Bachelor Degree Project

Hierarchical Temporal Memory Software Agent

- in the light of general artificial intelligence criteria

Author: Jakob Heyder
Supervisor: Johan Hagelbäck
Semester: VT/HT 2017
Subject: Computer Science

Abstract

Artificial general intelligence is not well defined, but attempts such as the recent list of "Ingredients for building machines that think and learn like humans" are a starting point for building a system considered as such [1]. Numenta is attempting to lead the new era of machine intelligence with their research to re-engineer principles of the neocortex. It is to be explored how the ingredients are in line with the design principles of their algorithms. Inspired by Deep Minds commentary about an autonomy-ingredient, this project created a combination of Numentas Hierarchical Temporal Memory theory and Temporal Difference learning to solve simple tasks defined in a browser environment. An open source software based on Numentas intelligent computing platform NUPIC and Open AIs framework Universe was developed to allow further research of HTM based agents on customized browser tasks. The analysis and evaluation of the results show that the agent is capable of learning simple tasks and there is potential for generalization inherent to sparse representations. However, they also reveal the infancy of the algorithms, not capable of learning dynamic complex problems, and that much future research work is needed to explore if they can create scalable solutions towards a more general intelligent system.

Keywords: general artificial intelligence, machine learning, hierarchical temporal memory, autonomous agent, reinforcement learning, temporal difference learning, human-like thinking, and learning

Contents

List of Figures

List of Figures

Abbreviations

TD – Temporal Difference

HTM - Hierarchical Temporal Memory

SDR – Sparse Distributed Representation

RL – Reinforcement Learning

MC – Monte Carlo

DP – Dynamic Programming

ML – Machine Learning

TM – Temporal Memory

SP – Spatial Pooler

TP – Temporal Pooler

L # – Layer #

b/w - black and white

1 Introduction

Artificial intelligence research exploded in the recent years, and even nations have made it a priority to lead in this new technology paradigm ([2],[3],[4]). Successes with neural networks and mathematical optimization methods open up new possibilities by learning input-output mappings. Computers can learn patterns to generate speech or label images([5], [6]), by merely being trained on enough data. However, besides the massive required training data, they have other downsides such as problems to transfer learning or to process more complex tasks that require abstract generalizations. Thus there are many approaches to develop machines that think and learn more like humans. In a recent paper published by Lake and his colleagues, they examine the current state of technologies and develop a list of ingredients needed for a more general intelligent system [1]. The paper was widely recognized and commented in the community and therefore a valuable context for this study.

In the development of new machine learning algorithms, there is much inspiration from neuroscience and brain models. They differ in the level of detail in biological accuracy. Experimentalists try to find equations and algorithms that predict the measured outcome of the experiments. Other approaches offer a higher level of abstraction, such as spiking neural networks, which include spike time dependence in their model. On the contrary side, there are mathematical approaches, which do not attempt to model biology and argue they would include too many details to be scalable.

This study chooses to use an algorithm which is based at the middle on this scale of abstraction. It attempts to be biologically plausible but only adapts the core computational principles and abstracts the details that are thought not to be necessary from the current point of research literature. The algorithm is interpreted in the context of the mentioned list of ingredients. For this purpose, it is advanced to integrate a reward mechanism that an agent can utilize to train on a task. The task is designed to be simple and challenges the agents' cognitive abilities. The results are analyzed and evaluated to form the basis for a discussion on how these ingredients can be applied in such an architecture.

1.1 Problem formulation

Cognitive scientist Brendon M. Lake et al. have published a paper on "ingredients for building machines that learn and think like humans", which

was widely commented by the AI-community and is used as context reference in this study [1]. Besides other components, the authors emphasize composition, causality, and learning-to-learn. On the other hand - Numenta, a silicon valley based company - tries to understand the neocortex and use this knowledge to build artificial intelligent systems.

Numentas hierarchical temporary memory theory and its implementation platform NUPIC have been successfully used for pattern recognition and anomaly detection of streaming data in the past([7],[8],[9]). Additionally new discoveries on the theoretical side attempt to integrate intrinsic sensorimotor sequences and pair extrinsic sequences with a location signal to infer features and objects([10],[11]).

In this study, I attempt to create a neural systems architecture for an agent based on a combination of existing NUPIC HTM technology with reinforcement learning. The agent is built with one-shot character recognition in mind and will attempt simplified challenges to test characteristics of the system in the context of Lake et al. proposed experiment. How far are Lake et al. ingredients of intelligence considered in HTM? How does the reinforcement learning agent based on HTM technology learn in static and dynamic environments?

As mentioned in the commentary from Deepmind's team on Lake's paper autonomy was one of the key missing aspects. The experiment will make use of the published omniglot dataset and attempt to model a similar setup as it was for the human participants [12]. However, as character recognition is not a standard problem in reinforcement learning and the challenge is not designed to be solved by an autonomous agent, it is different from the original setup. The evaluation is primarily concerned with the limitations and possibilities of the architecture and refers to the criteria of transfer-learning and composition in a more general perspective. Clear limitations are made in the scope of the thesis.

1.2 Motivation

Artificial general intelligence is one of the most thought of and interesting problems of our time. With recent successes in deep learning, we probably found a representation of patterns generated by large sets of data, but it is missing characteristics for more general learning. While the research diverges into many different subfields and approaches, there are attempts from the community to define characteristics that constitute more general intelligence [13]. The list from Lake et al. is not meant to be complete

but together with the extensive commentary from the community forms a good basis for criteria of such algorithms. They mostly investigate how the deep learning field is already incorporating some of it, thus it is of great interest to also evaluate Numenta's alternative approach. The comparison is accomplished by designing a similar experiment and using the same data set as in their original study. According to their paper, it requires much of these principles for a successful result on it and a more human-like thinking and learning.

Numentas algorithms are based on fundamental research and science rather than aiming for scalable industrial solutions. Thus they are still very much in their infancy, but possibly this study can contribute to algorithms that built the foundation for the next generation of artificial general intelligence.

Additionally, this study attempts to extend the unsupervised learning from the HTM system that outputs predictions and detects anomalies in a data-stream with a reward system to utilize the predictions. This would largely extend the functionality of HTM and is closely related to current research in sensorimotor-theory integration. The proposed framework is not thought to be complete or finished, rather it can be the first stepping stone for a general theory combining HTM and RL. For this reason, the implementation is entirely based on an extended version of NUPIC the open source platform of Numenta and all developed tools and environments are shared. This is of great interested for the scientific community to enable future research, which can built on top of the results.

1.3 Objectives

The main goal of this work is to explore the possibilities of current HTM theory combined with temporal difference learning, to propose an architecture for an HTM based autonomous agent. Additionally, all experiments and developed tools should be reusable to make it easy for possible future work to build on top.

- In-depth study of HTM as an alternative unsupervised learning algorithm.
- Exploration of an architecture combining HTM with TD for an autonomous agent capable of character recognition.
- Package all developed software and tools platform independent for open access and future research.

- Interpretation of results in the context of the Lake's mentioned criteria for human-like learning and thinking [14].

As HTM is still in a very early phase and research is done at a fundamental level rather than trying to archive state of the art performance on specific tasks, it is to be expected that the experimental results will not be astonishing. Additionally as mentioned in the Related Works Section 1.5, principal components are still missing such as temporal pooling, which raises problems in areas such as delayed reinforcement problems or generalization of concepts. However, especially with respect to the current research in the sensorimotor part of the theory about the integration of an allocentric location signal into mini-columns object representations, it is interesting to explore how a general RL integration could be.

1.4 Scope/Limitation

Due to the extensive size of the field and strong time limitations of 5 months part-time work, this study is strongly limited to a specific niche problem and setup. The scope is to present an autonomous agent architecture that uses reinforcement learning to utilize the predictions from HTM and learn simple tasks. It is limited to the model-free reinforcement learning approach, specifically Temporal Difference Learning Lambda with a Mechanistic (Backwards) View and Replacing Traces. This specific choice is taken to align with the design principle of HTM to use algorithms that are biologically plausible as proposed by Kaan. However, any proof of such and biological evaluation of the architecture is out of scope for this work. Furthermore, it orients on the current research of integrating sensorimotor theory into HTM, but as it is unpublished and still developing it does not attempt coherence and inclusion of this features. The study focuses on HTM as a real-time unsupervised learning algorithm and does not compare it broadly with other existing algorithms in that category. Furthermore, since the list of ingredients is extensive and it is unclear how a system can incorporate them all, they are only vague used to describe how they are possibly incorporated in HTM theories and the proposed agent. Furthermore, the experiments are designed to test the properties in a much simpler manner than the original omniglot character challenge proposed by the Lake et. al.

1.5 Related work

In the best of my knowledge, there are only three works to this point concerning a combination of HTM and reinforcement learning. In chronological order, the first is a MSc thesis from Marek Otaḥal in 2013 [15]. It investigates the concepts of Cortical Learning Algorithms and its applications for agents to produce behavior. Secondly, a BSc Project from Antonio Sa̧nchez Go̧mez in 2016, exploring a general combination of HTM with RL on a small set of problems [16]. The third is a comprehensive work from Sungur Ali Kaan in 2017, utilizing HTM for an autonomous agent in partially observable video game environments [17]. The previous studies, architecture designs, results and discussions are taken as inspiration and guided the way to the agent design for this research. Especially Ali Kaan's proven design for an autonomous agent was re-implemented in NUPIC and used as a basis for further exploration. Additionally, there is a great open source community around HTM and in the forum are some in-depth discussions and conceptualizations of a combination with RL and integration for autonomous agents. Lastly, the current research from Numenta on sensorimotor theory is closely related to the generation and execution of actions. Even though most of it is not in a final stage yet and unpublished, the open research manner of Numenta allows early participation which led to inspiration in this study.

1.6 Target group

The targeted audience of the study is the computer science community, preferably with a background in machine learning and knowledge of concepts such as neural networks, reinforcement learning or unsupervised learning algorithms. Further, as neurobiology inspires the presented algorithm, it can be helpful to have a basic understanding of neuroscience such as the process of nerve cell communication over synapses.

1.7 Outline

The next section of the report provides the background knowledge for Hierarchical Temporal Memory and Temporal Difference Learning, which are the Machine Learning techniques utilized in this study. Section 3 is concerned with the method used in this study. It describes in detail the experimental setup for the problem and its relation to the character challenge by Lake et al., how results were conducted, reliability and validity of the results and any ethical concerns. In section 4 the architecture of the agent is

described in detail and makes use of temporal difference learning and hierarchical temporal memory described in the background section. It contains a high-level overview and a more detailed description of each layers functionality and implementation. Further implementation details can be found in the appendix and in the published open source code. Section 5 is showing all measured results and describes them objectively, whereas section 6 is analyzing the data and interpreting them in the context of the task. Latter also includes conclusions for the experiments and an opinion to the results. In the discussion, the analysis is evaluated in the broader scope of the study and work in the field. Here especially the novel combination of reinforcement learning and hierarchical temporal memory, with its drawbacks and promising parts are discussed. In the final part, a conclusion of the study is made, summarizing the report and findings and reflecting on the progress of the project.

2 Background

Machine learning is a very broad field and evolved from the study of pattern recognition and computational learning theory in artificial intelligence [18]. It is concerned with the study and design of algorithms that can learn from and make predictions on data, without being explicitly programmed. It uses statistical techniques to infer structure from patterns and interpret the data. In the subbranch of supervised learning, the computer can learn to predict the output from a given input by building a model on a set of training data that infers a structure on the data. As shown in Figure 2.1 this changes the task for the programmer fundamentally as they do not have to program the computer explicitly anymore and instead let the computer build a model of the input and output relation of the data.

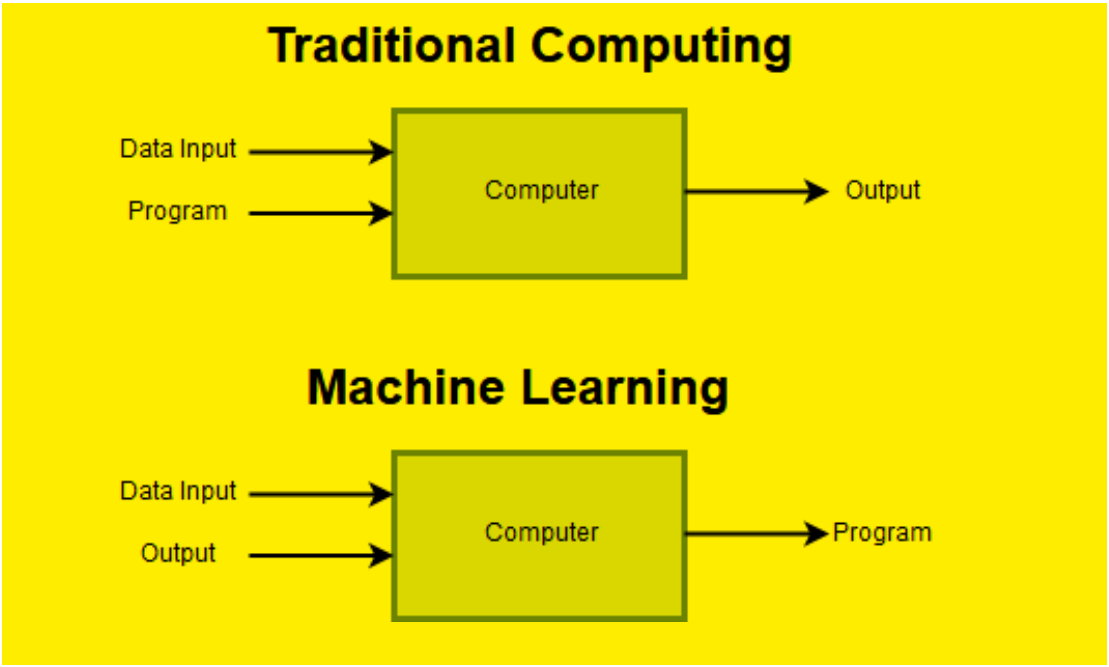

Figure 2.1: A simplified overview of the concept of machine learning in comparison to traditional computer programming.

This study is concerned with unsupervised learning, an area of machine learning that focuses more on exploratory data analysis and is used to find meaningful anomalies in data. Since examples are unlabeled, there is no direct evaluation of the accuracy of the model that is output by the algorithm. However, by a combination of unsupervised learning and reinforcement learning, intelligent software agents can be build that can be evaluated on a task using a reward function.

Reinforcement learning is an area of machine learning inspired by behaviorist psychology. It is concerned how software agents can choose an action in an environment and maximize an accumulated reward. This study

uses a model-free approach, where the agent is learning online from trial-and-error experience the selection of an action in contrast to model-based approaches where the agent exploits a previously learned model for a task [19].

Artificial intelligence and machine learning have a long history of being inspired by neuroscience - the study of the human brain [20]. There are many different approaches for general artificial intelligence systems, as Figure 2.2 from Perez shows [13]. However, with recent successes in deep learning, a machine learning method which was originally inspired by neurobiology, thought leaders in the community agree that biological inspiration can lead to faster development of such algorithms.

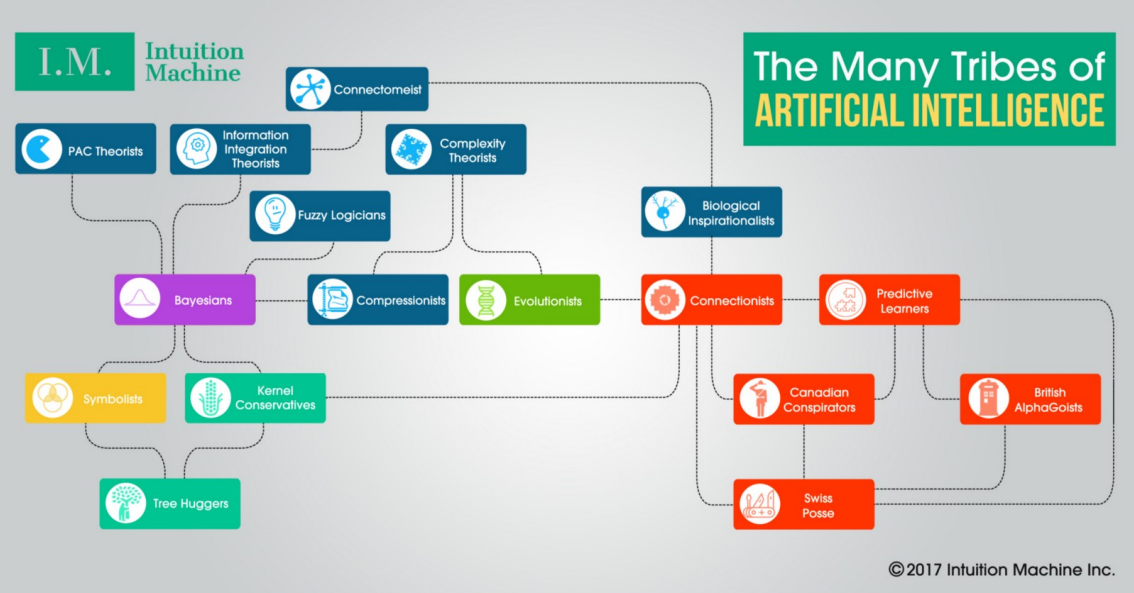

Figure 2.2: Different approaches in the artificial intelligence community to build more general intelligent systems [13].

Numenta is a company founded in 2005 in Redwood City, CA with the vision to become a leader in a new era of machine intelligence. Their belief is that the brain is the best example of an intelligent system and thus researching, understanding and re-engineering common principles of it is the most promising way to more general intelligence. Their theory focuses on the neocortex, a part of the mammalian brain that is involved in higher brain functions such as sensory perception, cognition, generation of motor commands, spatial reasoning, and language. As shown in Figure 2.3 it makes up the most substantial part of the human brain and is divided into six layers in biology. The division of layers is taken on in the terminology of the study. Numenta developed a biologically constrained unsupervised learning algorithm based on a theory which they termed Hierarchical Tem-

poral Memory. It is utilized in this study to develop a software agent.

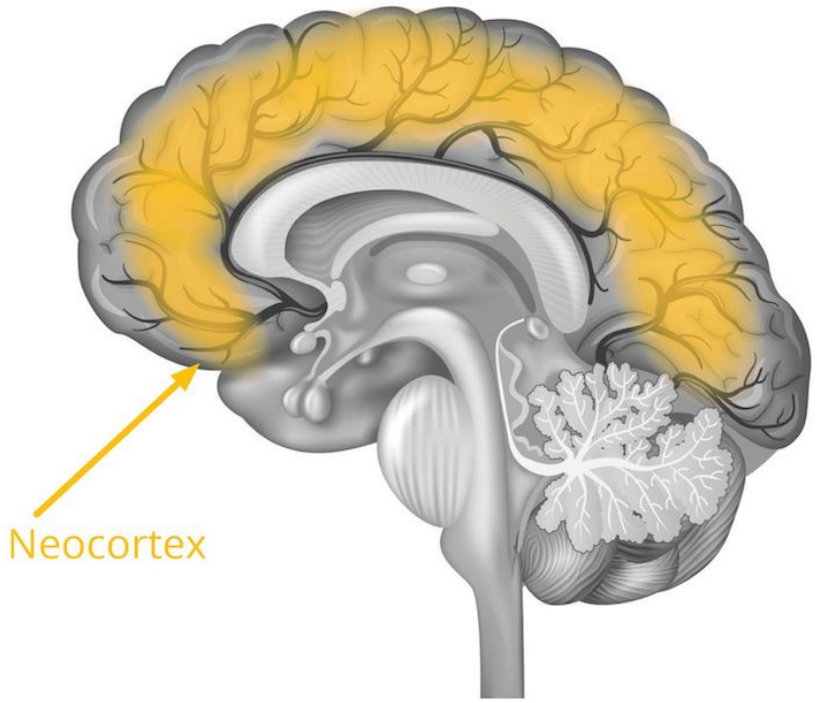

Figure 2.3: Picture of the part of the human brain that makes up the neocortex. [21]

Leaning on recent successes of the combination of deep learning and reinforcement learning - deep reinforcement learning - to develop software agents that were able to outperform humans in games such as GO or Chess [22], this study attempts an architecture to combine the Hierarchical Temporal Memory theory of Numenta with reinforcement learning. The developed design is based on Sal Kaan's master thesis that is one of the first attempts for an architecture of an autonomous agent with HTM [17].

The main reason this study is based on Numenta's theories is their foundation in neuroscientific research and the intersection to engineering, showing a promising future path for artificial intelligence research. However, this results in an alternative, partially complex model, which is different from common machine learning theories and neuroscience terminology. It requires a thorough introduction of the necessary background in HTM and temporal difference learning in the following section. Introducing all concepts in detail is out of the scope of this thesis and certain background knowledge is required. More details on the target audience are described in Section 1.6.

2.1 Hierarchical Temporal Memory

Hierarchical Temporal Memory is a continuous, online and unsupervised learning algorithm which is biologically constrained. It has local, Hebbian-like learning rules and models with the HTM-Neuron, a pyramidal neuron as the core component. A pyramidal neuron model is depicted in Figure 2.4 in comparison to a classical neuron model used in artificial neural networks, the Perceptron.

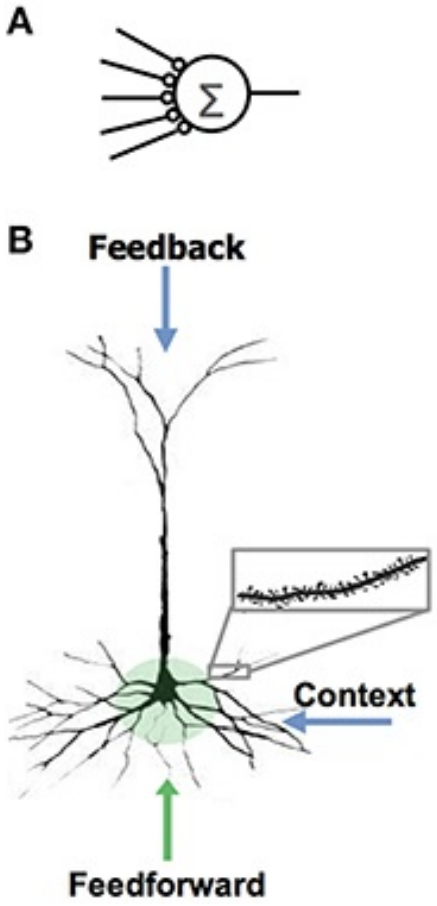

Figure 2.4: A) A classical Perceptron commonly model used in deep learning and neural networks. B) A pyramidal neuron model. It has proximal dendrites that receive feed forward input from lower layers or sensory. Distal segments of synapses that receive context information and apical feedback connections to higher layers.

The connections of the neuron are encoded as a set of sparse distributed representations (SDRs) that are essentially binary vectors and build the basic computational unit for the system. Hebbian learning is a theory in neuroscience that explains the adaptation of neurons in the brain during learning. It describes the underlying mechanism of synaptic plasticity, which can be interpreted in the context of HTM such that a synapse strengthens if it was correctly active and weakens inactive ones otherwise [23].

Figure 2.5 shows a standard model used in biology dividing an area of

the neocortex into a representative vertical column. A region of the neocortex consists of many cortical columns. Based on this cortex structure an HTM network consists of possibly multiple cortical columns, that each contains cortical layers labeled L6-L1. The theory is based on the assumption that the different cortex areas underlie a universal computational principle. Each layer consists of mini-columns, which represent neurons with the same perceptual field. In its currently published form, the main components of the algorithm include the HTM-Neuron, HTM-Spatial-Pooler, and HTM-Temporal-Memory. They are explained in detail in the following paragraphs.

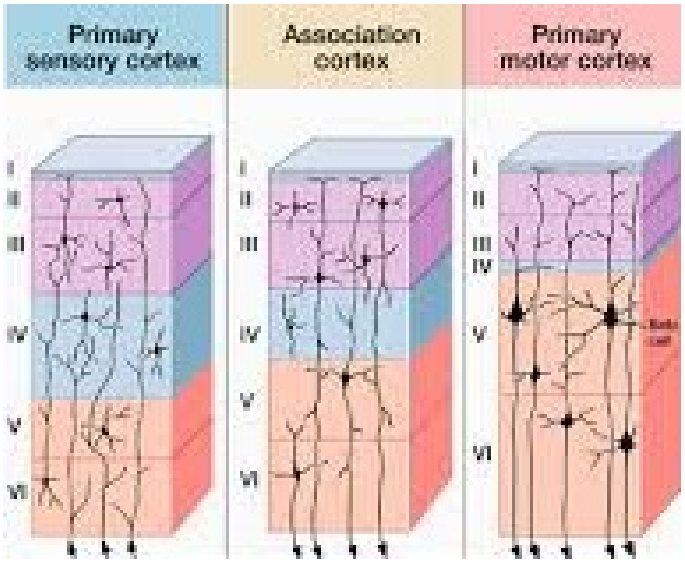

Figure 2.5: A standard model from biology which divides each cortical column of the neocortex into 6 layers.

HTM Neuron The HTM Neuron is the basic building block for the system. It is named after the hierarchical temporal memory theory to distinguish from biological and Perceptron models, and first introduced in Numenta's white paper [24]. The neuron is modeled from the biological pyramidal neuron which is taken on in most of the used terminology. In general, it can be thought of as a coincidence detector using its connections to determine its state. It can be in three different states: active, inactive or predictive. The Neuron, as shown in Figure 2.6, has three types of input connections:

- **Proximal dendrite**: Simple feedforward input from lower layers or senses. It can change the neuron to become **active**.
- **Distal dendrite**: Through the lateral connections the neuron can receive context information. The dendrite is modeled as a set of seg-

ments. Each segment can be thought of as a context and contains connections to other neurons. The segments are combined with a logical OR-Connection, if enough connected neurons in one segment become active, it will propagate the signal over the distal dendrite and bring the neuron in a **predictive state**.

- **Apical dendrite**: The apical connections work identically to the distal ones. The only difference is that they are feedback from higher layers or different cortical areas to be integrated into the interpretation of the current input. They can also bring the neuron into a **predictive state**.

The inputs on all connections are sparse distributed representations (SDRs). These binary vectors represent connections to a set of neurons that are either inactive(0) or active(1). At a specific point in time just 2 percent are active, so they remain sparse and distributed. Each active bit is thought to represent some meaningful semantic feature. Similar features are closer together, while dissimilar features are wider distributed. This clustering makes it extremely robust to errors. It is the task of an encoder to convert data of any format into SDRs with the desired properties for the network to use. The sparsity of the representations leads to several other very useful mathematical properties ([25], [26]):

- Capacity of SDRs and the probability of mismatches
- Robustness of SDRs and the probability of error with noise
- Reliable classification of a list of SDR vectors
- Unions of SDRs
- Robustness of unions in the presence of noise

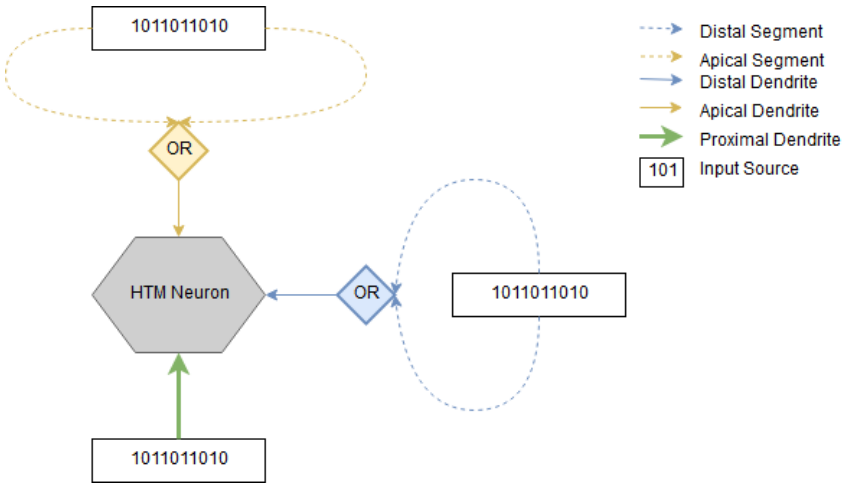

Figure 2.6: Overview of the HTM-Neuron. It has proximal dendrites that receive feed forward input from lower layers or sensory. Distal segments of synapses that receive context information from neighbouring mini-columns and apical feedback connections to higher layers. The input on all levels are SDRs.

HTM Spatial Pooler (SP) In a typical HTM-System as depicted in Figure 2.7, the spatial pooler takes input from an encoder and outputs to the temporal memory [27]. The main functional goal is to convert the given binary vectors to a semantically identical SDR with fixed sparsity. It consists of a set of mini-columns, typically about 2048. A mini-column consists of a set of HTM-Neurons, typically about 32 that share the same perceptual field. The perceptual field describes the input bits a mini-column can connect to. After initialization, the algorithm of the SP repeatedly iterates through three phases for each new input: Overlap calculation with Boosting, Inhibition and Learning through permanence adaptation [28]. An overview of the structure is given in Figure 2.8.

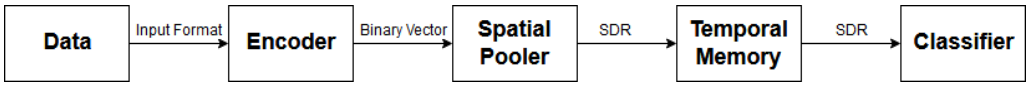

Figure 2.7: Overview of a typical HTM System. Firstly the data is encoded into a binary format of arbitrary size with some sparsity. The spatial pooler has the main function to convert the binary vector into an SDR with fixed 2 percent sparsity by keeping the semantic features. It usually inputs to the Temporal Memory, which interprets the input in learned contexts and makes predictions. Lastly, the predictions and detected anomalies can be extracted using some classifier.

Phase 0: Initialization HTM can run with or without topology option, which defines how the receptive field and furthermore all neuron connections are constrained in their connectivity to the input. Topology is an important

part of the HTM theory but is not used much in released systems due to the extra computational complexity and a lack of spatial patterns in input data such as scalar values. The algorithm will be described independently but with topology in mind.

Initially, each mini-column associated with the input has a receptive field assigned. A mini-column forms a potential pool of connections to a percentage of the input bits. It is used to model synaptic plasticity by adding and removing connections. Each potential connection has a permanence value assigned, and if this value crosses a permanence threshold, it becomes an actual connection. Initially, the permanence values are assigned randomly with a Gaussian distribution around the threshold and biased towards the center of the receptive field.

After initialization, the algorithm follows the phases 1-3 for each input at time t.

Phase 1: Overlap calculation and Boosting First, the overlap score of each mini-column to the current input will be calculated. The overlap score is defined by the number of connections, which are potential connections that have a permanence value higher than the threshold, to active input bits. Next, a boosting factor is applied to ensure a higher variety of mini-columns can represent features, which would otherwise be continuously suppressed by the inhibition phase. It is proportional to how often the column was active during the last duty cycles. The pseudocode of Phase 1 is given in Algorithm 1.

Algorithm 1 <SP: Phase 1>

for each c **in** columns **do**
  $overlap(c) \leftarrow 0$
  for each s **in** connectedSynapses(c) **do**
    $overlap(c) \leftarrow overlap(c) + input(t, s.sourceInput)$
  end for
  $overlap(c) \leftarrow overlap(c) * boost(c)$
end for

Phase 2: Inhibition Each mini-column has a local inhibition area, defining the columns it competes with to become active. It ensures that only a sparse set of columns becomes active and the radius is determined dynamically by the spread of input bits. A parameter *numActiveColumnsPerInhArea* sets k for the k-winners-take-all activation in each inhibition area that determines the sparsity of the SP. A column becomes active if its overlap

score is higher than the stimulus threshold and it is ranked under the k'th highest of its neighbors. The pseudocode of the Inhibition phase is given in Algorithm 2.

Algorithm 2 <SP: Phase 2>

for each c **in** columns **do**
  $overlap(c) \leftarrow kthScore(neighbours(c), numActiveColumnsPerInhArea)$
  if $overlap(c) > stimulusThreshold \ \& \ overlap(c) \geq minLocalActivity$ **then**
    $activeColumns(t).append(c)$
  end if
end for

Phase 3: Learning through permanence adaptation In the final phase, the local learning rules are applied. The learning happens only for the winning columns, the ones that won the competition and became active at time t. The potential connections permanence values to active input bits are incremented, and to inactive input bits are decremented. The permanence values are constrained to be between 0 and 1. Additionally, the boost values and inhibition radii are updated. Former increases the boosting factor of columns that have not been active often enough compared to its neighbors or decreases on the contrary case. It is an exponential function with the activity from a column and the average activity in its neighborhood as inputs. Furthermore, if a columns activity drops below a threshold, its permanence values are boosted. The inhibition radii are dynamically recomputed based on the input distribution. The pseudocode of phase 3 is given in Algorithm 3.

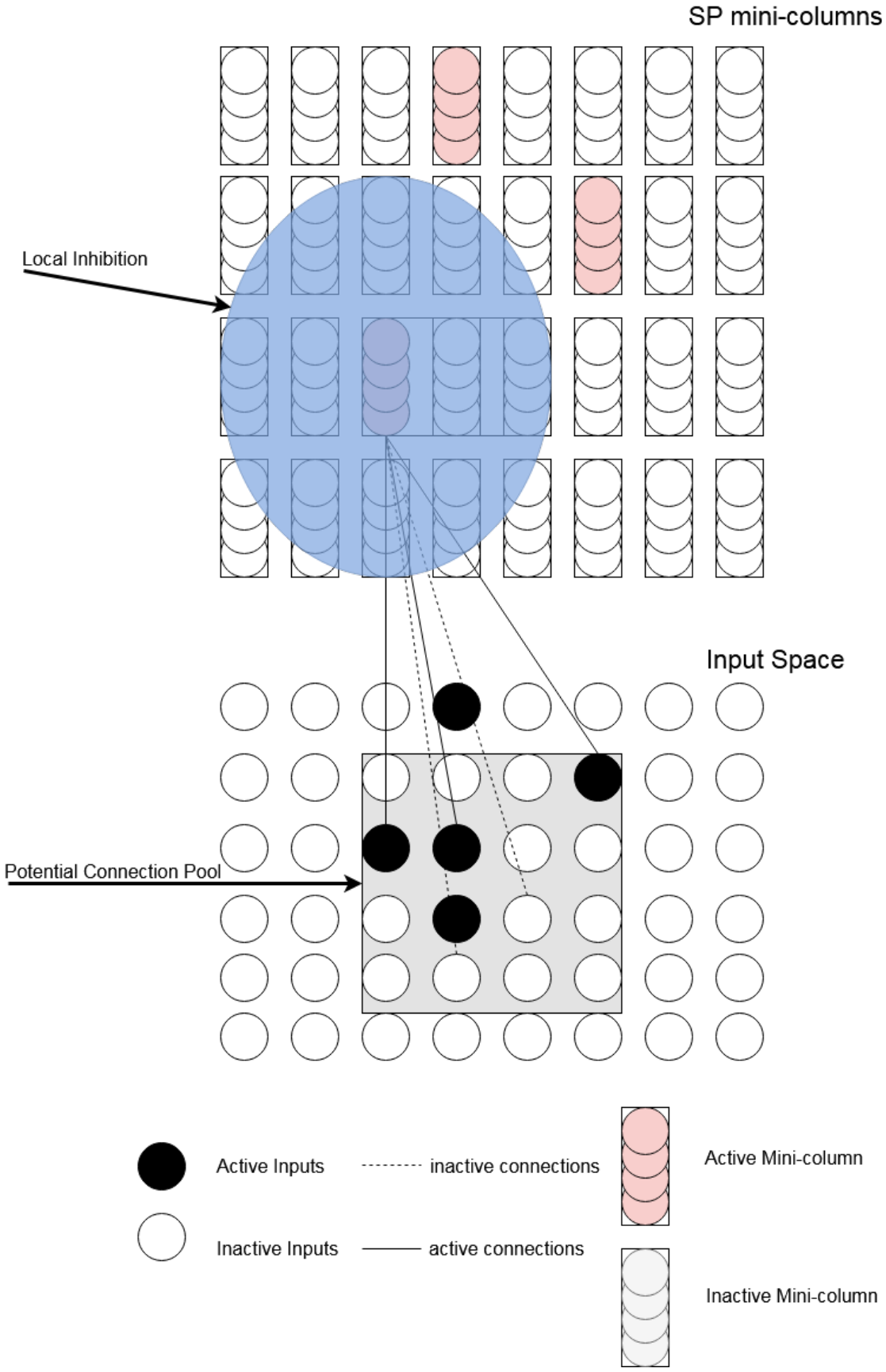

Figure 2.8: Overview of the HTM-Spatial-Pooler. Each mini-column has a pool of potential connections to the input space. If the amount of active connections crosses a threshold the mini-column becomes active.

Algorithm 3 <SP: Phase 3>

for each c **in** activeColumns(t) **do**
  for each s **in** potentialSynapses(c) **do**
    if $active(s)$ **then**
      $s.permanence \leftarrow s.permanence + synPermActiveInc$
      $s.permanence \leftarrow min(1.0, s.permanence)$
    else
      $s.permanence \leftarrow s.permanence - synPermInactiveDec$
      $s.permanence \leftarrow max(0.0, s.permanence)$
    end if
  end for
end for
for each c **in** columns **do**
  $activeDutyCycles(c) \leftarrow updateActiveDutyCycle(c)$
  $activeDutyCycleNeighbors \leftarrow mean(activeDutyCycle(neighbors(c)))$
  $boost(c) \leftarrow boostFunction(activeDutyCycle(c), activeDutyCycleNeighbors)$
  $overlapDutyCycle(c) \leftarrow updateOverlapDutyCycle(c)$
  if $overlapDutyCycle(c) < minDutyCycle(c)$ **then**
    $increasePermanences(c, 0.1 * connectedPerm)$
  end if
end for$inhibitionRadius \leftarrow averageReceptiveFieldSize()$

HTM Temporal Memory (TM) The temporal memory algorithm learns sequences of Sparse Distributed Representations (SDRs) in a temporal context to make predictions about the next SDRs [29]. In practice, it works like a coincidence detector which recognizes patterns and regularities of SDRs appearing over time and thus building a context that can predict the next sequential step. It is a core component in the HTM system and with input from the SP archives some key machine learning properties([30], [31]):

- **On-line learning** - In can learn from continuous data streams and quickly adapt to learn new patterns.
- **High-order sequences** - Dynamically determine temporal context/ Conditioning on past events.
- **Multiple simultaneous predictions** - Forms unions of predictions based on SDR properties.
- **Local unsupervised learning rules** - No gradient descent and back propagation, instead local learning of temporal patterns and possible detection of anomalies in the learned patterns.
- **Robustness** - SDR properties guarantee extreme noise tolerance and robustness.

In the TM algorithm, when a cell becomes active, it forms connections to other cells that were active just prior. These learned connections enable a cell to predict when it should become active next. The union of all cells that predict to become active next forms the basis to predict sequences of SDRs. It works like a distributed memory over the individual cells which makes it very robust to noise and individual cell-failure. It heavily relies on the properties of the SDRs, given in the Listing 2.1. As every cell participates in many different distributed patterns, it has multiple dendrite segments. Each segment represents a different context in which the cell learned to become active. If a certain number of cells in one of the segments becomes active and crosses a threshold, then the cell will be in the predictive state as it assumes from previous learning to become active next for that specific context.

The input to the TM algorithm comes from the SP and is an SDR that represents the active mini-columns. At each time step it performs three phases, firstly the evaluation of predictions to choose an active cell per active mini-column, secondly strengthening and weakening of the connections from active cells per learning segment and thirdly activating dendrite segments to set predictive cells for the next time step.

Phase 1: Evaluation of predictions to choose active cells In the first phase, the algorithm will determine the representation of the input in a specific context. The Spatial Pooler inputs a set of e.g. 40 active mini-columns, each consisting of e.g. 32 cells. By specifying the activation to individual cells, we dramatically increase the number of possible representations. The same input pattern can be interpreted with completely different individual cells depending on the temporal context given from the prior input.

For each active mini-column, it checks if any cells are in a predictive state. For cells to be in a predictive state, they need to have at least one active distal dendrite segment. This cells will be the winner and become active. If no cells are in a predictive state, the column will burst, meaning all cells in the column will become active. The set of active winner cells forms the representation for the input in the conditioned of prior input.

The Algorithm 4 stores active distal segments rather than predictive cells. A cell is predictive if it has an active segment. The pseudocode of the functions and details of the algorithm can be found in Appendix Section A.1 and the algorithm paper [29].

As depicted in Figure 2.9 and described in Algorithm 4 each mini-column can be handled for three different cases:

- Active and has one or more predicted cells - Predicted cells become winner cells
- Active and has no predicted cells - Bursting
- Inactive and has at least one predicted cell - False prediction, decrements permanence of connection

In the case of a mini-column bursting, meaning the input is not learned yet in this context, a winner-cell needs to be chosen to represent this pattern. It determined the winner-cell with the criteria in the following order:

1. Check if there is a cell that had a segment with active synapses, but not enough to cross the threshold and let the segment become active to bring the cell in a predictive cell. Choose the cell that had the segment with most such active connections as a winner.
2. If there were no active connections in any segment at all choose the cell which has the least segments, breaking ties randomly, and grows a new segment to all previously active cells. It now represents the input in this specific temporal context.

After the first phase, the set of active cells representing the input and winner cells is determined.

Algorithm 4 <TM: Phase 1>

for each column **in** columns **do**
  if $column$**in**$activeColumns(t)$ **then**
    if $count(segmentsForColumn(column, activeSegments(t-1))) > 0$ **then**
      $activePredictivedColumn(column)$
    else
      $burstColumn(column)$
    end if
  else
    if $count(segmentsForColumn(column, matchingSegments(t-1))) > 0$ **then**
      $punishPredictedColumn(column)$
    end if
  end if
end for

Phase 2: Learning per active segment In the second phase, the cells learn to make more accurate predictions the next time such a sequence appears. Only active segments or respectively the winner cells will learn. They

increase and decrease the permanence values of connections based on the correctness of the prediction and form new synapses to a sample of prior active cells that have not been included in a segment before.

Each active column has at least one winner cell. If it had predicted cells, it could be multiple, and in case it was bursting it will have chosen one cell to represent the input in that context. Each of this winner cells is represented as a learning segment. The segment is originated from the prior active cells. On each of this learning segments:

- Increase the permanence value on every active connection
- Decrease the permanence value on every inactive connection
- If a segment has fewer than SYNAPSE_SAMPLE_SIZE active connections, grow new connections to a subset of the winner cells.

By applying the local learning rules, the algorithm allows synaptic plasticity. This means it can quickly grow new connections and remove old ones or keep them to make multiple predictions with different segments. It enables fast adaptation to new sequence patterns and continuous learning.

Phase 3: Predict cells for next time step In the last phase, the algorithm calculates the active segments representing the predicted cells for the next time step.

- It iterates over every cell in the layer and counts for every segment of that cell the number of connections to currently active cells.
- If the number of active connections in a segment exceeds a threshold, then the segment is marked as active, and the cell will be in a predictive state for the next time step.

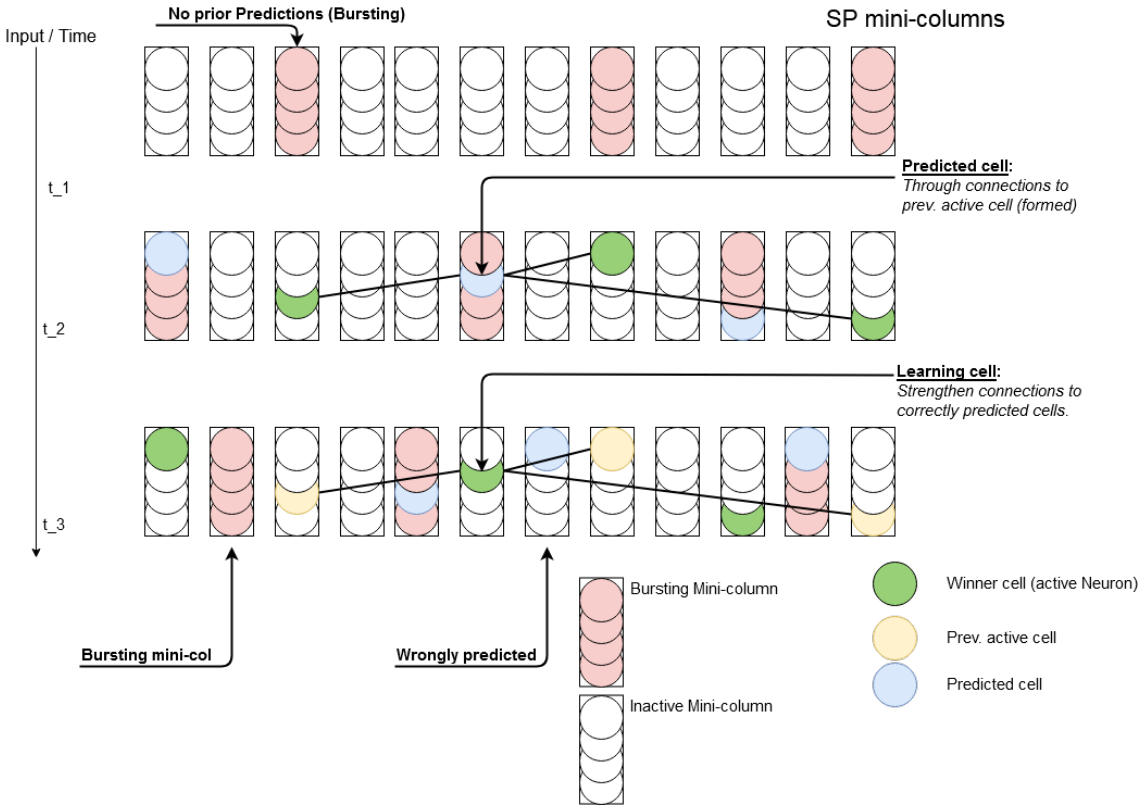

Figure 2.9: Overview of the HTM-Temporal-Memory. Each active mini-column is sparsified to individual cells representing the input in the specific temporal context. The three possible cases for a column are shown: (1) It has correctly predicted cells and is active, (2) It has no predictive cells and bursts and (3) It is inactive and has falsely predicted cells. The learning in every active segment reinforces connections to prior active cells and decrements inactive connections. Additionally, new synapses can be grown for each segment.

2.2 Temporal Difference Learning

Reinforcement learning (RL) is concerned with how software agents choose among actions in an environment to maximize some notion of reward signal. As the algorithm is integrated into an HTM system, the choice is aligned with their design principles. Thus the study only investigates a biological plausible model of temporal difference learning, a model-free approach of RL.

Temporal Difference (TD) learning is an approach to learn how to predict a quantity that depends on future values of a given signal [32]. It is often described as a combination of the traditional Dynamic Programming (DP) and Monte Carlo (MC) learning and consists of a variety of algorithms [33]. As the name suggests, it works with the differences of estimates at different time steps [34]. Recent successes have shown human-level control in deep reinforcement learning enabling a single agent to out-

perform on a variety of different unrelated tasks only from the visual input and a game score, and control over continuous action spaces ([22],[35]). The predictions in HTM need to be utilized to enable an agent to choose among a variety of actions towards a goal.

In the most famous versions, namely Q-Learning and Sarsa, states and actions are coupled to compute the rewards. However neuroscientific research indicates, and it is part of further sensorimotor inclusion in HTM theory that is currently researched, that the output of a layer is itself encoding the action. Additionally, neurobiological research in dopamine regulation is focused on on-policy methods, thus also taken as a limitation in this study continuing the work from Kaan [17]. It narrows down the choice for the Reinforcement Learning algorithm to $TD(\lambda)$ with eligibility traces, for on-policy learning with states representing actions.

TD Lambda There are different versions of the algorithm, but this study will only concentrate on $TD(\lambda)$ with backward view and replacing traces. Instead of accumulating the eligibility trace for a state each time it is visited we simply reset it to 1, enabling faster and more reliable learning [36].

In Temporal Difference learning, every state has a long-term reward value associated with it. This value V(s) is updated with an error signal δ_t at each time step, to refine towards the true value. In contrast to Monte Carlo learning, it uses the estimates of the future states to improve its prediction, so-called **bootstrapping**. It allows learning continuously and without the need to wait until the current sequence is terminated and an actual reward is given, as it can work entirely with estimates. The exemplified comparison of the two approaches is depicted in Graph 2.10 for a predicted travel time one would need from office to home [37]. Furthermore, it is **sampling** episodes and not exploring the full space as Dynamic Programming attempts. This enables it to work on partially observed environments exploiting the Markov property and building implicitly a Markov model, without being too heavy computationally.

The calculation of the error signal as shown in Equation 1 sums a possibly given reward R_{t+1} with the difference of the long-term reward estimate of the next state $V(S_{t+1})$ versus the current state $V(S_t)$, weighted by a discount factor γ that makes further rewards less important. V(S) is the value of a state S at time t. R_{t+1} is the actual stimulated reward at time t+1 and γ the discount factor on the rewards.

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{1}$$

The idea of $TD(\lambda)$ is to keep an eligibility trace based on the occurrence recency of a state. Using this trace, it is possible to update all recently visited states in one episode instead of updating only the current state and needing multiple episodes propagating the estimates backward in time. The trace keeps track of the recency, how long in time ago, and frequency, how often, a state was visited in the episode. The λ parameter controls the decay of the eligibility trace. If set to 1 it works equivalent to Monte Carlo learning, deferring the credit until the end of an episode not decaying the updates. On the other hand, if set to 0 it will update only the most recent state. The Equation 2 shows the update rules for the eligibility trace $E_t(s)$ of state s at time t with a discount factor γ.

$$E_0(s) = 0 \tag{2}$$

$$E_t(s) = \begin{cases} \gamma\lambda E_{t-1}(s) & \textbf{if } S_t = s \\ 1 & \textbf{if } S_t = s \end{cases} \tag{3}$$

Every state gets updated at a time step t. The error signal is weighted by the learning rate α and the eligibility trace $E_t(s)$ of the state. The Equation 4 shows the state value update rule.

$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s) \tag{4}$$

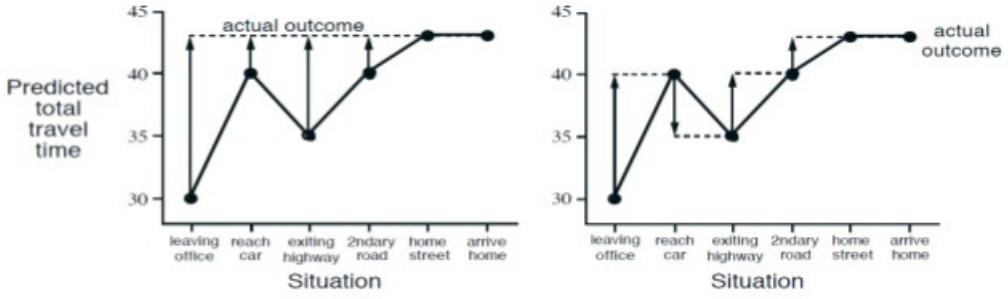

Figure 2.10: Exemplified comparison from MC (left) and TD(right) approach [37].

3 Method

This study uses two main methods for acquiring knowledge and conducting results:

- Extensive Literature Review
- Controlled Experiment

An extensive literature study will be conducted to acquire the background knowledge of HTM and reinforcement learning and to be able to construct an architecture for an autonomous agent. The review is limited only to the necessary parts to design the agent and experiment. It includes amongst other things articles, online-lectures and discussions especially in the machine learning domain.

The controlled experiment is based on Lake et al. study on the omniglot dataset for one shot character recognition [12]. In their research, they asked participants on the "Amazon Mechanical Turf" platform to do one-shot character recognition. More specifically they choose an evaluation set of 10 alphabets to form the ten-within classification task. Each task had 40 trials where a test image was shown and 20 characters from the same alphabet. The original setup is depicted in Figure 3.11. The drawings were produced by four relatively typical drawers and picked to maximize diversity if an alphabet consisted of more than 20 characters. The four drawers were paired randomly and in each pair one drawer produced the examples for the test images of 20 tasks and the other the training images. Example alphabets with 20 characters created by four drawers are shown in Figure 3.12.

Instead of constructing a confidence matrix for each test letter to the 20 training image classes, the experiment attempts to model the human participants' conditions. The agent has just the visual screen as input and receives a positive reward if it selects the similar image with a mouse click. It does not understand the instructions given, so it needs to learn the desired behavior from initial random trial and error. Similar to the original experiment 30 alphabets could be used for training and the agent evaluates on the evaluation set with other alphabets. This will require learning the patterns of lines and their composition to transfer the learned features to new problems in similarity-based one-shot comparisons.

On the technical side, the experiment is realized using OpenAIs gym environment with the Universe extension. It is a framework developed in 2016 for reinforcement agents to learn universally on benchmarks for games or other tasks. On top of this extension, a team around Andrej

Figure 3.11: The experimental setup for human participants in Lake at al. paper. Marked with a red square we can see the test image. The participant had to choose the similar character from the 20 characters of the alphabet, drawn by another person than the test image [12].

Figure 3.12: Three alphabets of the omniglot data set exemplified. Each shows 20 characters drawn from 4 different people [12].

Karpathy developed "World of Bits", that let the agents work on browser tasks with mouse and keyboard instructions [38]. It connects the client via VNC to a remote environment which launches a Chrome instance with the given task. The reward is read from the screen and together with the observations send back to the client. The agent can send actions towards the remote to change the environment. The infrastructure from a universe setup is shown in Figure 3.13. It works in real time, which can lead to problems especially in more complex game agents if the actions are generated too slowly. This was one of the reasons why it is not under development any further since 2018.

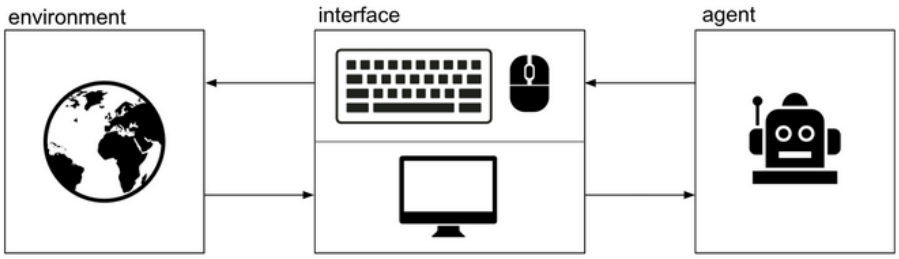

Figure 3.13: The infrastructure imposed by the universe library.

The tasks are defined with HTML, CSS, and Javascript for a mobile-sized screen which allows easy customization of your experiment.

3.1 Dynamic character experiment

This study replicated the experiment as shown in Figure 3.14 by constructing a browser task which shows a test image in the upper row and 20 images from the alphabet in the lower four rows. The choice of the pictures is similar as in the original experiment, in such the test image is produced by a different drawer than the 20 images to choose from.

However, as this task is very complicated and would need much computational power and time to train an agent, in this study the experiment is simplified. Instead of 20 images the agent only sees two possible images in the bottom row and the similar test image in the upper row that indicates which of the two letters to click. It receives a positive reward if it clicks the most similar character representation compared to the test image and otherwise a negative reward. Next, to the task the most recent reward, an average of the ten last trials and a timer is shown. If an agent exceeds the specified maximum time for a task to select an image it will also receive

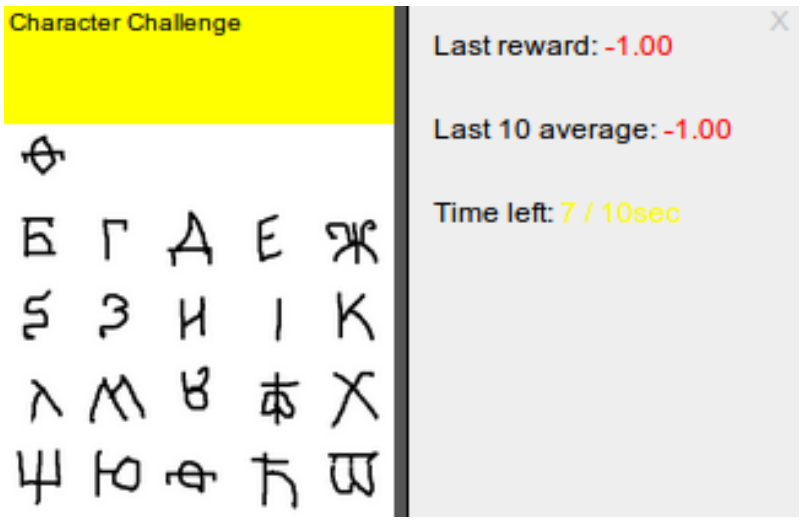

Figure 3.14: The original experiments human conditions from Lake et. al. replicated as a browser task for an agent to solve. The character in the first row on the left is the test image. It has to choose the most similar character from the following 20 letters.

a negative reward. This is necessary to force behavior of the agent. Additionally, the reward is between [-1,1] and is proportional to the rest time which favors a fast choice. An image of an exemplified task is shown in Figure 3.15. Further, the experiment is simplified such that learning and testing are done in the same run. This is due to untested serialization of the network and to reduce the complexity of the task. Thus no one-shot learning is evaluated, only the same letters are shown from a repeated alphabet set.

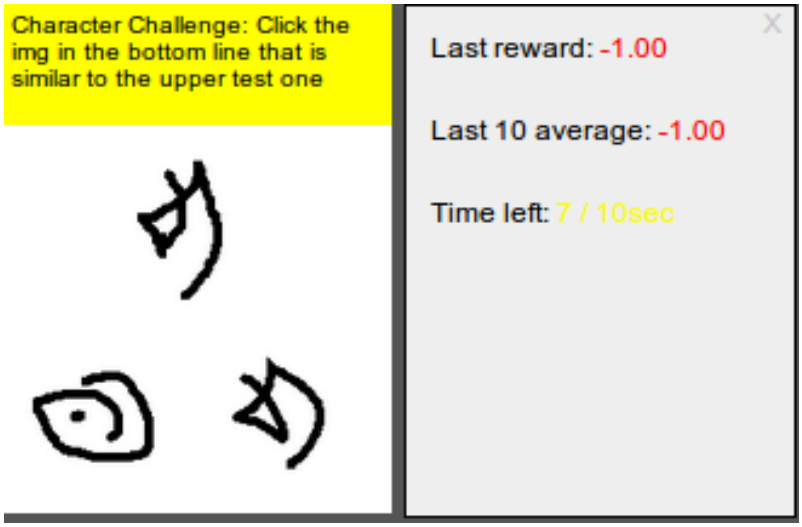

Figure 3.15: An example of a task the agent had to solve. The character in the first row in the middle is the test image drawn by a different person as the bottom row pictures. It has to choose the most similar character from the two letters in the bottom row.

3.2 Static square experiment

The described task requires the agent to interpret a randomly changing environment and interact with it based on the features of the complex character drawing. For an analysis and comparison of the framework it is also tested on a static problem, where the environment is not changing and the agent only has to learn a mapping from its actions to environmental change. The agent perceives 4 black squares on a white background and has to learn to click the upper left one to get a positive reward, a click anywhere else results in a negative reward. The agent learns continuously without any reset of the cursor or the layer states. Figure 3.16 shows the task that the agent has to solve.

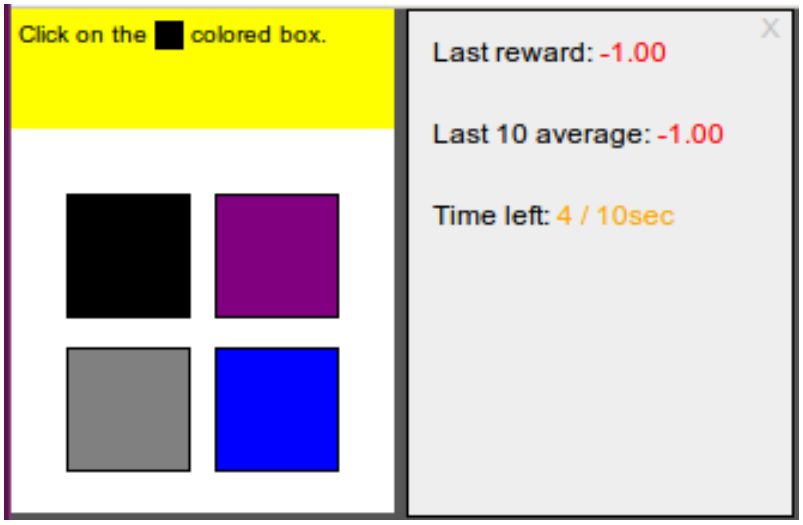

Figure 3.16: An example of a static task the agent had to solve. The encoder will convert the image into black and white which reduces the color for the perception of the agent. The goal of the task is to click the upper left square to get a positive reward.

The measured results at each iteration include the non-zero reward, the TD-Error, the neural activation and correct predicted activation of the layers, L4, L5, D1 and the number of voluntary excited Motor layer neurons. It helps to analyze the performance and the learning progress of an agent in the experiment. Neural activity is measured by the number of active neurons.

3.3 Reliability and Validity

This study tries to ensure a high quality of reliable and valid results. It requires a simple replication process of the task and results, thus next to the implementation details in the Appendix all source code is published open

source and can be used to replicate results or for further exploration. Except for the Motor-Layer component, network serialization is implemented to save and load agent models. Once finished this enables reproducing an agent that was trained before and test it on the same task.

For the validity of the results, the environment, as well as the agent, are packed into docker images, a mechanism that allows platform independence. This makes it possible to run experiments on different machines and independent of the local environment. As most challenges are randomly initialized or generated in each iteration, complete equality in the results is hard to ensure, but the result data is also published as comma separated text file to allow as much transparency as possible. Concerns regarding the experiment could be the restriction of the action space to be discrete and rendering the environment in real time with possible connection delays that lead to volatility in the results.

Additionally to the open sourced implementation, the data set was taken from an open-access site, which enables open research and comparison with the study and all documented results. Furthermore, the conclusions drawn from the results take the method and implementation details into account, use standard terminology for description and conclusions about HTM system properties are supported by related studies in peer-reviewed papers. The experimental setup was chosen to allow a degree of generality and comparison to associated experiments with the same data set.

3.4 Ethical considerations

As the algorithms become more powerful and artificial intelligence is becoming ever more important in the next century, ethical considerations can not be left aside. I believe that a societal discussion is needed to address the challenges and share the benefits of the emerging technologies. It is important for every researcher to question the impact and direction of the studies and the proposals towards a code of ethics should be continued. Furthermore, I think that an open research manner, supported by open access journals, as well as open data is crucial for a fair distribution of wealth and powers developing such algorithms. In this study, all code is open source, and Numenta publishes all their work in open access journals, which widely supports this view. They enhance collaboration and underscore the importance of educating about this key technologies through their channels.

4 Architecture and Implementation

In this chapter, the general architecture of the agent is explained. First, an overview is given, which is followed by the implementation details. Further information such as used parameters can be found in the Appendix. The architecture is developed on the basis of Ali Kaans research [17].

4.1 Overview

As described in Chapter 3, the agent does interact with the environment via a client-remote setup. A universe environment is used for building the remote environment. The system architecture of the client is receiving a reward $R \in [-1, 1]$ and an observation of 1028x768 RGB-pixels saved as array structure with values from 0-255 for each of the three color channels. After the computations, the system has to return an action. The discrete action space includes:

- Move mouse right/left/top/bottom by 10px
- Click mouse
- No action

The client is connecting to the environment via VNC, passing the observation and reward on to the network and in the end interpreting the SDR output with a constant conversion table to derive an action that is sent back to the remote. The flow is depicted in Figure 4.17. As stated earlier the remote environment is running in real time and the client receives up to 60 frames per second. If it is not processing them fast enough, they are skipped and discarded.

The neural network architecture consists of 8 layers that can be separated into three main functional parts and layer 5 in between where everything integrates. As this study leans on HTM theory and related studies that attempt to be biologically plausible, the naming of the layers is based on biological models. Usually, they include layer 2-6 in a cortical column of the cortex and D1/D2 for the Go/No-Go pathways in the Striatum of the Basal Ganglia respectively. In the theories former is responsible for pattern recognition and latter for dopamine control modeled with reinforcement learning in this study. A sequential description of the layers functional interplay is given in the following listing:

1. The first two layers, namely the encoder and layer 4, are for interpreting the raw sensor data in the context of the previous motor output.

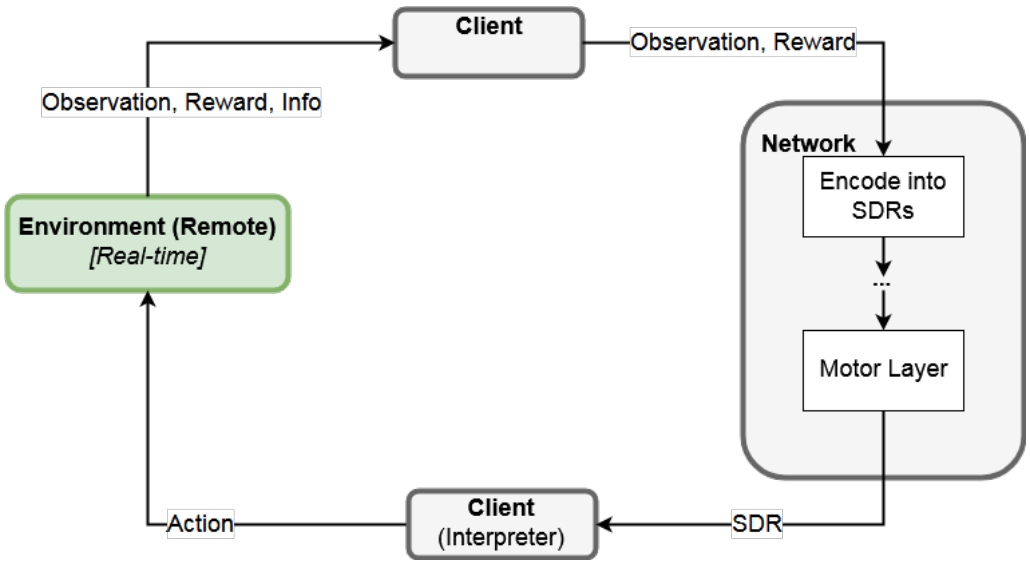

Figure 4.17: System flow where client-remote communication and network processing is shown.

2. The second part consists of layer 2 and 3 that form higher representations of the sequences independent from immediate time.

3. Layer 5 integrates the immediate predictions from new sensory information together with high order predictions to form the current agent state. This state is used in temporal difference learning with the given reward.

4. The third part is responsible for using all information to determine an action that will lead to a positive state. It utilizes the Layers D1 and D2 to excite or inhibit neurons in the Motor layer that lead to a positive or negative state in previous iterations respectively. The neurons from layer 5 learned to associate themselves with the motor neurons that produced them. Thus if they get voluntary active, they will excite or inhibit neurons in the Motor layer that will finally output the action that mapped to the desired state.

The crucial part is the interaction from layer 5 predictions in D1 and D2 towards an excitation or inhibition of neurons in the motor layer, which were learned to produce that state. This mechanism is supposed to generate voluntary behavior. The architecture is visualized in Figure 4.18.

A pivotal difference to Ali Kaans original architecture is that a part of the layer 5 functionality is directly merged into the motor layer which receives input from layer 5, D1 and D2. This was mainly done due to limitations in the NUPIC framework and to simplify the model, and changes association from layer 5 and motor neurons. Furthermore, there are a lot

of implementation details that differ such as global decay, weighing in the spatial pooling and resetting of layer states. More information can be found in the implementation details and parametrization.

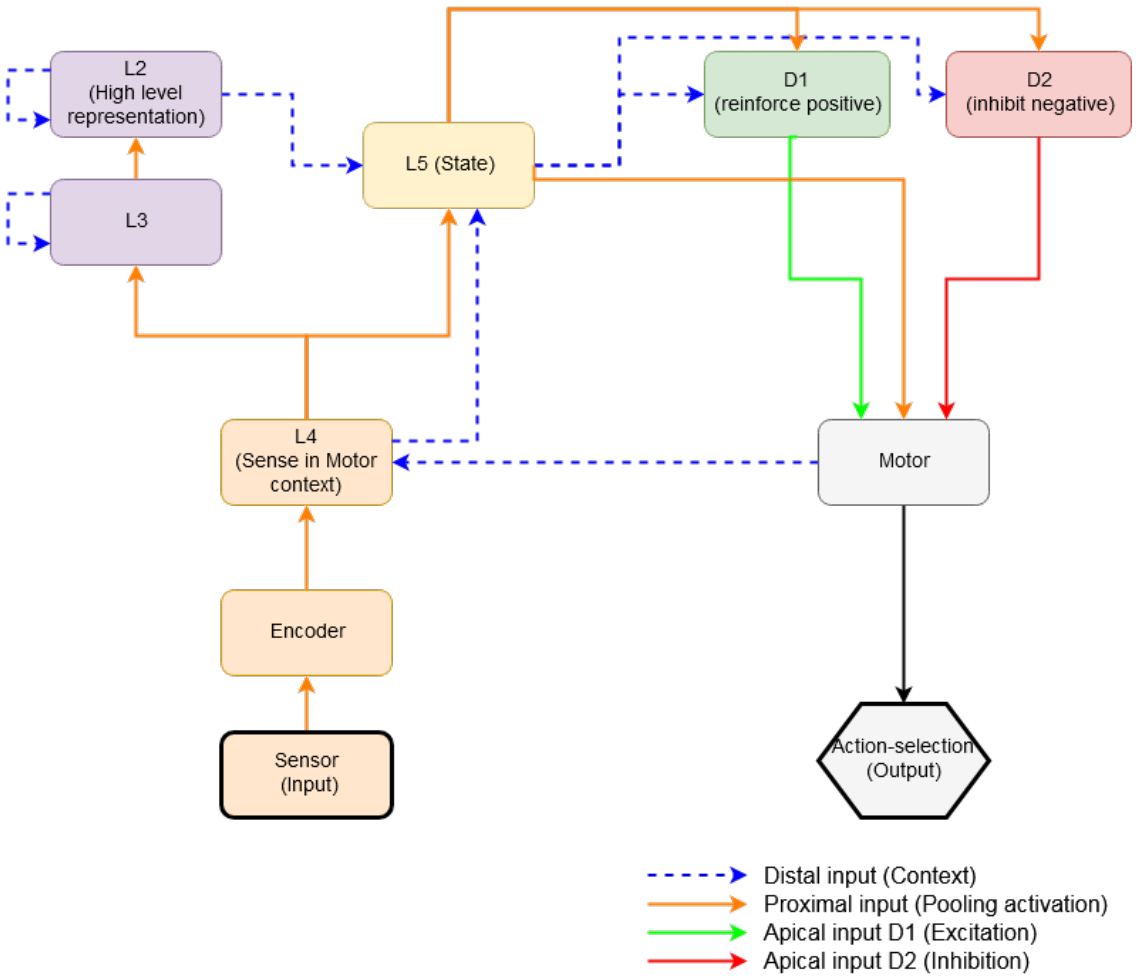

Figure 4.18: An overview of the neural architecture. The different functional parts are distinguished by color and connections between the layers are interpreted according to the legend.

4.2 Layer design

In the following subsections the different layers and their functionality is described in more detail.

4.2.1 Encoder/L4:Sensory integration

As mentioned in the method section the input to the network consists of raw RGB-pixel data. The encoder first crops the image to the meaningful part that the agent perceives, ignoring all pixels that are not part of 160x160 pixels with the task as exemplified in Figure 3.15. Further, it converts RGB to a greyscale image and then to black(1)/white(0) binary pixels using a threshold.

The encoder is very important as it defines how the agent perceives the environment. It needs to be sensitive to change of important variables but also generalize the visual input to let the agent build an invariant representation. A general encoder is an own research topic, and there are many attempts to build encoders close to the retina of the eye. In this study, a simpler encoder is used that is more task-specific and uses human engineering. Additionally to the encoded pixel data the agent needs a focus on the mouse, as its movements and clicks are the main determiners for the reward. Thus it is important to give the actual mouse position a greater weight than just a point on the pixel screen. Several solutions are possible such as encoding the mouse coordinates, using a smaller perceptive field around the mouse that moves or appending an additional focus field. In this framework, the latter method was taken and to the full encoded image of the task, a smaller radius around the mouse is appended in black and white encoding. This encoding is shown in Figure 4.19. It is important how to weight the different features, by default as greater the input length of the SDR is as more the feature is weighted in the spatial pooler. In this experiment, the ratio was 160x160 pixels to a radius of 40 resulting in 80x80 pixels. The full image takes 25600 input bits of the SDR, the cursor focus with 6400 bits and 6400 bits to encode mouse coordinates result in a total of 38400 input bits. The scalar coordinates were encoded as 6400 additional bits using the Numenta Coordinate Encoder for SDRs with a sparsity of 4 percent. The weights and the general encoding is a crucial experimental parameter.

The encoder provides the proximal input for layer 4, which is pooled with an HTM Spatial Pooler to create an SDR as described in Section 2.1. The activated mini-columns are fed into the Temporal Memory which additionally receives basal input from the Motor layer. This enables the Tem-

Figure 4.19: The agents perceptual field consists of an 80x80 focus area around the mouse and the full task 160x160 black and white pixels. The focus field helps it to identify changes in the environment when moving the mouse cursor with generated behaviour.

poral Memory as described in Section 2.1 to interpret the sensory information in the context of the recent motor behavior and make a prediction about the next sensation.

The neural activation of layer 4 is used as proximal input for layer 3 and 5, driving the activation of mini-columns.

4.2.2 L2/3: High level representations

In HTM theory there is the notion of **Temporal Pooling**, a pooling layer such as the Spatial Pooler but consistent over time allowing high-level predictions from the temporal sequences. However, there is no real solution yet how to form such a representation, so the most common method that is used for this feature is a **Union Pooler**. It utilizes the strong union properties of SDRs, which can be combined to a larger union and still operate functionally.

HTM Union Pooler The Union Pooler receives the neural activations and successfully predicted neurons as input from the Temporal Memory of another layer. Then, such as in the Spatial Pooler it calculates the overlap from each mini-column with the input. The overlap, called pooling value, is determined for each neuron with respect to the weighting parameters of active and predictive, active neurons. These values are added to the moving average of previous pooling values, after a decay which ensures more recent activations weight more. The Equation 5 shows the calculation of pooling values for each mini-column i, with ActiveOverlap being the number of active input neurons it is connected to and PrediciveOverlap

the number of correctly predicted, active input neurons it is connected to. The weighting parameters have to be defined at initialization. Additionally the process is visualized in Figure 4.20.

$$PoolingScore_i = decay(PoolingScore_i) + ActiveOverlap * ActiveWeight + PredictiveOverlap * PredictiveWeight \quad (5)$$

After the values are calculated the union of mini-columns with the highest activation are chosen to become active. Thus it is possible to form a high-level representation that can be consistent over time for a changing input. This is necessary for example to represent a constant object even though the sensory information is changing over time.

In this experiment, the input for layer 3 comes from the neural activation and prediction from layer 4 Temporal Memory. After forming a Union SDR, it will apply Temporal memory and feed the neural activation into layer 2. Here the same process is repeated allowing an even better generalization and high-level interpretation of the input.

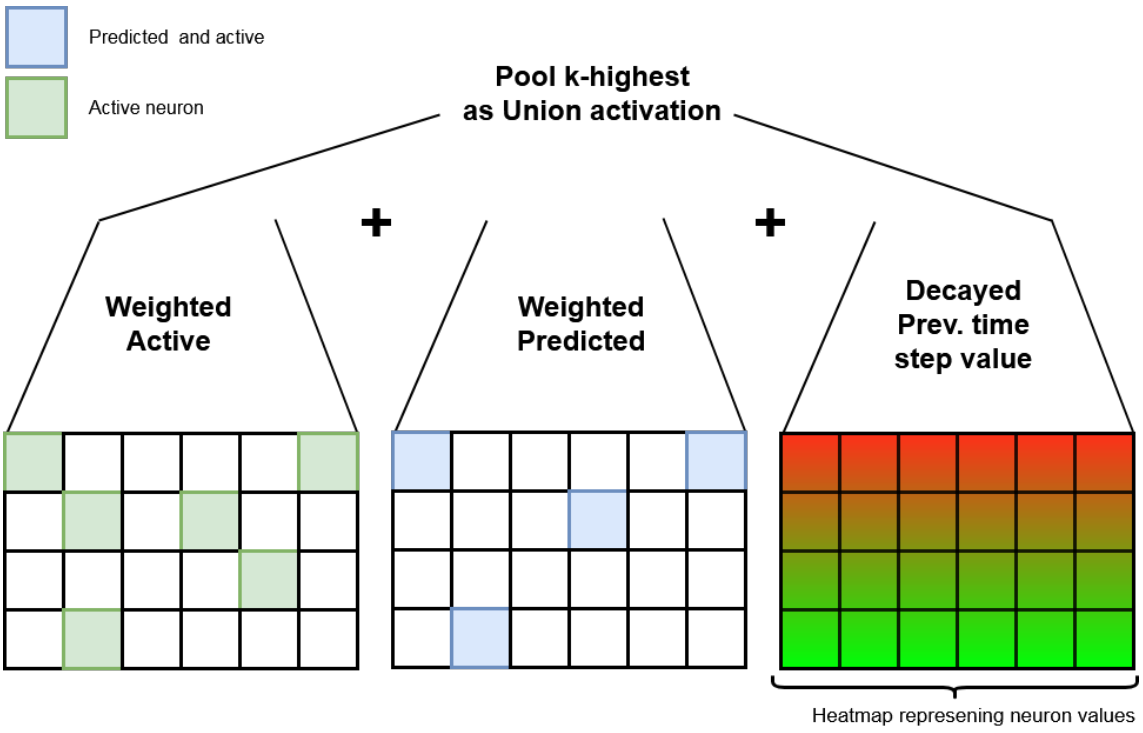

Figure 4.20: An overview of the neural architecture. The different functional parts are distinguished by color and connections between the layers are interpreted according to the legend.

4.2.3 L5: Agent state

Layer 5 is doing the integration of all the information towards a state of the agent. It receives the neural activation from layer 4 as feedforward input and uses high-level representations from layer 2 and neural activation from layer 4 to contextualize this activation in the Temporal Memory.

It is essential that the layer offers a large enough capacity representing the state of the agent and environmental model distinguishable. The capacity of a layer can be calculated as shown in the mathematical properties of SDRs [26]. The Equation 6 shows the calculation for a layer R with n neurons where w are active. The number of neurons is determined by multiplying the number of mini-columns with the number of neurons per mini-column.

$$R(n, w) = \binom{n}{w} = \frac{n!}{w!(n-w)!} \tag{6}$$

In an example with standard parameters operating with 2048 mini-columns, 32 neurons per mini-column and sparsity of 2 percent, it would have 65536 neurons where 1311 are active. Resulting in a huge capacity, enough to representing the different states in a simple environment.

The mini-column activation is forwarded to D1 and D2, to be utilized to make predictions about the next state and depolarize the motor layer such that a favorable state is reached. Additionally, the neural activation is given as distal input to make the predictions more accurate.

4.2.4 D1/2: Beneficial behaviour

The layers D1 and D2 are at the core of the reinforcement integration in HTM. Their goal is to excite/inhibit motor neurons to generate the behavior to reach a more favorable state. They are provided with the input from layer 5, the current state of the agent. They learn to use the mini-column proximal activation and neural distal activation to make predictions of the next state. Additionally, they grow apical connections to the motor layer and strengthen/weakening these connections dependent on the TD-Error.

This requires next to the computation of the Spatial Pooler and Temporal Memory the calculation of the state values for Temporal Difference learning. As explained in Section 2.2 it needs to keep track of the eligibility traces and values. The state value is the accumulated average of all individual neuron values.

At every time step first the eligibility trace for each neuron is updated with the mentioned update rule as shown in the equation. The hyperparameters TD Discount and TD Trace Decay are defined at initialization. The motor layer is also mirroring the mini-column and distal activation of layer 5 to calculate the excitation of its neurons after apical excitement.

$$Trace_i = \begin{cases} 1 & \textbf{if } Neuron_i = active \\ Trace_i * TDDiscount * TDTraceDecay & \textbf{else} \end{cases} \tag{7}$$

Secondly, the average state value V for the current time step t is calculated as given in Equation 8. It is the summation 1..n of all active neuron values of the current time step weighted. The weighting is defined such that predicted neurons that became active can express themselves more than activation through bursting.

$$V_t = \sum_{i=1}^{n} Value_i * Weight_i \tag{8}$$

Calculating the state value V at t-1 for the previous neural activation, representing the expectation of reward at the preceding time step and the newly computed state value it is possible to determine the TD-Error. We apply the TD Discount hyperparameter and add the received reward R. It is important that we newly calculate the state value for the previous activation at this time step. As we updated the neuron values at the end of the previous calculation of the average state value, it uses the previous values to calculate the temporal difference error. Both are variants for temporal difference learning and were tested in experiments. Equation 9 shows the calculation using the average state value from the current activation compared to the previous activation.

$$TDError = R + TDDiscount * (V_t - V_{t-1}) \tag{9}$$

In the last step, the neuron values are updated such that the expectation comes closer to the actual reward - more accurately guessed by the next time step. It uses the calculated TD-Error and applies the learning rate and eligibility trace before adding it to the previous value for each neuron i.

$$Value_i = Value_i + TDLearningRate * TDError * Trace_i \tag{10}$$

The permanences of apical connections are saved in the Motor layer, where the TD-Error is forwarded to and the activation of D1/D2 as apical input.

4.2.5 Motor: Action generation

The Motor layer calculates the final output of the network. It mirrors the state of layer 5 basal and proximal activations to calculate an overlap with the apical connections it forms to D1 and D2. However, the proximal input is only used in learning apical connections and the basal input to calculate the intersection with apical depolarized cells. It utilizes the TD-Error to learn apical connections, that help to decide which motor neurons to active through excitation and inhibition. The layers computational flow is depicted in Figure 4.22 and explained in the sequentially in the next steps.

Learn Apical Connections Firstly, receive apical input from the D1/D2 layers and the current TD-Error. Utilize it to increment/decrement the permanence values of the connections between neurons in the copied mini-columns of layer 5. The connections work as in the Temporal Memory by forming and removing links on segments. The permanence increment for an active connection follows the rule given in Equation 11, dependent if it comes from the D1 or D2 layer. The decrement is given in Equation 12 and is slightly boosted by a factor of 2 to filter more reliable the connections that drive action and prune others. Additionally, there is a hyperparameter to decrement incorrectly predicted neuron connections, as otherwise only activated neurons learn. These steps enable the learning of apical connections from D1 and D2 to the motor layer dependent on the state they produce.

$$ApicalPermInc = \begin{cases} TDError * TDLearningRate & \textbf{if } D1 \\ -TDError * TDLearningRate & \textbf{if } D2 \end{cases} \tag{11}$$

$$ApicalPermDec = |TDError| * TDLearningRate * 2 \tag{12}$$

Calculate voluntary neural activation In the second step the copied basal depolarized cells from layer 5 are used to calculate the intersection with the depolarized apical cells. This was taken in Ali Kaans framework from neuroscientific evidence and can be interpreted as filtering the predictions of the sampled state to a more reliable subset using context information from layer 4 and layer 2. This layers were responsible for the activations in layer 5 in the first place and include high level and encoded sensory representations in motor context implicitly. The second step produces a

set of voluntary neural activations in the dimensional size of layer 5 and distinguishable by their apical origin D1 or D2.

Learn association from Motor and L5 activity In the third part, the association between the previous action and the layer 5 state it produced are learned. This is a critical step as it connects the previous active motor neurons with the active layer 5 neurons from the current iteration. This enables the architecture to excite or inhibit motor neurons that are associated with producing the computed voluntary active layer 5 neurons which ultimately should lead to producing that state. The connections are learned to the activation of layer 5, while the excitation and inhibition is only from voluntary active neurons through apical and distal depolarization. In Figure 4.21 the concept of producing behavior is depicted in a conceptual way. It requires the agent to learn the state transitions in L5, D1, and D2, the mapping of motor commands to states that are produced and the mapping of apical connections to the neurons that produce a valuable or harmful state. Once learned the agent is able to generate behavior that can be beneficial for maximizing the reward.

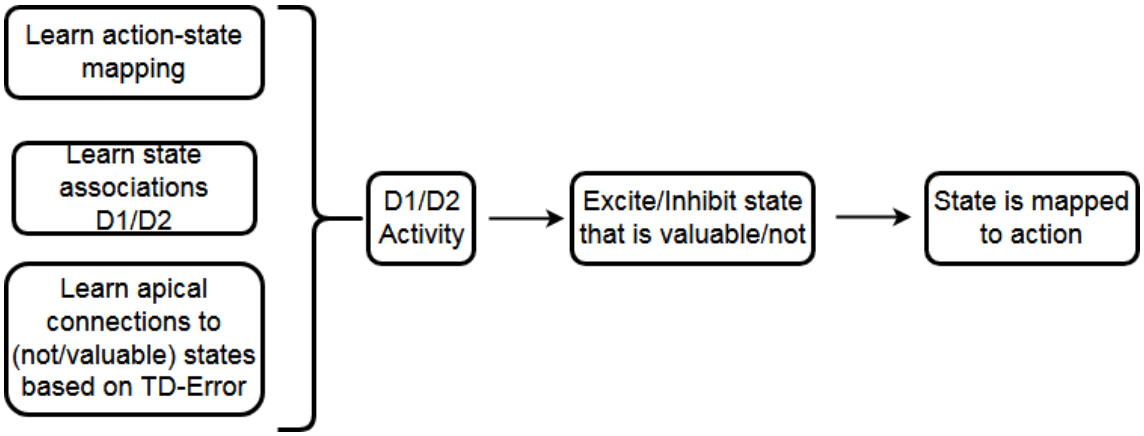

Figure 4.21: The simplified concept of how the agent is able to produce benevolent behavior based on learned state transitions and the excitement or inhibition of motor neurons that produced a state.

Excite/Inhibit mapped motor neurons In the last part we use the set of motor neurons that have apical connections to layer 5 neurons of states their action produced. They represent the action space implicitly as their activation output has a constant mapping to the action sent by the client. Each has an excitation level associated which is randomly initialized at each iteration following a normal distribution. The random initialization is needed

to produce arbitrary behavior and explore the action space when no voluntary behavior is learned. The values of the motor neurons get excited for all voluntary neural activations that origin from D1 and inhibited for the ones from D2. This requires a mapping from the layer 5 dimension to the motor neurons, which utilizes the learned connections from layer 5 state neurons to motor neurons. In this way, layer 5 excites and inhibits the action for a state it wants to produce. The excitation and inhibition of a motor neuron i follows the in Equation 13 shown rule with the hyperparameters ExcitationRate, InhibitionRate, Mean, Variance for the initial random value produced by the function randn.

$$ExcitationLevel_i = randn(Mean, Variance) + \sum_{ActiveMappedCells} \begin{cases} ExcitationRate & \textbf{if } D1 \\ -InhibitionRate & \textbf{if } D2 \end{cases} \tag{13}$$

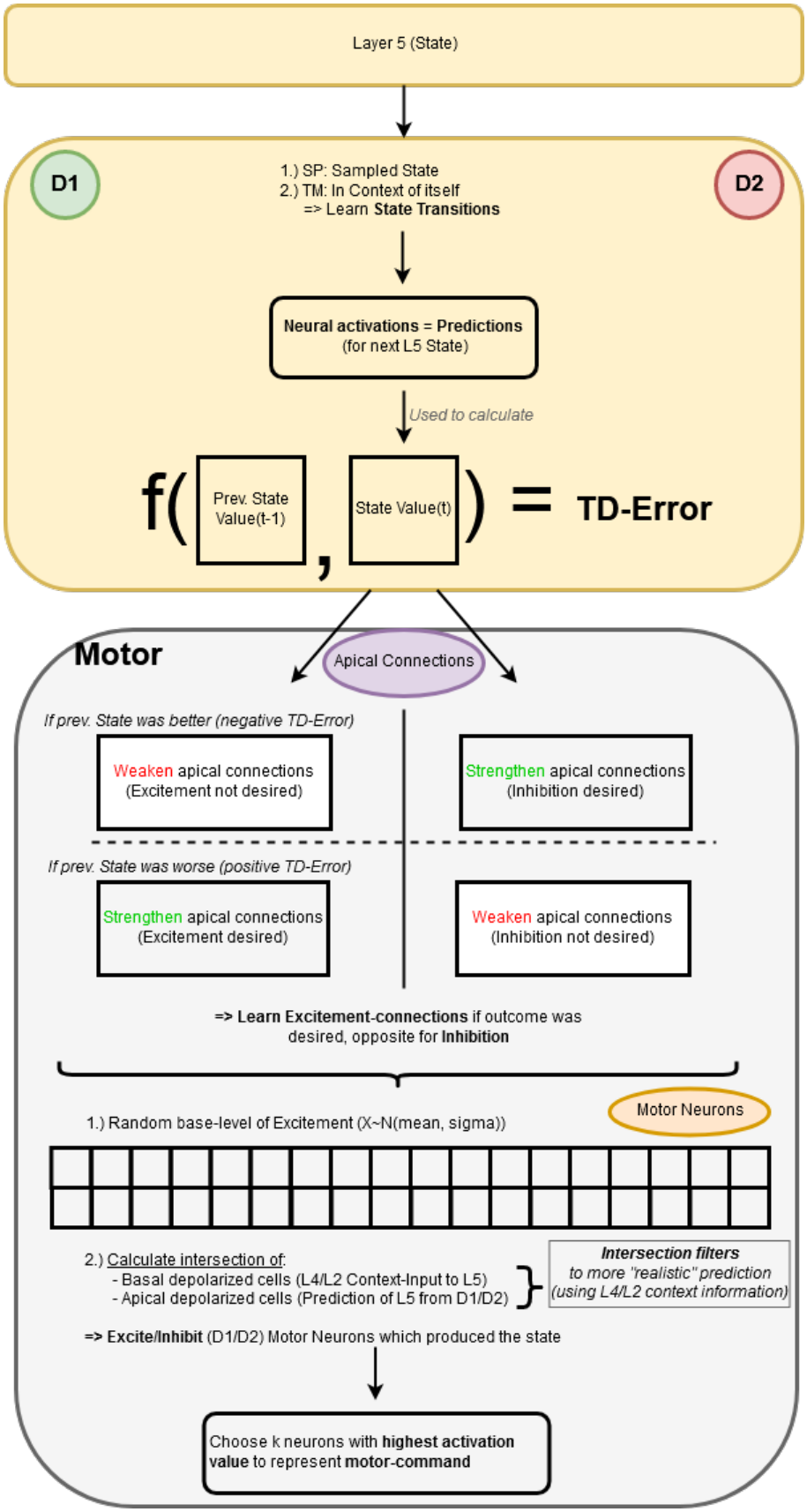

Figure 4.22: An overview of the flow and main computational parts of the motor layer and overall reinforcement learning integration. This is the core part to generate voluntary behavior.

4.3 Implementation

The software system is implemented using NUPIC, Numentas Platform for Intelligent Computing. It is an open source machine learning platform implementing the HTM algorithms in python and C++. Additionally to the published code for common components such as the standard Spatial Pooler and Temporal Memory, they share experiments in an open research manner. This allowed the use of experimental components from the htm-research repository such as the Union Pooler, Extended Temporal Memory which includes basal and apical connections and encoders. In general, the attempt was to use the Network API as much as possible to have a simplified, scalable and reusable design that integrates with the existing framework. However, custom components on a region and algorithmic level had to be designed for the reinforcement learning calculations and extended functionality such as the motor generation layer.

Besides the motor module, every layer consists of a pooling part, being either the Spatial Pooler or the Union Pooler for temporal pooling, and a Temporal Memory that supports basal and apical dendrites if needed. Additionally, the D1 and D2 layers have a modified version of Temporal Memory that includes the reinforcement learning functionality to keeps track of the state values, eligibility traces and calculate the TD error.

The pseudocode of the agent system is given in Algorithm 5. After initializing the remote environment, it checks if there is a serialized version of the network on the file system which could be loaded. If not it creates a new Network from scratch.

The environment is running in real time, so we update our states each iteration and process the observation and reward to submit an action. It resets all layers if a reset signal from the environment was received indicating the new characters are shown. This happens after the time expired of the agent did a click. In both cases, it will have received a reward in the observation prior. In theory, it increases performance and learning for the agent to use resets as otherwise, it will interpret the current challenge in the context of the last one, accumulating the information and learning one long sequence.

We update the sensor values that are encoded and passed to layer 4 initially. Then every layer is processing its input in the order of the input-output linking between the regions defined in the Architecture Section 4.2.

When the state of layer 5 is calculated it can be processed in the reinforcement regions D1 and D2. They calculate the TD-Error, update the state values and eligibility traces according to the algorithm design shown

Algorithm 5 Agent System Pseudocode

```
INITIALIZEENVIRONMENT( )
if !savedNetwork then CREATENETWORK( )
elseLOADNETWORK( )
end if
for each Observation, Reward in EnvironmentStates do
    if newCharacters then RESETACTIVITY(allLayers)
    end if
    UPDATESENSEDVALUE(Observation, Reward)
    for each layer in layers{L4, L3, L2, L5} do
        SPATIALPOOLER(layer)
        TEMPORALMEMORY(layer)
    end for
    for each layer in layers{D1, D2} do
        SPATIALPOOLER(layer)
        TEMPORALMEMORY(layer)
        UPDATEELIGABILITYTRACES(layer)
        CALCULATESTATEVALUE(layer)
        CALCULATETDERROR(layer)
        UPDATESTATEVALUES(layer)
    end for
    winnerCells ← MOTORLAYERCALCULATE( )
    Action ← GETACTION(winnerCells)
    UPDATEENVIRONMENTSTATES(Action)
end for
```

in Section 4.2.4.

Lastly, the motor layer can determine the active winner cells for this iteration that determines the action and send to the environment to update the current state. The motor layer pseudocode is given in Algorithm 6.

Algorithm 6 Motor Layer Pseudocode

```
function MotorLayerCalculate( )
    CopyActiveColumns(L5)
    CopyBasalConnnections(L5)
    ApicalLearningD1(TDError)
    ApicalLearningD2(TDError)
    RandomlyInitializeMotorCells(layer)
    D1Apical, D2Apical ← CalculateIntersection(Apical, Basal)
    ExciteMotorCells(D1Apical)
    InhibitMotorCells(D2Apical)
    winnerCells ← SelectHighestActivations(MotorCells)
    return winnerCells
end function
```

It copies the active columns and basal depolarized cells from layer 5 to use them for learning the apical connections and calculating the intersection with apical depolarized cells respectively. It will use the TD-Error to update the permanence values of apical connections to D1 and D2, including growing new synapses and segments if necessary.

In the next step, it is ready to calculate the excitation level for the different motor cells. Therefore it first initializes them all randomly, such that random behavior is produced to explore the environment in case the agent did not learn any voluntary actions. The intersection from basal and apical depolarized neurons builds the basis of neurons that will get excited or inhibited depending on the origin of the apical connections. In the last step, it calculates the k-winner cells that represent the motor action for this iteration. They will be continuously mapped to the discrete action space in the agent system.

Equation 14 shows the mapping of a winner cells index to the corresponding action space. The span of the winner cells corresponds to the number of motor cells. The action span corresponds to the 6 constant actions as mentioned in Listing 4.1. The action which has most associated winner cells will be submitted from the agent, and ties are broken randomly.

$$actionIndex = round((winnerIndex+1/winnerSpan)*actionSpan)-1 \tag{14}$$

4.3.1 Visualization and Debugging

To experiment and debug the implementation continuously a visualization and output from the network while running needs to be observable. NUPIC does not provide very advanced visualization tools, and in general, it is challenging to visualize artificial neural networks due to the sheer complexity of nodes and connections. Additionally, in HTM the connections can be of different kinds and are divided into segments for each cell. In this study, basic tools were used to track the state of the network during run-time.

The layer state and individual computation features are written directly into the terminal. This includes among other features indices of active cells, predicted cells, TD-Error, and motor activation. It highlights more relevant information such as the depolarization of motor neurons in yellow or red for improved readability.

Additionally, neuronal activations of the layers can be plotted with matplotlib. It slows down the computational process significantly but enables a better understanding of the online learning from the network. Figure 4.23 shows exemplified 3 regions neural activation and the sensor input that layer 4 receives. During the development process, any region could be displayed. In the neural activations, it is easy to see that the bulks represent a column bursting while more fine-grained patterns are coming from predicted neurons. This can help to see how far an agent is anticipating the next state already and which sensor input might change the activation significantly.

4.3.2 Serialization and Infrastructure

For sufficient exploration and reproducibility, the agents' network state needs to be able to serialize and deserialize. NUPIC uses capnp, a C++ serialization library with its python extension. However, it is only implemented for specific algorithm components and not on the network level. The serialization got extended for all custom and experimental regions used, such as the Reinforcement layer, Temporal Pooler or layer 5 except the Motor layer. This is due to missing serialization support in a crucial base component responsible for managing the connections. However adequate testing of the serialization process to ensure total equivalence from the state after loading to the saved network was out of the scope of this thesis. Thus the serialization was not used in any of the experiments in this study but could help any future attempts.

In the late stage of the project, the implementation and dependencies

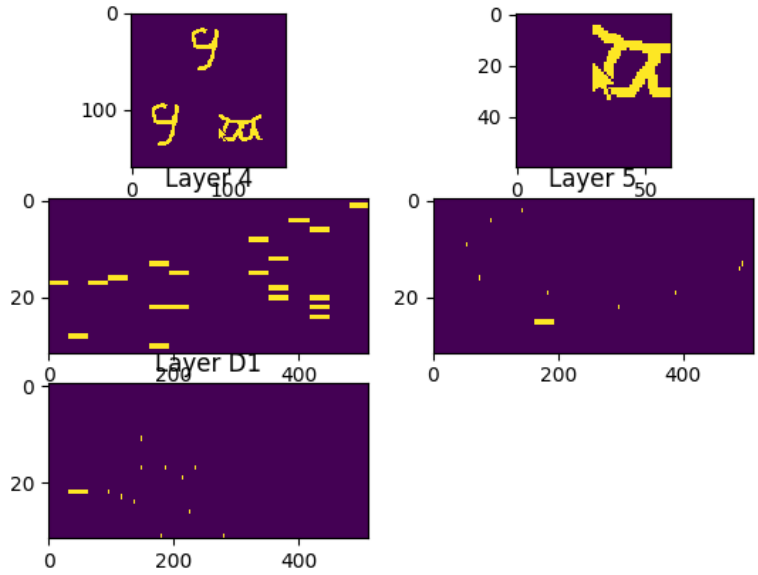

Figure 4.23: An exemplified picture of the implemented network plotting using matplotlib. It shows the activations and pixels as black and white matrix. On top the sensor region is displayed in two parts, which shows the agents perceptual field and input to layer 4. The other regions are layer 4, layer 5 and a reinforcement region.

were bundled into a docker image to ensure it can be reproduced on any platform. This also allows much easier experimentation by training in a cloud or running multiple instances in parallel. It was tested at the end of the project even though the code was not optimized for parallelism. This made it possible to run instances in Docker containers in the university cloud and observe the experiments via the VNC daemon that is created by the environment. Figure 4.24 shows an easy to manage interface and access via VNC to all running experiments in the cloud. Each experiment is creating a Docker container for the agent and for the environment. Theoretically, it could have been optimized to have a single agent container running multiple agents in parallel.

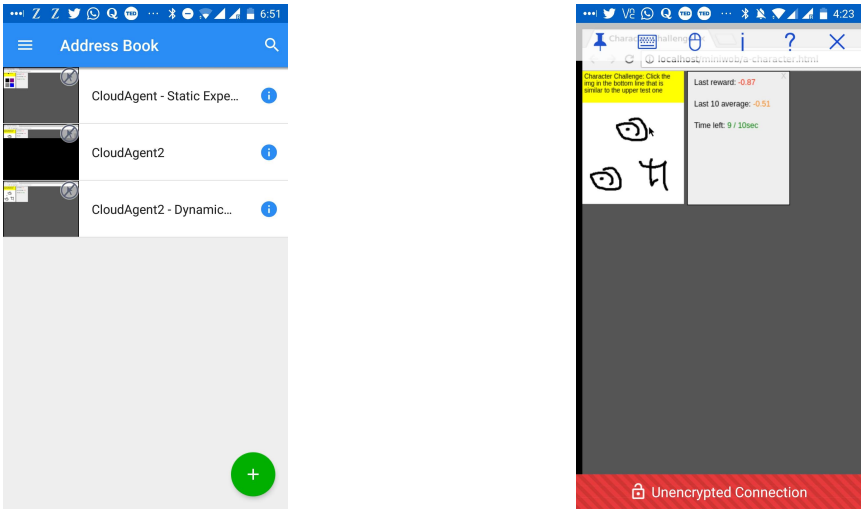

Figure 4.24: On the left a list of launched experiments is shown. On the right the view of the experiment when connected via VNC to observe its current state.

5 Results

In this study, many different configurations were used for experimentation and analysis of the agent implementation and architecture. Main components and parameters that were modified during the experimentation process include:

- Encoder: features, weight, radius
- NUPIC Layer parameters: permanence increment and decrement, threshold.
- Motor Layer parameters: number of motor neurons and number of winner cells
- TD-Parameters: trace decay, discount, learning rate, the calculation formula
- Reset: reset layer states at sequence ends

There are many more parameters that could have been considered but exceeded the scope of this project. They include for example a color encoding or the use of topology from pooling the input layer.

In this section, an overview of the results of the two experiments is given. The most important parameters that vary from the standard configuration are described, and the full list of default parameters is in the appendix.

5.1 Static square experiment

This section includes the results for the static square experiment as described in Section 3.2. In the experiment, the mouse cursor was not reset at the end of a sequence, and layer states were not reset either. The agent could observe a static environment changing only due to its generated behavior. Also in this experiment layer 2 and 3 were excluded as temporal abstractions were not required. All other parameters have the default values which can be found in the appendix.

The first Graph 5.25 shows the moving average over the accumulated non-zero reward of the agent in the 160,000 iterations it interacted with the environment. It received a higher reward for solving a task faster. First, the distribution is very volatile, but after about 50,000 iterations it is close to 1. Between iteration 18,000 and 35,000, it shows a stagnation at -1. The

minimum reward was -1, but it could exceed this value if the connection slowed down and the environment accumulated multiple negative rewards.

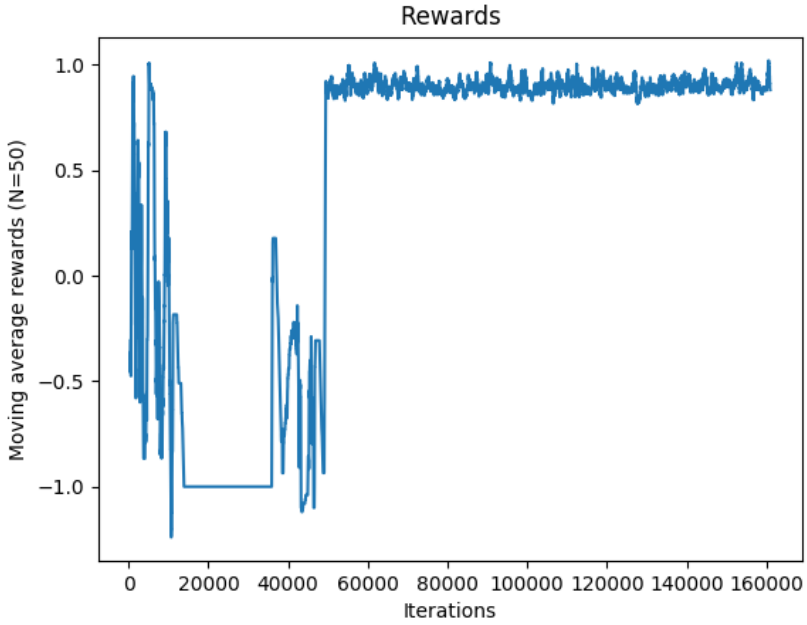

Figure 5.25: The moving average over 50 instances from the accumulated rewards from the agent.

Figures 5.26 and 5.27 show the neural activation and correct predictions in the layers 4,5 and D1. In general, the activity is less in the higher layers as the Spatial Pooler filters to a higher sparsity. However, the graphs of the different layers are correlated and following the same trend at most iterations. After 50,000 iterations the number of active neurons stagnates.

The last two Graphs 5.28 and 5.29 it shows the Temporal Difference Error and the voluntary active motor neurons respectively. Former is very volatile and growing at the beginning and then approaches zero after 50,000 iterations. Latter shows the same pattern as the number of active neurons fluctuates but stalls after 50,000 iterations.

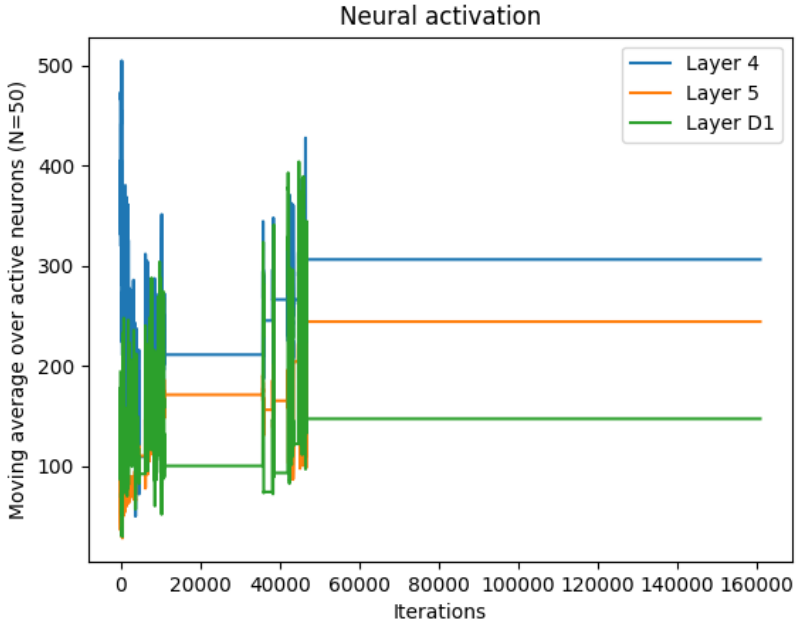

Figure 5.26: The moving average over 50 instances from active neurons in the layers 4,5 and D1.

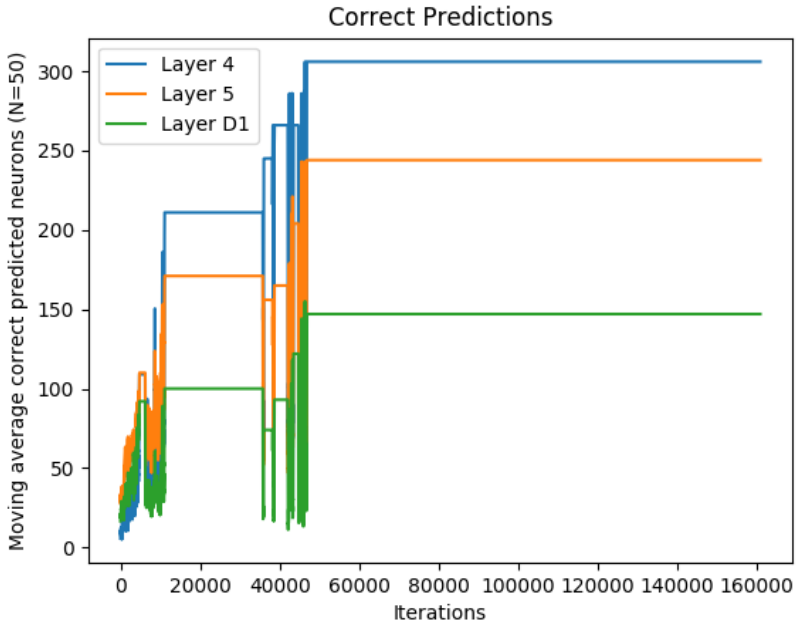

Figure 5.27: The moving average over 50 instances from active, predicted neurons in the layers 4,5 and D1.

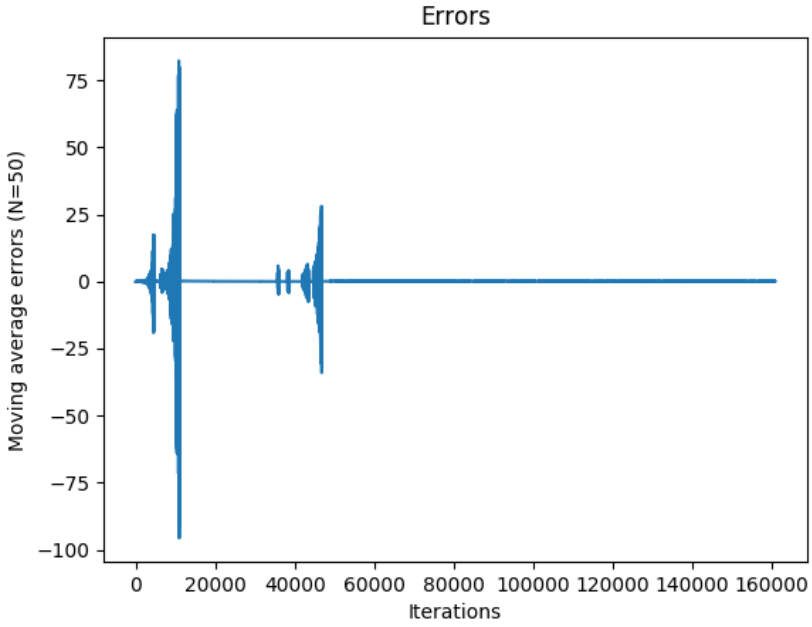

Figure 5.28: The moving average over 50 instances of the TD-Error calculated in Layer D1.

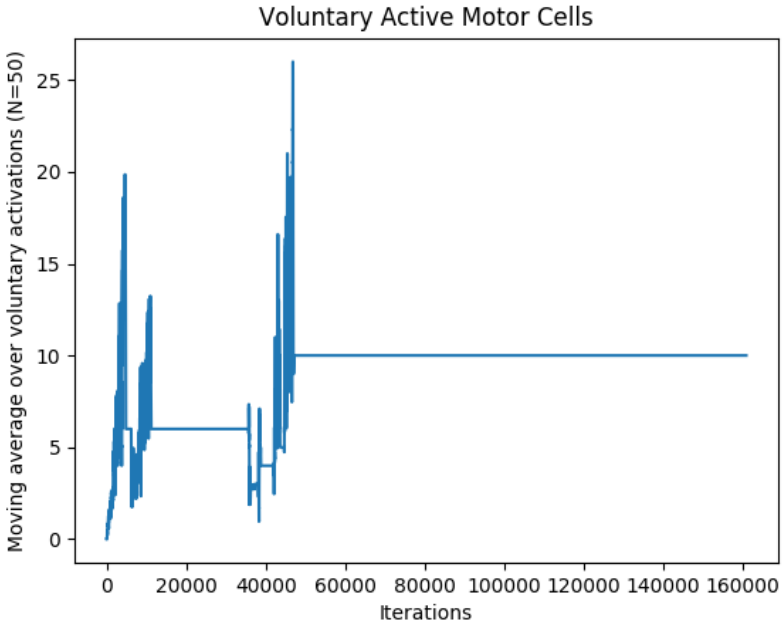

Figure 5.29: The moving average over 50 instances from voluntary active neurons in the motor layer through basal and apical depolarization.

5.2 Dynamic character experiment

Following are the results for the dynamic character experiment described in section 3.1. The configuration included the abstraction layers 2 and 3 with the Temporal Pooler component. It does also not reset any layer states or the mouse cursor. In the experiment, the TD-Parameters were set higher as this resulted in faster learning and enhanced updating of previous states that led towards the reward.

- TDTraceDecay (Lambda) : 0.2
- TDDiscount : 0.2
- TDLearningRate : 0.2
- globalValueDecay : 0.0001

Figure 5.30 shows the moving average of accumulated rewards. It is very volatile in the 250,000 iterations and does not yield towards a value. The average was taken over 1000 instances to account for the number of iterations and filter extremes.

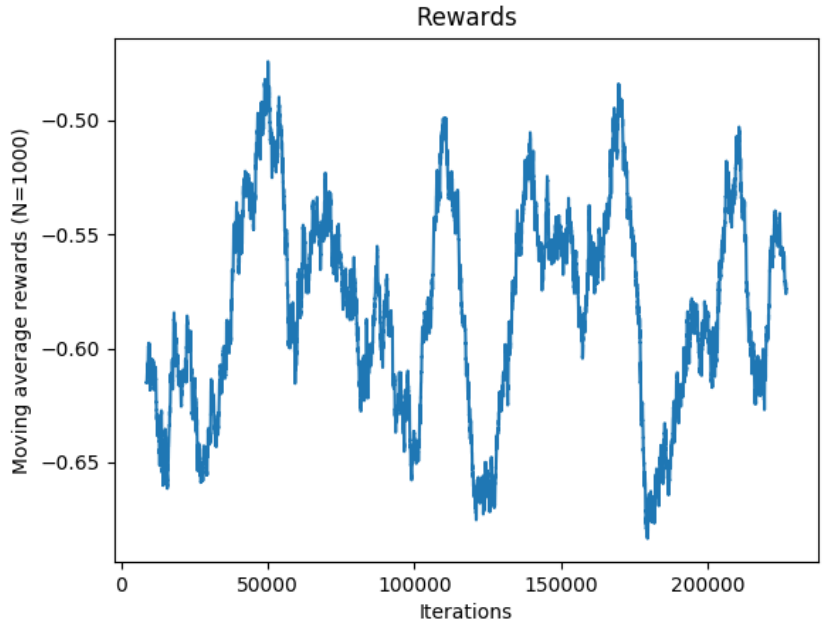

Figure 5.30: The moving average over 1000 instances from the accumulated rewards from the agent.

Figure 5.31 displays the moving average over 1000 instances of the TD-Error. It is very volatile and continuously switching from negative and

positive values. It grows exponentially towards infinity, and the measurements of beginning iterations seem to be around zero as the difference is much smaller but follow the same pattern.

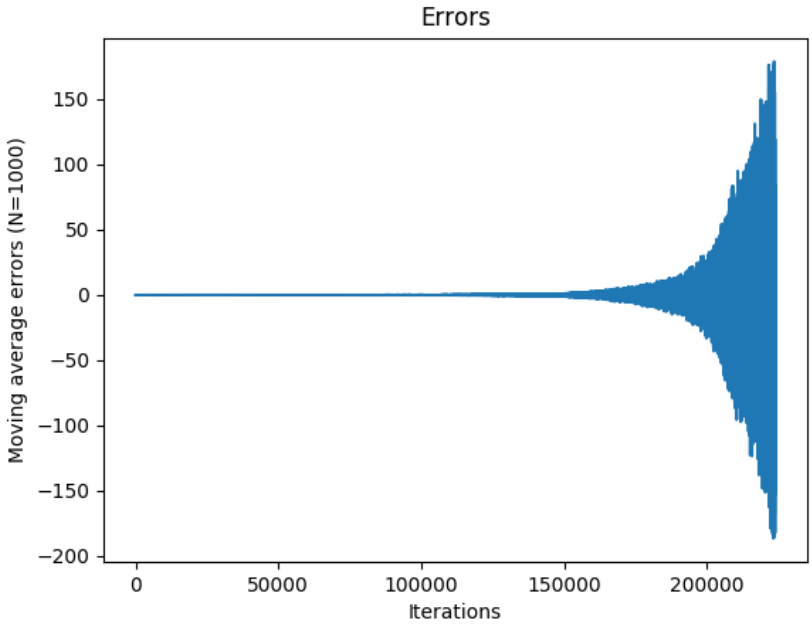

Figure 5.31: The moving average over 1000 instances from the TD-Error calculated in Layer D1.

In Figure 5.32 we can observe a deficient activation with a maximum average value of 3 overall 250,000 iterations. The graph follows a positive trend until the number of voluntary active neurons shrinks again after 65,000 iterations.

The last two Gaphs 5.33 and 5.34 present the neural activation and predicted neural activation in layers 4,5 and D1 respectively. The activity is unstable and does not follow a clear trend. The correct predictions are very low with a maximum average value of 18. In contrast, the minimum average number of active neurons is 330. The correct predictions of layer 5 and D1 initially follow a light trend upwards but seem to stall around 15.

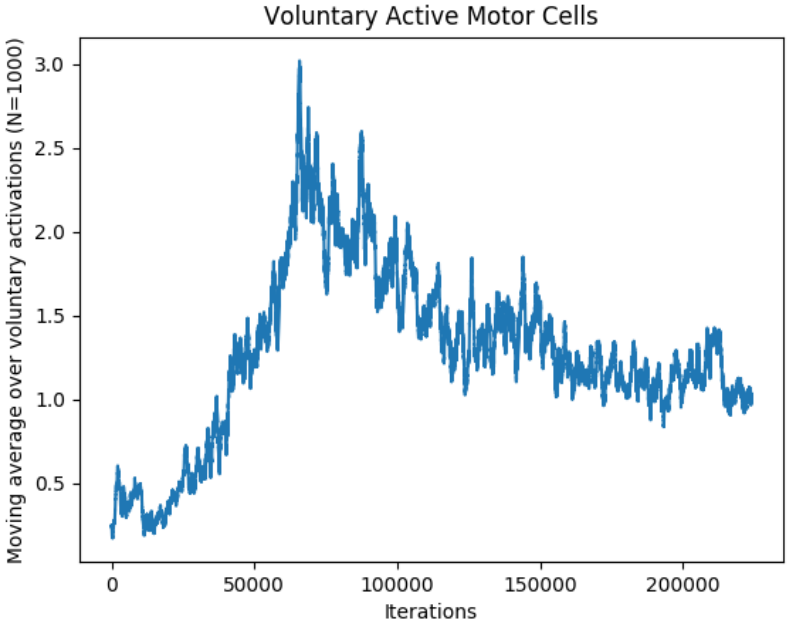

Figure 5.32: The moving average over 1000 instances from the voluntary active neurons in the motor layer depolarized by basal and apical connections.

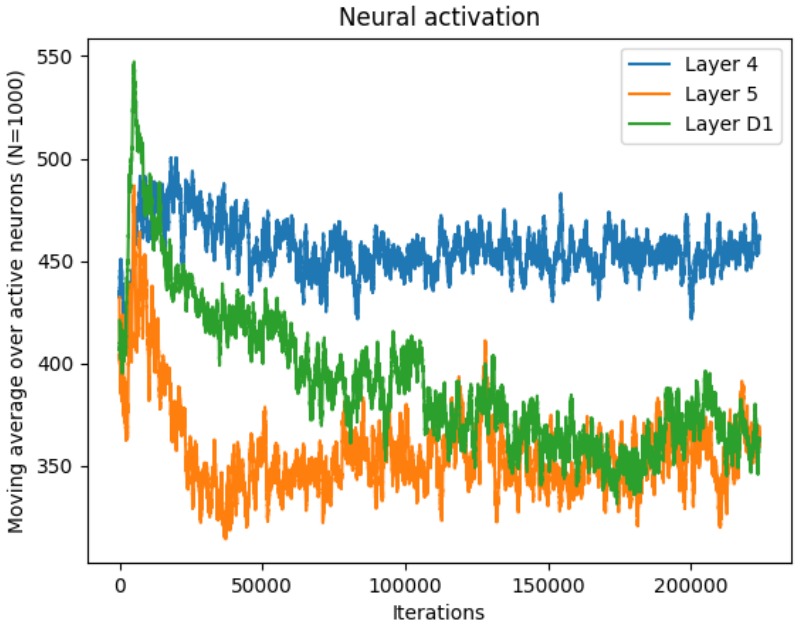

Figure 5.33: The moving average over 1000 instances from the active neurons in layers 4,5 and D1.

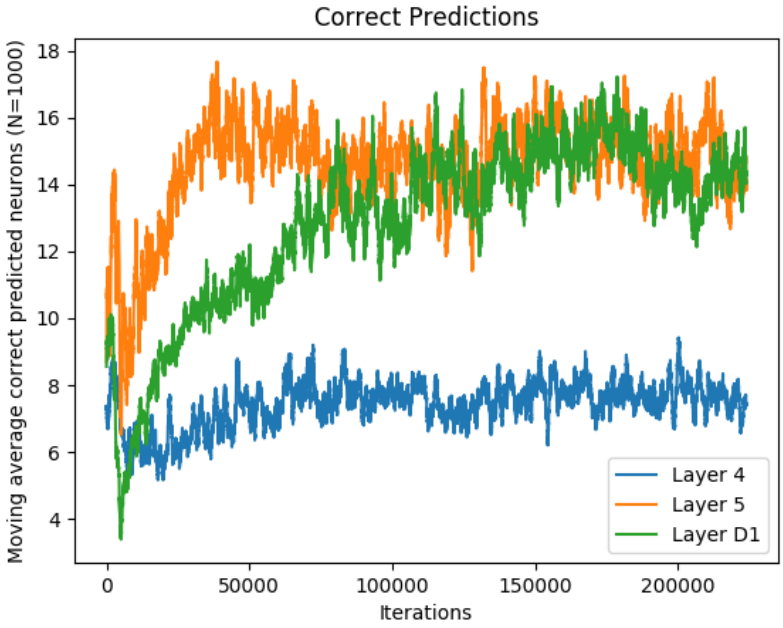

Figure 5.34: The moving average over 1000 instances from the active, predicted neurons in layers 4,5 and D1.

6 Analysis

The analysis is divided into a general part that is concerned with the architecture and a part that evaluates how the agent learns in the different experiments.

6.1 General Analysis of the Learning Agent

During development and experimentation, there have been many observations of how the theory relates to the actual practical results. The most important ones are described in the following paragraphs.

Interpretation and correlation of results On the static experiment the neuron activity was more accessible to interpret and observe as the state only changed due to generated behavior. As shown in Figure 4.21 the agent needs to learn a stable representation of the state changes in L5, D1, D2 and which actions led to which state. This can be well observed in the number of correct predictions in this layers that raises as shown in the results Figure 5.27. Layer 4 and 5, D1 are highly correlated as they are learning Layer 5 state transitions from the same input source of layer 4 activation. The number of correct predictions is logically also highly correlated with the number of active neurons, due to bursting. If a column has no predicted neurons, it will burst and results in more activity, which is a satisfactory indicator to examine the learning progress of the network building up a model of the environment changes. The TD-Error is very volatile and growing fast at the beginning when the state changes are high due to less correct predictions and more bursting columns. However as the state is relatively stable, it is closer to zero, especially in the static environment as the agent only exploits and accumulates expected rewards.

Specific neurons to distinguish states In the static task, the mouse cursor can be in any of the four different squares and the difference in the state is mostly made out of the different encoding from the coordinates, as the agent does not perceive the image detailed enough to detect the mouse position on the full image. Thus the agent has to learn particular active neurons that distinguish the specific state to represent the voluntary activation and generate the desired motor behavior. Dependent on which square it is in it has to either click or move the cursor. As the initial connections are entirely random, this learning process is quite complicated and needs many iterations to learn the correct connections. The speed of adaptation

and learning can be influenced by TD, threshold and permanence parameters. This complexity of learning the mapping of actions and state transitions requires even more computation when the environment is dynamic and changing not only with agent interactions. The results of the experiment with dynamic environment show clear limitations of the agent to distinguish between intrinsic generated environment changes and extrinsic sequences. In the static experiment with reset, the agent learns successfully to be randomly initialized by the environment. However, this is much simpler than learning a randomly changing character representation that determines the outcome.

Encoder determines perception One of the most important parts of the system is the encoder. It determines what the agent perceives from the environment. The output from the encoder is used in the Spatial Pooler of layer 4 which makes an abstract representation of the input with the goal to preserve the semantic features. However, the encoder determines the semantic features of the data by how many input bits are active and how many total bits represent the input. In case the whole image is encoded in black/white pixels, detailed, complex structures such as the mouse cursor or a character structure are not represented meaningfully in the abstraction. In the results, the agent can not even perceive a change of the mouse cursor. Thus the encoding needs to be task specific and in the case of a human-like challenge somehow account for detailed focus and attention mechanisms that an eye has. The encoder used 25% for the coordinate representation, 25% for a more focused radius around the mouse cursor and 50% for the full image representation. This enabled the agent to distinguish between states where the mouse cursor is in different positions but still remained difficult to differ between states of complex structures such as omniglot characters, especially when they are not in the focus radius.

Association with action space A requirement to generate benevolent actions is a learned correct mapping of the actions to the state they produce. As all action-state mappings are randomly initialized, and the action space randomly explored the size of the action space is influential. In this study, a discrete action space with 6 possibilities was defined and a uniform distribution of the actions. This is a relatively small number, but the agent has to explore the mapping from most possible state transitions dependent on the actions. This results in many possible combinations, especially when high order sequences in layer 2 and 3 are turned on and distinguish states partly

by the past active states. Even more, if the environment is dynamic and changes unrelated to agent actions. So the agent has to learn a mapping to every possible probabilistic outcome from a state and action. However, the described difficulty to distinguish between very similar states by individual neurons is also the strength to generalize them and possibly transfer behavior that applies to two similar states. This reduces the complexity to learn different actions for every possible state as similar ones share much overlap in neuron representation, which can help to learn the desired behavior faster if it leads to a similar outcome.

High order sequences In the experiments with a dynamic environment, the agent needs to learn to click the correct bottom row character dependent on the character is shown in the upper row. It does require movement of the cursor towards one side and then possibly click. Multiple steps that require associating the intermediate steps with the final reward. This is inherent in TD Lambda dependent on the trace discount parameter previous state values will be updated for their influence on the final reward. The higher level sequences as shown in the results can improve this effect as they make it possible to distinguish states in Layer 5 dependent on past activity. This is a generalization process as the union of the states in the Temporal Union Pooler, which is filtering for more recent activations through applying a linear decay, is not representing the order of the previous activations.

6.2 Results on experiments

In the static experiment, the agent learns successfully the mapping of an action and the state it results in. The correct predictions are on a constant high level once it is exploiting the learned model. In the initial phase it is exploring the environment and learns the different state transitions along with the reward it receives. This is also indicated by the volatile TD-Error which switches with unexpected state changes. The voluntary motor activation gives an indicator how much the agent is attempting to influence the state towards a possibly beneficial one by inhibiting and exciting motor neurons.

In contrast in a dynamic environment, we can observe a much larger volatile phase and the agent has problems to identify patterns between its activity and the resulting state. It is a much more difficult task to learn the mapping as the state change is also influenced by random environment parameters, each time the top character changes. The correct predictions

following a positive trend but are much lower than in the static experiment. Especially as layer 4 is not able to predict the changes within the context information of the previous motor command. Layer 5 is integrating the new activation with the previous layer 4 values but can not create a stable representation. Since the state transitions are not learned, the agent is unable to produce any benevolent behavior and stagnates exploring the environment. The TD-Error grows continuously towards infinity as the state values are increased and changing frequently and unexpected.

7 Discussion

The discussion will be divided into two parts, the general architecture of a reinforcement agent with HTM technology and the results in the context of the list of ingredients for intelligent machines.

7.1 A Reinforcement Agent with HTM

The list of related works given in Section 1.5 is short, and this study attempts to extend and explore the framework proposed by Kaan to go one step further towards the integration of reinforcement learning into HTM theories. It would extend the functionality of HTM systems by far as it could be applied to problems besides anomaly detection. It is exciting in the light of the sensorimotor theory integration that Numenta is currently working on. Critics of Numenta raise the issue frequently that their algorithms do not scale on any benchmark problems. However, the research of Numenta is on a fundamental level and has to be interpreted in this context. It has several very promising characteristics such as it is learning in real time online with continuous adaptation to a changing environment, has the temporal component as crucial component inbuilt and is using sparse representations that inherently form a possibility for generalization by having overlapping bits for semantically similar features. These properties are very interesting for building a reinforcement agent, that learns in a continuous environment and has to generalize its knowledge base for possible one-shot recognition.

Nonetheless, the promising characteristics, the results of this study confirm the previous experiments that the current architecture does not scale for more complex tasks. One shot recognition is a severe problem that requires an agent to incorporate various of the properties for a more general intelligent system. It requires the agent to generalize its knowledge and build a causal model to understand the relationship between its action and the outcome dependent on the state of the environment. The experiments show that it is possible to build an agent based on HTM which is capable of learning simple tasks, but it will need further exploration and a more meaningful encoded model of the environment to scale for more complex problems.

7.2 Targeting goals: List of Ingredients

In the paper "Building Machines That Learn and Think Like People" from Lake et al. they describe criteria and challenges towards the goal of gen-

eral intelligent machines [14]. Summarized they emphasize the following criteria:

1. **Causality**: build causal models of the world that support explanation and understanding, rather than merely solving pattern recognition problems
2. **Intuition**: ground learning in intuitive theories of physics and psychology, to support and enrich the knowledge that is learned
3. **Compositionality & Transfer Learning**: Harness compositionality and learning-to-learn to rapidly acquire and generalize knowledge to new tasks and situations

They argue that human-like learning and decision making does rely on rich internal models, perception, and learning must be informed and constrained by prior knowledge and for cognitive inspiration in artificial intelligence.

In their commentary, the Google Deepmind team explains why an approach with minimal human hand engineering and a focus on autonomy has the highest chance to scale success in this challenges [1]. So while they agree on the identified desiderata, they think autonomy is the right way to approach it. This study tried to relate to their discussion and the peer commentary from the AI community and emphasized the following criteria as targeting goal for an architectural design:

- Compositionality
- Causality
- Transfer Learning
- Autonomy

The idea of rich internal models, as well as cognitive and neural inspiration, are aligned with the ideas of Numenta. However, it has to be emphasized that HTM theory is yet still very much in an initial phase and not as far developed and scalable as many deep learning technologies. Thus the discussion around the list of ingredients is more thought of having the right fundamental approach to go further instead of being applicable and complete to the designed architecture.

The model they presented for the omniglot character challenge got often criticized for incorporating too much domain knowledge in their design,

for example about drawing characters and their composition. The experimental setup attempts to align with the challenge presented but focusing on autonomy to remove a possible bias in the design towards a specific domain. The agent learns solely from the observations and the action space it operates in and tries to build an internal model of the challenge it is facing. The reinforcement learning rules let it learn voluntary behavior which comes to a more favorable outcome - a higher reward.

As described in the analysis, the SDR properties inherently support a generalization of concepts based on similar semantic features. In theory, this can be utilized by the agent to generalize the actions taken in similar environment states or to reuse learned connections of the shared neurons for transferring knowledge. However, this system property is only applicable when the encoding of similar environment states results in similar SDRs and different states indistinguishable ones. Often domain knowledge is integrated into the encoder to bypass this difficult problem. This should be avoided and instead a more general way is needed to form meaningful SDRs where the overlap property can be exploited. Recent research from Numenta is attempting a spatial invariant object representation formed by each cortical column using a grid cell-like location and orientation integration. Furthermore, this study only used one cortical column, and it was out of scope to experiment with any hierarchical networks that have multiple cortical columns voting on a shared state of the environment. This could also result in a more detailed representation of the environment as the cortical columns can filter on different scaling levels. This study did also not experiment with any topology encoding. As the input is highly spatial with 2D observational data, this would also be an option to introduce a better encoding of the environment.

Nevertheless, the criteria from Lake et al. are thought of in the HTM algorithms and are partly inherent by design. Causality is at the heart of the temporal dimension of the system, Compositionality is supported by the union and overlapping properties of SDRs and inbuilt to sparse representations and transfer learning, and generalization is in open research for a temporal pooling layer which forms an abstract representation of a sequence independent of time. In its current form, the spatial dimension for invariant representations is not much developed in HTM and is required for composition and generalization of features. It is the main area of research at Numenta currently to integrate an allocentric location signal on a cortical column level, and future studies will show if it helps to incorporate the characteristics on a system level that an agent can utilize them.

8 Conclusion

This study set out to explore a novel combination of HTM and RL to build a software agent in the context of the widely discussed paper of Lake et al. about Building Machines that Think and Learn like Humans [14]. The paper was chosen purposely to have an intersection with the broader machine learning community and encourage a discussion about the theories developed at Numenta, which are alternative to current deep learning techniques. It was a very ambitious project, and thus clear limitations were made to account for the short duration.

In the project a software agent in Numentas open source platform NUPIC was built, that combined HTM with Reinforcement learning. The neural architecture was based on Kaan's exploratory design for an autonomous NPC agent [17]. It demonstrated how reinforcement learning techniques of model-free prediction and control can be integrated into the unsupervised, online learning algorithm of Numenta to create an agent capable of solving simple tasks. It analyzed the architecture and examined the results to gather insights of the agents learning.

On the other hand, it revealed limitations of the design to be applied to more complex challenges and dynamic problems. It investigated main obstacles for the agent on this challenges, which included a deficit in sensing the important details, generalizing spatially and making temporal abstractions. It also proposed a possible mechanism for improvement of these shortages that can be explored in future research.

Following the objective to create reusable software and tools to enable future research, the study produced a valuable setup for any future exploration of a combination from RL and HTM. With NUPIC and the Universe environment from OpenAI, it used open source software to ensure compatibility and easy adaptability for people coming from other machine learning areas. The final implementation enables researchers to merely create an experiment in a combination of Javascript, HTML, and CSS to switch the custom task of an agent. Furthermore, all code is packaged into a docker image and can thus be run in a typical cloud infrastructure. The VNC connection of an agent to the environment allows observing all experiments in real time without any other complex hardware or software needed than an orderly mobile phone. NUPICs network API gives an abstraction layer that makes it effortless to change connections of layers or add and remove them. This framework can be helpful for anyone that attempts further research of an agent with HTM technologies.

The experiment and agent were designed with Lake et al. criteria list

and one shot character recognition challenge in mind. It did not attempt a full analysis of the integration of the criteria in an HTM system, but it related and interpreted how most of them are thought of in the very design of their algorithms. The agent could not demonstrate properties such as an advantage through transfer learning or composition, but the analysis showed that sparse representations are theoretically very supportive of it. However, the main obstacle is the correct encoding to create meaningful representations that preserve the SDR properties and to predict neural activation based on the correct context information. Here the research of Numenta regarding the integration of an allocentric location signal and a hierarchically structured network with multiple cortical columns are fascinating to explore further.

8.1 Future work

Developing the architecture and experimentation in NUPIC was challenging, most components are in an experimental stage and not supported, tools are missing for detailed analysis on a network layer and the experiments had to be formulated and realized. Additionally, the environment build with Universe from Open AI became deprecated and support shut down during the project. This required the development of many components, refinement of faulty components and composition of infrastructure. An intensive theoretical study initially and a late start of the implementation postponed a final usable system implementation to the end phase of the project, which resulted in clear time limitations for explorations. Important parts that were not realized in the software and would have the potential to exaggerate the experimental process including:

- **Serialization**: Saving and loading the network with the validated integrity of the systems state boosts the experimentation process enormously.
- **Player guided exploring**: For complex problems, the initial random phase is taking the most exploration time to build a model of the environment and create benevolent behavior from scratch.
- **Computational Power**: As shown by popular deep reinforcement learning experiments and discussed in the analysis section, exploring the environment and possible state transitions takes much time and computational power.

- **Initial Implementation**: The architecture of this study was based on Kaan's design, but there was no previous open source implementation of such or any other reinforcement learning agent in NUPIC. An initial implementation to start from would have fastened development and understanding.
- **Visualization tools**: Neural networks have the inherent property to be hard to visualize. In HTM each neuron can have three different states and types of connections. Simplistic tools for analyzing the network were used, more advanced visualization tools could have improved the exploration process drastically.

Besides the shortcomings in the software implementation, the analysis showed many theoretical aspects that have to be further explored. They included the design of the encoder, the integration of a location signal for spatial representations, a hierarchical network structure with multiple cortical columns and the design of a temporal abstraction component. Numenta is working on several of them to extend their theories, such as temporal pooling that could make an abstraction of a sequence in time and send a reset signal in case a new sequence is identified. However, these ideas are not matured enough and need further research to be integrated with a possible reinforcement learning agent.

References

[1] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Ingredients of intelligence: From classic debates to an engineering roadmap," *Behavioral and Brain Sciences*, vol. 40, p. e281, 2017.

[2] t. E. C. i.-h. t. t. European Political Strategy Centre (EPSC), "The age of artificial intelligence - towards a european strategy for human-centric machines," European Political Strategy Centre, Tech. Rep., 2018, accessed: 18/05/2018. [Online]. Available: https://ec.europa.eu/epsc/sites/epsc/files/epsc_strategicnote_ai.pdf

[3] P. Hong, D. Alarcón, M. Bruckner, M. LaFleur, and I. Pitterle, "The impact of the technological revolution on labour markets and income distribution," *FRONTIER ISSUES*, 2017. [Online]. Available: https://www.un.org/development/desa/dpad/wp-content/uploads/sites/45/publication/2017_Aug_Frontier-Issues-1.pdf

[4] O. Churchill, "China's ai dreams," Nature, Tech. Rep., 2018, accessed: 18/05/2018. [Online]. Available: https://www.nature.com/articles/d41586-018-00539-y

[5] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016. [Online]. Available: http://arxiv.org/abs/1609.03499

[6] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, "CNN-RNN: A unified framework for multi-label image classification," *CoRR*, vol. abs/1604.04573, 2016. [Online]. Available: http://arxiv.org/abs/1604.04573

[7] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *Neural Computation*, vol. 28, no. 11, pp. 2474–2504, 2016, pMID: 27626963. [Online]. Available: https://doi.org/10.1162/NECO_a_00893

[8] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134 – 147, 2017, online Real-Time Learning Strategies for Data Streams. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231217309864

[9] A. Lavin and S. Ahmad, “Evaluating real-time anomaly detection algorithms - the numenta anomaly benchmark,” *CoRR*, vol. abs/1510.03336, 2015. [Online]. Available: http://arxiv.org/abs/1510.03336

[10] J. Hawkins, S. Ahmad, and Y. Cui, “A theory of how columns in the neocortex enable learning the structure of the world,” *Frontiers in Neural Circuits*, vol. 11, p. 81, 2017. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncir.2017.00081

[11] S. Ahmad and J. Hawkins, “Untangling sequences: Behavior vs. external causes,” *bioRxiv*, 2017. [Online]. Available: https://www.biorxiv.org/content/early/2017/09/19/190678

[12] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “Human-level concept learning through probabilistic program induction,” *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015. [Online]. Available: http://science.sciencemag.org/content/350/6266/1332

[13] C. E. Perez, “The many tribes problem of artificial intelligence,” MEDIUM, Tech. Rep., 2017, accessed: 18/04/2018. [Online]. Available: https://medium.com/intuitionmachine/the-many-tribes-problem-of-artificial-intelligence-ai-1300faba5b60

[14] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, “Building machines that learn and think like people,” *CoRR*, vol. abs/1604.00289, 2016. [Online]. Available: http://arxiv.org/abs/1604.00289

[15] M. Otahal, “Architecture of autonomous agent based on cortical learning,” 12 2013, accessed: 15/04/2018. [Online]. Available: https://www.researchgate.net/publication/261699568_Architecture_of_Autonomous_Agent_Based_on_Cortical_Learning

[16] A. S. Gomez and J. Shapiro, “Hierarchical temporal memory as a reinforcement learning method,” 2016, accessed: 15/04/2018. [Online]. Available: http://studentnet.cs.manchester.ac.uk/resources/library/3rd-year-projects/2016/antonio.sanchezgomez.pdf

[17] A. K. Sungur, “Hierarchical temporal memory based autonomous agent for partially observable video game environments,” accessed: 13/04/2018, Version 0.5. [Online]. Available: http://etd.lib.metu.edu.tr/upload/12621275/index.pdf

[18] W. L. Hosch, "Machine learning," Encyclopædia Britannica, inc., Tech. Rep., 2016, accessed: 09/05/2018. [Online]. Available: https://www.britannica.com/technology/machine-learning

[19] P. Dayan and Y. Niv, "Reinforcement learning: The good, the bad and the ugly," *Current Opinion in Neurobiology*, vol. 18, pp. 185–96, 09 2008.

[20] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick, "Neuroscience-inspired artificial intelligence," *Neuron*, vol. 95, pp. 245–258, 07 2017.

[21] M. Malik. (2017) Accessed: 09/052018. [Online]. Available: https://hackernoon.com/machine-learning-in-brain-120d375eccd1

[22] D. S. A. A. R.-J. V. M. G. B. A. G. M. R. A. K. F. G. O. S. P. C. B. A. S. I. A. H. K. D. K. D. W. S. L. . D. H. Volodymyr Mnih, Koray Kavukcuoglu, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529–533, 2015.

[23] D. O. Hebb, "Hebb, d. o. organization of behavior. new york: Wiley, 1949, pp. 335, $4.00," *Journal of Clinical Psychology*, vol. 6, no. 3, pp. 307–307, 1950. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/1097-4679%28195007%296%3A3%3C307%3A%3AAID-JCLP2270060338%3E3.0.CO%3B2-K

[24] A. S. Hawkins, J. and D. Dubinsky, "Cortical learning algorithm and hierarchical temporal memory," 2011, accessed: 25/03/2018, Version 0.2.1. [Online]. Available: http://numenta.org/resources/HTM_CorticalLearningAlgorithms.pdf

[25] S. Ahmad and J. Hawkins, "Properties of sparse distributed representations and their application to hierarchical temporal memory." *arxiv*, 2015. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1503/1503.07469.pdf

[26] ——, "How do neurons operate on sparse distributed representations? a mathematical theory of sparsity, neurons and active dendrites." *arxiv*, 2016. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1601/1601.00720.pdf

[27] Y. Cui, S. Ahmad, and J. Hawkins, "The htm spatial pooler—a neocortical algorithm for online sparse distributed coding," *Frontiers in Computational Neuroscience*, vol. 11, p. 111, 2017. [Online].

Available: https://www.frontiersin.org/article/10.3389/fncom.2017.00111

[28] M. T. . Y. C. S. Ahmad, "Spatial pooler algorithm," 2017, accessed: 26/03/2018, Version 0.5. [Online]. Available: https://numenta.com/assets/pdf/spatial-pooling-algorithm/Spatial-Pooling-Algorithm-Details.pdf

[29] S. A. . M. Lewis, "Temporal memory algorithm," 2017, accessed: 10/04/2018, Version 0.5. [Online]. Available: https://numenta.com/assets/pdf/temporal-memory-algorithm/Temporal-Memory-Algorithm-Details.pdf

[30] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Frontiers in Neural Circuits*, vol. 10, p. 23, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncir.2016.00023

[31] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *Neural Computation*, vol. 28, no. 11, pp. 2474–2504, 2016, pMID: 27626963. [Online]. Available: https://doi.org/10.1162/NECO_a_00893

[32] B. A. Sutton, R.S., *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press., 1998.

[33] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, Aug 1988. [Online]. Available: https://doi.org/10.1007/BF00115009

[34] H. M. Schwartz, *Multi-agent machine learning : a reinforcement approach*. John Wiley & Sons, Inc., 2014.

[35] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[36] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, no. 1, pp. 123–158, Jan 1996. [Online]. Available: https://doi.org/10.1023/A:1018012322525

[37] David Silver. (2015) Lecture 4: Model-free prediction. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MC-TD.pdf

[38] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang, "World of bits: An open-domain platform for web-based agents," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 3135–3144. [Online]. Available: http://proceedings.mlr.press/v70/shi17a.html

[39] N. Inc. (2018) Nupic documentation 1.0.4. [Online]. Available: http://nupic.docs.numenta.org/1.0.4/

A Appendix 1

A.1 Additional Pseudocode

Pseudocode which is not included here can be found in the documentation of Numenta for Spatial Pooler and Temporal Memory component ([28],[29]).

Algorithm 7 <TM: Phase 2 - activatePredictedColumn>

```
function ACTIVATEPREDICTEDCOLUMN(column)
    for each segment in segmentsForColumn(column, activeSegments(t-1)) do
        activeCells(t).add(segment.cell)
        winnerCells(t).add(segment.cell)
        if LEARNING_ENABLED then
            for each synapse in segment.synapses do
                if synapse.presynapticCell in activeCells(t − 1) then
                    synapse.permanence+ = PERMANENCE_INCREMENT
                else
                    synapse.permanence− = PERMANENCE_DECREMENT
                end if
            end for
            newSynapseCount = (SYNAPSE_SAMPLE_SIZE − numActivePotentialSynapses(t − 1, segment))
            growSynapses(segment, newSynapseCount)
        else
            if count(segmentsForColumn(column, matchingSegments(t − 1))) > 0 then
                punishPredictedColumn(column)
            end if
        end if
    end for
end function
```

A.2 Default parameters

A.2.1 Encoder

The features encoded include:

- Full 160x160 observational data encoded to 25600 1D SDR.
- A RADIUS*2 x RADIUS*2 encoded focus area around the mouse coordinates. If the image boundary is reached the area is smaller accordingly. The radius default value is 40, which results in a 6400 1D SDR.

- The mouse coordinates encoded using NUPIC Coordinate Encoder to a 6400 1D SDR with 257 active bits.

They are appended to a single SDR which is received as input in Layer 4 from the sensor region.

A.2.2 Layer parameters

A description of all common NUPIC parameters can be found in their documentation [39]. Custom parameters are described in Section B. The width of the layers is set to the output dimensions of the input regions as described in the network architecture.

Layer 4 - Spatial Pooler (SP)

- columnCount : 512
- globalInhibition : 1
- synPermConnected : 0.2
- synPermActiveInc : 0.02
- synPermInactiveDec : 0.02
- potentialRadius : 128
- potentialPct : 0.15

Layer 4 - Extended Temporal Memory (ETM)

- columnCount : 512
- cellsPerColumn : 32
- activationThreshold : 2
- minThreshold : 1
- initialPermanence : 0.2
- connectedPermanence : 0.2
- permanenceIncrement : 0.04
- permanenceDecrement : 0.08
- predictedSegmentDecrement : 0.0008
- formInternalBasalConnections : False

Layer 2/3 - Temporal Union Pooler (TP)

- columnCount : 512
- historyLength : 1
- minHistoy : 0
- leaningMode : True
- inferenceMode : False
- activeOverlapWeight : 0.4
- predictedActiveOverlapWeight: 0.6
- exciteFunctionType : linear
- decayFunctionType : linear
- decayLinearConst : 0.9
- synPermConnected : 0.2
- synPermActiveInc : 0.04
- synPermInactiveDec : 0.004
- globalInhibition : 1
- potentialRadius : 128
- potentialPct : 0.15

Layer 2/3 - Temporal Memory (TM)

- columnCount : 512
- cellsPerColumn : 32
- learningMode : True
- permanenceMax : 10
- initialPerm : 0.2
- connectedPerm : 0.2
- permanenceInc : 0.04
- permanenceDec : 0.08

- globalDecay : 0.000001
- newSynapseCount : 12
- minThreshold : 6
- activationThreshold : 9

Layer 5,D1,D2 - Spatial Pooler (SP)

- columnCount : 512
- globalInhibition : 1
- synPermConnected : 0.2
- synPermActiveInc : 0.04
- synPermInactiveDec : 0.004
- potentialRadius : 128
- potentialPct : 0.15

Layer 5 - Extended Temporal Memory (ETM)

- columnCount : 512
- cellsPerColumn : 32
- activationThreshold : 9
- minThreshold : 6
- initialPermanence : 0.2
- connectedPermanence : 0.2
- permanenceIncrement : 0.04
- permanenceDecrement : 0.08
- predictedSegmentDecrement : 0.0008
- newSynapseCount : 3
- formInternalBasalConnections : False

Layer D1,D2 - Extended Temporal Memory (Reinforcement) The layer have the same parameters as Layer 5 ETM, but with the additional configuration for temporal difference learning:

- TDTraceDecay (Lambda) : 0.1
- TDDiscount : 0.1
- TDLearningRate : 0.1
- globalValueDecay : 0.0001

The motor layer is one component but for understandability the parameters are divided into the two main computations Motor Layer - D1/D2 to L5 Apical

- columnCount : 512
- cellsPerColumn : 32
- activationThreshold : 9
- initialPermanence : 0.2
- connectedPermanence : 0.25
- minThreshold : 6
- learn : True
- synPermActiveIncMotor : 0.04
- synPermInactiveDecMotor : 0.08
- punishPredDec : 0.004

Motor Layer - L5 to Motor Apical

- motorCount : 32
- winnerSize : 6
- apicalGlobalDecay : 0.0001
- TDLearningRate : 0.2
- synPermActiveIncMotor : 0.04
- synPermInactiveDecMotor : 0.08
- punishPredDec : 0.004

A.2.3 Action mapping

- Actionspace : ['click', 'right', 'left', 'top', 'bottom', 'none']
- Distribution : Uniform, (Default 5 Motor Neurons per Action)

B Parameter Description

Temporal Union Pooler

- historyLength : The union window length.
- minHistoy : Minimum history length required to compute union.
- decayLinearConst : Linear decay rate applied in decay function.
- exciteFunctionType : Specify which excite function to use for computation. (Linear, Logistic, Fixed)
- decayFunctionType : Specify which decay function to use for computation. (Linear, Exponential, NoDecay)
- activeOverlapWeight : A multiplicative weight applied to the overlap between connected synapses and active-cell input.
- predictedActiveOverlapWeight : A multiplicative weight applied to the overlap between connected synapses and predicted-active-cell input.

Reinforcement Region (Based on Extended Temporal Memory)

- TDTraceDecay : TD Trace Decay used in eligability traces computation to decay the trace of each neuron.
- TDDiscount : TD Discount used in TD-Error computation.
- TDLearningRate : TD Learning rate (lambda) used how fast new state values are learned when updating of neuron values.
- globalValueDecay : Global decay applied to neuron values to prevent infinite growth.

Motor Layer computation

- motorCount : The number of motor neurons that map to the action space.

- winnerSize : Determines how many motor cells ultimatively become active. (k-cells with highest excitation)
- apicalGlobalDecay : Global decay applied to apical synapses of motor neurons at each iteration.
- TDLearningRate : The learning rate for temporal difference learning. Used for calculation of the increment/decrement of apical synapses from Layer 5 cells.
- synPermActiveIncMotor : Amount by which permanences of L5-Motor synapses are incremented during learning.
- synPermInactiveDecMotor : Amount by which permanences of L5-Motor synapses are incremented during learning.
- punishPredDec : Not correctly predicted motor cells will be decremented by this amount.