j w_{ij}^2 = 1$. This implies competition between afferent synapses. [GKNP14]

$$\frac{d w_{ij}}{dt} = \gamma (v_i v_j - w_{ij} v_i^2) \quad (2.3)$$

Another rate based model was developed and named after Bienenstock, Cooper and Munro (**BCM**). While Hebb's rule did only address LTP, the BCM model does also

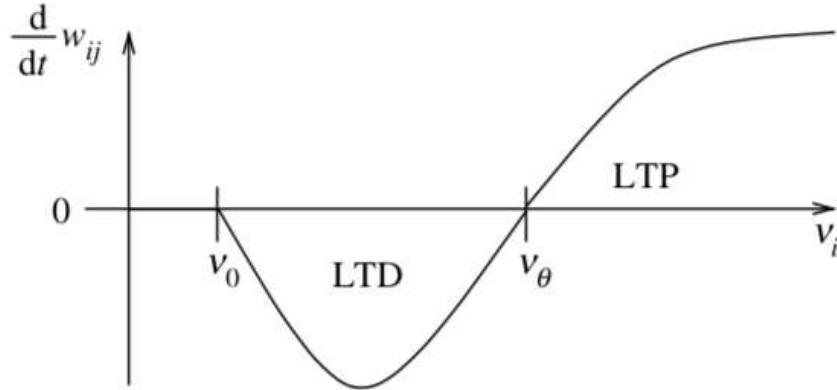


Figure 2.1: BCM update rule: Synaptic strength is weakened if postsynaptic activity is below threshold v_θ . Below v_0 no synaptic modification occurs. Source: [GKNP14]

allow for LTD whenever presynaptic activity is matched with just moderate postsynaptic activity. It also has an inherent soft bound limiting synaptic growth. See the plot in Figure 2.1. The following common expression ensures the desired behavior:

$$\frac{d w_{ij}}{dt} = \gamma v_i (v_i - v_\theta) v_j \quad (2.4)$$

v_θ is a reference rate marking the threshold between LTD and LTP and is computed as a function of the average output firing rate. The BCM model has e.g. been used to describe orientation selectivity in receptive fields. [GKNP14]

Spike-Time-Dependent-Plasticity In STDP models, the relative timing of pre- and postsynaptic spikes is used for computing the updates. These types of learning rules are not considered in this thesis but will be explained shortly for completeness. The simplest approach is to consider a pair-based update rule which computes updates each time a pre- or postsynaptic spike is fired. Updates are then dependent on the temporal difference between the two: $\delta t = t_{post} - t_{pre}$, t_{post} and t_{pre} being the spike times. Due to the non-continuity at $\delta t = 0$ (shift from LTD to LTP) the rule consists of a positive and a negative part:

$$\begin{aligned} \delta w_+ &= A_+(w) e^{-\frac{|\delta t|}{\tau_+}} && \text{at the postsynaptic spike for } t_{pre} < t_{post} \\ \delta w_- &= A_-(w) e^{-\frac{|\delta t|}{\tau_-}} && \text{at the presynaptic spike for } t_{post} < t_{pre} \end{aligned}$$

$A_\pm(w)$ is a factor depending on the weight itself and τ_\pm determines the slope of the update magnitude, i.e. how small δt has to be for the update to still have significance.

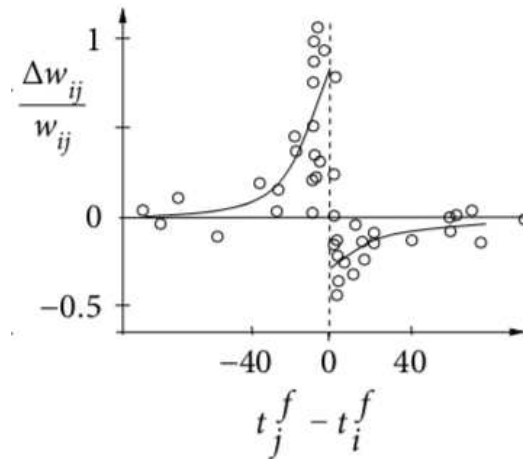


Figure 2.2: Pair-based update rule (lines) plotted alongside experimental data (circles).
Source: [GKNP14]

This rule models behavior observed in pairing experiments with cultured hippocampal neurons. Figure 2.2 shows both the plot of the function and experimental data for comparison. [GKNP14]

2.2 Reinforcement Learning

Reinforcement Learning (RL) and especially Deep Reinforcement Learning (DRL) is a branch of Machine Learning (ML) and constitutes the state of the art in automated controller synthesis. In this section, the most important concepts are shortly introduced by taking a look at the history of RL, followed by a closer look at more recent developments that are relevant to this thesis. Finally, the parallels between RL and Neuroscience are shown and the implausibility of *backpropagation* is discussed alongside its most popular alternatives.

2.2.1 History of RL

The term *Reinforcement* was initially described by E. Thorndike at the end of the 19th century. He used the term for modes of conditioning that led to an increase in behavior (in contrast to *Punishment*). Later, it was extensively investigated by B.F. Skinner who introduced the term *operant conditioning* subsuming both types of conditioning. Considered a field of psychology thereafter it was not until the 1980s that it was discovered by Computer Scientists as a potential learning scheme for machines. From this point on the term *Reinforcement Learning (RL)* was used due to the way feedback was given to the machine. It was heavily influenced by statistics and control theory, most notably

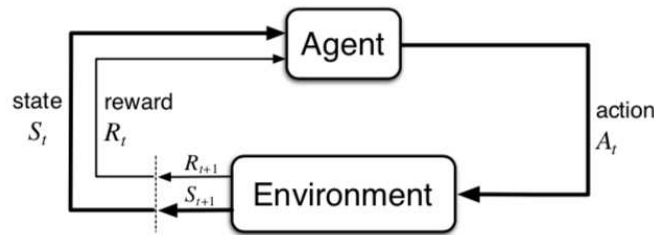


Figure 2.3: *Markov Decision Process*: the agent finds itself in *state* S_t and takes *action* A_t which makes the environment go to *state* S_{t+1} and yields a *reward* R_{t+1}

by the work of Richard Bellman. [Bel53] The textbook “Reinforcement Learning: an introduction” by Sutton and Barto constitutes a milestone in the field.[SB98]

Markov Decision Process A task that can be solved using RL can be modeled as a Markov-Decision-Process (**MDP**) which is the discrete stochastic version of the optimal control problem. While optimal control in a fully observable environment can be achieved by solving the underlying differential equations, MDPs can be addressed, for example, by means of Markov Chains (or Hidden Markov Chains, in case of partially observable MDPs).

MDPs require sequential decision making from the *agent* navigating an *environment*. After each timestep (passage of fixed amount of time), the agent has to decide what to do based on what it sees and what has happened so far. After taking an *action* the agent is given a *reward* evaluating its decision. Additionally, the agent receives the next *state* of the *environment*, now affected by its *action*. The agent has to figure out which *actions* to take in order to maximize the *reward*. [SB98] See Figure 2.3 for a visual representation.

An MDP consists of a set of *states* - one of which is the *initial state*, a set of *actions* an agent can take, the *state transition* function which determines the probability of ending up in *state* s' after taking *action* a in *state* s , and a reward function which assigns a numerical reward to each *action* in every *state*.

Table 2.1: *Markov Decision Process* formal definition

S	set of all states
s_0	initial state
A	set of actions
$R : S \times A \mapsto \mathbb{R}$	reward function
$T : S \times A \times S \mapsto \mathbb{R}$	state transition function

For a simple example of an MDP consider a robot that has to navigate on a chessboard-like map containing obstacles (tiles that cannot be walked on). The possible actions would be to move up, right, down or left. A reward is given each time the robot reaches a goal tile. In order to punish useless detours, a small negative reward is given each

time the robot does a move. Suitable RL algorithms will find an agent that maximizes the total reward by choosing the shortest path to the goal tile while avoiding all the obstacles.

Initially, such MDPs were solved by *tabular* methods: in a table, the agent keeps track of all the *states* it has visited, of all the *actions* it took inside those *states* (at first they are chosen at random), and of all the *rewards* received afterwards. Those recordings can then be used to estimate the *values* of *states* and *actions* which later lead to more sophisticated decisions. Unfortunately, tabular methods are limited to discrete environments only. Continuous environments (like the world we live in) do have an infinite number of possible *states* and, depending on the task, an infinite number of *actions*. Tabular methods fail on continuous problems as this would require infinitely large tables to store all the information needed.

Consider the following example of a continuous MDP. A pole mounted on top of a cart has to be balanced upright (See Figure 2.4). The *state* of the environment is summarized in a vector containing the cart's position, its velocity, the pole's angle, and the pole's angular velocity. A *reward* of 1 is given at each time-step unless the cart's position or the pole's angle leave the predefined bounds which terminates the *trial*. The agent controlling the cart has to decide in which direction to push the cart and the magnitude of the push (e.g. pick a value inside the range $[-1,+1]$). Since the *state* and *action* values are real numbers the *state-space* as well as the *action-space* are continuous.

In order to handle continuous spaces, parameterized functions are used. One example is the policy $\pi_{\theta}(s)$ function which computes the *action* to take in each *state* according to the parameters θ . Another example is the *value* function $V_{\theta}(s)$ where the *value* represents the maximum expected *reward* that can be gathered from this *state* on. Here, artificial neural networks (ANNs) come into play since they are basically non-linear function

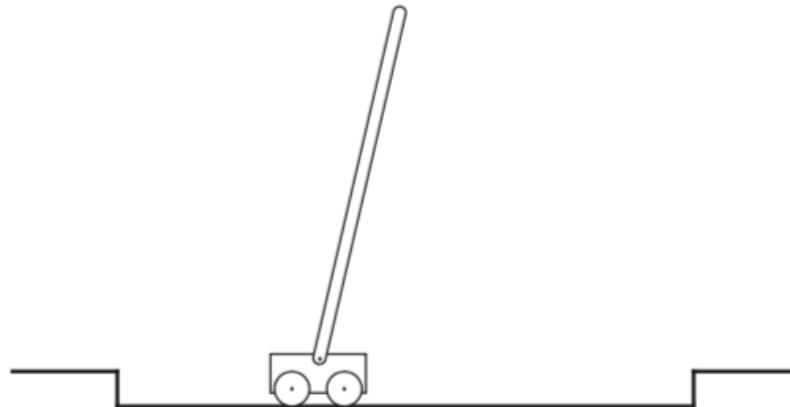


Figure 2.4: Pole-balancing: a continuous MDP

approximators with the synaptic weights being their parameters. The *policy*-function can subsequently be thought of as an ANN taking the current *state* as input and returning the *action* to take. Those ANNs can be trained using backpropagation similarly to Supervised Learning. [SB98]

Recurrent Neural Networks (RNNs) are a special case of ANNs where connections between the nodes form a directed graph that may contain cycles. They can process sequences of data while retaining an inner state ("memory") and they therefore exhibit temporal dynamic behavior which enables them to make accurate time-series predictions. Arguably, the most famous RNNs are the LSTM networks that were invented by Hochreiter and Schmidhuber in 1997 and which are ubiquitous in recent speech recognition applications. [HS97]

If the *state* does not exhaustively capture all the variables needed for describing the underlying physics of an environment one speaks of *Partially Observable (PO)*MDPs. The need for extracting information from the sequence of preceding *states* makes them considerably harder to solve than MDPs, although in many cases, RNNs provide satisfactory solutions. The *cartpole* task for instance, becomes a POMDP by taking just the cart's position and the pole's angle as the *state*.

Bellman equation The *policy* and *value* functions mentioned above are related to each other via the Bellman equation [Bel53] which recursively defines the *value* in terms of the *reward* of the current *state*, the *policy*, the *transfer* function and the *value* of its successor states:

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_{\pi}(s')] \quad (2.5)$$

$v_{\pi}(s)$	is the state-value function
$\pi(a s)$	denotes the stochastic policy
$p(s',r s,a)$	denotes the stochastic transfer function
γ	is a discount factor

In the equation above, the *state-value* gives the numerical *value* of *state* s , the stochastic policy returns the probability of doing *action* a in *state* s and the stochastic transfer function gives the probability of ending up in *state* s' and receiving *reward* r after doing *action* a in *state* s . The discount factor $0 \leq \gamma \leq 1$ decides how strongly the algorithm favors short-term rewards. Equation 2.5 serves as a basis for many different algorithms used in RL. A *policy* is called optimal if it maximizes the *value* of each *state*. While an MDP can have more than one optimal *policy*, they all share the same optimal *value* function

Policy Iteration Utilizing this equation, Bellman introduced the first Reinforcement Learning algorithms called dynamic programming (DP). An optimal *policy* can be found by means of the *policy iteration* algorithm which alternates between calculating a *value*-function with a fixed *policy* and using that obtained *value*-function to update the *policy*.

While *policy* iteration is an example for an *on-policy* algorithm there are different approaches that work in an *off-policy* setting, which means the agent can be taught using samples from any *policy* acting on the environment. For an algorithm to be successful it has to make a trade-off between *exploration* and *exploitation*. This means new *actions* have to be explored in order to find better *policies* but well-working *actions* should not be disregarded at the same time. A *policy* is called *greedy* if it always chooses the action considered best, meaning that no *exploration* takes place at all.

Notice, that all of the DP methods require an explicit *model* (i.e. *transfer function*) of the underlying MDP. This limitation is not present when using *model-free* approaches like Monte Carlo methods or Temporal-Difference Learning, for which it is sufficient to merely create samples of the underlying distribution.

Monte Carlo, TD and Policy Gradient Monte Carlo methods use generated sample episodes (also called *trials*) to learn *value*-functions and optimal *policies*. This is a significant advantage over DP as sample models are easier to construct than explicit *transition* models. It is worth noting that Monte Carlo methods do not use "*bootstrapping*", which means calculating approximations on the basis of approximations. DP, in contrast, is using *bootstrapping* as in each cycle a *value*-function is approximated using an approximated optimal *policy* and vice-versa. (This is also referred to as "*semi-gradient*")

Temporal-Difference (TD) Learning combines ideas from DP and Monte Carlo methods in the sense that it works by learning from experience without the need of a model, but does also use bootstrapping. TD algorithms are on-line which make them applicable to a wider range of problems as they do not rely on completing an entire episode prior to computing the updates. After each *action*, the gained experience is used to update the *value* estimates. Equation 2.6 defines the *TD-error* for *state-value* prediction that is used in TD-learning. It is a measure of how accurately the reward received at a particular time-step was predicted.

$$\delta = r + \gamma \hat{v}_\theta(s') - \hat{v}_\theta(s) \quad (2.6)$$

One prominent TD-algorithm is called *Q-Learning* and uses a *state-action-value* function instead of the *state-value* $Q_\theta(s, a)$ function introduced above. Using *values* for (*state*, *action*) pairs makes Q-Learning off-policy since said *values* are independent of the active *policy*.

Finally, Policy Gradient (PG) methods can be used for environments with continuous *state*-spaces. The *policy* is in this case a parameterized (non-linear) function. Parameters

are in most cases the weights of a neural network. By constructing a performance measure J and taking its gradient with respect to the parameters θ the network can be trained using gradient ascent. This approach learns a policy without the need of a *value*-function. Nonetheless, the most successful Policy Gradient algorithms, e.g. actor-critic, do learn an additional *value*-function which assures faster convergence.

The Actor-Critic algorithm A particularly interesting Policy Gradient algorithm is the so-called *actor-critic* (AC) algorithm. This algorithm makes use of both a *value* function and a *policy*. The *policy* is also called the *actor* as it is used for computing the *action* and the *value* function is called *critic* since it is used for evaluating the *actor's* decisions. After taking the *action*, the *value* function is then used to assess the *values* of the old *state* and the resulting new *state*. Those two *values*, together with the *reward*, are used to calculate the *TD-error*.

The *TD-error* is a measure for the accuracy of the *reward*-prediction - therefore also considered a Reward-Prediction-Error (RPE) - and is used for updating both the *actor* and the *critic*, i.e. it acts as a reinforcement signal. [SB98] Note the difference between the *reward* signal and the *TD-error* as a reinforcement signal: if the *reward* is predicted perfectly by the *value* function no reinforcement takes place whereas the absence of a predicted *reward* leads to negative reinforcement. Although *actor-critic* was devised through pure mathematical considerations, the algorithm interestingly very much resembles the dopamine feedback-loops found in the human brain.

2.2.2 Deterministic AC variants

The basic *actor-critic* algorithm is a PG method in which a value and a *stochastic* policy function are trained in parallel. *Stochastic* in this context means that the outputs of the *actor* specify the probability distribution from which the *action* is sampled. On the other hand, *deterministic* policies compute the exact action to take. When using a deterministic policy, noise is usually added at training time in order to ensure adequate *exploration*.

Deep Deterministic Policy Gradient One special derivative of *actor-critic* is the Deep Deterministic Policy Gradient (DDPG) algorithm. This variant can deal with continuous action spaces and, as the name suggests, it uses a *deterministic* policy as opposed to a *stochastic* one. [LHP⁺19]

DDPG uses Q-Learning to approximate the optimal *action-value* function Q^* which is defined analogous to equation 2.5, although given as expected value here.

$$Q^*(s, a) = \mathbb{E}_{(s', r) \sim p(s, a)} \left[r + \gamma \max_{a'} Q^*(s', a') \right] \quad (2.7)$$

From this, the **mean-squared Bellman error (MSBE)** for an approximate *action-value* function Q_θ is derived.

$$MSBE(\theta, \mathcal{B}) = \mathbb{E}_{(s,a,r,s',d) \sim \mathcal{B}} \left[\left(Q_\theta(s, a) - (r + \gamma(1-d) \max_{a'} Q_\theta(s', a')) \right)^2 \right] \quad (2.8)$$

where \mathcal{B} is a batch of samples taken from a **replay buffer** and $d = 1$ if s' is *terminal*, 0 otherwise. The use of Q-Learning makes DDPG an off-policy algorithm meaning that the replay buffer should contain old experience collected with outdated *policies*. The replay buffer size should be chosen as to incorporate a wide range of experience in order to avoid overfitting to the most recent data, while not slowing down learning. By minimizing $MSBE(\theta, \mathcal{B})$ the optimal *action-value* function is approximated.

Since the updates for θ computed from 2.8 depend on θ itself, learning becomes unstable. This can be avoided by maintaining additional **target networks** $Q_{\theta'}$, $\pi_{\phi'}$ that lag behind the ones used for optimization. The target networks are updated using exponential averaging:

$$\theta' = \rho\theta' + (1 - \rho)\theta \quad \rho \in [0, 1] \quad (2.9)$$

When using target networks for both functions in equation 2.8 we arrive at the *loss* function used in DDPG:

$$\mathcal{L}(\theta, \mathcal{B}) = \mathbb{E}_{(s,a,r,s',d) \sim \mathcal{B}} \left[\left(Q_\theta(s, a) - (r + \gamma(1-d)Q_{\theta'}(s', \pi_{\phi'}(s'))) \right)^2 \right] \quad (2.10)$$

The updates for θ are computed by gradient descent on the *loss*.

$$\theta \leftarrow \theta - \eta \frac{d\mathcal{L}(\theta, \mathcal{B})}{d\theta} \quad (2.11)$$

Finally, the parameters of policy $\pi_\phi(s)$ are updated by gradient ascent on the *action-value* $Q_\theta(s, \pi_\phi(s))$:

$$\phi \leftarrow \phi + \eta \frac{dQ_\theta(s, \pi_\phi(s))}{d\theta} \quad (2.12)$$

Twin-Delayed-DDPG (TD3) DDPG is prone to drastically overestimating Q^* which leads to undesirable learning behavior. This issue is addressed by TD3 which introduces three changes: First, TD3 learns *two* Q -functions and uses the lower of the two when calculating the MSBE. Second, the *policy* π_ϕ is updated less frequently than the Q -functions (usually one *policy* update for two Q -function updates) assuring more stable updates for θ . Third, noise is added to the target action $\pi_{\phi'}$ in order to to make exploitation of Q -function errors less likely.[FvM18]

2.2.3 Eligibility traces

Eligibility traces are another central concept of reinforcement learning. Often, *rewards* are delayed in RL tasks meaning they are not given immediately after the *action* accountable for them, but instead later on after some insignificant *actions*. This does not pose a threat to off-line algorithms (e.g. *Monte Carlo* methods) since each weight update is computed after a complete episode where all the future rewards are known. However, on-line algorithms need some way of attributing *rewards* to past *actions*. One possibility is to delay the calculations of updates for a fixed number of steps which is called *n*-step bootstrapping. However, this approach comes with significant computational drawbacks.

Another way of factoring in future *rewards* is by using **eligibility traces**. The main idea is to keep a running average of each synapse's contribution to the network's output which can be thought of a *short-term* memory paralleling the *long-term* memory represented by the synapse weights. Eligibility traces can be used to unify and generalize Monte Carlo and TD methods. [SB98]

For example, the TD(λ) algorithm makes use of eligibility traces multiplied with the TD-error (2.6) for computing the weight updates. By accumulating the gradients of the *state-value* function, as can be seen in equation 2.13, each weight's contribution is recorded. The trace \mathbf{e} decays with factor $\gamma\lambda$ where γ is the discount factor of equation 2.6.

$$\begin{aligned}\mathbf{e} &= \gamma\lambda \mathbf{e} + \nabla \hat{v}_\theta(s) \\ \theta &\leftarrow \theta + \eta \delta \mathbf{e}\end{aligned}\tag{2.13}$$

Johard and Ruffaldi used two separate eligibility traces to approximate the gradient of the discounted reward in a connectionist actor-critic setting. [JR14] They used a modified cascade-correlation algorithm for training which outperformed regular PG.

2.2.4 Correspondence to Neuroscience

Dopamine and Learning Dopamine is a neurotransmitter found in the central nervous system of mammals. It is widely agreed upon that dopamine plays an important role in rewards and reinforcement. The molecule is released by specialized neurons located primarily in two areas of the brain: the substantia nigra zona compacta (SNc) and the ventral tegmental area (VTA). [Wis04] These neurons have large branching axonal arbors making synapses with many other neurons, mostly in the striatum and the pre-frontal cortex. Those synapses are usually located at the dendritic stems of glutamate synapse spines and can therefore effectively influence synaptic plasticity. [SB98]

Whenever receiving an unexpected reward, dopamine is released which subsequently reinforces the behavior that led to the reward. However, if the reward is preceded by a conditioned stimulus, then it is released as soon as the stimulus occurs instead of when receiving the expected reward. If then the expected reward is absent, levels will even

drop below baseline. [Wis04] Various experiments have shown that dopamine is crucial for stamping-in response-reward and stimulus-reward associations which in turn is needed for motivation when confronted with the same task in the future. For example, moderate doses of dopamine antagonists (neuroleptics) will reduce motivation to act. Habitual responding declines progressively in animals that are treated with neuroleptics. [Wis04]

Brain structures There is evidence that certain brain structures could be implementing *actor-critic* methods. The striatum is involved in “both motor and action planning, decision-making, motivation, reinforcement, and reward perception” [noa19] and it is also heavily innervated by dopamine axons coming from the VTA and the SNc. It is speculated that dopamine release in the ventral striatum “energizes the next response” while in the dorsal striatum it acts by stamping in the procedural memory trace, “establishing and maintaining procedural habit structures“ [Wis04]. Subsequently, the ventral striatum would correspond to the *critic* and the dorsal striatum to the *actor* of the algorithm [SB98]. Dopamine would then correspond to the *TD-error* which is used to update both the *actor* and the *critic*. As dopamine neuron axons target both the ventral and dorsal striatum, and dopamine appears to be critical for synaptic plasticity, the similarities are evident. Furthermore, the *TD-error* and dopamine levels are both encoding the RPE: they are high whenever an unexpected reward is received and they are low (or negative in case of the *TD-error*) when an expected reward does not occur. [SB98] These similarities could be beneficial for RL as well as for Neuroscience as advances in either field could lead to new insights beneficial to the other.

2.2.5 Implausibility and alternatives to Backpropagation

In order to do gradient descent (ascent), gradients are calculated using *backpropagation*, or *backpropagation through time* (BPTT) in case of RNNs. While *backpropagation* allows for mathematically proven solutions and has many working real world applications, it also has some drawbacks. It uses global rather than local information for computing the updates which means they cannot be computed in parallel.

More importantly, there is no evidence for *backpropagation* in biology since it requires some characteristics not found in biological neural networks. The two main objections are the necessity for: (1) *shared weights* for forward and backward connections and (2) *reciprocal error transport* meaning that there is a form of propagating back the errors without interfering with neural activity. [BSR⁺18]

Admittedly, a variety of alternatives to *backpropagation* exists and the search for biologically plausible learning algorithms remains subject of ample investigation. There are two alternatives to *backpropagation* worthy of mentioning:

Feedback Alignment (FA) By using random weights in the backward propagation, FA gets rid of the need for *shared weights* listed above, while still being able to learn the objective. See figure 2.5 for a visual representation. Note that FA still makes use of *reciprocal error transport*. [BSR⁺18]

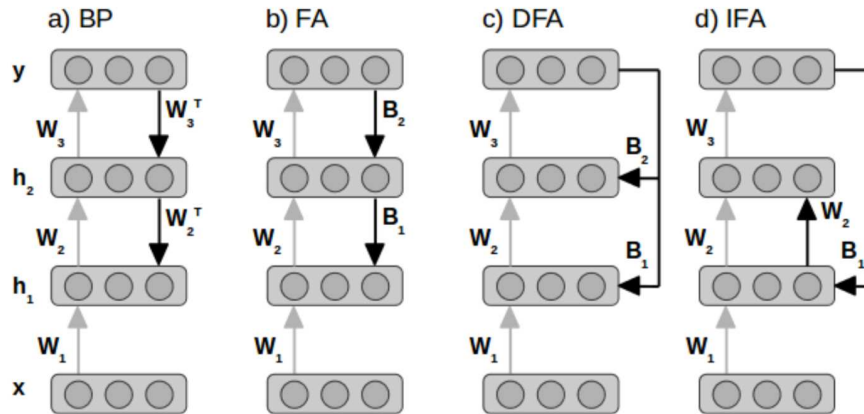


Figure 2.5: Error transport in FA and its variants: a) backpropagation uses weights symmetric to the forward weights W_i ; b) FA uses fixed random weights B_i per layer; c, d) Direct and Indirect FA are variants that project back from the output layer only.

Source: [Nøk16]

Target Propagation (TP) In TP *target activities* are propagated back instead of the errors which rids it of biologically implausible weight transport. The propagation is realized through learned connections that are different than the weights used in the forward pass. These weights are trained to approximately invert the forward computation, similar to auto-encoders, in order to calculate target activities for the preceding layers. Accordingly, each layer tries to minimize two losses at once: a *reconstruction* loss, e.g. MSE between output and inverted input - and a *forward* loss, e.g. MSE between output and target. [BSR⁺18]

Both approaches have been shown to accomplish test accuracies comparable to backpropagation for simple supervised learning tasks but have not been successfully applied to more sophisticated problems yet. [BSR⁺18]

2.3 Neural Regulatory Networks

A Neural Regulatory Network (NRN) is a biophysical model of non-spiking neurons as found in the retina of larger species or in small species, like the *C. elegans* roundworm. NRNs, also referred to as Liquid-Time-Constant (LTC-)RNNs [HLA⁺18], constitute a special case of *continuous time* RNNs but were shown to be more succinct when used for function approximation. [Gro20] Hasani et. al. showed that LTC-RNNs are nonetheless universal function approximators. [HLA⁺18]

In this section the LTC-RNN model will be introduced, after which it is shown how NRNs generalize other models of neural networks.

2.3.1 Liquid-Time-Constant RNNs

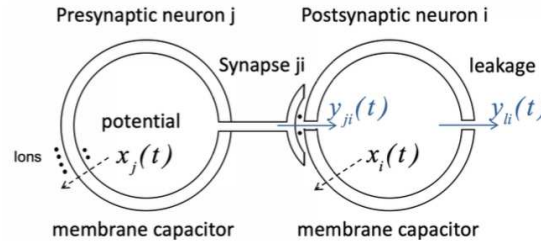


Figure 2.6: schematic of a biological synapse

A cell in a LTC-RNN models the membrane potential dynamics of one biological cell that is connected to other cells via chemical synapses and gap junctions as shown in Figure 2.6. *Note:* Gap junctions are omitted in the figure. They function as resistor connecting the membrane capacitors.

The membrane potential v_i constitutes the cell's state and is governed by the *single-compartment membrane equation* as in 2.14. [KS98]

$$\begin{aligned}
 C_i^m \frac{dv_j}{dt} &= I_j^{leak} + \sum_{i=1}^n I_{i,j}^{syn} + \sum_{i=1}^n I_{i,j}^{gap} \\
 I_i^{leak} &= G_i^{leak} (E_i^{leak} - v_i(t)) \\
 I_{i,j}^{gap} &= G_{i,j}^{gap} (v_i(t) - v_j(t)) \\
 I_{i,j}^{syn} &= G_{i,j}^{syn} (E_{i,j}^{syn} - v_j(t)) \text{ sig}_{i,j}(v_i(t)) \\
 \text{sig}_{i,j}(v) &= \frac{1}{1 + e^{-\sigma_{i,j}(v - \mu_{i,j})}}
 \end{aligned} \tag{2.14}$$

I_i^{leak} , $I_{i,j}^{syn}$ and $I_{i,j}^{gap}$ are the input currents due to chemical synapses and gap junctions respectively and $\text{sig}(t)$ is the sigmoid function. Equations 2.14 constitute the model of neuronal dynamics that will be used throughout this thesis.

Each cell is associated with 3 *intrinsic* parameters and each synapse with 5 *synaptic* parameters (4 for each chemical + 1 for each electrical synapse) as shown in table 2.2. The total number of parameters in a network of a fully connected LTC-RNN of size n is $5n^2 + 3n$.

The name "Liquid-Time-Constant" stems from the fact that equations 2.14 can be reformulated to represent an ODE system with a nonlinearly varying time-constant

$$\tau_{system} = \frac{1}{1 + G_i^{leak}/C_i^m + G_{i,j}^{syn}/C_i^m \text{sig}(\sigma_{i,j}(v_i(t) - \mu_{i,j}))} \text{ when omitting gap junctions. [HLA}^+18]$$

Table 2.2: LTC-RNN parameters

	<i>intrinsic</i>
C_i^m	Membrane capacitance [F]
G_i^{leak}	Leak conductance [S]
E_i^{leak}	Leak potential [mV]
	<i>synaptic</i>
$G_{i,j}^{gap}$	Conductance of gap junctions [S]
$G_{i,j}^{syn}$	Conductance of chemical synapses [S]
$E_{i,j}^{syn}$	Post-synaptic potential of chemical synapses [mV]
$\sigma_{i,j}$	Steepness of synaptic activation
$\mu_{i,j}$	Center of synaptic activation [mV]

2.3.2 From ANN to NRN

This section summarizes the findings from the paper by Grosu et. al. [Gro20] starting with traditional Artificial Neural Networks and gradually refining them until we arrive at NRNs. Along the way it will become clear why NRNs are biologically more accurate than other network models.

Single Layer Perceptron Let's first consider a single neuronal layer as depicted in Figure 2.7. A forward pass is computed using the following equations

$$y_i^{t+1} = \sigma\left(\sum_{j=1}^n w_{ji}^t y_j^t, \mu_i^t\right) \quad \sigma(x, \mu) = \frac{1}{(1 + e^{-(x-\mu)})} \quad (2.15)$$

where y_i^t is the activation of neuron i at time t , w_{ji} is the synaptic weight of the synapse of neuron j onto neuron i , σ is any non-linear function such as the sigmoid function given here, whose center is determined by μ_i .

The weighted sum of the inputs from layer t is passed through the non-linear function, resulting in the output of layer $t+1$. This is done for each neuron of the layer. This basic cell is biologically implausible as synapses show non-linear behavior and cell bodies (i.e. axon hillocks) do the summation of the currents. It is also well known that a single Layer of such Artificial Neurons is not capable of expressing arbitrary functions (e.g. XOR).

Multi Layer Perceptron By adding an additional Layer, the flaw in expressiveness is fixed and also the biophysical incorrectness is mitigated. The resulting computation is sketched in Figure 2.8. Shifting the borders of the neurons hypothetically makes clear why the biological flaws are now corrected: the sigmoid function at each layer represents the accumulation of currents at the dendrites followed by the summation that can be considered to model the neuron's body.

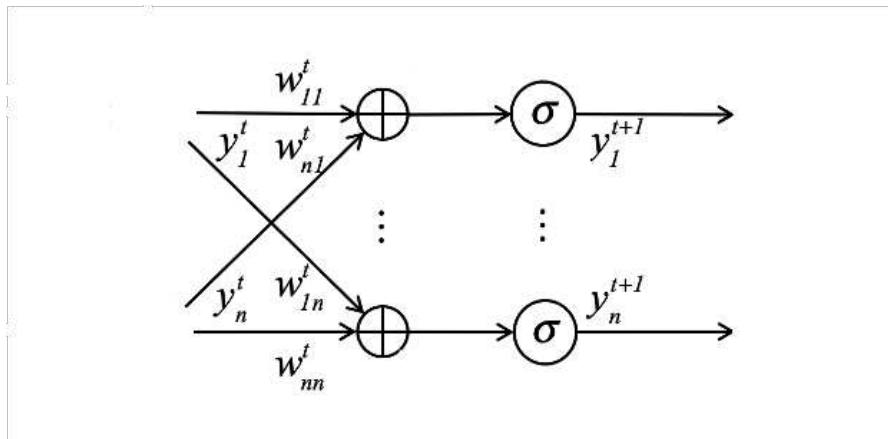


Figure 2.7: schematic of a single layer neural network

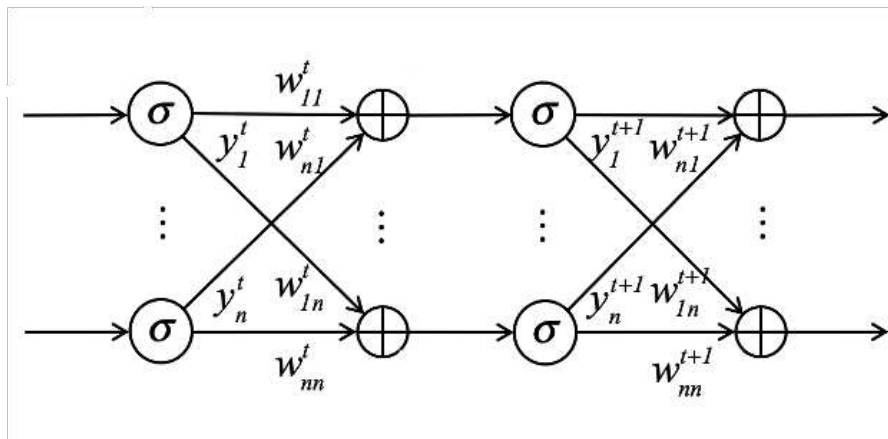


Figure 2.8: schematic of a multi layer neural network

ResNets When adding additional “skip” connections as depicted in Figure 2.9 we arrive at ResNets which were introduced in order to combat the degradation problem. The skip connections are represented by the blue arrows. The input of Layer t is added to the output. These additional connections actually lead to learning the parameters for the derivative of the actual function. This is visible when looking at the mathematical formulation.

$$x_i^{t+1} = x_i^t + \sum_{j=1}^n y_{ji}^t \quad y_{ji}^t = w_{ji}^t \sigma(x_j^t, \mu_j^t) \quad (2.16)$$

NeuralODEs If we now consider each layer to be a step in time and all the neurons of each layer to be the same the architecture resembles NeuralODEs. In the following

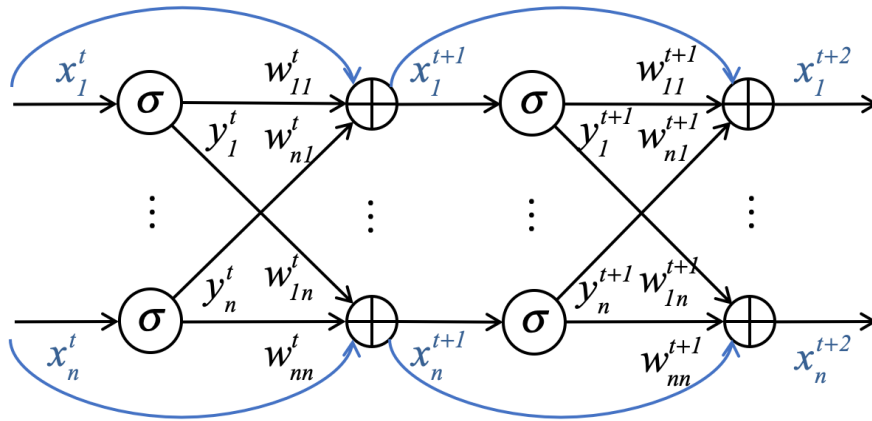


Figure 2.9: schematic of a ResNet layer

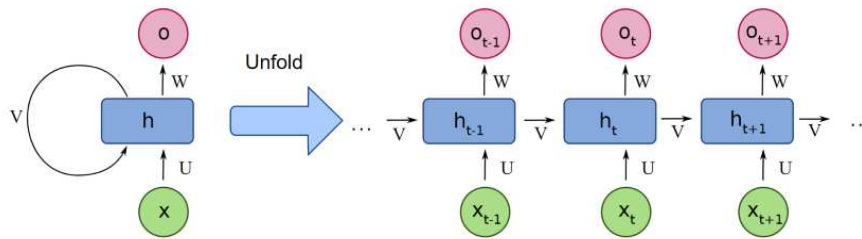


Figure 2.10: NeuralODEs are unfolded by repeatedly computing the activation function using the last outputs as additional inputs.

equations the activations of the neurons are represented as functions in time as opposed to indices for the layer number.

$$x_i(t) = \sum_{j=1}^n y_{ji}(t) \quad y_{ji}(t) = w_{ji}\sigma(x_j(t), \mu_j) \quad (2.17)$$

NeuralODEs take the same weights for every layer and a ResNet is obtained after unrolling the NeuralODE (See figure 2.10 for a visual representation). This makes NeuralODEs N times more succinct than ResNets, where N is the number of time-steps (or layers). This means N times less parameters have to be learned. However, learning might be harder than in ResNets.

CT-RNNs are a special kind of NeuralODEs that add an additional stabilisation term which ensures that the output returns to a resting state when no input is present.

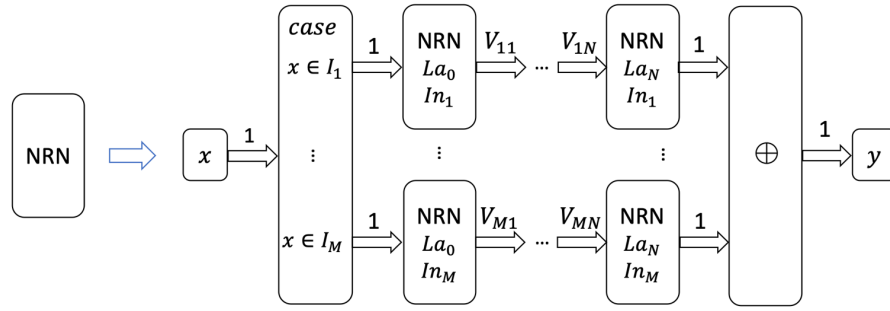


Figure 2.11: NRNs are unrolled into ResNets whose weights depend on the input. Each row shows one possible unfolding. I_j are the input partitions, La_j are the layers, In_j is the input and V_{ij} are the weights of each layer

$$\dot{x}_i(t) = -w_i x_i(t) + \sum_{j=1}^n y_{ji}(t) \quad y_{ji}(t) = w_{ji} \sigma(x_j(t), \mu_j) \quad (2.18)$$

NRNs Finally, Neural Regulatory Networks are similar to CT-RNNs. The output of each layer and the leakage term are additionally multiplied by a difference of potentials which leads to currents. They are therefore in accordance with the biophysical model of non-spiking neurons. Note that in equation 2.19 the membrane capacitance as well as gap junctions were omitted for simplicity.

$$\dot{x}_i(t) = -w_i(E_{li} - x_i(t)) + \sum_{j=1}^n y_{ji}(t) \quad y_{ji}(t) = w_{ji} \sigma(x_j(t), \mu_j)(E_{ji} - x_i(t)) \quad (2.19)$$

Due to the multiplication by a difference of potentials, the unrolling of the NRN leads to a ResNet where the weights additionally depend on the input. This can be formalized by a quotient structure that is induced by equivalence classes (modulo some tolerance ϵ) in the output space, dividing the input space into M partitions. Figure 2.11 depicts how a NRN is unrolled. At first the input decides which row to use and each row then corresponds to one specific ResNet. NRNs are therefore M times more succinct than NeuralODEs and $M * N$ times more succinct than ResNets.

2.3.3 Numerical Stability

When evaluating the differential equation in 2.19 using the forward euler method, the solution is very unstable at moderate values for dt , which leads to impractically high computation costs.

In order to improve numerical stability a *hybrid euler* discretization can be used as found in the appendix of [LHG18]. By assuming $v_i^{pre}(t) \approx v_i^{pre}(t - \delta t)$ the membrane potential can be computed as:

$$v(t) = \frac{\frac{C^m}{\Delta t} v(t - \Delta t) + G^{leak} E^{leak} + \sum G^{syn} sig(v_{pre}(t - \Delta t)) E^{syn} + \sum G^{gap} v_{pre}(t - \Delta t)}{\frac{C^m}{\Delta t} + G^{leak} + \sum G^{syn} sig(v_{pre}(t - \Delta t)) + \sum G^{gap}} \quad (2.20)$$

While more recently a fixed step ODE solver was used in place of the discretization above, this thesis stuck to the latter.

2.3.4 Ordinary Neural Circuits

While the LTC model for neuronal dynamics introduced above was derived from biology, it is also possible to repurpose network architectures as found in the *C. elegans* roundworm. More specifically, the roundworm's tendency to change its direction of movement when experiencing vibration (i.e. when the plate they are sitting on is tapped), was found to originate from a small part of its nervous system. This neuronal sub-circuit was termed "Tap-Withdrawal" (**TW**)-circuit accordingly and consists of 11 *sparsely* connected neurons. [WRR96] The layout is depicted in figure 2.12. PVD, PLM, AVM and ALM are input neurons sensing vibrations and FWD and REV are command neurons for forwards and backwards motion respectively. The 5 remaining nodes are inter-neurons. Note that there exist different depictions of this circuit throughout the literature. However, in order to maintain biological realism one particular constraint is met by the circuit in figure 2.12: each neuron has only one type of efferent chemical synapses (either inhibitory or excitatory but never both).

Network architectures that are found in nature were recently termed *ordinary neural circuits* (**ONCs**) by Hasani et. al.[HLA⁺20]. They further showed that the TW circuit exhibits a higher *flow-rate* as compared to randomly connected sparse networks of the same size, depicting the amount of (information) flow between nodes within the network. They also used the TW circuit in control tasks, for which they were trained using a search-based RL algorithm, and were able to outperform randomly wired networks of the same size and, in many cases, contemporary deep learning models that require much larger networks.

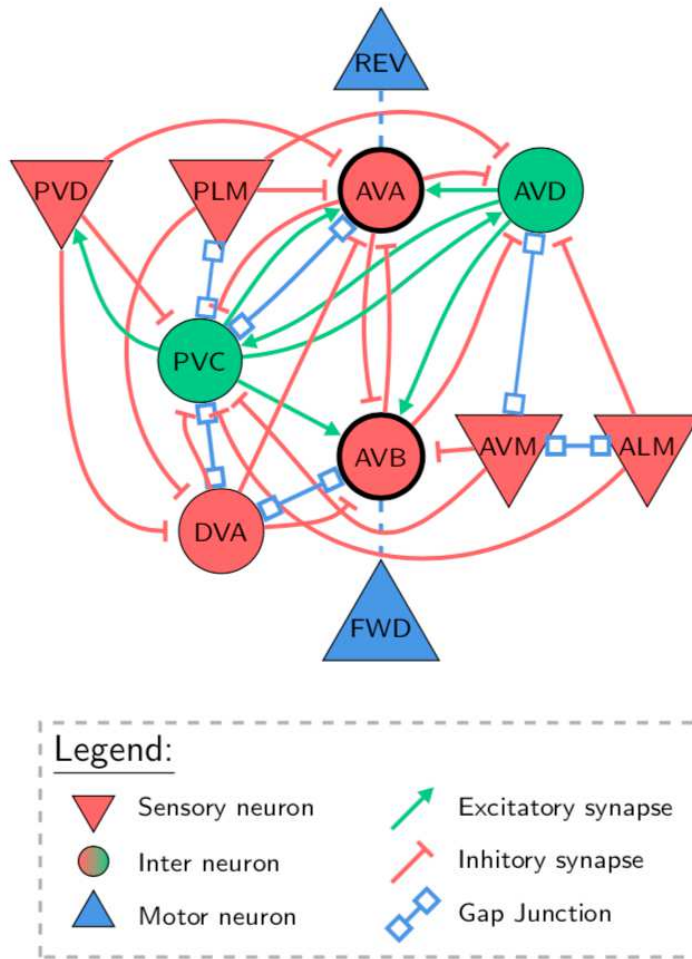


Figure 2.12: Tap-Withdrawal circuit as found in the *c. elegans* roundworm.
Source:[LHG18]

Methods

In order to explore the usage of synaptic plasticity rules in NRNs, two different approaches were chosen and tested for this thesis. First, the gradients for the synaptic weights G^{syn} were replaced by the value of an *eligibility trace* that is calculated as an additional RNN state. Second, a continuous plastic component is added to the model in line with [MRCs19] making use of the same eligibility trace.

For assessing the viability of *biological* learning rules a neural network architecture as can be found in nature seems favorable. The Tap-Withdrawal ONC introduced in Section 2.3.4 is ideal to these ends as it was used as a controller for RL tasks in the past while being entirely derived from observations in biology. Accordingly, the TW circuit acts as the *policy* in a RL algorithm. The RL algorithm of choice was TD3 since it implements a deterministic *actor-critic* scheme which is biologically plausible.

The novel approach was tested by applying it to a continuous *cartpole* balancing task using the cart's position and the pole's angle as inputs (cf. 2.2.1). Since NRNs, like RNNs, retain an inner state throughout the entire trial they are capable of extracting information from sequences. This is the first time the LTC model was used in an *off-policy* algorithm and training is not consistently converging even when using *backpropagation*. Nonetheless, the metrics used for evaluation were chosen in order to make appropriate comparisons as depicted in chapter 4.

3.1 Backpropagation with custom gradients

The first approach of this thesis was to replace the actual gradients for G^{syn} in the actor updates with a *local* term that is computed using locally available information only. This changes the update term for the actor from equation 2.12 to:

$$\phi[G^{syn}] = \phi[G^{syn}] + \eta \frac{dQ_{\theta}(s, \pi_{\phi}(s))}{da} trace^{syn} \quad (3.1)$$

$trace^{syn}$ is a form of eligibility trace (cf. section 2.2.3) and is computed each time-step using some type of a local learning rule. One possibility is a simple hebbian plasticity rule with decay term.

$$\dot{trace}_{ij}^{syn}(t) = \gamma (x_i(t-1)x_j(t) - trace_{ij}^{syn}(t-1)) \quad (3.2)$$

γ is a new (hyper-)parameter and is the same for the entire network. In order to avoid exploding gradients it was necessary to clip the trace to remain inside $[-1, 1]$.

Alternatives to the simple hebbian trace are discussed in the next section. The state of the NRN, which formerly was just the membrane potential $x_i(t)$ is extended to include the trace which makes it a tuple $[x_i(t), trace_{ij}^{syn}(t)]$. Note that LSTM cells do also have a twofold state.

3.2 LTC with Plasticity Dynamics

For the second approach in this thesis, additional *plasticity dynamics* are introduced to the LTC model as defined in 2.3 similar to the framework used by Miconi et. al. [MRCS19]. The synaptic weights $G^{syn,pd}$ are computed by adding the plasticity trace $trace^{syn}$ to the static part of the weights G^{syn} , changing equations for $x_i(t)$ from 2.14 to

$$\begin{aligned} C_i^m \frac{dx_j}{dt} &= I_j^{leak} + \sum_{i=1}^n I_{i,j}^{syn} + \sum_{i=1}^n I_{i,j}^{gap} \\ I_i^{leak} &= G_i^{leak} (E_i^{leak} - x_i(t)) \\ I_{i,j}^{gap} &= G_{i,j}^{gap} (v_i(t) - v_j(t)) \\ G_{i,j}^{syn,pd} &= G_{i,j}^{syn} + \alpha_{ij} trace_{ij}(t) \\ I_{i,j}^{syn} &= G_{i,j}^{syn,pd} (E_{i,j}^{syn} - v_j(t)) sig(\sigma_{i,j}(v_i(t) - \mu_{i,j})) \end{aligned} \quad (3.3)$$

where $trace^{syn}(t)$ is computed the same way as before and α_{ij} is another new parameter of the model that decides how "plastic" a synapse is.

3.3 Learning Rules

Table 3.1 shows the three different learning rules that were tested. They were used to calculate the trace(s) used for adjusting G^{syn} in both approaches described above. Refer to section 2.1.4 for their neuro-scientific background.

In total there are 6 different *gradient types* considered in this thesis. In the following, they are denoted by the learning rule used, followed by the suffix '_pd' in case of the *plasticity*

Table 3.1: Learning rules for G^{syn}

<i>hebb</i>	$\dot{trace}_{ij}^{syn}(t) = \gamma (x_i(t-1)x_j(t) - trace_{ij}^{syn}(t-1))$
<i>oja</i>	$\dot{trace}_{ij}^{syn}(t) = \gamma x_i(t-1)x_j(t) - x_j(t)^2 trace_{ij}^{syn}(t-1)$
<i>bcm</i>	$\dot{trace}_{ij}^{syn}(t) = \gamma x_j(t) (x_j(t) - x_\theta(t)) x_i(t-1)$ $x_\theta(t) = \eta * (-x_\theta(t-1) + x_j(t)^2)$

dynamics mode or no suffix in case of *custom gradients*. In *custom gradients* mode the hyper-parameters γ and η were fixed ahead of training while they were optimized using backpropagation in the *plasticity dynamics* mode.

Table 3.2: *gradient_types*

	<i>hebb</i>	<i>oja</i>	<i>bcm</i>
<i>custom gradients</i>	hebb	oja	bcm
<i>plasticity dynamics</i>	hebb_pd	oja_pd	bcm_pd

3.4 RL setting

The framework of this thesis builds on the TD3 algorithm with LSTM networks (see 2.2.2). The critic network was kept the same and the actor network was replaced by the TW ONC. All hyper-parameters associated with TD3 were left untouched. Their values are given in 3.3.

The continuous *cartpole* environment as provided by the `Roboschool` library served as a benchmark. This particular environment takes one float in the interval $[-1, 1]$ as *action* at each time-step and returns a five-tuple for its *state* $(x, \dot{x}, \cos(\theta), \sin(\theta), \dot{\theta})$. The goal is to balance the pole in an upright position for a duration of up to 1000 steps, the *reward* being the total number of steps. To make the task a POMDP we only take the x and $\sin(\theta)$ as the partially observed *state*.

Mapping onto the TW circuit The TW circuit has 4 input nodes which the *state* was mapped onto, and 2 output nodes which were mapped to the *action* in a similar fashion to Lechner et. al. [LHG18]. Each of the two input variables was mapped on either one of two input nodes depending on the sign: x was mapped to PVD if positive and ALM if negative, θ was mapped to AVM if positive and PVM if negative. The mapping was done in a linear way mapping the intervals of $[0, \frac{x_{max}}{50}]$, $[0, \frac{\theta_{max}}{50}]$ for positive values and $[\frac{x_{min}}{50}, 0]$, $[\frac{\theta_{min}}{50}, 0]$ for negative values, to $[-1, 1]$ and were clipped to the same range if outside the given intervals. In any case the opposite neuron was kept at -1 . The input neurons remained "clamped" at those values throughout the steps of the ODE solver.

Table 3.3: TD3 hyper-parameter values

<i>critic network</i>	
<i>observation embedding layer nodes</i>	400
<i>action embedding layer nodes</i>	None
<i>joint embedding layer nodes</i>	300
<i>output embedding layer nodes</i>	100
<i>LSTM size</i>	40
<i>replay buffer capacity</i>	10000
<i>exploration noise std</i>	0.1
<i>target update τ</i>	0.05
<i>target update period</i>	5
<i>actor update period</i>	2
<i>actor learning rate</i>	1e-4
<i>critic learning rate</i>	1e-3
<i>gamma</i>	0.995
<i>train steps per iteration</i>	200
<i>train sequence length</i>	10
<i>batch size</i>	64

3.5 Implementation

The `BiORL` framework implemented for this thesis was used to produce the results in chapter 4. It is based on *tensorflow* 2.0 in conjunction with the *tf-agents* library. Its folder structure is given below.

```

├── config .....config files for neuronal parameters bounds and initialization
├── model
│   ├── pyobject ..... object-oriented implementation of the ltc model
│   └── ltc ..... matrix implementation of the ltc model
├── pybnn .....import functionality for bnn files
├── rl
│   ├── pyobject ..... Basic RL algorithms using the object-oriented model
│   ├── a2c_nrn ..... Basic RL algorithms using the matrix model
│   └── tf_agents SOTA RL algorithms using the matrix model and tf-agents
├── README.md ..... usage instructions
└── run_experiments.py helper function for running multiple experiments

```

While implementing the framework I learned a lot about the nature of RNNs, NRNs and the quirks of *tensorflow* which caused me to scratch my progress and start anew

a few times. `pyobj`: At the very beginning I tried implementing the LTC model in an object-oriented manner by having each neuron and synapse modeled as a separate entity. The resulting LTC model was very slow since no optimizations for matrix multiplication were available which made it unfit for actual experimentation. `a2c_nrn`: I quickly moved on to representing the neuron parameters and states as vectors and the synaptic parameters as matrices to harness the accelerated matrix operations provided by `tensorflow`. For training the model, I initially tried to use the quite basic *advantage actor-critic* (A2C) algorithm, but failed to reliably train the model even when applying regular backpropagation updates. `tf_agents`: Eventually, I stumbled across *tf-agents*, a reinforcement learning library built with *tensorflow*. They provide working examples of state-of-the-art RL algorithms employing DNN or RNNs. I modified the TD3 RNN example and replaced the actor RNN with the TW circuit modeled as an NRN. This approach did eventually bring the desired performance when trained with *backpropagation* and was then used for experimentation.

LTC model The LTC model was implemented as a subclass of `keras.layers.Layer` and has up to 11 trainable parameters, namely: `Mu`, `Sigma`, `W_syn`, `W_gap`, `Erev`, `Vleak`, `Gleak`, `Cm_t`, `Gamma`, `Alpha` and `Eta`. By providing a whitelist containing the names when creating a new instance, individual parameters can be set to being non-trainable. Additionally, the `gradient_type`, the parameter constraints, and their initialization bounds can be specified through constructor arguments.

Function `_ode_step(inputs, state)` implements the hybrid euler discretization from equations 2.20, including modifying `W_syn` on each step if operating in *plasticity dynamics* mode. The number of solver steps can be specified via constructor argument. `_calc_traces(v_pre, v_post, traces)` is called on each step implementing the calculation of the traces for the learning rules from 3.3. Finally, `_ode_step_custom_grad(inputs, state)` is used in *custom gradients* mode, which calls `_ode_step(inputs, state)` under the hood but replaces the gradients of the `W_syn` parameter with the eligibility trace by using the `@tf.custom_gradient` annotation.

TW circuit The structure of the TW circuit (see 2.3.4) was imported from [LHG18] since their source code was available on GitHub. `pybnn_deserializer_ltc.py` provides a function for importing `bnn` files used to store network information in their framework. Upon import, some or all of the parameters can be randomized, but the sparse connectivity is kept in any case.

Parameter initialization Parameters were randomly initialized according to table 3.4. The sparse connectivity of the TW circuit was maintained by setting all components that do not correspond to any synapse in the ONC to zero. The table also lists the bounds used for constraining the parameters during training. The constraints were enforced by clipping the values after applying the gradients.

Table 3.4: LTC-RNN parameter bounds and initial values

	<i>Parameter</i>	<i>lower bound</i>	<i>upper bound</i>	<i>init min</i>	<i>init max</i>
<i>intrinsic</i>	C_i^m	$1e^{-4}$	1000	0.1	0.5
	G_i^{leak}	$1e^{-3}$	100	0.1	1
	E_i^{leak}	-	-	-1	0
<i>synaptic</i>	$G_{i,j}^{gap}$	$1e^{-3}$	100	0.1	1
	$G_{i,j}^{syn}$	$1e^{-3}$	100	0.1	1
	$\sigma_{i,j}$	-	-	3	8
	$\mu_{i,j}$	-	-	0.3	0.8
<i>plasticity</i>	γ	0	1	0.001	0.001
	η	$1e^{-3}$	1000	0.1	0.5
	$\alpha_{i,j}$	0	100	0.01	1

Visualization For visualizing the results `tensorboard` was used. It supports recording of scalar and histogram values alongside hyper-parameter information such as the *gradient type* used. The information can be displayed and downloaded through a web interface featuring plots of scalars and histograms and grouping by hyper-parameter values. Additional features of `tensorboard` include model profiling and visualization of computation graphs and multidimensional data. The data can be downloaded as `json` through `http` requests which was used for plotting with `matplotlib`.

Results and Discussion

4.1 Experiments

For comparing the different learning rules introduced in the last chapter, the following two metrics were used to assess performance of the training algorithm: the *average episode length* at the end of training and the number of *environment steps* until the task appeared to be solved (all evaluation episodes reach 1000 steps). Evaluation was done every 1000 episodes by averaging the episode length of 10 episodes. The networks were trained a total of 50000 episodes, 200 train steps each with a batch size of 64 and a train sequence length of 10 time-steps (since we are using RNNs). A total of 10 experiments each were conducted.

The outcomes are summarized in table 4.1 where *backprop* was recorded using backpropagation and serves as reference, *hebb*, *oja* and *bcm* were conducted using the *custom gradient* approach and *hebb_pd*, *oja_pd* and *bcm_pd* using the *plasticity dynamics* approach.

Table 4.1: Results (best in bold)

	<i>average episode length</i>	<i>average steps until solved</i>	<i># solved (out of 10)</i>
<i>backprop</i>	836.06	32200	5
<i>hebb</i>	156.19	-	0
<i>oja</i>	279.26	41000	1
<i>bcm</i>	779.61	18714	7
<i>hebb_pd</i>	774.94	33333	3
<i>oja_pd</i>	771.12	34000	3
<i>bcm_pd</i>	718.00	31600	5

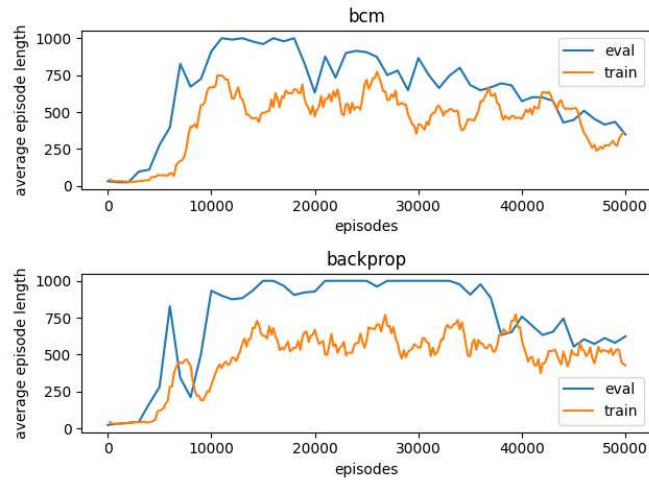


Figure 4.1: Performance degradation after successfully solving the task occurred in all approaches (shown: *bcm* and *backprop*).

Performance degradation After successfully solving the environment, performance nonetheless decreased in some cases as shown in Figure 4.1, even when using *backpropagation*. This issue might stem from the use of the off-policy TD3 algorithm. Another possible cause could be an inadequately high learning rate which causes the training network to converge too early. However, this does not diminish the findings of this thesis, since the proportion of runs that showed this phenomenon appears to be equal among the different approaches.

4.1.1 Parameter and gradient distributions

When looking at the evolution of parameters and their gradients throughout the training process we can gain insights into causes for failed experiments. While gradients for W_{syn} , fluctuate a lot in the *plasticity dynamics* approach and span a broader range than in the *custom gradient* approach (akin to gradients calculated using *backpropagation*), gradients change more slowly in the latter case (see figures 4.2 and 4.3). All of the *custom gradient* runs seem to exhibit unconstrained growth for some synapses while this does not occur in the *plasticity dynamics* runs.

4.1.2 Looking at the traces

The framework implemented for this thesis supports recording the eligibility traces used in the *learning rules*. Figures 4.4 and 4.5 show eligibility traces that were calculated during one single trial. It immediately comes to attention that in many cases the traces would have grown past the clipping range of $[-1, 1]$. Only exception is the *bcm* trial, for which traces never grew past the upper bound.

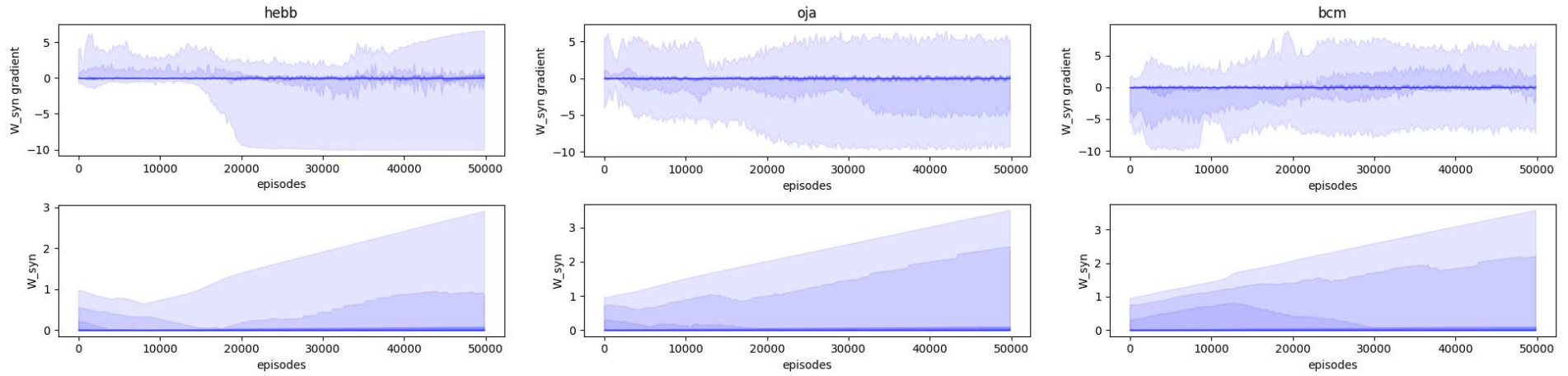


Figure 4.2: Gradients and values for W_{syn} during one run for *hebb*, *oja* and *bcm*

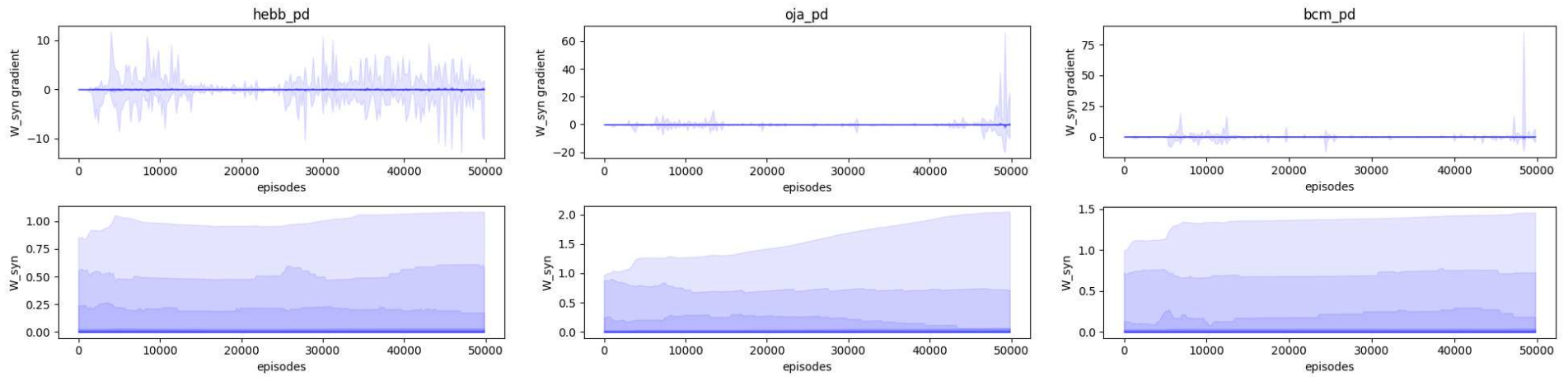


Figure 4.3: Gradients and values for W_{syn} during one run for *hebb_pd*, *oja_pd* and *bcm_pd*

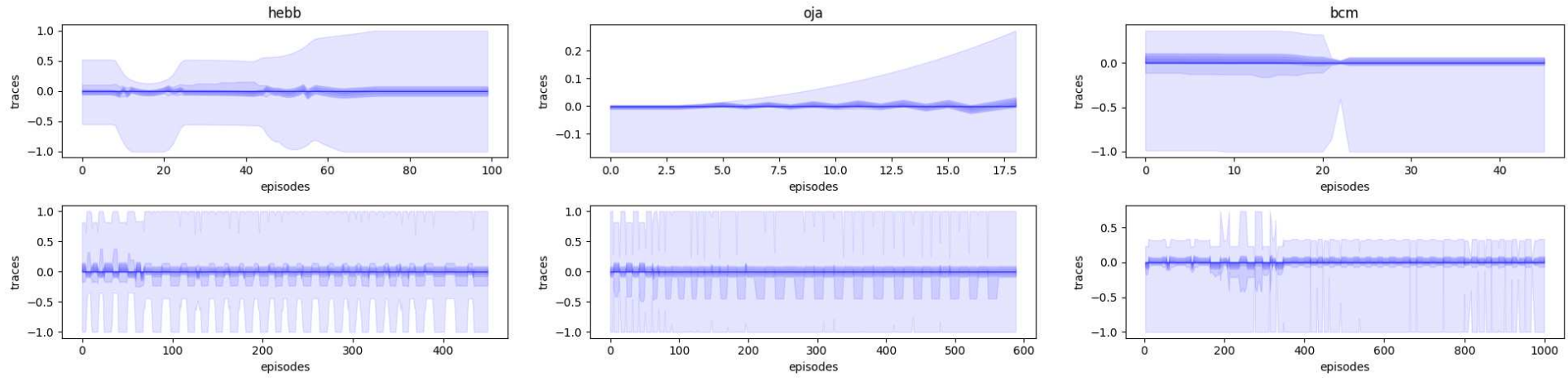


Figure 4.4: Eligibility traces for W_{syn} at the beginning (top) and at the end (bottom) of training for *hebb*, *oja* and *bcm*

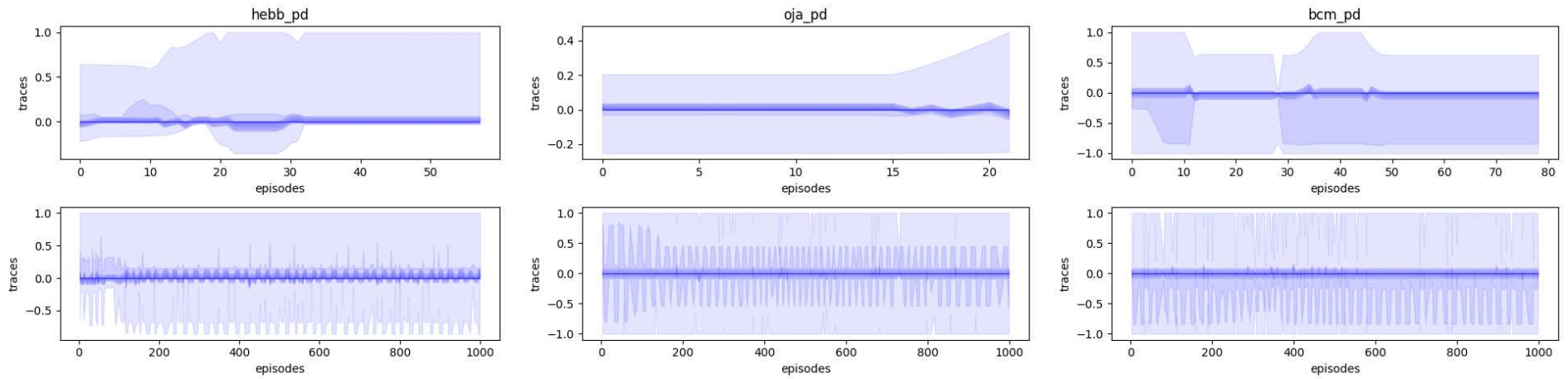


Figure 4.5: Eligibility traces for W_{syn} at the beginning (top) and at the end (bottom) of training for *hebb_pd*, *oja_pd* and *bcm_pd*

4.2 Discussion

This thesis takes a look at different local learning rules for synaptic weights and how they might be implemented in a RL setting using NRNs with *eligibility traces*. By comparing them to regular backpropagation their viability is assessed. Two different approaches of incorporating the rules into a learning regime were explored: replacing the true gradients with the trace, and adding a learned proportion of the trace to the fixed (learned) weights.

The results presented in this chapter suggest that the *custom gradient* approach can lead to improved performance, but may also thwart learning altogether depending on the *learning rules* used. The *bcm* rule proved to be the best choice when looking for fast convergence. It's success may come from the fact that the rule shows *potentiation* and *depression* characteristics as opposed to *potentiation* only for the *hebb* and *oja* rules.

The *plasticity dynamics* approach showed performance comparable to, or slightly worse than, *backpropagation*, depending on the *learning rule* used. Again, the *bcm* rule led to a faster solving of the environment. Whether the bounds for eligibility trace clipping had any impact on the training performance remains unanswered.

4.3 Outlook

The approach of this thesis is a step towards implementing learning rules that do not require backpropagation and are acting throughout the entire lifespan of the agent.

As this thesis only provides preliminary results, extensive hyper-parameter tuning should be conducted to further explore the novel approach. An in-depth analysis of the parameter and gradient evolution may provide insights into the differences of the *learning rules*, alongside additional information needed for adequate parameter constraints. Comparing direction and magnitude of the gradients to the ones calculated using *backpropagation* could provide a deeper understanding of the rule's workings.

In order to assess the potential of generalization to many different environments a Meta-learning approach that builds on the present framework may be tested. For this, the same circuit would have to be trained across different environments. This would hopefully result in generalized parameters for the learning rules and enable transfer learning.

Since they don't use *backpropagation* at all, *Genetic Algorithms* could be used instead of predominant *Policy Gradient* algorithms (like TD3). They are *search-based* algorithms that employ heuristics inspired by Charles Darwin's theory of natural evolution, which seems promising considering the biological motivation of the work. Previous work by [SSR18] featured a *genetic* algorithm in conjunction with *plasticity dynamics* that could serve as basis for further investigations.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

2.1	BCM update rule: Synaptic strength is weakened if postsynaptic activity is below threshold v_θ . Below v_0 no synaptic modification occurs. Source: [GKNP14]	7
2.2	Pair-based update rule (lines) plotted alongside experimental data (circles). Source: [GKNP14]	8
2.3	<i>Markov Decision Process</i> : the agent finds itself in <i>state</i> S_t and takes <i>action</i> A_t which makes the environment go to <i>state</i> S_{t+1} and yields a <i>reward</i> R_{t+1}	9
2.4	Pole-balancing: a continuous MDP	10
2.5	Error transport in FA and its variants: a) backpropagation uses weights symmetric to the forward weights W_i ; b) FA uses fixed random weights B_i per layer; c, d) Direct and Indirect FA are variants that project back from the output layer only. Source: [Nøk16]	17
2.6	schematic of a biological synapse	18
2.7	schematic of a single layer neural network	20
2.8	schematic of a multi layer neural network	20
2.9	schematic of a ResNet layer	21
2.10	NeuralODEs are unfolded by repeatedly computing the activation function using the last outputs as additional inputs.	21
2.11	NRNs are unrolled into ResNets whose weights depend on the input. Each row shows one possible unfolding. I_j are the input partitions, La_j are the layers, In_j is the input and V_{ij} are the weights of each layer	22
2.12	Tap-Withdrawal circuit as found in the c. elegans roundworm. Source:[LHG18]	24
4.1	Performance degradation after successfully solving the task occurred in all approaches (shown: <i>bcm</i> and <i>backprop</i>).	32
4.2	Gradients and values for W_{syn} during one run for <i>hebb</i> , <i>oja</i> and <i>bcm</i>	33
4.3	Gradients and values for W_{syn} during one run for <i>hebb_pd</i> , <i>oja_pd</i> and <i>bcm_pd</i>	33
4.4	Eligibility traces for W_{syn} at the beginning (top) and at the end (bottom) of training for <i>hebb</i> , <i>oja</i> and <i>bcm</i>	34
4.5	Eligibility traces for W_{syn} at the beginning (top) and at the end (bottom) of training for <i>hebb_pd</i> , <i>oja_pd</i> and <i>bcm_pd</i>	34
		37



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

2.1	<i>Markov Decision Process</i> formal definition	9
2.2	LTC-RNN parameters	19
3.1	Learning rules for G^{syn}	27
3.2	<i>gradient_types</i>	27
3.3	TD3 hyper-parameter values	28
3.4	LTC-RNN parameter bounds and initial values	30
4.1	Results (best in bold)	31



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [Bel53] Richard Bellman. An introduction to the theory of dynamic programming. Technical report, RAND CORP SANTA MONICA CA, 1953.
- [BSR⁺18] Sergey Bartunov, Adam Santoro, Blake A. Richards, Luke Marris, Geoffrey E. Hinton, and Timothy Lillicrap. Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures. *arXiv:1807.04587 [cs, stat]*, November 2018.
- [FvM18] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv:1802.09477 [cs, stat]*, October 2018.
- [GKNP14] Wolfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, Cambridge, 2014.
- [Gro20] Radu Grosu. ResNets, NeuralODEs and CT-RNNs are Particular Neural Regulatory Networks. *arXiv:2002.12776 [cs, q-bio]*, March 2020.
- [HLA⁺18] Ramin M. Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid Time-constant Recurrent Neural Networks as Universal Approximators. *arXiv:1811.00321 [cs, stat]*, November 2018.
- [HLA⁺20] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. The Natural Lottery Ticket Winner: Reinforcement Learning with Ordinary Neural Circuits. *Proceedings of the International Conference on Machine Learning*, 119, December 2020.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.
- [JR14] Leonard Johard and Emanuele Ruffaldi. A connectionist actor-critic algorithm for faster learning and biological plausibility. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3903–3909, Hong Kong, China, May 2014. IEEE.

- [Kan13] Eric R. Kandel. *Principles of Neural Science*. McGraw-Hill Medical, New York; Toronto, 2013.
- [KS98] Christof Koch and Idan Segev. *Methods in Neuronal Modeling: From Ions to Networks*. MIT Press, Cambridge, MA, USA, second edition, 1998.
- [LHG18] Mathias Lechner, Ramin M. Hasani, and Radu Grosu. Neuronal Circuit Policies. *arXiv:1803.08554 [cs, q-bio]*, March 2018.
- [LHP⁺19] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, July 2019.
- [MI95] Jorge H. Medina and Ivan Izquierdo. Retrograde messengers, long-term potentiation and memory. *Brain Research Reviews*, 21(2):185–194, September 1995.
- [MRCS19] Thomas Miconi, Aditya Rawal, Jeff Clune, and Kenneth O Stanley. BACK-PROPAMINE: TRAINING SELF-MODIFYING NEURAL NETWORKS WITH DIFFERENTIABLE NEUROMODULATED PLASTICITY. page 15, 2019.
- [noa19] Striatum. *Wikipedia*, September 2019.
- [Nøk16] Arild Nøkland. Direct Feedback Alignment Provides Learning in Deep Neural Networks. *arXiv:1609.01596 [cs, stat]*, December 2016.
- [SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass, 1998.
- [Sho07] Harel Z. Shouval. Models of synaptic plasticity. *Scholarpedia*, 2(7):1605, July 2007.
- [SSR18] Andrea Soltoggio, Kenneth O. Stanley, and Sebastian Risi. Born to Learn: The Inspiration, Progress, and Future of Evolved Plastic Artificial Neural Networks. *Neural Networks*, 108:48–67, December 2018.
- [Wis04] Roy A. Wise. Dopamine, learning and motivation. *Nature Reviews Neuroscience*, 5(6):483–494, June 2004.
- [WRR96] Stephen R. Wicks, Chris J. Roehrig, and Catharine H. Rankin. A Dynamic Network Simulation of the Nematode Tap Withdrawal Circuit: Predictions Concerning Synaptic Function Using Behavioral Criteria. *The Journal of Neuroscience*, 16(12):4017–4031, June 1996.