Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Memory in Deep Learning

Bc. Tomáš Paleček

Supervisor: Ing. Jaromír Janisch
Field of study: Open Informatics
Subfield: Artificial Intelligence
August 2020

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Paleček Tomáš** Personal ID number: **457070**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master's thesis details

Master's thesis title in English:

Memory in Deep Learning

Master's thesis title in Czech:

Paměť v hlubokém učení

Guidelines:

Memory is a crucial component of models based on neural networks, if they are to succeed in complicated tasks, such as meta-learning. Meta-learning is a method of machine learning, in which an agent *learns to learn*. Feed forward neural networks can be used to solve many contemporary and important tasks, such as classification, playing chess or driving a car. However, they cannot solve problems which require a memory. A simplest problem is to repeat a given sequence. For the model to succeed in this task, it has to *learn to memorize* the input and then repeat it. A different problem can be to navigate a maze, topology of which changes between lifetimes of the agent.
Memory in neural networks can take many different forms: external memory, recurrency in the network, neural plasticity or their combination. However, little is known about what these mechanics actually encode and what tasks are they suitable for.
The goals of the thesis are:
- Review the state-of-the-art of different types of memory in neural networks (external memory, recurrent neural networks and neural plasticity).
- Implement two existing problems (from meta-learning) requiring memory to be solved.
- Benchmark different memory types (and their combinations) on the problems.
- Combine different memory types (e.g., RNN + plasticity) and evaluate their relative contribution to solving the tasks.
- Based on the knowledge obtained from the experiments, explain the merits of the various memory types and the tasks they are suitable to.

Bibliography / sources:

Thomas Miconi, Jeff Clune, and Kenneth O Stanley. Differentiable plasticity: training plastic neural networks with backpropagation. arXiv preprint arXiv:1804.02464, 2018.
Thomas Miconi, Aditya Rawal, Jeff Clune, and Kenneth O. Stanley. Backpropamine: train-
ing self-modifying neural networks with differentiable neuromodulated plasticity. In Inter-
national Conference on Learning Representations, 2019.
Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho,
John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. Nature, 538(7626):471, 2016.
Florentin Hennecker. Meta reinforcement learning. https://github.com/fhennecker/meta-reinforcement-learning, 2017.
Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation,

9(8):1735–1780, 1997.
Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient
descent. In International Conference on Artificial Neural Networks, pages 87–94. Springer,
2001.
Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In International conference
on machine learning, pages 1842–1850, 2016.
Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matthew Botvinick. Learning to reinforcement learn. CoRR, abs/1611.05763, 2016.

Name and workplace of master's thesis supervisor:

Ing. Jaromír Janisch, Artificial Intelligence Center, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **04.02.2020** Deadline for master's thesis submission: **14.08.2020**

Assignment valid until: **30.09.2021**

Ing. Jaromír Janisch Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I thank my supervisor for the topic he had chosen for me and for the consultations, during which he had helped me with problems I had encountered during my work on the project.

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ....
signature

Abstract

In our work, we study neural networks utilizing memory, more specifically, plastic neural networks, recurrent neural networks, and their combinations. The networks are used in the meta-learning and meta-reinforcement learning domains. In the Binary Sequences environment, the task of the algorithm is to remember binary patterns and reconstruct them upon a presentation of their degraded versions. The goal of the algorithm in the Maze environment is to maximize the reward obtained from finding a reward in the maze, where the position of the reward differs for each episode. In the Binary Sequences environment, the plastic neural networks outperformed the recurrent neural networks. In the Maze environment, the networks achieved similar performance. For the Maze environment, we devised two tests, which show that the plastic neural networks are capable of better exploring the maze, while recurrent neural networks better adapt to the environment during the episode. The combination of plastic and recurrent networks decreases the performance in the Binary Sequences environment, while increasing the performance in the Maze environment. In the last experiment, we investigated the different modifications for the plastic neural networks, and shown their added value to the network.

Keywords: Memory, Plastic neural networks, Recurrent neural networks, Meta-learning, Meta-reinforcement-learning

Supervisor: Ing. Jaromír Janisch

Abstrakt

V předkládané práci zkoumáme neuronové sítě využívající pamět, konkrétněji rekurentní neuronové sítě, plastické neuronové sítě a jejich kombinace. Výše zmíněné sítě testujeme v doménách meta-learning a meta-reinforcement learning. V prostředí Binary Sequences, které jsme použili pro testování, je cílem algoritmu zapamatovat si binární vzory, které jsou mu ukázany a následně si je vybavit po tom co je mu ukázána degradovaná verze jednoho z nich. Cílem algoritmu v prostředí Maze je maximalizovat odměnu získanou v bludišti, kde se pozice odměny nachází na jiném míste v každé testované epizodě. V prostředí Binary Sequences dosahovaly plastické neuronové sítě lepších výsledků, než rekurentní neuronové sítě. V prostředí Maze měly oba přístupy srovnatelné výsledky. Pro prostředí Maze, jsme navrhli dva testy, které ukázaly, že plastické sítě zvládají lépe prozkoumávat bludiště, zatímco rekurentní sítě se dokáží lépe přizpůsobit prostředí během jedné epizody. Po kombinaci rekurentní a plastické sítě, výsledná neuronová síť dosahovala horších výsledků v prostředí Binary Sequences a lepších výsledků v prostředí Maze. V posledním experimentu, jsem zkoumali různé modifikace plastických sítí a ukázali jsme jejich přidanou hodnotu.

Klíčová slova: Pamět, Plastické neuronové sítě, Rekurentní neuronové sítě, Meta-učení, Posilované-meta-učení

Překlad názvu: Paměť v hlubokém učení

Contents

Chapter 1

Introduction

In the following work, we aim to obtain useful insights into how plastic neural networks and recurrent neural networks work and use the obtained knowledge to explain the merits of individual approaches, compare them, and possibly increase their performance based on the insights obtained. To obtain knowledge, we visualize the inner workings of the network and create environment-specific experiments to test the algorithms' properties. The primary motivation is to provide new knowledge regarding the recently introduced plastic neural networks [29], [30]. Both plastic neural networks and recurrent neural networks utilize memory mechanisms. Introducing memory to neural networks is a powerful tool that allows them to use information acquired earlier during an episode for current and future decisions. Memory has been used in the domains of meta-learning ([29],[37]), language modeling ([30], [35]), reinforcement learning ([43]) and multiple other domains. In our work, we focus on the domain of meta-learning (Section 2.5) and meta-reinforcement learning (Section 2.6.3). The goal of meta-learning is to train a model capable of quickly adapting to new environments not seen during training and learn new concepts only after minimal interaction with the environment. The question that arises is how memory is used in meta-learning. To answer this question, we will briefly introduce the Maze environment used in our work. In the following environment, the agent's goal is to maximize its total reward over 200 steps (one episode). For each episode, a new random position for the reward tile is chosen, and after finding the reward, the agent is teleported to a new location. By using memory, the agent can remember the reward's position after it finds it and uses the acquired knowledge to reach the reward faster in subsequent runs and thus obtain a higher final reward.

A general introduction to memory in neural networks is presented in Section

2.1. There exist various ways how to provide neural networks with memory: recurrent neural networks (Section 2.2), plastic neural networks (Section 2.3) and external memory(Section 2.4). The focus of this work is solely on the plastic and recurrent neural networks and their combinations. For the plastic networks, we had decided to use the differentiable plasticity framework introduced in [29], provided in Section 2.3. The recurrent neural networks used are LSTM [15] and vanilla RNN [8]. There are two main goals that our experiments aim to achieve. The first goal is to provide a comparison of the two memory approaches and explain their merits. This goal is achieved by first testing the algorithms' overall performance and then creating environment-specific experiments to understand their properties better. The second goal is to investigate the properties of the plastic neural network. The plastic neural networks were only recently introduced and had not been well studied in literature yet. Hence we see this as an opportunity to provide new insights into this promising framework. We do so by first measuring the relative contribution of plastic and recurrent neural layer, in plastic neural network. In the next experiment, we divide the plasticity framework into individual parts and measure their contribution to the final result.

In the Binary Sequences environment, introduced in [17], the goal of the agent is to remember binary patterns uniquely generated for each episode and reconstruct them upon a presentation of their degraded versions. The following problem is an instance of supervised learning problems and tests the capabilities of the algorithms to store information to memory and associate the stored information with new inputs. The main findings from the experiments are that plastic neural networks outperformed the recurrent neural networks (Section 3.2). The recurrent layer in plastic neural networks is decreasing the performance (Section 3.3). The plasticity learning rate is the most important parameter of plastic neural networks (Section 3.4), for the Binary Sequences environment.

In the Maze environment, introduced in [29], the goal of the algorithm is to maximize the reward obtained from finding a reward in a maze, where the position of the reward differs in each episode. The following problem is an instance of reinforcement learning problems and tests the algorithms' capabilities to adapt to the changes in the environment rapidly. The main findings from the experiments are that recurrent neural networks and plastic neural networks achieve similar performance (Section 4.2). The recurrent layer in plastic neural networks is increasing the performance (Section 4.3). The full plasticity is the most important component of the plasticity framework (Section 4.4), for the Maze environment. We found out that the plastic neural networks exhibited better exploration capabilities, while the recurrent neural network (LSTM) had been able to better adapt their policy after finding the reward (Sections 4.5, 4.6). The conclusions of our thesis are provided in Section 5.

Chapter 2

Theoretical background for our work

2.1 Memory in neural networks

Memory is a mechanism in neural networks that allows the network to store and actively use the information obtained earlier during observations. This is used in tasks where sequential data are present. Examples of such domains are time-series prediction [40] or language modeling [35]. By introducing the memory, the network can model nontrivial relationships between the observations. This allows the network to perform well in these environments compared to the simple feedforward network that does not benefit from past observations. Our focus in this work is the domain of meta-learning and meta-reinforcement learning thoroughly described in Section 2.5. In this domain, the memory is used to derive a complex learning algorithm within the network, which can quickly adjust to the changes in the environment([16], [43]). In the following three paragraphs, different approaches to implementing memory are described, with links to sections where each approach is introduced.

In the recurrent layer, in addition to standard output, outputs a hidden state which encapsulates information from previously seen observations, to be used for future decisions. The hidden state is created by multiplying the previous output by weights unique to the recurrent layer. In the next observation, the network has available the augmented data from previous observations in the form of a hidden state in addition to the standard input. A useful way to visualize this is that the hidden states are variables that change during the episode, and the standard weights are then programs that act upon them. A detailed introduction to recurrent neural networks is provided

in Section 2.2.

Another option to introduce memory is using plasticity, which is inspired by the working of the human brain. Plasticity is based on Hebb's rule [14]. In plasticity, the weights of the connections weaken or strengthen over time in response to the activity in them. The changing of weights during the episode allows the network to use them to store information in them and use it later. Neural plasticity is described in Section 2.3.

The next alternative is providing the network with external memory. This approach provides the network with an array used by the algorithm to store and retrieve information from past observations. In order to do so, the algorithm needs to have a hardcoded mechanism for information manipulation. Information manipulation can be done using multiple approaches they are summarized in Section 2.4.

2.2 Recurrent neural networks

Recurrent neural network solves the problem of retaining information regarding previous events to reason about the current events. The workings of the network are best presented in sequential tasks where we need to make predictions in multiple steps. In the first step, the network outputs its decision solely on the inputs it receives. However, in addition to this, it sends encoded information from the current step as an additional input to the network for step t+1. Then, in the next step, the network's decision is based on the current input and the encoded information retained by the network from step t-1. The information is sent through a connection between the recurrent layers. This connection has weights that are used to encode the information. The encoded information is usually called hidden state. The equations used to define vanilla RNN are provided in Equation 2.1. Where h_t is the hidden state at time t, x_t is the input at time t, and $h_{(t-1)}$ is the hidden state of the previous layer at time $t-1$ or the initial hidden state at time 0. W_{ih} are the weights and b_{ih} are the biases for the connections between the input and hidden state. W_{hh} are the weights and b_{hh} are the biases for connections between the hidden states.

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh}) \tag{2.1}$$

One way to get a better grasp of this concept is to imagine the network

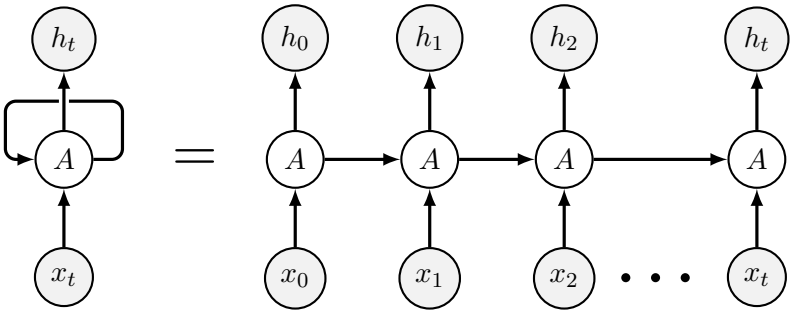


Figure 2.1: The following figure shows a vanilla recurrent network. On the left side we can see the network in it's "unrolled state" with a loop within itself. On the right side we can see a visualization of unrolling the network for t steps.

as multiple copies of the same network(for each step one copy), which are connected through episodes. The visualization of this process, referred to as unrolling, is provided in Figure 2.1. The recurrent neural networks are trained using backpropagation through time(BPTT [36], [44]). This method trains the weights of the neural network by unfolding the network for N time steps, thus creating a classic feedforward network and applying gradient descent. The problem of this approach are exploding\vanishing gradients ([3], [34]). A explanation of this problem is that when we apply the BPTT, we create a network with many layers with the same parameters. The problem is that the networks are duplicates(they have identical parameters). Because of this, when we apply the gradient descent, the resulting product will contain many instances of the same term. This can result in an unstable gradient which is either very large (exploding gradient) or a very small (vanishing gradient). One way to address this problem is by using LSTM (Long Short-Term Memory) [15], described in the next section. Another useful way to visualize the RNN is that the hidden states are variables that change during the episode, and the standard weights are then programs that act upon them.

2.2.1 Long Short Term Memory network

The Long Short Term Memory networks(LSTM) [15] address the problem of vanishing gradients and are designed to learn long-term dependencies. They achieve this by transferring a cell state between episodes in addition to the hidden state. The objective of the cell state is to store information for long periods of time. Gates manage the information that flows through the cell state. These mechanisms can manipulate the information inside the cell state. The LSTM consist of three gates forget gate, input gate, and output gate. The forget gate f_t decides what information to keep form h_{t-1} and x_t. Then follows the input gate i_t, which decides which values to update. Then the new values for storing are created using c_t. The values created using c_t are

then combined with i_t together using multiplication and are added to the information that went through the forget gate. The final gate is the output gate o_t, which decides the network's output based on the cell state that went through the forget and update gate. The exact equations for all the gates are provided in equation 2.2. For more detailed explanation on LSTM we recommend the following two resources [19], [33].

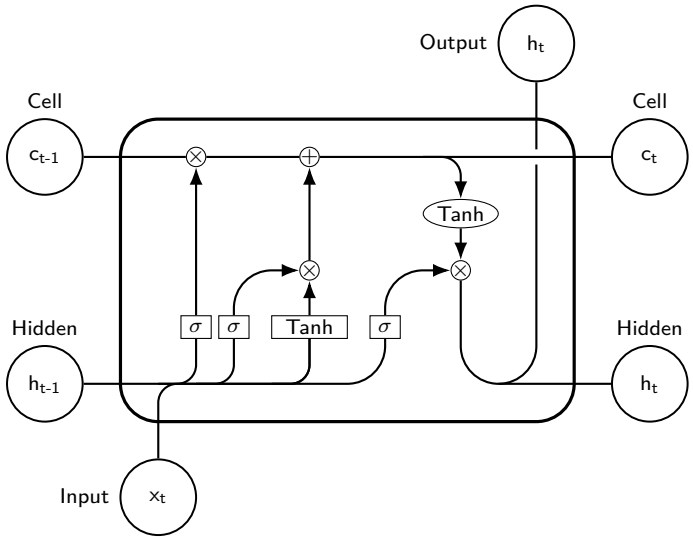


Figure 2.2: The following figure shows the LSTM network module.

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
\tag{2.2}
$$

Another popular approach for handling long term dependencies is the Gated Recurrent Unit(GRU) [4]. The GRU uses the gating mechanisms proposed in the LSTM, but their specifications are different. In our work, we will only work with LSTM. A comparison of LSTM and GRU and other architectures using the gating mechanism is provided in [13]. In this article, the authors tested multiple papers providing alternations to the gating mechanism including GRU and found that there are no significant improvements in using other versions that the original LSTM. Another interesting study regarding the recurrent neural networks is [5]. This study found that the results in RNN

architectures are driven primarily by differences in training effectiveness, rather than differences in capacity.

2.3 Neural Plasticity

Neural plasticity is a biologically inspired method for training the weights of connections in a neural network. From a biological perspective, neural plasticity is the ability of synapses to strengthen or weaken over time, in response to increases or decreases in their activity [18]. The ability of the synapses to self-modify their weights is one of the essential foundations of learning and memory in the brain. The neural plasticity is actively researched in the field of neurosciences, which aim to understand the neural system.

The findings from neuroscience led to creating new machine learning algorithms based on them. The currently most used algorithms utilizing plasticity are inspired by Hebb's rule [14]. Hebb states the definition as follows "Let us assume that the persistence of repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability. When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" [14]. The idea of Hebb can be extended to the mathematical formulation, as expressed in the following equation: $w_{i,j} = x_i x_j$. Where $w_{i,j}$ is the weight of the connection between neurons i and j, x_i is the output of the neuron i, and x_j is the output of the neuron j.

In our work, we will focus on plasticity defined in the following works differentiable plasticity [29] and its extension backpropamine [30], described in the next sections. The idea of combining standard neural network with plastic (Hebbian) parts had been used in several papers published during recent years. The general overview of these papers alongside with a comparison of their approaches and a summary of their results is provided in Section 2.3.3.

2.3.1 Differentiable Plasticity

The following work [29] proposes a new framework in which each connection in the neural network consists of plastic and non-plastic component. The plastic

component is done using Hebbian plastic connections and the non-plastic component consists of a recurrent neural network (vanilla RNN). In the work the components are also referred to as fixed and plastic parts. This term comes from the fact that the recurrent (fixed) part does not change weights during the episode, while the plastic part actively changes its weights. The plastic component consists of a Hebbian trace $\text{Hebb}_{i,j}$ which value varies during each episode based on the activations of neurons in the neural network. The final weight of the connection is determined as a sum of a fixed component and a portion of the plastic component. The exact formulation of the network is provided in Equation 2.3. The plasticity coefficient of α is a learnable parameter that is used by the network to determine the amount of plasticity used in a given connection. By introducing the plasticity coefficients, the network can determine during learning whether the connection should be fully plastic (set the value of $w_{i,j}$ to zero), or fully recurrent (set the value of α to zero) or a combination of both approaches. The plasticity coefficient can be either specified for each connection in the layer, in this case, we talk about Full plasticity. Or it can be specified as one value for each connection, and in that case we talk about Homogenous plasticity. The learning rate of plasticity η is a single scalar parameter for the entire network, which is learned using gradient descent. In the equation, σ denotes nonlinear activation function (tanh in our case).

$$x_j(t) = \sigma\{\sum_{i \in inputs} [w_{i,j} x_i(t-1) + \alpha_{i,j} \text{Hebb}_{i,j}(t) x_i(t-1)]\} \tag{2.3}$$

$$\text{Hebb}_{i,j}(t+1) = \eta x_i(t-1) x_j(t) + (1-\eta)\text{Hebb}_{i,j}(t) \tag{2.4}$$

The unique contribution of this paper is training the plasticity coefficients using gradient descent, hence the Differentiable Plasticity name. In the paper, authors have shown that using their algorithms can optimize the plasticity coefficient for large neural recurrent neural networks (millions of parameters). Furthermore, they show that by introducing full plasticity connections to the network (each connection having its unique plasticity coefficient), they show that they can achieve better results in several domains, including Pattern memorization of binary sequences and natural images, one-shot pattern classification and reinforcement learning. In each of these domains, the plasticity improved the performance of the neural network. The performance was further improved when full plasticity was used (Not a single shared plasticity coefficient for all connections). The exact working of the differentiable plasticity is still not fully discovered, and the authors propose several possible directions to expand the research. One of them is using plasticity in sequential domains. Another one is to introduce plasticity into LSTM. The final one and probably the most interesting from a biological perspective is to introduce neuromodulation into the network.

2.3.2 Backpropamine

After releasing the paper Differentiable plasticity [29], the authors extended their work and released the paper Backpropamine [30], which extends the differentiable plasticity framework with neuromodulation. Neuromodulation is a mechanism that allows the algorithm to modify its own plasticity based on its output. It can be seen as a function of its inputs and computations. By allowing the algorithm to modify its own plasticity based on its computations, it can combat catastrophic forgetting of previously acquired knowledge, by filtering out irrelevant events while selecting which new information to incorporate. Furthermore, this mechanism can also be used to implement a self-contained reinforcement learning algorithm. Neuromodulation is present in our brain and is vital for our brain to work properly. This algorithm tries to imitate a very simplified version of neuromodulation occurring in the brain.

The paper introduced two exact ways on how to introduce neuromodulation. The first way is replacing the learning rate of plasticity η with network-computed, time-varying neuromodulatory signal M(t). By doing so, we allow the network to devise its own algorithm for controlling forgetting and memorizing. From the implementation perspective, the neuromodulatory signal is achieved by introducing a new single neuron to the network's output layer and connecting it with the previous layers using a linear layer.

In neuromodulated networks the equation for the output of the network is identical to the differentiable plasticity's equation (Equation 2.3). The difference is in the equation for calculating the Hebbian trace (Equation 2.4 in differentiable plasticity). The changes are that the learning rate of plasticity η is replaced by the time-varying neuromodulatory signal M(t). And the resulting Hebbian trace is hard clipped to output values in specific a range, in our case [-1,1].

$$\text{Hebb}_{i,j}(t+1) = \text{Clip}(\text{Hebb}_{i,j}(t) + M(t)x_i(t-1)x_j(t)) \tag{2.5}$$

The second more sophisticated way to introduce neuromodulation is using retroactive neuromodulation and eligibility traces. This method is inspired by the short-term retroactive effects of neuromodulatory dopamine on Hebbian plasticity in animal brains [45]. This equation is a modification of Equation 2.4:

$$\text{Hebb}_{i,j}(t+1) = \text{Clip}(\text{Hebb}_{i,j}(t) + M(t)E_{i,j}(t)) \tag{2.6}$$

$$E_{i,j}(t+1) = (1-\eta)E_{i,j}(t) + \eta x_i(t-1)x_j(t)) \tag{2.7}$$

Where eligibility trace $E_{i,j}$ at connection i,j, is an exponential average of the Hebbian product of pre- and post-synaptic activity, with trainable decay factor η.

The performance of the following algorithm had been measured in three environments. The first environment is the Cue-Reward association. This environment is very similar to the Binary Sequences environment used in our work, the main difference is that it is a reinforcement learning problem. In the environment, the agent is first shown four binary patterns, and one of them is chosen as the target cue. The agent is then shown two patterns and must decide whether the target pattern is present or not. In this environment performance of modulated plasticity drastically outperforms non-modulated plasticity and non-plastic networks. The authors hypothesize that neuromodulation helps in memorizing reward associations with high-dimensional stimuli. The second environment is the maze environment, introduced in differentiable plasticity. Here the neuromodulated plasticity yields better results than the non-modulated plasticity. In the last environment, the authors use plasticity for language modeling. For this environment, they augmented LSTM with plasticity. This is the first time a differentiable plasticity framework had been used on a natural language processing task. In the results, they had managed to beat the performance obtained by [46].

The authors of the paper state that this is the first work introducing a neuromodulated plastic network that can be trained with gradient descent. They had taken inspiration from the field of evolutionary computation where the neuromodulated plasticity had been used. An overview of findings on neuromodulated plasticity from the view of evolutionary computation is provided in [41].

2.3.3 Hebbian learning recent work

In our work, we had decided to focus on the differentiable plasticity work described in Sections 2.3.1, 2.3.2. However, there are multiple other papers utilizing plasticity to increase the performance of the models. The goal of this section is not only to show a different way to implement plasticity, but also to provide the reader with a better understanding of the possible benefits of introducing plasticity, as shown by other works. In the paragraphs below we provide a summary of each paper and their reported findings.

The work that shares the most similarities with differentiable plasticity framework is Using Fast Weights to Attend to the Recent Past [1]. In the

following work, authors propose using "fast weights" that are used to store temporary memories. The definition of "fast weights" is practically similar to the definition of simple plasticity. The main difference, when compared to differentiable plasticity is, that there is only a single plasticity coefficient used in this work, and it is not trained using backpropagation. Both methods are applied to recurrent neural networks. The following method had been tested in the following four environments: associative retrieval, visual attention, facial recognition, and reinforcement learning. In all the environments, the "fast weights" augmentation yielded better results than the LSTM. In the associative retrieval, the weights enabled the RNN to work more effectively with the recurrent units. They also outperformed LSTM with associative memory [6]. The increase in performance was most visible when the neural network had a low number of recurrent units. In the glimpses environment, the "fast weights" were able to store information about object parts. In the facial recognition domain, it helped with the score and outperformed LSTM. However, ConvNet architecture was still better. In reinforcement learning, the network played a partially observable variant of the game "Catch" described in [32]. There "fast weights" helped the agent to learn faster. A useful finding was that the improvements obtained by fast weights were more significant on larger versions of the game.

In the work Hebbian descent [27], the authors propose a learning rule, which is a combination of gradient descent and Hebbian learning. They claim that their algorithm can be used as a replacement for gradient descent as well as Hebbian learning, especially in the domain of online learning and one-shot learning. They provide arguments for why they think the algorithm will succeed, and then they empirically test them. This paper provides a detailed comparison and also contains a debate and remarks regarding each experiment. Each learning method (Hebb's rule, gradient descent, covariance rule, and Hebbian descent) is tested in multiple environments, including hetero-associative learning, classification, auto-associative learning. This paper helps in understanding the properties of each learning algorithm.

Another use of plasticity is in the Fast Parametric Learning with Activation Memorization [35]. In the work, the authors address the problem of identifying underrepresented classes. This problem is, for example, prevalent in language modeling, which is the main focus of the following work. The authors propose a combination of plasticity and gradient descent called Hebbian Softmax, which is a modification of softmax, with a new learning rule. In the new learning rule weights are first learned using Hebb's rule for a given class. Moreover, after a certain number of class occurrences, the learning algorithm switches to gradient descent completely. The findings are that the following method can retain information over longer time intervals than traditional memory and does not require additional space or compute. The environments used for testing were: Omniglot image curriculum task, language models

on news reports (Giga-Word), books (Project Gutenberg), and Wikipedia articles (WikiText-103).

2.4 External Memory

Neural networks can be extended with external memory to help them with retaining larger amounts of information and increase their performance in domains including NLP [10], Meta-Learning [37] and multiple other domains where memory is used. External memory can be incorporated into a neural network using multiple approaches.

The first approach we will focus on is improving neural language models with a continuous cache [10]. In this work, the authors introduce a memory augmented network, which stores past hidden activations as memory and accesses them through a dot product with the current hidden activation. They had tested their approach on language modeling tasks the Penn Tree Bank [25] and the wikitext2 [28] datasets. This paper is mentioned for two reasons. Firstly it provides a simple approach to augment the neural network with memory while having competitive results. The second reason is that it had been used in the same environments that were used in papers using plasticity. The first comparison is with backpropamine [30], one of the approaches utilizing plasticity studied in our work. Both papers use the Penn Tree Bank training dataset. Unfortunately, the approaches use different underlying LSTM models. However, we can measure the relative increase in performance gained from introducing the memory mechanisms to the LSTM. In this comparison, the cache model yields better results than backpropamine. Another comparison of the external memory approach with plasticity is provided in the paper [35] they compare the results on dataset WikiText-103, where plasticity achieves better results. What is more interesting is that they show that the two approaches can be combined together to obtain better performance than if used individually.

The second approach discussed here is neural turing machines(NTM) [11] and their improvement differentiable neural computer [12]. The neural turing machines consist of two parts: the neural network controller and a memory bank. "Like most neural networks, the controller interacts with the external world via input and output vectors. Unlike a standard network, it also interacts with a memory matrix using selective read and write operations." These networks had been used in the domain of meta-learning in the following article [37]. This work provides us with a comparison between the approach of using external memory and plasticity. The following work and differentiable

plasticity 2.3 had been both tested in the Omiglot environment [22]. For the 5-way, 1-shot omniglot tasks, differentiable plasticity had been able to obtain a 98.3% score while neural turing machines obtained a score of 82.8%. This difference is quite significant. However, it should be noted that the performance of NTM increases with the number of "shots"(episodes) given. For 5-shots, the reported score is 94.9%.

2.5 Meta-learning

Meta-learning algorithms can quickly adapt to new environments not seen during training and learn new concepts only after minimal interaction with the new environment. One of the inspirations for creating this class of algorithms is from observing how fast humans can of learn and adapt to new unseen events. For example, when we know how to play one shooting game, we can rapidly learn another shooting game by looking at it and realizing that we already know how to play something similar. This ability to quickly learn new concepts by utilizing previous training experiences is one of the aims of this technique. Furthermore, it is the reason why meta-learning is also sometimes called "learning to learn." The ability of "learning to learn" is quite different from the direction the current research in AI is heading towards. The current AI algorithms usually focus on mastering a single skill such as Go [39]. Nevertheless, they are unable to learn another similar game quickly. Another technique trying to address the problems above is transfer learning [2]. The main difference between the two methods is that transfer learning aims to reuse parts of an already trained model on a new similar problem, while meta-learning aims to optimize the model to learn new concepts from the initial training.

To train a meta-learning algorithm, we need to create a dataset $\boldsymbol{D}$ containing multiple different tasks which share high-level similarities. We then choose the parameters $\boldsymbol{\theta}$ of the network in such a way to minimize the loss $\boldsymbol{L}$ across the dataset. This is captured in Equation 2.8. The difference between the meta-learning approach and standard approach for training is that in the case of standard learning, we train to minimalize on samples from one dataset. While in meta-learning, we minimize across different datasets and each dataset is counted as a sample [37].

$$\theta^* = \underset{\theta}{argmin}\, \mathbb{E}_{D\sim p(D)}[L_\theta(D)] \tag{2.8}$$

There are countless approaches to meta-learning, which vary greatly in the way they perform meta-learning. The approaches can be loosely divided

into three groups: Model-based, Metric-based, and Optimization-based, as is reported in [9]. The focus of this work will be on the model-based approaches utilizing memory.

In the model-based approaches, where the model is created with mechanisms that are capable of fast adaptation. It had been shown that the LSTM, with a specific setup, is capable of devising its own learning algorithm from scratch [16]. This paper introduces a setup for supervised learning in the domain of non-stationary time series prediction. The above-mentioned work had been further improved by using memory augmented neural networks in the work [37]. The setup can be extended to the domain of reinforcement learning ([43], [7]). Another approach capable of meta-learning is introducing plasticity into neural networks [29]. In this case, the plastic weights are working as a fast memory capable of quickly adapting to the current environment.

The next group of meta-learning algorithms is referred to as metric-based. These algorithms learn a metric which is then used to rank similarity between inputs. After learning a proper metric, the metric is used by the network to generalize across datasets and act even on never before seen data. An example of using this method in one shot-classification is [21].

The last group of meta-learning algorithms is referred to as optimization-based. These algorithms perform meta-learning by optimizing the backpropagation of gradients. An example of a paper using this approach is [9]. In this work, the authors propose an algorithm that tries to find optimal θ^* for that network that can be then rapidly fine-tuned using a few steps of gradient descent to perform optimally in a given task.

2.6 Reinforcement learning

2.6.1 Introduction

Reinforcement learning is a method used for solving problems that are defined as finite Markov decision processes(MDPs). Markov decision processes are used to formalize the problem of sequential decision making, where the future states of the process depend only on the current state. This property of MDPs is known as the Markov property. Because the future states of the Markov decision process are affected by the current state, a delayed reward needs to be taken into consideration in order to solve MDPs optimally. MPDs

are defined as a 5-tuple (S,A,P,R,γ), where S is a finite set of states, A is a finite set of actions, P is the probability of taking action a in state s_t and ending in state s_{t+1}. R is the reward obtained after transitioning from state s to state s'. γ is the discount factor used for defining the trade-off between immediate and delayed rewards.

The purpose of reinforcement learning is to maximize the cumulative reward obtained by the agent. This is done by controlling the agent's actions based on inputs received from the environment. The term agent refers to the part of the algorithm interacting with the environment. The agent receives information from the environment about the state $S_t \in S$ it is currently in, and based on the information received, selects an appropriate action $A_t \in A(s)$ to perform. After performing the action, the agent receives a numerical reward, $R_{t+1} \in R$ and information regarding the state S_{t+1} it is in after performing the action. The agent chooses which action to perform based on his current policy π. Agents policy defines what action the agent will perform in each state. The optimal policy tries to get the agent into states with the highest expected, accumulative, discounted reward. The state value function $V_\pi(s)$, represents the value of the accumulative sum of rewards the agent will obtain if it is in the state s and follows the policy π thereafter. We will also define the Q function $Q_\pi(a, s)$, which represents the value of the accumulative sum of rewards of taking action a in state s and thereafter following policy π. Another important definition used later in the text is the Advantage function $A(s, a)$, which is used to measure how much better is taking action a in comparison to other actions available in state s. The advantage function is obtained using the following equation 2.9. The transition function t, defined in equation 2.10 records the probability of transitioning from state s to s' after taking action a while obtaining reward r.

$$A(s, a) = Q(s, a) - V(s) \tag{2.9}$$

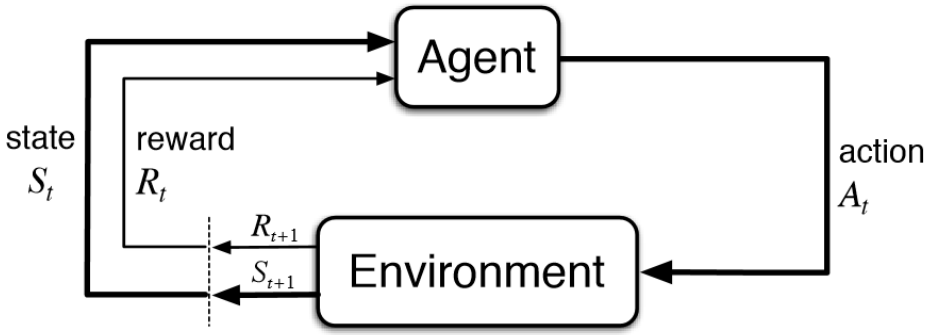


Figure 2.3: Interaction of the agent with the environment, from [42]

$$t(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a] \tag{2.10}$$

2.6.2 A2C

Deep reinforcement learning is a combination of reinforcement learning and deep learning. The neural networks from deep learning are used to approximate the value of a state based on the input provided. Function approximation using neural networks enables reinforcement learning to be used in domains with large state spaces and continuous values. There are multiple approaches currently used to perform reinforcement learning. We recommend the following overview of deep reinforcement learning techniques [24]. In our work, we focus on the Advantage Actor-Critic (A2C) algorithm a synchronous version of A3C [31]. The reason for using this algorithm is that it is used in the meta-reinforcement-learning method we will use [43]. The following algorithm belongs to the policy gradient methods optimizing the policy π of the agent, in respect to the parameters of the neural network θ. These methods use the policy gradient theorem 2.11, proof provided in [42]. In the equation, t refers to the transition function shown in equation 2.10. The policy gradient theorem is used to estimate the gradient for the policy π outputted by the neural network. The neural networks outputs in addition to the policy π also the Value function V, which is used to calculate the Advantage function A, shown in formula 2.12. To train the Value function V the MSE loss function is used on the difference between the output of the network for V and the actual value for the state. The Advantage function A is used to evaluate the resulting policy. The A2C algorithm is an on-policy learning algorithm, which means that it is able to learn only from data produced by its own policy.

$$\nabla_\theta J = \mathop{\mathbb{E}}_{s,a\sim\pi_\theta,t}\Big[A(s,a) \cdot \nabla_\theta \log \pi_\theta(a|s)\Big] \tag{2.11}$$

$$A(s,a) = r + \gamma V(s') - V(s) \tag{2.12}$$

2.6.3 Meta-reinforcement learning

There are multiple definitions of meta-reinforcement learning. In our work we will use the specification provided in papers ([43], [7]). The goal of

meta-reinforcement learning is to address problems of reinforcement learning mentioned in [23]. The two most important deficiencies of reinforcement learning, the meta-reinforcement learning addresses are the massive volume of training data required to train a reinforcement learning algorithm and the ability to adapt to changes in the environment conditions [43]. These problems can be seen in the Maze environment, the environment changes after each episode because the reward in the maze is located to a new random position. The agent then must locate the reward to remember its new position and adapt its model to incorporate the new knowledge. By doing so, the agent will maximize the number of times the reward is reached within an episode. In comparison, a non meta-learned A2C algorithm would fail because it would not be able to adapt to the change in the environment fast enough. In the paper [7], the authors state that classic RL algorithms fail because they lack a good prior, which results in deep RL agents needing to rebuild their knowledge about the world from scratch.

This brings up the question of how specifically the meta-rl algorithm can cope with this problem. "In meta-reinforcement learning, the dynamics of the recurrent network come to implement a learning algorithm entirely separate from the one used to train the network weights. Once again, after sufficient training, learning can occur within each task, even if the weights are held constant. The algorithm is a full-fledged reinforcement learning algorithm that negotiates the exploration-exploitation trade-off and improves the agent's policy based on reward outcomes" [43].

The meta-rl maximizes the reward over a distribution D over Markov Decision Processes (MDPs). And in each step the action a_t is executed as a function of the whole history $H_t = x_0, a_0, r_0, ..., x_{t-1}, a_{t-1}, r_{t-1}, x_t$. This is achieved by using a network with memory and providing the network with additional input in addition to the standard input s_t (state of the environment in time t). The additional input includes the action at the previous step a_{t-1} and the reward at previous step r_{t-1}. The final step required to implement meta-reinforcement learning is to reset the memory when new MDP is sampled. The A2C algorithm is used for training the network, introduced in Section 2.6.2. However, other algorithms can be used, for example, Trust Region Policy Optimization (TRPO) [38] as used in [7].

Chapter 3

Binary Sequences

3.1 Environment overview

3.1.1 Environment specification

In the Binary Sequences environment, our goal is to test how well the algorithms can remember binary patterns and reconstruct them after being presented with their degraded versions. The main challenge in this task is the random generation of patterns for each episode, which requires the algorithm to be capable of rapidly storing them instead of remembering them from previous episodes. Another obstacle is that although each pattern is unique, they can share a high level of similarity, this can be challenging because the algorithm needs to remember each bit precisely to correctly reconstruct the pattern. In each episode the algorithm is presented with five patterns, after showing the patterns once they are presented the second time in a different order. After the two presentation cycles, a degraded version of one of the previously seen patterns is provided to the algorithm, and its goal is to output its full version. Each pattern has a size of 50 bits and is shown for six steps followed with six steps of empty input. The role of the empty input is to test whether the algorithm can retain the information even in the absence of input. The degraded pattern is created by degrading 50% of the bits. An example of one episode is provided in Figure 3.1. The Binary Sequences environment is specified using multiple parameters, their exact values are specified in the Table 3.1.

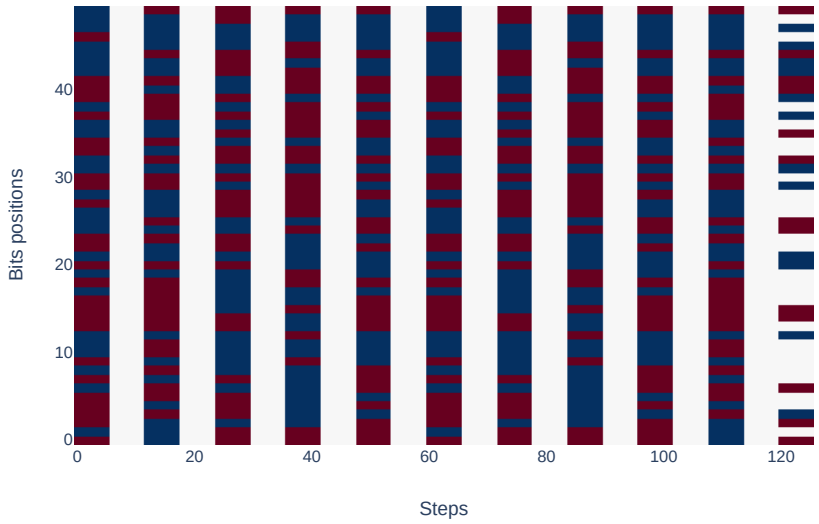


Figure 3.1: The figure shows one episode of the Binary Sequences environment. The episode contains five binary patterns. The patterns are shown in 2 presentation cycles. Each presentation cycle has a different order of the patterns. The size of a pattern is 50 bits. Each pattern is displayed for six steps after presenting a pattern, a phase of six steps where an empty pattern is shown. At the end of the presentation cycles, a degraded pattern is shown, in our case, 50% of bits of the test pattern are degraded.

- pattern size - determines the number of bits the pattern is composed of
- number of patterns - defines the number of patterns that will be shown during an episode
- probability of degradation - sets the probability of a bit to be set as unknown in the final pattern.
- presentation time - each pattern is shown for x steps in succession.
- presentation time degraded target pattern - how many times is the degraded target pattern shown
- delay between presentations - after showing an binary pattern x steps of empty output are shown.
- number of presentation cycles - each presentation cycle consists of showing all the patterns once. Each presentation cycle has a random order of patterns.
- episode length - total length of the episode.

pattern size	50
number of patterns	5
probability of degradation	0.5
presentation time	6
prestime degraded target pattern	6
delay between presentations	6
number of presentation cycles	2
episode length	126

Table 3.1: The exact values of parameters for the Binary Sequences environment

3.1.2 Neural networks used

In the Maze environment, we test the following recurrent neural networks vanilla RNN (Section 2.2), LSTM (Section 2.2.1). For the plastic neural networks we test the combination of plasticity with multiple different layers. The first combination is with the recurrent layer proposed in the original work [29] (RNN + PLAST). The second combination is with the feed forward layer (FF + PLAST). The last combination is unmodified input (PLAST). The specification for the recurrent layer is provided equation 2.3. To get the equation for FF we replace the recurrent layer $w_{i,j}x_i(t-1)$, with a feedforward layer $w_{i,j}x_i(t)$ and for PLAST we replace the recurrent layer with $x_i(t)$. Note that the feedforward layer can be trained to output unmodified input and thus can work as PLAST. For every combination of plasticity we also test different plasticity modifications and in comparisons take the best performing modification. More specifically homogenous plasticity (one plasticity coefficient for the whole network), full plasticity (each connection has its plasticity coefficient), and neuromodulated plasticity.

Each neural network architecture is tested with the number of neurons for each layer corresponding to the binary pattern size and two layers, except the PLAST network which has only its specific layer. The reason for PLAST having only one layers is that we wanted to test solely the Hebbian Learning. The first layer of each architecture is a unique layer for the given architecture. The second layer is a feed forward layer. Adding more layers to the network did not improve the performance. Setting the same number of neurons and layers is unfair to some architectures, which are less computationally demanding or has fewer parameters. For example, if we consider RNN and LSTM, LSTM has four times more parameters. Another option would be to calculate the number of neurons for each architecture so both networks would have the same number of parameters. The third option would be to compare the networks based on computational time.

The activation for all architectures is hyperbolic tangent(tanh). Each network is trained using the mean square error loss function(MSE). The MSE is calculated from the difference between the network output and the target pattern. The backpropagation is optimized using Adam optimizer [20]. All the hyperparameters are summarized in Table 3.2.

Net types	FF, RNN, LSTM, FF+PLAST, RNN+PLAST, PLAST
Plast types	Homogenous, Fully plastic, Neuromodulated
Learning rate	0.001, 0.00055, 0.0001
Loss function	Mean Squared Error
Batch size	32
Optimizer	ADAM
Activation function	TANH
First Layer	Net type specific
Second Layer	Linear
Layers sizes	50, 50
Start eta	0.01
Start alpha	0.01

Table 3.2: Hyperparameters of the neural network used for the Binary Sequences environment

3.1.3 Environment analysis

The Binary Sequences environment has several properties that can make it challenging for the algorithm. In the following section, we provide an analysis of the environment. Our goal here is to specify the obstacles the algorithm needs to overcome in order to succeed in this environment. By doing so, we also showcase properties of different approaches for using memory. This environment had been proposed in the work of [17], where it had been shown that hand-designed recurrent neural network with homogenous plasticity can solve this problem.

Probably the most significant obstacle is that in the default definition of the environment, each episode contains different patterns with a different order and is shown only once. The algorithm needs to be able to do one-shot learning to solve this environment successfully. In order to do so, the method should be able to learn new patterns quickly. In the case of plasticity, this mechanism is present in the form of plastic connections, where it had been shown that the connections are capable of storing information rapidly. For recurrent neural networks, the network needs to encode the information from previous steps into a hidden state using recurrent connections. It could be challenging for the recurrent neural networks to learn useful weights from scratch, because there are almost none general concepts to learn between episodes to encode

information more efficiently. In fact trying to learn concepts between episodes could hurt the performance. For example, if the algorithm observed in one episode, that first pattern is always one and then learned it and used it in another episode, where this was not true.

The next challenging aspect of the Binary Sequences environment is that the binary patterns can have a large number of identical bits. Thus the algorithm needs to be able to store into memory each bit perfectly to be able to distinguish between the individual patterns.

Another problem with the size of the pattern is that the network needs to store all the patterns and recall them at the end of the episode. This can be done by storing all patterns or creating an encoding and decoding mechanism which can retrieve them. To store all the patterns in an unchanged way, we need a memory of the size equal to $number\ of\ patterns * pattern\ size$. This could pose a problem for a standalone recurrent network, which only has a hidden state (memory size) equal to the number of neurons. The environment is also suitable for finding how memory works. We can measure how long it can retain information by extending the length of the episode. The task of the empty patterns is to see how the memory behaves in the absence of input.

One of the reasons for choosing this environment is that the algorithms' memory and output can be clearly visualized and interpreted. When determining the final output of the network, we look at each neuron's value and set the final pattern to 1 if the value is positive and -1 if it is negative. This means we can intuitively measure how confident the network is about its decision based on the value. If the value is close to zero, the network is not that certain about it, and the higher it gets, the more confident the network is about it. We can also measure the role of memory based on this. For example, if plasticity always has a higher value of the output. It is the dominant memory used.

Favorable property of the environment is the low computational cost. In order to simulate an episode of the environment, we only need to generate random binary patterns. In contrast, for example, if we had tried to solve a problem in the reinforcement learning domain, we would also need to simulate the environment. Along with the low computational cost, the environment offers a wide variety of parameters that can be used to test the memory properties. For example, if we increased the number of patterns shown during each episode, the neural network would need to retain more information in the memory.

3.1.4 Experiments performed

Our first steps were conducting a general comparison of the performance of different approaches. The goal of this experiment is to get an overview of how well each approach performs in the environment. Our findings are described in Section 3.2.

The experiment's results are that the solutions using plasticity dominated the non-plastic solutions. In the next experiments, we try to show why plasticity is so efficient in this domain and the reasons behind the low performance of non-plastic solutions. For understanding previous results, we use several methods. The first method is looking into what is present in the network's memory during the episode by analyzing the hidden state of the recurrent network and the weights of plastic connections. Our findings are shown in Section 3.4.2. Another way to dive deeper into the problem is by measuring the contribution of different types of memory (plasticity, recurrency). In this environment, this can be done by analyzing the output strength of each system. We researched this approach in Section 3.3.

Afterward, we looked more closely into the separate methods. In plastic networks, we had looked into how different parts and definitions of plasticity compare. For example, we measured how making the network fully plastic instead of homogenous improves the performance. Plasticity specific experiments are located in Section 3.4. A summary of the most important findings is provided in Section 3.5.

3.2 General comparison

In the following section, we compare the final performance and learning speed of the algorithms. To achieve the best performance for each algorithm, we had searched for the optimal learning rate. When searching for hyper-parameters of algorithms, there are usually more of them searched. Those hyper-parameters include layer size, number of layers, activation functions, or batch size. In our case, we did not search the optimal number of layers and their respective sizes. The number of neurons was set to the size of binary patterns, which is 50. The full specification for neural networks used in this experiment is provided in Section 3.1.2. Each neural network configuration was executed three times with different seeds. The average of the rewards had been taken as a result. We found this setting sufficient because the difference between individual runs was low.

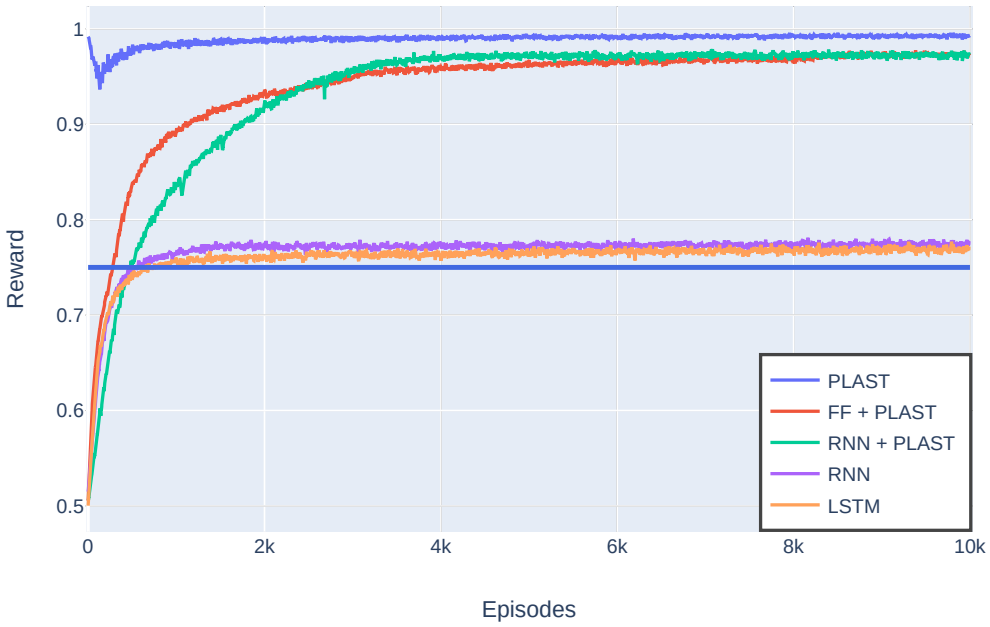


Figure 3.2: The following figure shows the best performance obtained by each architecture in the Binary Sequences environment. The reward measures how many percent of the wanted pattern the network successfully remembered. The reward of one signifies 100 percent correct answers. The results of each architecture are taken as the mean from 3 independent runs. The rewards were collected for each episode and then smoothed using a moving average from 10 episodes. The blue horizontal line represents the baseline performance.

Figure 3.2 shows the results of the experiments. The baseline for the algorithms is 75% percent of correctly remembered bits of the target pattern. This baseline corresponds to outputting the non-degraded bits of the degraded target pattern and randomly filling in the remaining bits. The baseline for 50% stands for outputting a random pattern without utilizing any information from the degraded target pattern. We will start with the commentary of the best performing method to the worst-performing.

The experiment shows that the standalone differentiable plasticity was able to outperform the other solutions. It achieved a performance of 98% correctly remembered bits during the first episode without any training (updating the coefficients of plasticity through gradient descent). After the first episode, there is a decrease in performance. This is the effect of training the plasticity coefficients using gradient descent. After 500 episodes, we can

Net type	Mean reward	Max reward
PLAST	99.3 ± 0.04	100.0
FF + PLAST	97.3 ± 0.08	98.8
RNN + PLAST	97.3 ± 0.11	99.3
RNN	77.4 ± 0.10	80.6
LSTM	76.9 ± 0.11	79.9

Table 3.3: The table contains information regarding the rewards obtained by each approach. The mean reward is calculated from the last 100 episodes. The ± shows the 95 % confidence interval over the last 100 episodes. The max reward is the maximal reward obtained in any episode.

see that the gradient descent finds sufficient coefficients, and the performance of plasticity starts to increase again and outperform the initial performance. The increase in plasticity performance compared to the initial setting shows that introducing differentiable plasticity improves its performance.

The second-best performance was achieved by a feedforward neural network combined with plasticity. Its performance is very similar to the one obtained by vanilla RNN combined with plasticity. Both networks were able to obtain solid results in this environment. The faster speed of training for feedforward networks was expected because it is a much simpler architecture and does not need to train to encode information from previous states. However, it is surprising that the recurrent neural network did not obtain better performance after sufficient training. We can see that after 4000 episodes, the increase in the reward obtained by the network is minimal. In theory, the recurrent neural networks should work better because they can use the hidden state for storing information through observations. Another important observation is that although combining plasticity with layers trained using gradient descent obtained good results compared to standalone plasticity, it seems that combining these two methods is not helpful for this problem. This is a useful finding because we can see that combining these two architectures is not always beneficial. It will be interesting to explore in which environments these two approaches complement each other. The first reason behind the decrease in performance could be that plasticity is enough for solving this environment. Another explanation is that the recurrent neural networks are insufficient for solving this environment, as can be seen from their respective results. Testing only feedforward network does not make any sense, because there are no mechanisms for storing history.

The last group of experiments consisted of recurrent neural networks. This mechanism for memory has not performed very well in this environment. It had been able to achieve an average reward of 77.4 for RNN and 76.9 for LSTM. These scores are only slightly better than repeating the degraded

Learning rate Net type	0.00010	0.00055	0.00100
PLAST	98.8	99.2	99.3
FF + PLAST	92.8	97.3	97.3
RNN + PLAST	90.9	97.3	97.1
RNN	76.7	77.3	77.4
LSTM	76.1	76.9	76.9

Table 3.4: The table shows the average reward obtained from the last 100 episodes for each network and learning rate. We can observe that the optimal learning rate is not influencing the resulting performance significantly.

pattern. In our experiments, we had tried to increase the training duration ten times to 100000 episodes. Nevertheless, the final reward practically did not improve. In the analysis of the environment, we had provided several reasons why this environment can be challenging for the recurrency.

3.3 Memory systems contributions

In order to investigate how the combination of non-plastic and plastic layers works, we had visualized the outputs of each component during the episode. Our aim in this experiment is to show how these two approaches work together. First, we had visualized the final output of the network, provided in Figure 3.3.

We will start examining the experiment by discussing what happens when the degraded pattern is shown in step 120 to step 126. It can be seen that the network had learned to repeat the non-degraded parts of the pattern and is significantly more confident in them then in the degraded parts. When filling in the degraded parts, it is visible that the network starts with low confidence, but as the pattern is repeatedly shown, network output confidence starts to increase. This effect is most visible in the case of neuron in position 12. Here we can see that the neuron first outputs the wrong bit but gradually changes its decision to the correct one. We can see that this event only occurs when the previous bit in the same position has the opposite value to our bit. This raises a question of which exact role the gradual weakening of the inputs occurring during the presentation of empty patterns play. Another thing to notice is that the network benefits from seeing one pattern multiple times in succession. This can be observed when the network is shown a new pattern. In the first output, the network is less confident than in the successive outputs. Also, note how the outputs of the network behave in the absence of input

(presentation of an empty pattern). We can see that the network still outputs the pattern previously seen but in much lower intensity. When observing the output of the network when an empty pattern is shown, we can see that the output is higher during the second presentation cycle.

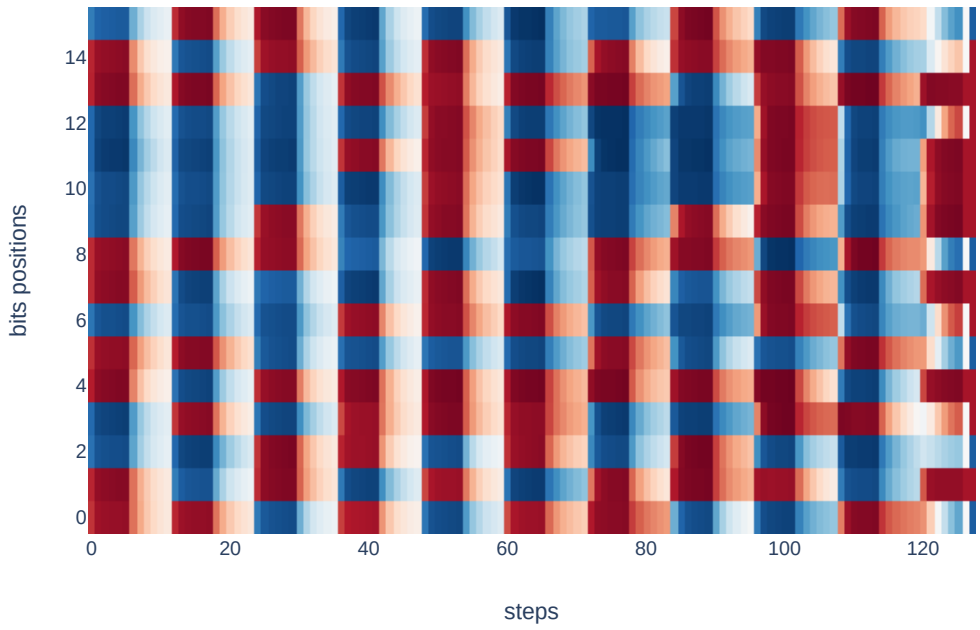


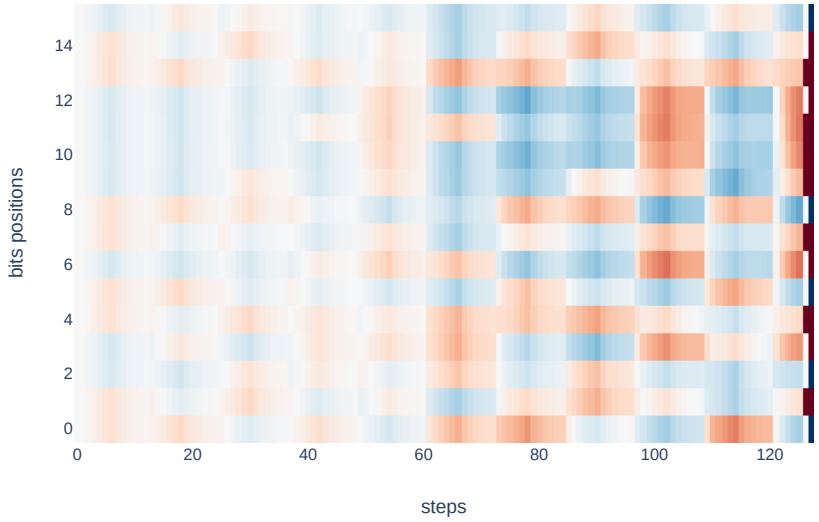
Figure 3.3: The following figure captures the outputs of the neural network during the episode. Note that the last two steps (127, 128) are not actual outputs but represent the degraded pattern shown to the network and its non-degraded version. We had added these to the figure to improve the interpretability of results. Another augmentation for better readability was that we only show the first 16 neurons. In the figure, the blue color corresponds to positive values and the red color to negative. The more intense the color is, the more positive/negative the value is. This can also be interpreted as the neural network is more sure about the value for neuron. The degraded pattern is shown to the algorithm in steps 120 to 126.

After the initial analysis, we had visualized the contribution of each approach separately. A distinct separation of the two approaches can be done for interference. If we look at the equation for implementing differentiable plasticity 2.3, we can see that the output is done by adding the plastic and fixed parts of the network together. Combine this with the interpretability of binary sequences environment where the output directly corresponds to the confidence, and we can clearly decide the individual contributions. For measuring the contributions, we take the values before applying the tanh

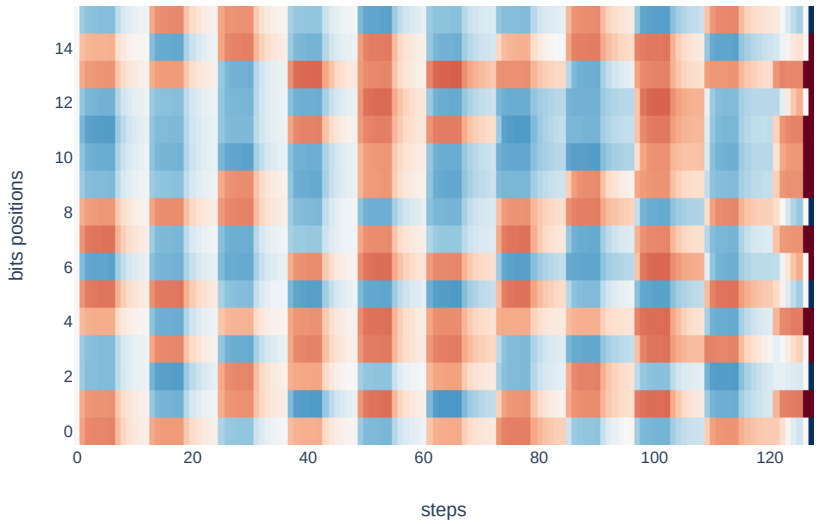
function. A visual comparison of the contributions is shown in Figure 3.4.

From the visualization, it seems that the recurrent part's contribution is to repeat the observed pattern during each turn. However, this can be achieved simply by utilizing a feedforward network. By looking closer at the behavior of the recurrent neural network during filling in the degraded pattern, we can see that it can negatively impact the outcome. This is visible when observing how the layer behaved when the bit in the pattern shown before the degraded target pattern had a different bit. In this case, the recurrent layer first outputs the incorrect solution, while the plastic layer outputs the correct answer. If we look at the plasticity, we can see that it has a significantly smaller impact on the output during the presentation phase. However, when the algorithm starts to decide the missing bits in the degraded pattern. It has a higher value than the recurrent layer in all bits. This means that the decision regarding the missing bits is influenced solely by the plastic part.

The following experiment's main result was that the recurrent neural layers in this environment function as a simple feedforward layer. In cases when the pattern before the target pattern had an opposite bit value, they even decreased the performance. The choice of filling in the missing patterns is made purely by the plastic part.



(a) : Plastic part



(b) : Reccurent part

Figure 3.4: The figure shows the contribution of plasticity and recurrency to the output of the neural network. The intensity of color represents the network certainty with the bit value, the higher, the better. When measuring the contribution, look at the intensity of the bits in the same position and step in the two different layers. The presentation of the degraded pattern is from steps 120 to 126. The steps 127 and 128 are the degraded pattern and its full version.

3.4 Plasticity analysis

The experimental results and the analysis of the inner working of the neural network showed that plasticity is the best performing approach for this problem, shown in Section 3.3. It had been able almost entirely to remember and associate the binary patterns, but it still did not manage to learn to remember every pattern without an error. The plasticity framework can be dissected into multiple parts. For example we can use full plasticity or homogenous plasticity, use differentiable plasticity coefficients α or not and multiple other augmentations. In Subsection 3.4.1, our goal is to get insight into how different plasticity parts affect the performance and what do they achieve. In the other part of this Section, we will take a closer look into the trained plasticity in plastic neural network 3.4.2. The experiments will be done on the PLAST neural network introduced in Section 3.1.2.

3.4.1 Plasticity components comparison

The plastic neural network is composed of several components which are added together and form the resulting layer. In this section, we will conduct a comparison of the attribution of the components to the overall performance of the algorithm. The most simplified formula for plasticity is using just the hebb rule specified in Section 2.3 (HEBB). The next four architectures will all use the plasticity defined in Section 2.3.1. The first architecture will have static plasticity coefficients α and static plasticity learning rate η (Hebb Static Eta), where alpha is set to 1 (one plasticity coefficient for the whole network), eta was found using grid search to 0.01. The second architecture have the same α as the previous one, but the parameter η is optimized using gradient descend (Hebb Differentiable Eta). The third architecture optimizes both the α and η using gradient descent and has one α for the whole network (Homogenous Plasticity). The fourth architecture optimizes both the α and η using gradient descent and has α for each connection instead of only one for the whole network, as was the case in the previous architecture (Full Plasticity). The last architecture is specified in Section 2.3.2, which extends the architecture defined in Section 2.3.1 with neuromodulation (Modulated Plasticity).

The full plastic network has obtained the best final performance. The network has lower performance in the initial episodes because it contains more parameters to learn than other methods. It takes approximately 5000 episodes to learn the parameters to a sufficient degree, and afterward, it performs better than the others. This proves that adding full plasticity

helps the network perform better at the cost of slightly slower learning. The training speed of homogenous plasticity is almost similar to the speed of full plasticity. This is interesting when taken into consideration the overall number of parameters for each approach. The modulated plasticity is the most sophisticated form of plasticity in this experiment. So the slower learning speed was expected. However, we also expected that the additional complexity of adding neuromodulation would increase the final score, which did not hold.

Probably the most significant finding that came out of this experiment is how crucial is the η parameter for solving this task. This is most visible in the case of differentiable η. We can see that the following solution is able to obtain almost perfect scores even in the first episodes, and its final performance is on par with more complicated solutions. This suggests that the η parameter contributes more to the performance than the plasticity coefficients. This claim is further supported if we look at the performance of static *eta* and hebb without *eta*. Here we can see that by introducing the plasticity learning rate into the equations, we obtain a 98% score in comparison to the score of 75% without them.

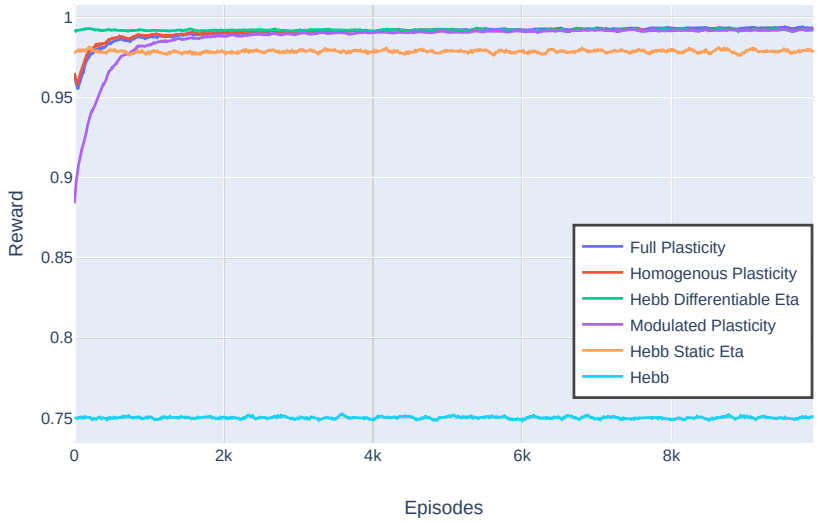


Figure 3.5: The following figure depicts the results of comparing the relative contribution of different plasticity components and augmentations. The reward corresponds to the percentage of correctly recalled bits of the degraded pattern. Where 1 signifies 100% correctly recalled bits. The reward is calculated as a moving average from 10 episodes. For magnified version, please refer to Figure 3.6

Figure 3.6: The figure is a magnified version of Figure 3.5, focused on better capturing the differences between the best performing plasticity augmentations. The x-axis starts at step 1000 and ends at 10000.

Plasticity type	Mean reward	Max reward
Full Plasticity	99.34 ± 0.04	100.0
Homogenous Plasticity	99.24 ± 0.04	100.0
Hebb Differentiable Eta	99.23 ± 0.04	100.0
Modulated Plasticity	99.20 ± 0.05	100.0
Hebb Static Eta	97.90 ± 0.08	99.94
Hebb	75.09 ± 0.13	77.44

Table 3.5: The table contains information regarding the rewards obtained by plasticity modifications. The average reward is calculated from the last 100 episodes. The ± shows the 95 % confidence interval over the last 100 episodes. The max reward is the maximal reward obtained during one episode.

3.4.2 Plasticity visualization

The main improvement of the differentiable plasticity framework is the introduction of differentiable plasticity rates for each connection. In the following experiment, our goal was to observe the final values of plasticity coefficients after training. The visualization is provided in Figure 3.7. As can be seen from the visualization, the plasticity coefficients share similar values except for the coefficients on the diagonal. The diagonal plasticity coefficients are the coefficients for connections between the same neuron but in different episodes. This means that for determining the output of a plastic neuron in the problem of binary sequences, the previous value of the neuron is most important. We could hypothesize that the following rule was created to copy the non-degraded patterns when the degraded pattern is shown. Another useful insight gained from this experiment is that we see that the resulting plasticity coefficients do not differ significantly from the uniform fully plastic parameters, except for the diagonal. We believe that the larger plasticity coefficients in the diagonal are the reason behind the better performance of differentiable plasticity in comparison to standard plasticity (Section 3.4.1).

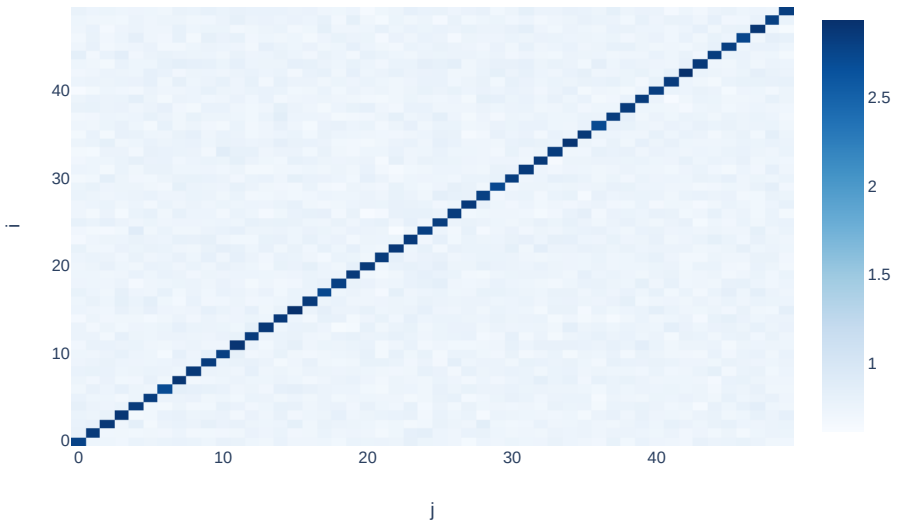


Figure 3.7: The Figure shows the final values for plasticity coefficients α. We can observe a clear pattern in the plasticity coefficients, at the diagonal.

3.5 Summary

In the Binary Sequences environment, the neural networks using plasticity had outperformed the recurrent neural networks in a significant manner. The plastic neural networks had almost correctly recalled the degraded pattern with an accuracy of 99.3%. In comparison, the standalone recurrent neural networks obtained an accuracy of 77.4%. Afterward, we had further analyzed what contributes most to the performance of plastic neural networks. When analyzing the performance, we found out that the combination of recurrent layers and plastic layers is not required in the Binary Sequences environment. The combination of these two layers results in slower learning and decreases the accuracy to 97.3%. The following fact was found by calculating each layer's contribution to the output of the network. We had found that the final output is only affected by the plastic layer. This is an interesting finding because using only differentiable plasticity with a feedforward network was not considered for the Binary Sequences in work [29]. Moreover, it brings out the question of how well the combination mentioned above performs in other environments.

The plasticity framework consists of multiple components and parameters. Our next line of research was to determine which component affects the performance of plasticity the most. From our experiments came out, that the most influential parameter for plasticity performance is the η plasticity learning rate. When the parameter η was set to its optimal value, the plastic network without any differentiable part had reached an accuracy of 97.9%. By introducing differentiable plasticity coefficients α and differentiable η, the accuracy improved to 99.3%. Adding neuromodulation to the network decreased the accuracy to 99.2%. Furthermore, we investigated the final α coefficients of the network. Moreover, after training them using gradient descent, we found out that the resulting values are for most connections identical, except for the connections between the neurons at the same positions but in different layers. The relatively simple structure of the pattern in the resulting α coefficients could signify that the Binary Sequences environment is not challenging enough to bring out the full potential of introducing differentiable plasticity.

From the findings above, we conclude that the plastic neural networks were able to outperform the recurrent neural networks in this environment due to algorithm predispositions and not because of the differentiable plasticity. We hypothesize that the weak performance of the recurrent neural networks is caused by the size of their hidden state and the incapability of learning useful weights for connections in this environment due to the randomness in it.

Chapter 4

Maze

4.1 Environment overview

4.1.1 Environment specification

In the following environment, the agent's goal is to explore a maze and find the reward within it. After finding the reward, the agent is transported to a new location. The reward remains at its original position during the episode. The goal of the agent is to maximize the reward obtained during an episode. After each episode, the reward's position is changed to a random empty space in the maze. The structure of the maze remains unchanged thorough episodes. An episode of the environment is provided in Figure 4.1. In each step, the agent obtains an input containing its current view, previous action(encoded in binary), reward obtained in the previous step, current timestep and a bias neuron. Resulting in the total input size of 17. The view contains the 3×3 neighborhood of the agent, with the agent being in the center. The walls are represented by 1 and empty tiles by 0. The reward is not visible to the agent. A list of the environment parameters is provided below. Their exact values are provided in Table 4.1.

- reward - the reward obtained from collecting the reward
- wall penalty - penalty introduced for moving into the wall

- maze size - the size of the maze including the outer walls
- episode length - the length of one episode
- view range - the area the agent can see around itself
- reward visible - if the reward is visible to the agent when it is in its view range
- agent respawn position - if the agent always respawns on a fixed/random position.

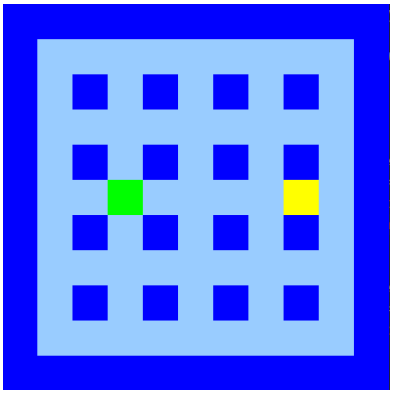


Figure 4.1: The figure shows the maze environment. The environment consists of the 9×9 grid occupied with walls on the sides. The green tile represents the agent. The reward is depicted as the yellow tile. The goal of the agent is to maximize the reward. After reaching the reward, the agent is transported to a random empty tile in the maze. The reward stays at the same location. Each episode has a duration of 200 steps.

Reward	10
Wall penalty	0
Maze size	11
Episode length	200
View range	3
Reward visible	False
Agent respawn position	Random
Number of actions	4
Input size	17

Table 4.1: The table contains the exact values for the Maze environment parameters.

4.1.2 Neural networks used

For the following task, we use a three-layer neural network trained using the A2C algorithm described in Section 2.6.2. The first layer of each network is

realized using a linear layer, followed by the activation function. The second layer of the network is unique for each network. The final layer is a linear layer. Each network outputs 4 values for each action and 1 value for the state value function. The agent takes action based on the probabilities obtained by applying softmax to the outputted values for actions. The networks using neuromodulated plasticity have 1 additional output used for neuromodulation. The neural network architectures are described in Section 3.1.2.

Net types	FF, RNN, LSTM, FF+PLAST, RNN+PLAST
Plast types	Homogenous plasticity, Fully plasticity, Neuromodulated
Learning rate	0.0001
Batch size	16
Optimizer	ADAM
Activation function	TANH
First Layer	Linear
Second Layer	Net type specific
Third Layer	Linear
Second Layer size	100
Output size	5
Start eta	0.01
Start alpha	0.01
RL algorithm	A2C
Bent	0.03
Blossv	0.1
Discount factor	0.9
Gradient norm clipping	4.0
L2 norm	0.000003

Table 4.2: The table contains hyperparameters of the neural network used for the Maze environment

4.1.3 Environment analysis

Reward analysis

To establish the algorithm's performance, we need a way to measure the maximal possible reward that the agent can obtain. To do so, we had implemented an agent that always uses the optimal path to the reward. We achieved this by providing the agent with a view of the whole maze and planning his movements using breadth-first search (BFS). Although this solution gives us a better understanding of the maximal possible reward, it is not precise. There are two reasons why the solution is imprecise. The first is that the agent must first locate the reward by exploring. The second is the

teleportation after finding the reward. To make the estimated maximal reward more accurate, we need to consider the cost of exploring the environment. The environment consists of 65 empty tiles, and the reward is not visible to the agent. In the worst case, the agent must visit all the 65 tiles to find the location of the reward. Also, note that visiting 65 tiles cannot be done in 65 steps, because of the walls. The cost of teleportation is harder to approximate, because first of all it is impossible to determine exact location after teleporting, because most states share representation (detailed explanation provided in Section 4.5). Secondly, the agent can be teleported to a location that was not explored during the exploration occuring before finding the reward. Because of these obstacles, we will subtract a reward of 50 from the performance obtained by the BFS agent. To account for the randomness in the maze, we had run the BFS agent for 1000 episodes and took the mean reward. The final averaged reward of the agent was 294. This means the approximated optimal reward is 244.

Another baseline that we tested was a random agent. The random agent obtained a mean reward of 2.3 from 1000 runs. Another interpretation for this value is that the agent collected the reward in 1 out of 5 episodes. In the default setting of the maze, environment reward is only obtained for collecting the reward. This result is useful for two reasons. The first one is that we can see that all the networks can beat random policy after a few episodes. The second is that we can determine the initial difficulty of the environment. This environment is an instance of reinforcement learning (RL) problems. This means that in order for the network to learn a well-performing policy, it needs to receive a reward to evaluate the utility of its actions. When a neural network is untrained, its first policy is usually very similar to a random one. Thus, we can conclude that this environment provides enough feedback in terms of rewards for the algorithm to learn in its default settings without any alterations.

The last baseline used for interpreting the results was the baseline of following the wall in the Maze. The following baseline achieved a reward of 62 and was used by a trained feed-forward network.

4.2 General comparison

The following section contains a comparison of performances for individual approaches specified in Section 4.1.2. To compare the performance, we had taken the average performance of each algorithm executed with three different seeds. The plot capturing the experiment is shown in Figure 4.2. A comparison of results is provided in Table 4.3. For plasticity, we only used the best performing variants for the comparison. A full overview of performance for different plasticity variants is given in Section 4.4.

In the experiment, the approaches: LSTM and RNN with modulated plasticity were able to obtain the best performance. Their resulting rewards were 199 and 200, respectively. This suggests that the following algorithms are suitable for this environment. However, they are not able to solve the environment optimally. We approximated that the maximal reward achievable in the Maze environment is 244 (Section 4.1.3). When comparing the two methods, we can observe that the approach utilizing RNN with plasticity exhibits faster learning speed in the earlier episodes. This could support the claim that plasticity is able to learn faster in the following environment. The evidence further supporting the claim can be gained by observing the learning speed of RNN in comparison to FF with Plasticity, where the approach utilizing plasticity learns significantly faster. The RNN with plasticity is a combination of these two approaches.

The next approach tested was a feed-forward network with plasticity. The following approach obtained a reward of 189, and during the first 40 000 episodes, it learned significantly faster than all the other approaches. This result shows that plasticity without any recurrency can cope well with this problem and even bring competitive results in a more complex environment than the Binary Sequences. In other environments, the benefit of fast learning speed could outweigh the lower final performance.

The vanilla RNN network obtained the worst results from all networks utilizing memory, with a reward of 118. When comparing the performance to the performance of LSTM, the lower reward could be expected, as LSTM had been shown to be better suited for meta-learning in [16]. The performance of vanilla RNN is also useful for measuring the benefits of adding plasticity (the recurrent neural network used with plasticity is vanilla RNN). From the results, we can see that combining these two approaches increases their overall performance. The last tested network was a feed-forward network. This network was added to show how networks without memory mechanisms perform in the Maze environment and to show how much adding plasticity to

this network improves the performance.

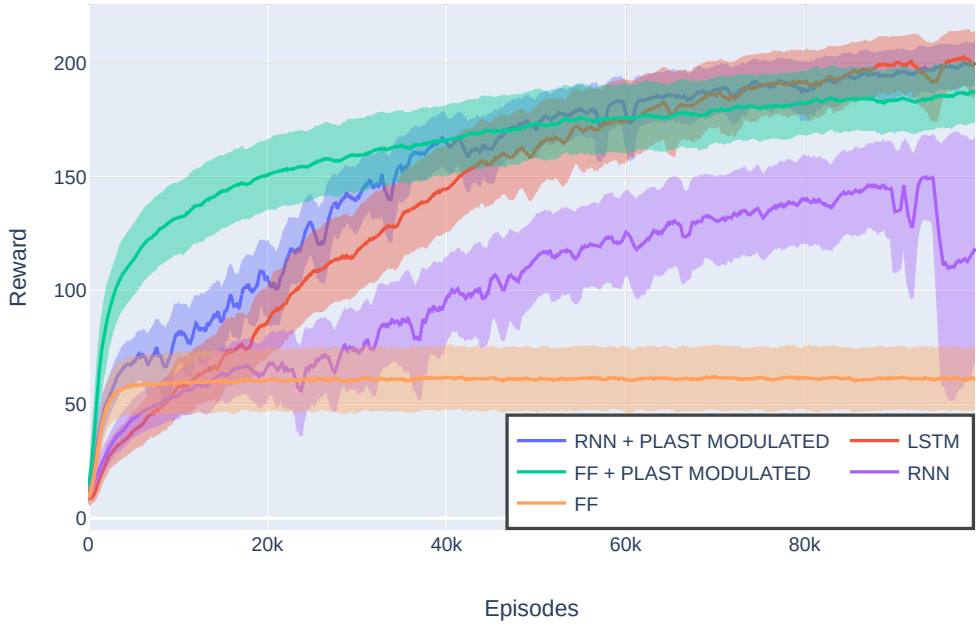


Figure 4.2: The Figure shows the reward of algorithms obtained in the Maze environment. The reward is calculated as a moving average from 1000 episodes from the mean of three independent runs. The mean reward is provided alongside the value of 1 standard deviation added to both sides.

Net type	Mean reward	Max reward	Min reward
RNN + PLAST MODULATED	200.2 ± 0.7	254	140
LSTM	199.1 ± 0.6	254	132
FF + PLAST MODULATED	187.1 ± 0.6	249	126
RNN	118.4 ± 1.8	226	45
FF	61.6 ± 0.6	150	13

Table 4.3: The Table contains information regarding the rewards obtained during the last 1000 episodes by each algorithm. The ± shows the 95 % confidence interval over the last 1000 episodes.

4.3 Memory systems contributions

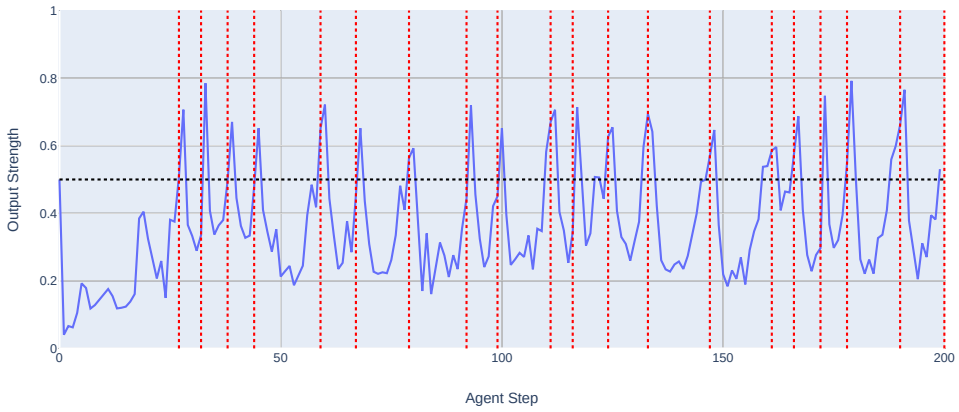
To get a better insight into each memory system's role when they are combined, we visualized their neural activity and detected to what stimuli each system

reacts. It is more complicated to measure the contribution of the memory system in Maze than in Binary Sequences. The problem is that the network outputs values for the agent's four possible actions, which are then transitioned into probabilities using the softmax algorithm. The softmax of values is problematic because applying softmax on the outputs produced by each memory system separately is not the same as when applying softmax on the combined outputs of the systems. We had solved this problem by using the values before softmax. This alteration should have no impact on the experiment because we only consider the action that is the agent most likely to take. This action corresponds to the action with the largest value before softmax. Although we only show the memory system contribution from one run of the episode for each architecture, we had checked that the presented findings hold true for other runs with the same configurations.

The neural activity is provided in Figure 4.3. From the figure, we can observe that the memory systems work together; there are only two steps where one system has almost 100% control over the decision. However, there are steps during episodes where one system is more important. Through the agent's analysis during the episode, we found out that the spikes in neural activity of plasticity (better visibility in the case of RNN + PLAST MODULATED) corresponds to the agent finding the reward. The steps when a reward is reached are displayed using a red dotted line. From the observations, it seems that the role of plasticity in the system increases before reaching the reward and decreases after finding it. On the other hand, the contribution of recurrent neural network or feed-forward neural network is higher when the agent moves towards the reward. One possible explanation could be that the plasticity is capable of remembering states before reaching the reward and reacts in response to them. This should be possible because the reward location remains the same during the episode. The next observation supporting this claim is that the plastic network activity is significantly lower before the first time the reward is found than after it. The resulting role of the RNN then would be to guide the agent through the maze until a state remembered using plastic memory is encountered.

When observing the results of the combination of FF and plasticity, we can notice that the plasticity is the more dominant memory in most episodes. On the other hand, when combining RNN with plasticity, plasticity is the less dominant memory in most episodes. Also, the relative contribution of plasticity is higher in the case of combination with FF. This difference can be attributed to the fact that the FF network does not have any memory, while in the case of RNN, both approaches are capable of utilizing memory.

(a) : FF + PLAST MODULATED



(b) : RNN + PLAST MODULATED

Figure 4.3: The figure represents the relative contribution of plasticity in comparison to the recurrent or the feed-forward layer. The contribution is measured by calculating the contribution of the individual systems to the agent's most valued action. The y-axis represents the relative contribution where 1 stands for 100% and 0 stand for 0 %. The black dotted line represents 50 %. When the plot is above the black line it means the plasticity is the dominant memory system. The red dotted line represents the episodes when the agent reached the goal.

4.4 Plasticity analysis

The plastic network that we use for testing the performance in Section 4.2 consists of multiple improvements to the plain plastic network. These improvements include a differentiable η parameter, full plastic connections and neuromodulation. In this section, we analyze their contribution to the performance by gradually adding them to the basic plastic neural network. The exact specifications for each improvement are provided in Section 3.4.1. We test the added value of improvements on a plastic network combined either with a recurrent neural network (RNN + PLAST) or with a standalone plastic network (FF + PLAST). The results for the standalone plastic network are provided in Figure 4.4 and Table 4.4. The results for the recurrent neural network combined with the plastic network are provided in Figure 4.5 and Table 4.5.

When we look at the results, we can observe that each improvement added to the plasticity increases the final performance. What is more interesting is that the learning speed of more complex algorithms (more trainable parameters) is faster than the simpler ones. Next important finding is that the FF network combined with plasticity has more stable training than the combination of RNN with a plastic neural network. The FF network with plasticity obtained better final performance for homogenous plasticity and differentiable eta. This finding suggests that the full plasticity has at least two roles in the network. The first one is that it allows the plastic layer to model more complex relationships. The other is that it promotes the combination of plasticity with other types of layers, by allowing each neuron to define its own combination of the layers. The layers used in the combination does not need to be recurrent. We can see that when plasticity is combined with a more powerful network (RNN compared to FF), the relative increase in performance is much higher. In the case of combination with RNN, the reward increased from 103 to 180, which is a relative increase of 77. While in the combination of FF, the reward increased from 138 to 169, which is a relative increase of 31. When observing the results of homogenous plasticity, we can see a decrease in performance when combined with the recurrent layer. Standalone RNN obtained a reward of 118.4 (Section 4.2), while when combined with homogenous plasticity, it obtained a reward of 103.1. These results suggest that full plasticity is required for combining the networks properly.

From the experiments, we found out that the neuromodulated full plasticity achieves the best performance. One of the reasons for the better performance of neuromodulated plasticity is the ability to explore the environment better (Section 4.5).

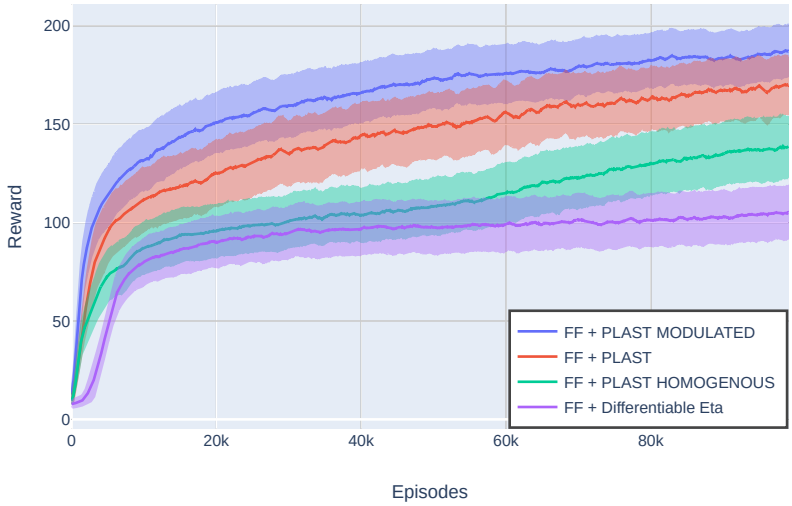


Figure 4.4: The figure provides a comparison of performance for different plasticity modifications in a standalone plastic network. The reward is calculated as a moving average from 1000 episodes from the mean of three independent runs. The mean reward is provided alongside the value of 1 standard deviation.

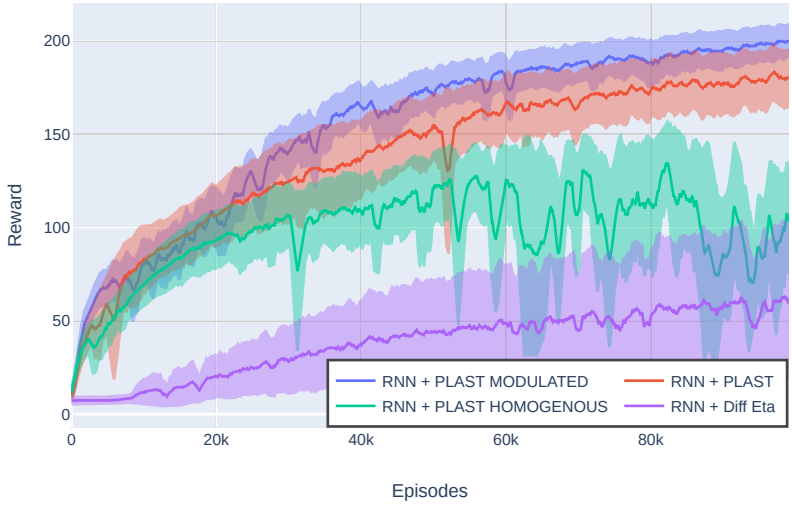


Figure 4.5: The figure provides a comparison of performance for different plasticity modifications in a recurrent + plastic network. The reward is calculated as a moving average from 1000 episodes from the mean of three independent runs. The mean reward is provided alongside the value of 1 standard deviation.

Net type	mean	max	min
FF + PLAST MODULATED	187.1 ± 0.6	249	126
FF + PLAST	169.8 ± 0.7	239	108
FF + PLAST HOMOGENOUS	138.7 ± 0.8	211	72
FF + Differentiable Eta	105.2 ± 0.7	182	46

Table 4.4: The table contains the mean, max, and min rewards obtained by different plasticity modifications from the last 1000 episodes. The ± shows the 95 % confidence interval over the last 1000 episodes.

Net type	mean	max	min
RNN + PLAST MODULATED	200.2 ± 0.7	254	140
RNN + PLAST	180.6 ± 0.7	243	103
RNN + PLAST HOMOGENOUS	103.1 ± 1.5	218	14
RNN + Differentiable Eta	59.3 ± 1.5	119	12

Table 4.5: The table contains the mean, max, and min rewards obtained by different plasticity modifications from the last 1000 episodes. The ± shows the 95 % confidence interval over the last 1000 episodes.

4.4.1 Visualizing plasticity

In order to better understand plasticity, we had visualized the final plasticity coefficients α in Figure 4.6. Our goal was to observe if there is any visible structure after displaying the values, as was the case in the Binary Sequences environment 3.4.2. In the case of the Maze environment, we had not been able to observe any visible structure. We believe that a more complex analysis of the plasticity coefficients could bring useful insight into plasticity's inner working. An example of such analysis would be, to sum up all the plasticity coefficients for one neuron and compare it to sums of other neurons.

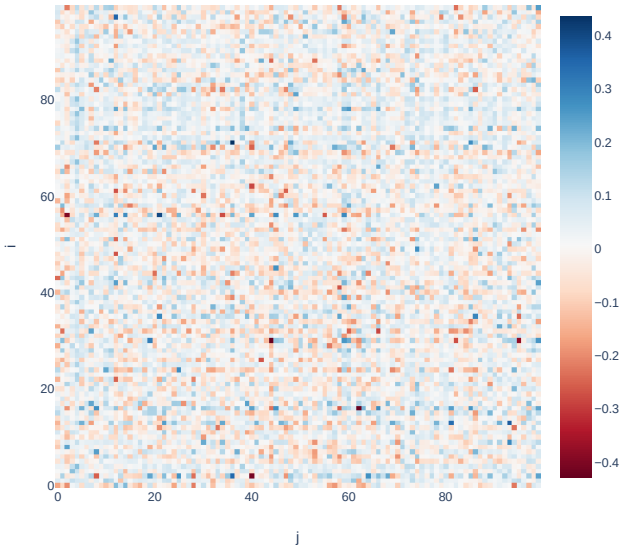


Figure 4.6: The figure shows the final plasticity coefficients α. There is no visible pattern in the coefficients.

4.5 Exploration capabilities

To better compare the individual approaches, we had devised a test where we did not put any reward into the maze. The goal of this test is to measure the exploring capabilities of each solution. In order for the algorithm to succeed, it needs to be able to perform state aliasing [26]. State aliasing is a problem that occurs when two or more different states share the same representation in models representation space. The algorithm must use memory to succeed in the problem of state aliasing. Although our maze consists of 65 cells that the agent can visit, there are only 14 observations the agent can get from the 3×3 view the agent receives. The four corners of the maze have unique observations. Then the agent can distinguish if he is at the tile next to the outer wall. In this case, the observation also is unique for each wall so the agent can better localize itself. These observations make for the next eight unique observations (4 walls and agent can be next to the inner wall in the maze or not). Notice that in this case, the agent cannot know without memory on which tile he is next to the outer wall. The last two unique observations are from the 33 maze cells, which creates the whole "inner maze." The results are shown in Table 4.6. From the results, we can observe that the approaches utilizing plasticity explore the maze better. Also, when comparing the results

of modulated and non-modulated plasticity, we can conclude that modulated plasticity improves the algorithm's exploration capabilities. We can further observe that there is a significant difference in the exploration capabilities of a vanilla RNN 78.8% and LSTM 96.6%.

Net type	Explored
RNN + PLAST MODULATED	$98.9 \pm 0.2\%$
FF + PLAST MODULATED	$98.5 \pm 0.2\%$
FF+PLAST	$97.0 \pm 0.4\%$
RNN+PLAST	$96.8 \pm 0.3\%$
LSTM	$96.6 \pm 0.4\%$
RNN	$78.8 \pm 1.7\%$
FF	$62.0 \pm 0.7\%$

Table 4.6: The table describe how well each algorithm performs in the exploration of the Maze. The ± shows the 95 % confidence interval over the 300 runs.

4.6 Adaptation capabilities

This experiment's goal is to test how well and fast the algorithm can incorporate knowledge of the reward position. The experiment is based on the premise that to get a high reward, the agent must use the shortest possible path to the reward. To calculate the shortest path, the agent must be aware of its current position and the reward position. In the experiment, we measure how often the agent reached the final reward in a specific range. Our finding are provided in Table 4.7 and Figure 4.7. The most crucial finding from this test is that the LSTM achieves a reward of 300-450 in the most percentage of runs compared to other methods. This means that the LSTM can better adjust its algorithm after finding the reward than the other approaches.

Net type	0	10-100	100-200	200-300	300-450
RNN + PLAST MOD	1.53	4.40	30.90	61.47	1.70
FF + PLAST MOD	1.90	11.57	32.33	52.67	1.53
LSTM	3.87	3.53	29.97	60.07	2.57
RNN	23.40	20.47	20.83	34.60	0.70
FF	49.73	25.73	17.13	7.23	0.17

Table 4.7: The table contains information regarding the percentage of runs the agent ended within a defined reward range. The data had been calculated from 3000 episodes for each algorithm, obtained from 3 independent runs. Plast mod stands for modulated plasticity.

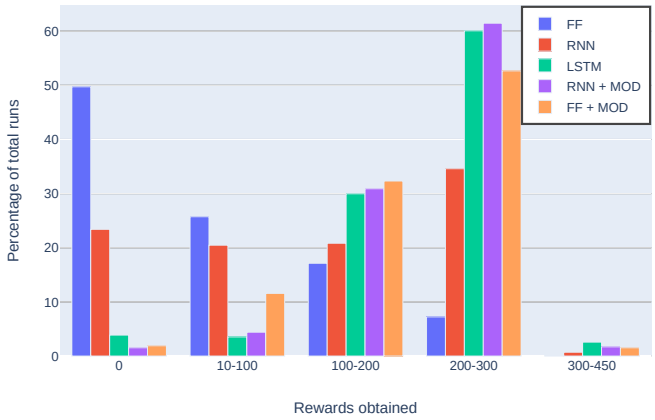


Figure 4.7: The figure provides a histogram of the percentage of runs the agent ended in a defined reward range. The data from Table 4.7 are used.

4.7 Summary

For the Maze problem, we first started with comparing the overall performance of each algorithm (Section 4.2). In the following experiment, both approaches utilizing memory (plasticity and recurrency) achieved similar performance. The best plastic neural network (RNN + PLAST MODULATED), had achieved a reward of 200.2. The best recurrent neural network (LSTM) had achieved a reward of 199.1. The experiment also showed that a feed forward network combined with modulated plasticity (FF + PLAST MODULATED), obtained a reward of 187.1. Furthermore, when comparing the learning speed of the approaches, we can see that the approaches utilizing plasticity learn faster compared to approaches using recurrency. To better understand the agents' performance, we had done a theoretical analysis of the environment and approximated that the optimal reward achievable is around 244 (Section 4.1.3).

To establish the differences between the algorithms, we had devised two new tests. The first test measures the exploration capabilities (Section 4.5), by placing the agent into the Maze with no reward and checking if the agent visits all tiles. From the following test, we had found that none of the approaches explores 100% of the Maze. Our Maze is challenging to explore because most states share the same representation (Section 4.5). The RNN + PLAST MODULATED explored 98.9% of the Maze, while LSTM explored 96.6% of the Maze. This result suggests that plasticity can explore this environment

more efficiently (State aliasing, Agents position). The second test conducted measured how well the agent learns within an episode (Section 4.6). More specifically, we looked at the distribution of the rewards obtained after each episode from 3000 runs. The LSTM had achieved reward in the range from 300 to 450, in 2.57% of the runs, while plasticity only in 1.70% of runs. The second experiment results suggest that the recurrency is able to better learn within the episode (utilize the knowledge regarding the reward position).

In our next experiment, our goal was to measure and determine the role of each memory type (plasticity, recurrency), when they are combined (Section 4.3). We measured the contribution of each memory by its impact on the network's output. More specifically, we measured the relative contribution to the agents' most probable action. In the experiment, we found out that the two systems work together, rather than each of them having a specific role. Work together means that the relative contribution of memory was withing a range of 30% - 70%, which means it played a significant role in the final output. We had found out that the contribution of plasticity starts to increase the closer the agent is to the reward (up to 70% of relative contribution) and decreases after finding it. In comparison, the importance of recurrent memory starts to gradually increase to 70% after finding the reward and then begins to decrease after the agent is close enough to the reward. One possible explanation for this finding is that the plasticity's role is to remember the states near the reward, and the role of the recurrent layer is to navigate the agent until a state remembered using plasticity is encountered.

The last experiment's goal was to measure the importance of different plasticity improvements (Section 4.4). From the results, it came out that each improvement to the plasticity increased the final performance and the learning speed. The increase in learning speed was not expected, because each improvement increases the total number of trainable parameters. Another result that came out of this experiment is that introducing full plasticity into the network not only increases the performance of the plastic layer but also improves the cooperation between the plastic layer and other layers.

Chapter 5

Conclusions and future work

The goal of the thesis was to get a better insight into the neural networks utilizing memory and compare plastic and recurrent neural networks. After reviewing the state of the art for using memory in neural networks, we became interested in the plastic neural networks. More specifically, in the recent work of differentiable plasticity [29] and its extension backpropamine [30]. In the mentioned works, the authors combine recurrent networks with Hebbian plastic connections and achieve competitive results in the meta-learning and meta-reinforcement learning domains. When examining the experiments provided in the paper, we wanted to learn more about how plasticity works, what contributes most to its performance, and how the recurrent network works together with plastic connections. To do so, we had devised our own experiments and obtained valuable insights into the plasticity framework, which are provided in the next paragraphs. In addition to getting better insight into the plasticity, we wanted to know how well the plasticity neural networks compared to other methods. We decided to test them against recurrent neural networks, more specifically vanilla RNN and LSTM. In the theoretical part, we introduced all the concepts required for understanding our work. More specifically vanilla recurrent neural network, long short term memory network, differentiable plasticity framework, backpropamine (modulated plasticity), reinforcement-learning, meta-learning and meta-reinforcement learning. For plasticity (Hebbian learning), we provided a more detailed overview of the recent work, including domains it is used in.

In the Binary Sequences environment, we first compared the plastic networks, defined in [29], against vanilla RNN and LSTM. The plastic networks achieved success rate of 97.3%, while the vanilla RNN and LSTM achieved a success rates of 77.4% and 76.9% respectively, in recalling the degraded

pattern correctly. We established that the baseline for this environment could achieve a score of 75%. After the high success rate of plasticity, we wanted to investigate more how it works. We started by separating its recurrent layer from the plastic layer and measuring each layer's contribution to the final decision. We found out that the final output is decided by the plasticity only. Based on this finding we implemented a plastic network combined with feed forward network (97.3%) and plastic network without any other layer (99.3%). This result suggest that only plastic neural networks are required for solving Binary Sequences environment. Providing them with recurrent neural layer, only decreases the performance. In our next experiment, we investigated which parts and parameters of plasticity contributes most to the result. We found out that the plasticity learning rate η is the most important plasticity parameter for the Binary Sequences environment. The performance of plasticity with differentiable η and static α plasticity achieves a success rate of (99.23%). Adding full plasticity (differentiable α for each connection) improves the final score only slighty to (99.34%). The neuromodulated plasticity (99.2%) even sligthly decreased performance. From the results of this experiment, we concluded that plasticity learning rate η is the most important parameter. The plasiticity coefficient α which is the main addition of the work Differentiable Plasticity, where the plastic neural networks been introduced is not well utilized in the Binary Sequences environment. For future work we, plan to show how the total number of shown patterns affects the algorithm's performance. We believe it will help establish the limits of storing capabilities for each network architecture, especially the plastic neural networks.

In the Maze environment, we made contribution by showing that the LSTM recurrent neural network is capable of solving the environment comparably to the best plastic neural network. The two mentioned approaches obtained rewards of 199.1 and 200.2 respectively after 100000 episodes of training. To get a better understanding of the rewards obtained by the algorithms, we approximated the optimal solution for the Maze environment to be around 244. The results obtained from the comparison of the two methods suggest that the plastic neural networks exhibit faster learning speed than the recurrent neural networks. The fastest learning architecture by a huge margin in the first 40 000 episodes was the combination of feed forward network with plasticity. This network achieved a final reward of 187.1, which is only slightly worse than the best performing architectures. The combination of feed forward network with plasticity provides an interesting tradeoff between learning speed and performance, which could be used in other environments. From the experiment measuring the contribution of individual layers, when combined, we found out that the recurrent and plastic layer work together, rather than each of them having a specific role. When observing the individual layer contributions we noticed that the plastic layer contribution rises in the steps when the agent is closer to the reward and afterward decreases, while the influence of the recurrent layer rises after finding the reward and teleportation

to the random location. The following observation suggests that the main role of the plasticity is to remember how to navigate the agent in the states near the reward, while the main role of the recurrent neural network is to navigate the agent until a state remembered using plasticity is encountered. Our next goal was to devise new tests that would help us better understand the advantages and disadvantages of different memory types. The first metric devised measured the exploration capabilities of the agent. In this, test the agent was put in a Maze with no reward, and we calculated how many percent of the Maze the agent explored. The plastic neural networks explored 98.9% of the Maze, while LSTM explored 96.6%. With the second metric we tested how well the agent can learn within an episode. The experiment results showed that the LSTM can obtain the reward in the range of 300-450 in 2.57% of runs, while the plasticity only in 1.70% of runs. This suggests that the LSTM is capable of better adapting to the algorithm. In our final experiment, we tested which parts and parameters of plasticity contribute most to its performance. In the Maze experiment, both differentiable plasticity and neuromodulaton increased the final performance significantly. Furthermore we found out that introducing full plasticity into plastic network that is combined with recurrency increased the performance by 80, while in the case of combination with feed forward network the performance only increased by 30. This finding could suggests that full plasticity improves the cooperation of the recurrent and plastic layer.

From our experiments, we obtained significant knowledge regarding the properties of the algorithms in the tested environments. We will leverage the obtained knowledge to provide insights into applying the algorithms in other environments. When using plastic neural networks, it is important to assess whether the network used in combination with the plasticity provides the network with additional utility. This finding is based on the results from the Binary Sequences environment, where we found out that by replacing the recurrent layer with the feed forward layer, we increase the performance of the network. When combining networks, use full plasticity, which allows the network to choose for each neuron the contribution of different layers (Maze Section 4.4). Lastly, when considering which algorithm to use for a given environment, think about the plasticity also as an extension to another neural network architecture. This brings us to the question of what is gained by introducing plasticity to the network. From our experiments, we recommend using plasticity when working with binary patterns. This recommendation is based on the finding from Binary Sequences, where plasticity alone was capable of working with binary patterns effectively after minimal training. For reinforcement learning, our results suggest that plasticity can help with the state aliasing problem, this recommendation is based on the results provided in Section 4.5.

We believe that our experiments provided the reader with useful insights

into the workings of plasticity and showcased its properties. In our future work, we plan to extend our research by including the neural networks with external memory into our experiments

Appendix A

Bibliography

[1] Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. *ArXiv abs/1610.06258* (2016).

[2] Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. A theory of learning from different domains. *Machine Learning 79* (2010), 151–175.

[3] Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks 5*, 2 (1994), 157–166.

[4] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR abs/1406.1078* (2014).

[5] Collins, J., Sohl-Dickstein, J., and Sussillo, D. Capacity and trainability in recurrent neural networks, 2016.

[6] Danihelka, I., Wayne, G., Uria, B., Kalchbrenner, N., and Graves, A. Associative long short-term memory. *CoRR abs/1602.03032* (2016).

[7] Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. *CoRR abs/1611.02779* (2016).

[8] Elman, J. L. Finding structure in time. *Cognitive Science 14*, 2 (1990), 179–211.

[9] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR abs/1703.03400* (2017).

[10] Grave, E., Joulin, A., and Usunier, N. Improving neural language models with a continuous cache. *ArXiv abs/1612.04426* (2017).

[11] Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *ArXiv abs/1410.5401* (2014).

[12] Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwinska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. Hybrid computing using a neural network with dynamic external memory. *Nature 538* (2016), 471–476.

[13] Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., and Schmidhuber, J. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems 28*, 10 (Oct 2017), 2222–2232.

[14] Hebb, D. O. *The organization of behavior: a neuropsychological theory.* J. Wiley; Chapman & Hall, 1949.

[15] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9* (12 1997), 1735–80.

[16] Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. 87–94.

[17] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences 79*, 8 (1982), 2554–2558.

[18] Hughes, J. R. Post-tetanic potentiation. *Physiological Reviews 38*, 1 (1958), 91–113. PMID: 13505117.

[19] Karpathy, A., Johnson, J., and Li, F. Visualizing and understanding recurrent networks. *CoRR abs/1506.02078* (2015).

[20] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2015).

[21] Koch, G. R. Siamese neural networks for one-shot image recognition.

[22] Lake, B., Salakhutdinov, R., and Tenenbaum, J. Human-level concept learning through probabilistic program induction. *Science 350* (12 2015), 1332–1338.

[23] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *CoRR abs/1604.00289* (2016).

[24] Li, Y. Deep reinforcement learning: An overview. *CoRR abs/1701.07274* (2017).

[25] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics 19*, 2 (1993), 313–330.

[26] Mccallum, A. K., and Ballard, D. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, 1996. AAI9618237.

[27] Melchior, J., and Wiskott, L. Hebbian-descent. *CoRR abs/1905.10585* (2019).

[28] Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *ArXiv abs/1609.07843* (2017).

[29] Miconi, T., Clune, J., and Stanley, K. O. Differentiable plasticity: training plastic neural networks with backpropagation. In *ICML* (2018).

[30] Miconi, T., Rawal, A., Clune, J., and Stanley, K. O. Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity. *ArXiv abs/2002.10585* (2019).

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

[32] Mnih, V., Heess, N., Graves, A., and Kavukcuoglu, K. Recurrent models of visual attention. *CoRR abs/1406.6247* (2014).

[33] Olah, C. Understanding lstm networks, 2015.

[34] Pascanu, R., Mikolov, T., and Bengio, Y. Understanding the exploding gradient problem. *CoRR abs/1211.5063* (2012).

[35] Rae, J. W., Dyer, C., Dayan, P., and Lillicrap, T. P. Fast parametric learning with activation memorization. *CoRR abs/1803.10049* (2018).

[36] Robinson, A. J., and Fallside, F. The utility driven dynamic error propagation network. Tech. Rep. CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK, 1987.

[37] Santoro, A., Bartunov, S., Botvinick, M. M., Wierstra, D., and Lillicrap, T. P. One-shot learning with memory-augmented neural networks. *ArXiv abs/1605.06065* (2016).

[38] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization. *CoRR abs/1502.05477* (2015).

[39] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature 529* (01 2016), 484–489.

[40] Smyl, S. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting 36*, 1 (2020), 75–85.

[41] Soltoggio, A., Stanley, K. O., and Risi, S. Born to learn: the inspiration, progress, and future of evolved plastic artificial neural networks. *CoRR abs/1703.10371* (2017).

[42] Sutton, R., and Barto. *Reinforcement learning: An introduction. (Second Edition)*. Cambridge: MIT press, 2018.

[43] Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. *CoRR abs/1611.05763* (2016).

[44] Werbos, P. J. Generalization of backpropagation with application to a recurrent gas market model.

[45] Yagishita, S., Hayashi-Takagi, A., Ellis-Davies, G. C., Urakubo, H., Ishii, S., and Kasai, H. A critical time window for dopamine actions on the structural plasticity of dendritic spines. *Science 345*, 6204 (2014), 1616–1620.

[46] Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *ArXiv abs/1409.2329* (2014).

Appendix B

User guide

The project is implemented using Python 3.7. For training the neural networks, we use the PyTorch library. The resulting data from training and testing is visualized using Plotly. The visualizations are shown using Jupyter Notebooks.

The code for the project can be divided into three groups based on its functionality. The first group contains the scripts used for executing training, testing, and obtaining visualization data, located in the rnn directory. These scripts can be launched either locally or on a dedicated server. We used the services of MetaCentrum to calculate the computationally demanding tasks. The second group is the actual code implementing the neural networks and the environment, located in the code directory. The last group contains the visualization functions and functions for manipulating the data from experiments, located in the rnn_visualise directory. Results from training and testing are located in the rnn_results folder. We had provided one trained neural network for each environment. If needed, we can provide all our trained networks and data files. The rnn_vis_data is used for storing visualization data, such as the data for displaying the agent during an episode of the Maze environment. For creating the python environment, the list of requirements is provided in requirements.txt.

We had created a basic guide containing basic commands to run the scripts and a general overview of the code provided. If you have any questions regarding the project, contact us and we will gladly answer them.

The project contains some modifications of the code provided with the differentiable plasticity paper (https://github.com/uber-research/differentiable-

plasticity) and the backpropamine paper (https://github.com/uber-research/backpropamine).

The structure of attached files is provided below.

```
folder
├── rnn
│   └── code
├── rnn_results
├── rnn_vis_data
├── rnn_visualise
├── thesis.pdf
├── requirements.txt
├── user_guide.pdf
└── executed_jupyter_notebook_example.html
```