HIERARCHICAL TEMPORAL MEMORY BASED AUTONOMOUS AGENT FOR PARTIALLY OBSERVABLE VIDEO GAME ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS INSTITUTE
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ALİ KAAN SUNGUR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF MULTIMEDIA INFORMATICS

AUGUST 2017

HIERARCHICAL TEMPORAL MEMORY BASED AUTONOMOUS AGENT FOR PARTIALLY OBSERVABLE VIDEO GAME ENVIRONMENTS

Submitted by **ALİ KAAN SUNGUR** in partial fulfillment of the requirements for the degree of **Master of Science in The Department of Multimedia Informatics , Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Director, **Graduate School Of Informatics**

Assoc. Prof. Dr. Hüseyin Hacıhabiboğlu
Head of Department, **Multimedia Informatics**

Asst. Prof. Dr. Elif Sürer
Supervisor, **Multimedia Informatics**

Examining Committee Members

Assoc. Prof. Dr. Hüseyin Hacıhabiboğlu
Multimedia Informatics, METU

Asst. Prof. Dr. Elif Sürer
Multimedia Informatics, METU

Assoc. Prof. Dr. Aysu Betin Can
Information Systems, METU

Asst. Prof. Dr. Ufuk Çelikcan
Computer Engineering, Hacettepe University

Asst. Prof. Dr. Nurcan Tunçbağ
Health Informatics, METU

Date: 15.08.2017

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ALİ KAAN SUNGUR

Signature : 

ABSTRACT

HIERARCHICAL TEMPORAL MEMORY BASED AUTONOMOUS AGENT FOR PARTIALLY OBSERVABLE VIDEO GAME ENVIRONMENTS

Sungur, Ali Kaan
M.Sc., Department of Multimedia Informatics
Supervisor : Asst. Prof. Dr. Elif Sürer

August 2017, 103 pages

Believable non-player characters (NPC) can have a profound impact on the experience that a video game provides. This thesis presents an online, unsupervised and lifelong learning autonomous agent that the player can interact with. It has an architecture utilizing a combination of Hierarchical Temporal Memory and Temporal Difference Learning Lambda with the guidance of neurobiological research. The agent has a visual sensor with an online data stream. Input from this sensor feeds the architecture to model the surrounding environment. The goal of the agent is to learn rewarding sequences of behavior based on the stimulation it receives caused by its actions. It navigates in a procedurally generated three-dimensional environment and is in a continuous learning state adapting the synapses of its neural connectome. The architecture is also capable of being stored and loaded at any point allowing for persistent learning through multiple simulation sessions. The study presents the learning characteristics of the agent on a video game related learning task. We compared the data collected from the experiments with varying parameters along with providing the runtime and serialization performance. The proposed methodology results in an autonomous NPC that can learn rewarding behaviors without any supervision. Moreover, it is also capable of learning specific action sequences via player guidance. The result is a promising and novel NPC architecture that is also relatively open to incremental improvements through the relevant neurobiological studies and the advancements on the theory of Hierarchical Temporal Memory.

Keywords: hierarchical temporal memory, autonomous agent, reinforcement learning, temporal difference learning, lifelong learning

ÖZ

KISMEN GÖZLEMLENEBİLİR SANAL OYUN ORTAMLARI İÇİN
HİYERARŞİK ZAMANSAL HAFIZA TABANLI OTONOM AJAN

Sungur, Ali Kaan
Yüksek Lisans, Çokluortam Bilişimi
Tez Danışmanı : Yrd. Doç. Dr. Elif Sürer

Ağustos 2017, 103 sayfa

Bir oyunun sunduğu tecrübede bilgisayar kontrollü karakterlerin inandırıcı olması ciddi bir önem arz etmektedir. Bu tezde gerçek zamanlı, gözetimsiz, hayat boyu öğrenme kabiliyetleri olan ve kullanıcıların etkileşimde bulunabileceği bir otonom ajan mimarisi sunuyoruz. Hiyerarşik Zamansal Hafıza ve Zamansal Farklılık Öğrenmesi algoritmalarını, sinir bilim ve biyoloji araştırmaları doğrultusunda bir arada kullandık. Ajan gerçek zamanlı veri akışına sahip görsel sensörü aracılığı ile çevresinin bir modelini oluşturur. Ajanın temel amacı aksiyonları neticesinde aldığı ödüller doğrultusunda ardışık aksiyonlardan oluşan davranışlar öğrenmektir. Yöntemsel olarak oluşturulmuş, üç boyutlu ortamlarda hareket eden ajan, sürekli öğrenme halinde olup sinir ağını güncellemektedir. Ajanın sinir ağı herhangi bir anda kaydedilebilir ve geri yüklenebilir. Dolayısıyla farklı simülasyon seanslarında bile kaldığı yerden öğrenmeye devam edebilir. Farklı parametrelerle yapılan oyun içi öğrenme seanslarından derlenen veriler ile çalışma hızı, saklama, geri yükleme ve öğrenme kabiliyetleri ölçülmüştür. Önerilen yöntem doğrultusunda denetime ihtiyaç duymaksızın ödül getiren davranışları öğrenme yeteneğine sahip, bilgisayar kontrollü bir karakter oluşturulmuştur. Karakter buna ek olarak kullanıcı yönlendirmeleri ile öğrenmeyi de desteklemektedir. Sonuç olarak, bu çalışmada yeni bir yaklaşım aracılığıyla bilgisayar kontrollü karakterler için gelecek vaat eden bir mimari sunulmuştur. Sinir bilim ve biyolojik kaynaklar doğrultusunda Hiyerarşik Zamansal Hafıza teorisine dayandırılarak oluşturulan bu mimari, kademeli geliştirmeye de gayet açıktır.

Anahtar Kelimeler: hiyerarşik zamansal hafıza, otonom ajan, destekli öğrenme, zamansal farklılık öğrenmesi, hayat boyu öğrenme

this thesis is dedicated to all partially observing agents...

ACKNOWLEDGMENTS

I would like to start by expressing my gratitude to my advisor Assist. Prof. Elif Sürer for guiding me throughout the demanding process of writing this thesis. At times, she and the head of our department Assoc. Prof. Hüseyin Hacıhabiboğlu supported my academic adventure even despite my attitude. I would also like to thank him for being an exemplary and inspiring professor among Informatics Institute and for doing his best for the good of our department.

This thesis is an output of the combined knowledge that the passionate people of HTM community provide. Lead by Numenta employees, the topics talked about in the community forum range from neuroscience papers and theoretical arguments to practical experimentations. From the community, I would especially like to thank H. Suzuki, M. Taylor, J. Hawkins, A. Subutai, Y. Cui, A. Lavin, S. Purdy, F. Bryne, P. Lamb, J. Bruce and many others that need to be named here for our valuable discussions. I am grateful for the transparency of Numenta and that they publish their research for everyone, unlike many others.

I feel sorry for my mother G. Sungur, my sister Z. T. Sungur and my close friends for putting up with me during this time. Though I guess, that is what they are signed for in the first place, specifically A. S. Şahin. In addition, B. Ulaşoğlu aka. V. Kozmik and E. Gök deserve a special mention for the long hours of exciting discussions that they provided for this work.

Last but not least, I am grateful for a kind and understanding significant other that was blessed to me during this process. She had to go through this taxing process alongside me which seemed forever at times. Maybe, she will continue being there for me. E. S. Seçim, will you marry me?

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CLA	Cortical Learning Algorithm
CNN	Convolutional Neural Networks
DL	Deep Learning
FPS	Frames per Second
FOV	Field of View
GPe	Globus Pallidus External
GPi	Globus Pallidus Internal
HFOV	Horizontal Field of View
HMM	Hidden Markov Model
HTM	Hierarchical Temporal Memory
HUD	Heads-Up Display
LSTM	Long-Short Term Memory
MDP	Markov Decision Process
ML	Machine Learning
NPC	Non-Player Character
RL	Reinforcement Learning
PMN	Pyramidal Neuron
POMDP	Partially Observable Markov Decision Process
POV	Point of View
SNc	Substantia Nigra Pars Compacta
SNr	Substantia Nigra Pars Reticulata
STN	Subthalamic Nuclei
TD	Temporal Difference Learning
TDλ	Temporal Difference Learning Lambda
VFOV	Vertical Field of View
VOM	Variable Order Markov
WTA	Winner Take All

CHAPTER 1

INTRODUCTION

Non-player characters are important components of video games, even sometimes taking the spotlight on games such as Thief (1998), Fear (2005) and Stalker (2007). Believable characters that provide high levels of interactivity have an undeniable impact on the overall experience. The NPCs can have varying roles ranging from pets, companions, community members, assistants, mentors to evaluators, opponents and enemies. Some games such as Black and White (2001) takes one more step and builds the experience around its NPC by putting the player in control of guiding it. It utilizes a hybrid system by containing belief-desire-intention models, rule-based systems, decision trees and perceptron neural networks. However, designing games around its non-player characters and in general, artificial intelligence (AI) components requires the developer to craft the underlying engine accordingly. While the modern game engines such as Cry Engine, Unreal and Unity have advanced tools for building believable agents, these engines introduce constraints on the development due to their resource management and architectural limitations. Although they provide robust AI facilities such as behavior toolkits, pathfinding methods, conditional logic utilizing decision trees, they are not designed around the intelligence component itself. So, it is challenging to apply the state of the art advancements on machine learning (ML) and artificial intelligence in these modern game engines. AI centric experimental engines and platforms are essential for the industry to incorporate these advancements and provide experiences that utilize the latest methodology. Computer controlled video game agents are powerful platforms to communicate the capabilities of the most recent findings on AI and ML because they also provide scenarios that require active participation from the player.

In this thesis, a novel autonomous agent architecture is developed along with the necessary platform to test it in terms of suitability for video games, specifically as an NPC intelligence. Aim of the study is to provide an interactive agent that is capable of producing its own life cycle based on the reward stimulation from its surroundings and based on the player guidance. The study presents relevant neurobiological research to support the architecture and it utilizes a cortical computation approach - Hierarchical Temporal Memory (HTM). HTM is an unsupervised, continuous and online learning method for modeling spatiotemporal patterns and variable order Markov (VOM) sequences. The agent navigates a three dimensional and procedurally generated environment via online visual information stream. Therefore, it is a partially observing agent and the problem at hand can be classified as partially observable Markov decision process (POMDP). It is important to note that we decided to use HTM because

of its neurobiological relevancy and its potential to provide continuous learning in real-time. The traditional machine learning approaches to model partially observable environments, such as convolutional neural networks (CNN), deep learning (DL), long short-term memory (LSTM) are not viable contenders because of insufficient neurobiological evidence supporting them. In addition, real-time applications as well as continuous online learning are not their strong suit considering the required computational power. The data hungry nature of DL and its variations do not provide the best facility for an NPC because the input data is limited in real-time learning tasks. The objective is to create a lifelong learning NPC that functions in real-time which proves to be difficult to solve, especially with DL. On top of this, interaction with the player necessitates the model to allow for manipulation, which is again a problem considering the black box nature of the overmentioned method.

Hierarchical Temporal Memory attempts to build an abstracted version of the universal computation happening among the neocortex based on the six layer, mini and macro-column theories in neuroscience. As a result, it satisfies the neurobiological concerns of this study. While doing so, it also attempts to model the cortical computation at a functional level which allows for better under the hood transparency and opens up possibilities for in-depth visualizations as implemented in this study. The computational requirements of HTM allow for a real time implementation which is embedded into the custom game engine utilized for this thesis. Considering all, HTM theory enables a starting point for an autonomous agent that has a neurobiological base along with online, lifelong learning and interaction capabilities. Having a neurobiological base allows for universal incremental advances by taking the relevant research as the reference point. This study couples HTM with Temporal Difference Learning Lambda (TD(λ)) along with necessary additional features for an autonomous agent. The results are promising for the proposed methodology to be utilized as an NPC intelligence.

This thesis starts with presenting background on HTM, TD(λ) and the relevant neurobiological research in Chapter 2. In Chapter 3, problem, agent and the platform constraints are laid out to specify the scope of the study. We described the agent architecture from a mechanistic point of view in Chapter 4 along with the supplementary visual material. Chapter 5 focuses on the reproducibility of this study by presenting details about implementation and additional features necessary for an HTM and TD(λ) based autonomous agent. Study concludes after presenting results and providing relevant discussion in Chapters 6 and 7.

1.1 Scope of This Thesis

The thesis aims to model an autonomous architecture specifically for a video game NPC, not a general purpose agent. For this reason, the current environment is limited to video game worlds and to be specific; procedurally generated three-dimensional environments. One of the primary constraints on the scope emerges from neurobiological concerns. The functionality of cortical layers and computational models of basal ganglia are reference points for the architecture. This approach facilitates a starting point that is open to incremental improvements through the relevant neurobiological research. More research with this perspective may yield biology backed uni-

versal solutions and an incremental approach towards general intelligence while imitating the biological intelligence. Model transparency, serialization capability, player interaction and real-time performance are the constraints on the scope of this study. We designed the agent architecture and the testing scenario around these constraints which rather limit the problems that the agent can currently solve. In summary, we specify the scope as producing an unsupervised autonomous agent with real-time performance and neurobiological basis, in a lifelong learning video game scenario which includes player interaction.

1.2 Contributions

Up to the time of this study, there are no publications on HTM and TD(λ) based autonomous agents. The only relevant works on this topic were done by Otahal [1] for his MSc thesis and Gomez [2] in his BSc project report. The former argued about the design of a system combining HTM with reinforcement learning (RL) for a biologically plausible agent while providing some implementation notes. The latter combined HTM with RL and compared it to Q-Learning under a Markov Decision Process (MDP) setting. One of the strong suits of HTM is functioning well under noisy environments which is lacking in an MDP setting that was used by Gomez.

In 2016, Sungur and Surer [3] proposed an architecture by extending the HTM theory for producing voluntary behavior based on recent neuroscience findings. We realized the proposition in this thesis with results on a video game environment. Therefore, this study is novel in being the first publication on HTM based partially observing autonomous agents with learning results on a real task. It also presents the neuroscience material that directs the architecture which includes computational models of basal ganglia and research on cortical layer functionalities. The study provides the first findings on combining HTM with TD(λ) through learning tasks. Moreover, we propose extensions to the existing HTM implementation to allow for real-time functionality and present an implementation guideline for the whole architecture. Among the current HTM implementations, this thesis houses the only published real-time implementation of HTM algorithms in a continuous learning setting. As a result, this study is also the only platform realizing a lifelong learning NPC based on HTM via its serialization and player guidance capabilities.

CHAPTER 2

BACKGROUND

2.1 Background on Hierarchical Temporal Memory

The foundations of Hierarchical Temporal Memory date back to work done by P. Kanerva [4] on Sparse Distributed Memory. He argued about the sparse distributed representations being the underlying long term human memory model denoting both the data and the address. This memory model is one of the key concepts behind HTM. A neuroscience institute founded in 2002, Califonia, kick-started the research on the theory behind Hierarchical Temporal Memory. The mission of a non-profit organization, Redwood Neuroscience Institute, was to study biologically accurate and mathematically well founded, theoretical, large scale memory models. Kanerva was also a research affiliate for the institute. One of the key figures of the institute, Jeff Hawkins, went on to publish a book named On Intelligence about their memory prediction framework [5]. Together with Dileep George, they co-founded Numenta Inc. in 2005 following this publication. The goal was to advance the research based on the computational principles of the neocortex via experimental models. The paper by George and Hawkins [6] describes the basis of HTM and the underlying hierarchical Bayesian inference model. It describes the HTM node as a combination of Markov chain and coincidence detector. They went on to use their experimental model in the industry under the name Grok. It is a commercial algorithm focused on finding anomalies in data and utilized in predictions of IT analytics. The company Cortical.io also used this product for natural language processing problems. Since 2013, Numenta decided to publicize their model through an open source implementation which they called Nupic.

George was more focused on the mathematical background of the research [7] and as a result, the co-founders of Numenta split. He found another company under the name Vicarious which consists of impactful researchers within the field. However, unlike Numenta, Vicarious did not publish their model as of this study. Thus, Hierarchical Temporal Memory is one of the only applicable computational models of cortical learning at a functional level in machine learning supported by neurobiological evidence. The work done by Rinkus [8] links the sparse distributed coding models to the mini and macrocolumn theories of neuroscience which forms the basis of HTM along with the six layer theory of the neocortex. The HTM neuron model concurs with this study which states that neurons of macrocolumns have the same receptive fields while sparsely representing their input patterns through minicolumns (around 70). Neurons of a minicolumn (around 20) represent similar features and enforce

sparsity. Mumford presents a computational view of the six layer theory. He also describes the specialized functionalities of the cortical layers in his publications [9] [10]. HTM theory incorporates the functional principles of these layers.

On the side of traditional machine learning approaches, Marblestone et al. [11] argue about the integration of neuroscience to the theory of deep learning by comparing the current backpropagation implementations with the proposed biologically plausible ones. They concluded the default approach to be biologically implausible even if the brain is assumed to be doing backpropagation. It is important to note that the fundamental difference between HTM and other conventional neural networks based approaches is that HTM is built around neuroscience research while popular methods such as DL try to integrate it afterwards. Studies questioning the biological plausibility of backpropagation date back decades. For example, the paper by Mazzoni et al. [12] concludes it to be biologically implausible. The HTM theory does cortical computations without a backpropagation mechanism outlined in Y. Cui et al [13]. This paper dwells on the fundamentals of HTM sequence memory and compares it with statistical models, feedforward and recurrent neural networks on real world problems with continuous data streams. Hawkins and Ahmad present the underlying neurobiological evidence of the HTM theory in their publication [14]. They argue about the number of synapses a neuron needs for sequence learning and its implication in the context of cortical learning. Ahmad et al. also published a work [15] showing the unsupervised real-time anomaly detection performance of HTM.

2.1.1 Background on HTM and RL Coupling

The literature on combining HTM with reinforcement learning (RL) is fairly weak, an expected outcome considering its recency. We were only able to find two attempts on such online publications consisting of a Master's thesis by Otahal in 2014 [1] and a Bachelor's project by Gomez in 2016 [2]. Otahal's work explains how such a system works from a design perspective while providing some implementation guidelines. He conducts learning tests on a 2D grid world where the HTM is extended with -in his words- emotional guidance to result in a task based HTM agent. On the other hand, Gomez treats the HTM as the reinforcement learning itself and compares it to Q-Learning [2]. His agent model runs as a Markov Decision Process without being partially observing. However, one of the strong suits of HTM is being able to run on noisy environments with partial observation thanks to its noise robustness. HTM neuron uses its dendrites as coincidence detectors functioning in an all-or-nothing principle discarding activations under a set threshold. This mechanism allows HTM to be highly robust to noise. While providing the first valuable attempts at an HTM based agent, both studies do not share any supporting neurobiological evidence on their HTM and reinforcement learning coupling.

2.2 Related Neuroscience Research

This study aims to provide relevant neurobiological research along with the component explanations of the proposed architecture. The current HTM theory mainly

imitates the functionality of cortical layers 2/3. These layers are hypothesized to be doing variable order sequence learning [14], [13]. The recent HTM research areas include sensorimotor integration through extending the theory with layers 4, 5 and 6. There is especially an interest in layer 6 which is hypothesized by Numenta researchers to be doing transformations between world space inputs and object space inputs.

Regarding the proposed methodology in this study, research on cortical layers 1, 4 and 5 are significant in extending the current HTM theory to allow sensorimotor inference. The second area of interest is the computational models of basal ganglia. This subcortical structure interplays with the frontal cortex to generate actions.

2.2.1 Cortical Layers

Regions of neocortex such as V1 (visual cortex), S1 (somatosensory cortex) and M1 (motor cortex) constitute of six layers with differing functionalities. These layers are positioned on top of each other with layer 6 at the bottom and layer 1 at the top. The primary input layer of a region is layer 4 [16]. This layer integrates information from a variety of subcortical structures and other cortical regions gated by the thalamus. The input is specialized according to the cortical region type. For example, sensory thalamic input and motor activity information make up a portion of the information coming to M1 layer 4 [17]. The computational order of the six layers is still a debate among neuroscientists. The classical assumption states that the information travels to layers 2/3 after layer 4 and continues to layers 5/6 from there [18], [19]. The findings of Constantinople and Bruno [20] also suggest that layers 2/3 project to layer 5. However, this projection is modulatory and layers 5/6 cannot be driven by layers 2/3 because layers 5/6 are activated slightly before 2/3 according to their results. Moreover, layers 5/6 take direct excitatory thalamic input and the activity is not driven by layer 4. So they suggest that there are two seperate feedforward pathways in cortex: L4→L2/3 and L5→L6. On the other hand, the work by Schubert et. al. [21] states that layer 5a cells take excitatory input from layer 4 cells. According to the research presented in this section, Figure 2.1 shows the cortical computational flow assumed in this study.

Ramaswamy and Markram present the anatomy and physiology of the layer 5 pyramidal neuron in their work [22]. Their findings suggest that a layer 5 neuron integrates information across all cortical layers. Layer 5 also serves as the principal pathway for output among the region. It supplies information flow to motor areas and other subcortical structures such as basal ganglia as also stated by Naka and Adesnik [23]. In [24], Hosp and Loft experimented with teaching rats to reach a pellet. They found that pyramidal neurons (PMN) in layer 2,3 and 5 had enlarged dendritic fields after learning sessions. There was an increase in the synapses of layer 5 neurons promoting the idea that learning increases synaptogenesis. This finding is linked to the association by learning mechanism between layer 5 and motor neurons in our architecture.

In [25], Garcia-Munoz and Arbuthnot analyzed the structural composition of layer 1 and its effect on other cortical layers. They found layer 1 to be containing inhibitory neurons which influence layers 2, 3 and 5. Layer 1 is the terminal for the informa-

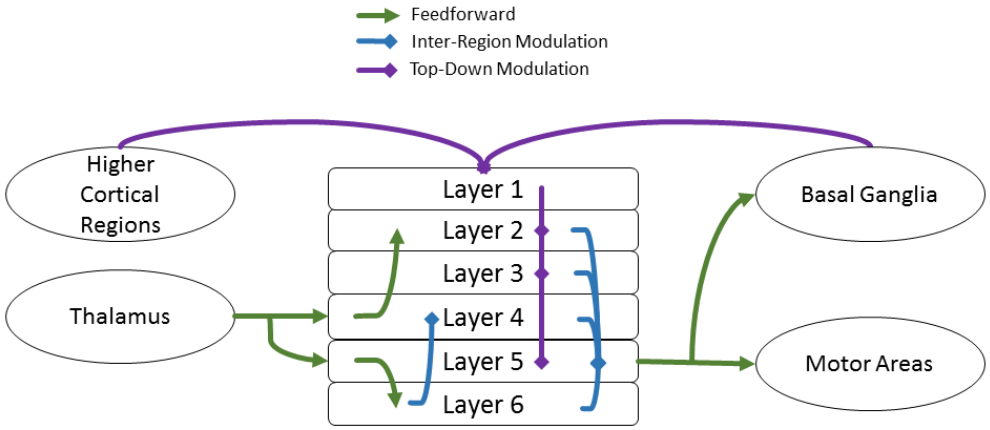


Figure 2.1: The abstracted computational flow of the cortical layers.

tion originating at basal ganglia and going through the thalamus. The modulation of the superficial layer 1 on other layers concurs with the findings of Gao and Zheng [26]. According to Kim [27], stimulating layer 1 yielded activity on apical dendrites of layer 5 cells. Functionally, layer 1 resolves the conflicting alternative responses of layer 5 ensuring the continuation of cortical function as stated by Edelman and Gally [28]. These findings form the foundation for the abstraction of layer 1 in our architecture.

Although layer 6 cells make up a significant portion of the cortical layers, their functionality is still not completely understood. Lee and Sherman [29] present their findings on layer 6 neurons having a modulatory role for layer 4 from an organizational perspective. The study also points out the possibility of tall pyramidal neurons of layer 6 exerting influence on the output of layer 5 which reaches thalamus, striatum and other subcortical structures as stated in [30] by Morishima and Kawaguchi. In another the study, layer 6 is found to have excitatory influence on layer 5a activations in mouse visual and somatosensory cortices [31].

2.2.2 Computational Models of Basal Ganglia

The neuroanatomy of basal ganglia presented in the publication by Lanciego et al. [32] shows the refined information flow through basal ganglia back to the cortex. They describe the afferents and efferents of the nuclei of basal ganglia: striatum, globus pallidus internal (GPi) and external (GPe), substantia nigra pars reticulata (SNr), substantia nigra pars compacta (SNc). They also present relevant material describing the direct pathway neurons of striatum which are D1 dopamine receptive medium spiny neurons (MSNs) and the indirect pathway neurons which are D2 dopamine receptive MSNs. According to another study, basal ganglia provides the main functionality of reward circuitry and controls the learning of behaviors via dopamine receptive cells [33].

The study by Helie et al. [34] compares 19 computational models of basal ganglia that are actuated by convolutional neural networks (CNN). It presents the limitations

of the existing models which had their functional schemes of basal ganglia. The comparison concludes that all the computational models agree on the functionality of direct pathway which is activating specific cortical representations. On the other hand, the models interpret hyperdirect and indirect pathways with major differences. The study conducted by Brown et al. [35] presents the mechanics of the interplay between frontal cortex and basal ganglia through the differences between planned and reactive saccades. They argue that plan execution is possible through the direct pathway of the ganglia. The book published by Prescott et al. on modeling natural action selection [36] describes the impact of dopamine changes on learning in Go (direct) and No-Go (indirect) pathways. The dips in dopamine indirectly excite D2 receptive cells and therefore supports learning in No-Go pathway. The burst of dopamine leads to the excitation of D1 receptive cells resulting in learning in the Go pathway. The competition between Go and No-Go pathways generate behavior, so any imbalance results in behavioral inconsistencies such as Parkinson's Disease and attention-deficit hyperactivity disorder [37]. Figure 2.2 shows the functional scheme of basal ganglia along with the direct and indirect pathways. The research related to these pathways constitute the reasoning behind the abstracted reward circuitry of the agent.

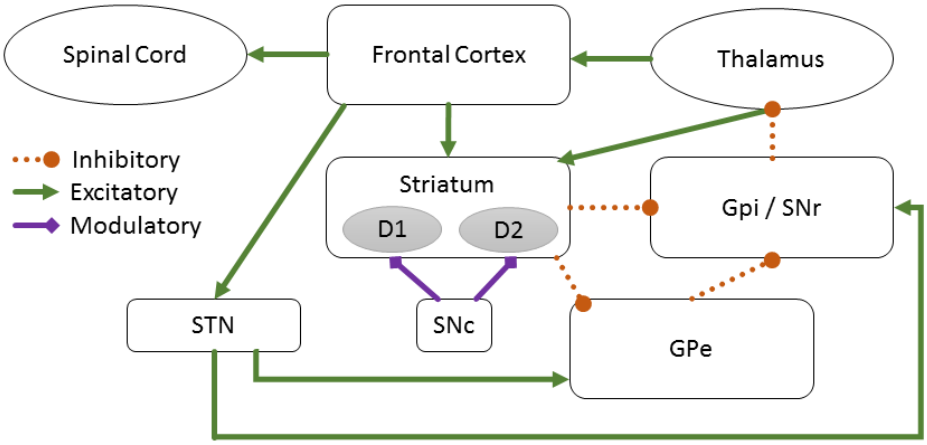


Figure 2.2: Functional scheme showcasing basal ganglia pathways.

- **Direct (GO) Pathway** : Striatum → GPi.
- **Indirect (NO-GO) Pathway** : Striatum → GPe → GPi.
- **Hyperdirect Pathway** : Cortex → STN → GPi.

Tewari et al. [38] present the differing functionality of basal ganglia input structures: subthalamic nucleus (STN) and striatum. They compare these structures in terms of their roles in motor excitation and inhibition. STN contributes to the global inhibition of all actions through the hyperdirect pathway until output conflicts are resolved. It states that STN increases the response thresholds to inhibit impulsive actions while the striatum does the opposite and decreases response thresholds for quicker activation of previously learned actions. Accordingly, ganglia are used less when a decision has been made. The study also states that striatum can make comparisons between

estimated reward and the reward during acting. Striatum neurons can encode both positive and negative results of action selection. Zavala et al. [39] present their findings on STN which is the main component of the hyperdirect pathway. They hypothesize that directly activating the STN causes the inhibition of all responses until the competition between cortico-striatal activation (direct pathway) and cortico-striatal inhibition (indirect pathway) results in the correct response. This behavior suppression mechanism is incorporated in one of the well known basal ganglia computational models by Frank et al. [40]. According to [41], the inhibition by STN allows striatum to integrate information from other cortical regions to resolve conflictions and output the correct response. We currently neglect the functionality of hyperdirect pathway because our architecture consists of a single cortical region in its current state.

In [42], Haber analyzes the communication of midbrain dopamine neurons and striatum. The study concludes that midbrain dopamine neurons output mainly to the striatum. It also states that thalamocortical pathway which receives the output of ganglia may be a simple one-way relay to cortical regions. The thalamus is currently not included in our architecture, and this assumption decreases the importance of thalamus on the communication between cortex and basal ganglia. The direct projection of layer 5 onto basal ganglia and the responding output of ganglia which terminates on the superficial cortical layer 1, forms the basis of reward circuitry loop. Other cortical layers are then modulated by layer 1 through their apical dendrites. Haber also suggests that dopamine cells may receive direct sensory input which is how our study integrates dopamine into the model.

2.3 Hierarchical Temporal Memory

Hierarchical Temporal Memory is a continuous, online and unsupervised neural computation method. It achieves learning via adapting the synaptic connectome of volumetric neural layers where connections form and decay continuously. Mini and macrocolumn theories [8] direct the structural composition of HTM layers. The activation of an HTM layer is a sparse distributed representation which describes the state. The general goal of HTM is variable order sequence modeling and generating predictions from this model [14]. It can model the change in streaming input patterns with temporal dependency. Depending on the current state and input, it outputs a prediction about the following state on each iteration. The learning mechanism imitates the functional principles of the cortical layers in the six layer theory of the neocortex, specifically layers 2 and 3. HTM neurons are capable of predicting their feature activity by learning the activations happen before them. At each iteration, the current activation forms connections with the previous activations. Every activation stimulates the possible following activations through these formed synapses. These stimulations depolarize neurons that represent the activations on the next step, and these neurons are prioritized to fire. If the feedforward input at the following iteration matches with the prediction, the neural activation becomes sparse. Due to this mechanism, the resulting active neurons represent the feedforward input in the context of previous activations. In short, HTM layers are capable of predicting their future state as in Figure 2.3, depending on the current state and the context.

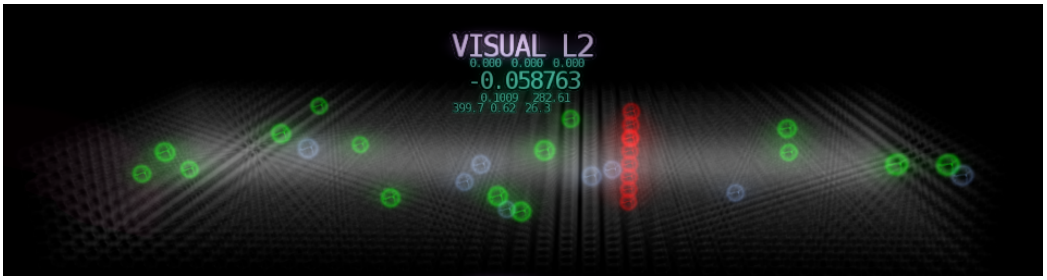


Figure 2.3: Activation and the prediction of an HTM layer. Red = unpredicted activation, green = successfully predicted activation and blue = predictive.

2.3.1 Structural Composition and HTM Neuron

The research in neuroscience identifies a similar arrangement of neurons in mammalian neocortex. According to the mini and macrocolumn theory of Rinkus [8], a minicolumn is a group of cells that are receptive to a similar feedforward input. The minicolumns form a macrocolumn that is represented by one or more cortical layers. The macrocolumn is a sparse distributed representation of the input via active minicolumns. The neurons of minicolumns function in a winner-take-all (WTA) fashion where an active neuron inhibits the rest. Minicolumns ensure sparsity among the macrocolumn while also representing the context via their active neurons. As a result, the neurons of HTM layer form a grid of minicolumns which represent the macrocolumn shown in Figure 2.4. All the neurons of a minicolumn share a similar synaptic receptive field which is called proximal dendrite. The individual neurons of minicolumns have synapse segments that are unique to them which are called the distal and apical dendrites. These provide the neurons with contextual information.

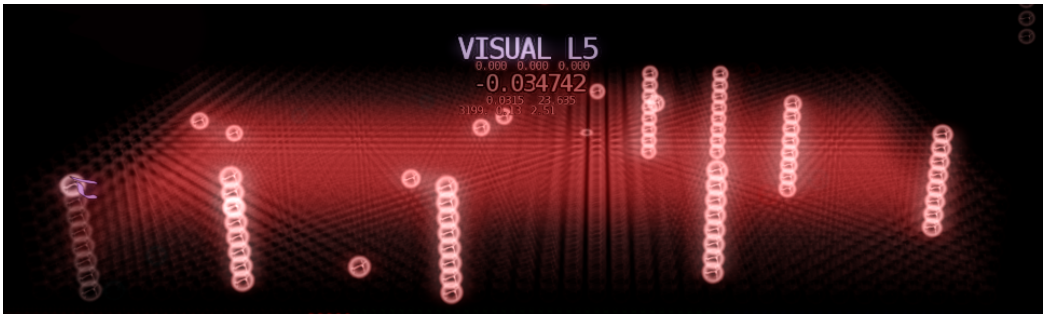


Figure 2.4: HTM layer composition; a single layer macrocolumn constituting minicolumns.

Guided by the neurobiology, the HTM neuron model has more functionalities compared to point neurons seen in conventional neural networks. The HTM neuron is an abstraction of the pyramidal neurons that make up the majority of the cells in cortical layers [43]. The dendrites incorporate the biological functionality of proximal and distal dendrites which have separate effects on the cell body. These dendrites have different activation processes and connection rules [14]. Figure 2.5 describes the HTM neuron model.

The HTM neuron has a single proximal dendrite, and all the neurons throughout a

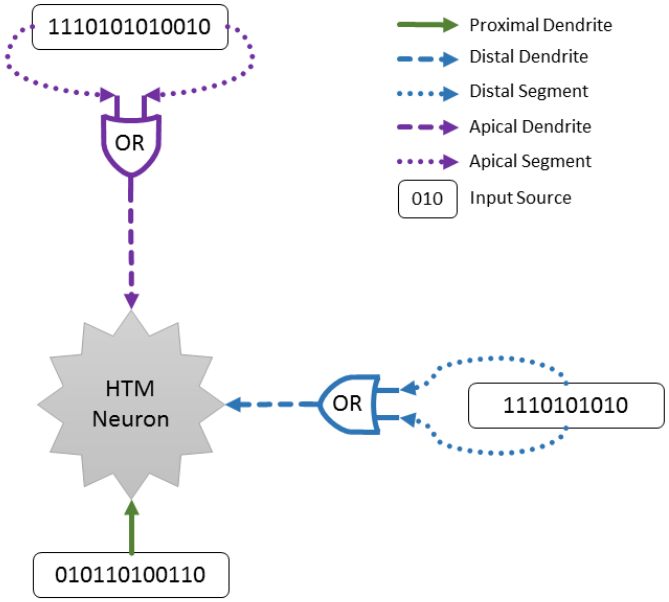
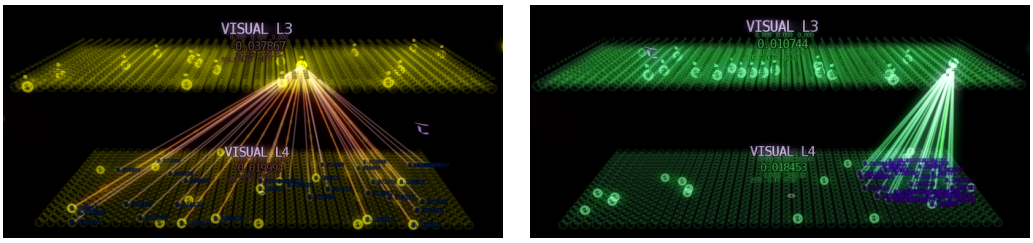


Figure 2.5: HTM neuron model. Proximal synapses are shared among all the neurons of a minicolumn and these synapses activate minicolumns. Distal and apical synapses carry contextual information and determine which neurons to fire among an activated minicolumn.

minicolumn share it. The synapses of the proximal dendrite have a pre-defined receptive field on the input data. This receptive field can represent any form of topology based on the position of the minicolumn. The input to an HTM layer is a binary representation of any type of data, including the activation of another layer. Given a minicolumn, the inputs falling inside its receptive field are the potential synapse locations of its proximal dendrite. Synapses of this potential pool are modified continuously by the source activity. Details about this adaptation process are explained in the Spatial Pooler algorithm in Section 2.3.2. Figure 2.6 shows the proximal dendrite visualization of two minicolumns with random and topological connections.



(a) Proximal dendrite with random sampling. (b) Proximal dendrite with local sampling.

Figure 2.6: A proximal dendrite with no topology is on the left while another with a positional topology is on the right.

A single neuron can recognize a multitude of activation patterns through its distal dendrites due to its nonlinear properties [13]. The cell develops dedicated segments for each pattern on its distal dendrites. If one of these patterns occur, the complementary distal dendrite is activated, and this puts the parent cell in a depolarized/predictive

state. These segments function in an all-or-nothing fashion: they ignore any activity if it does not stimulate the number of synapses required by the set threshold. Input overlaps below the threshold are treated as noise. Temporal Memory algorithm explains the details of this process in Section 2.3.3. In Figure 2.7, a single distal segment of a cell is chosen among many, and its synapses are visualized with their permanence (strength) values.

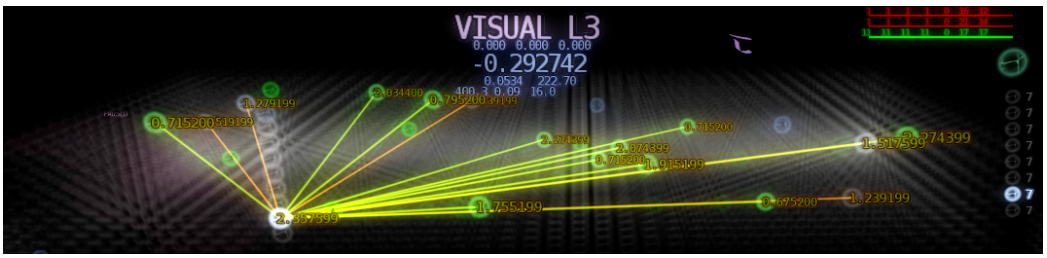


Figure 2.7: Distal segment of a cell sampling from neighboring neurons.

Apical dendrites are functionally similar to distal dendrite which depolarize their parent cells and provide context to the HTM activation. However, apical dendrites sample from further sources and they provide contextual information from higher cortical layers and subcortical structures such as basal ganglia. Distal segments lead to distal depolarization while the apical segments lead to apical depolarization. Figure 2.8 visualizes an apical segment of a neuron sampling from a distant source.

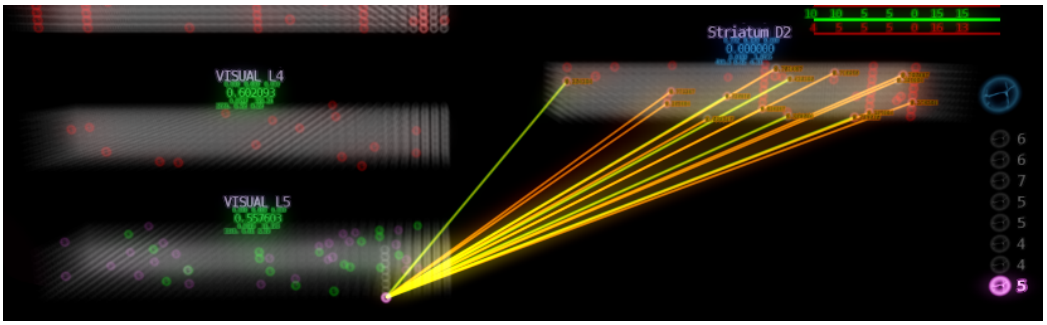


Figure 2.8: Apical segment of a cell sampling from reward circuitry layers of our architecture.

HTM computations are carried out via two main interleaving phases: Spatial Pooler (SP) and Temporal Memory (TM). Spatial Pooler decides which columns spatially represent the input. On the other hand, Temporal Memory activates neurons among these active columns based on temporal context.

2.3.2 Spatial Pooler

The first computational stage of HTM involves representing the input spatially. Every input pattern is represented by a set of active columns depending on the overlap between their proximal dendrites and the pattern. As stated earlier, all neurons share the same proximal connections among the minicolumn. Each column has a single feedforward proximal dendrite. In each HTM cycle, Spatial Pooler algorithm sequentially iterates the three phases explained below.

2.3.2.1 Phase 1 - Calculating Overlaps

The first phase calculates the overlap of proximal dendrites with the active inputs. Algorithm iterates over all the connected proximal synapses of columns. A synapse is connected when its permanence is above the connection threshold. Overlap of a column is incremented if the proximally connected input is active. The boost factor calculated in Phase 4 scales the final overlap to ensure that every column is utilized over time. Algorithm 1 presents the Phase 1 pseudocode and Figure 2.9 shows the calculated overlaps of all the columns for an HTM layer. *stimulationThreshold* denotes the threshold for treating the input overlap as noise.

Algorithm 1: SP - Phase 1

```
1 foreach col ∈ layer.columns do
2     col.overlap ← 0;
3     foreach syn ∈ col.proxDend.connectedSynapses do
4         col.overlap ← col.overlap + syn.active;
5     if col.overlap < stimulusThreshold then
6         col.overlap ← 0;
7     else
8         col.overlap ← col.overlap · col.boost;
```

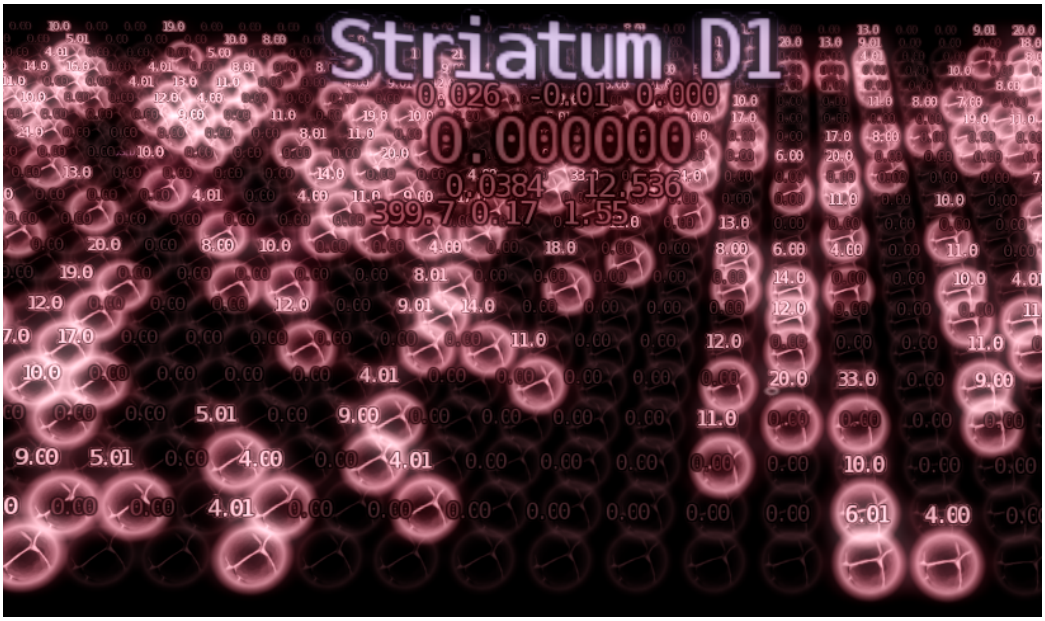


Figure 2.9: Calculated overlaps between the proximal dendrites of the columns and the input.

2.3.2.2 Phase 2 - Inhibition

There is competition among the columns to become active because the number of active columns is fixed at any time, and these active columns inhibit the rest. This

mechanism ensures that the input is converted into a sparse distributed representation. The inhibition can be carried out in a global fashion meaning all the columns of a layer is compared with each other, or it can be local within a columnar neighborhood. However, local inhibition severely impacts the performance, so the global inhibition is preferred in this study for real-time functionality. Algorithm 2 presents the Phase 2 pseudocode and Figure 2.10 shows the resulting active columns after applying global inhibition on the activation at Phase 1. *k_maxOverlap* function takes all the columns as input and calculates the minimum overlap among the highest overlapping k columns. k denotes the number of active columns at a time and is calculated by multiplying the column count with sparsity.

Algorithm 2: SP - Phase 2

```
1 foreach col ∈ layer.columns do
2     if col.overlap > k_maxOverlap(layer.columns) then
3         col.active ← true;
4     else
5         col.overlap ← 0;
```

(a) Active columns before inhibition.

(b) Active columns after inhibition.

Figure 2.10: Resulting columns after applying WTA inhibition to the activation at Figure 2.9

2.3.2.3 Phase 3 - Synaptic Adaptation

In this phase, the proximal synapses of the active columns resulting after the inhibition are adapted according to the input to imitate the synaptic plasticity of pyramidal cells. It is important to note that the algorithm only adjusts the synapses of active columns and the rest remains the same. For all the potential synapses of a column, the synaptic permanence is incremented if the source input bit is active. Permanence is decremented otherwise. This phase introduces synaptic plasticity that biases the

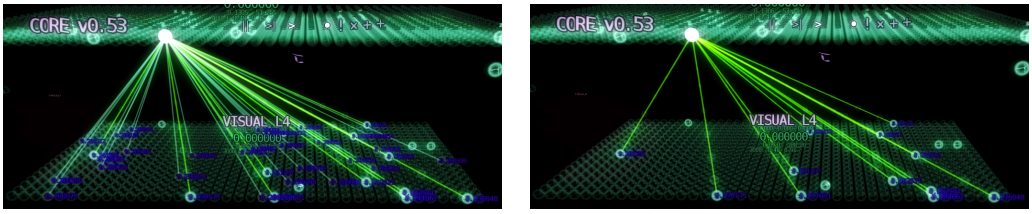
proximal synapses onto occurring inputs. It ensures that columns specialize to a group of input patterns over time. There are bumping and boosting mechanisms to imitate neural homeostasis. These provide a more uniform activation distribution among the layer.

- **Bumping** : All the proximal synapse permanences are incremented for the columns with lesser input overlap (*overlapdutycycle*). Thus, these columns are reinforced to have more connected synapses that can contribute to their overlap.
- **Boosting** : This mechanism scales the overlap score of every column with the column's boost value. This value depends on the activation frequency of the column (*activedutycycle*). Therefore, frequently activated columns have a lower boost value while lesser used columns have a higher boost value.

The activation (*adc*) and overlap duty cycles (*odc*) of a column are updated during this phase to bump the proximal synapses of the columns and compute the boost values for the next iteration's SP - Phase 1. Algorithm 3 presents the Phase 3 pseudocode and Figure 2.11 shows the synaptic adaptation of a proximal dendrite. *period* represents the moving average window of duty cycles with respect to iteration count. The function *minOdc* calculates the minimum overlap duty cycle among all the columns. Boost calculation had a recent update on the open source HTM implementation - Nupic. The up-to-date computation is an exponential function to provide a more uniform activation frequency distribution among the HTM layer. $boostStrength$ controls the intensity of leveling this distribution.

Algorithm 3: SP - Phase 3

```
1  foreach col ∈ layer.columns do
2      foreach syn ∈ col.proxDend.potentialSynapses do
3          if syn.active then
4              syn.permanence ←
                min(maxPermanence, syn.permanence + increment);
5          else
6              syn.permanence ← max(0, syn.permanence − increment);
7      col.odc ← (col.odc · (period − 1) + col.overlap)/period;
8      if col.odc < minOdc(layer.columns) then
9          foreach syn ∈ col.proxDend.potentialSynapses do
10             syn.permanence ← syn.permanence + increment · 0.1;
11     col.adc ← (col.adc · (period − 1) + col.active)/period;
12     col.boost ← exp(−boostStrength · (col.adc − localAreaDensity));
```



(a) Proximal dendrite before adaptation. (b) Proximal dendrite after adaptation.

Figure 2.11: A proximal dendrite adaptation example that results in substantial change in connected synapses.

2.3.3 Temporal Memory

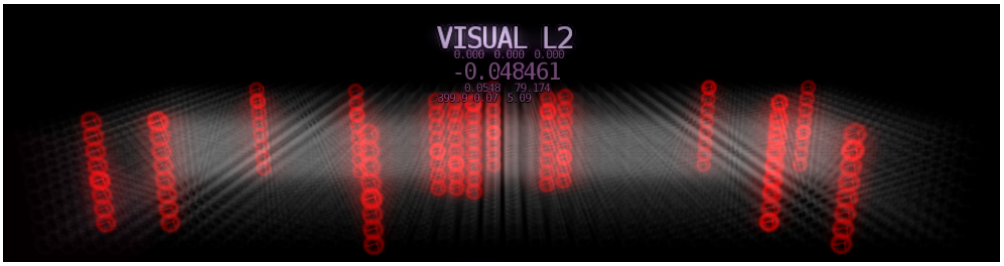
This stage picks which neurons to fire after computing the activated columns via the Spatial Pooler algorithm. By default, the distal dendrites of neurons potentially connect to the neighboring neurons of the same layer. These dendrites allow neurons to predict their activation by growing synapses to the previously active neurons. Therefore, a neuron is depolarized and put in a predictive state if its distal dendrites detect the activation that occurs before itself. The depolarized neurons are primed to fire in the following iteration if the feedforward columnar activation includes those neurons. If a depolarized neuron becomes active, it inhibits the rest of the neurons in the same minicolumn. All the neurons become active if there is no depolarization. The resulting neural activation computed on this stage represents the contextual information of the columnar activation. As a result, the same columnar activations may have different active neurons based on the previous activations. There are four Temporal Memory phases explained above that run sequentially in an HTM iteration.

2.3.3.1 Phase 1 - Neural Burst and Sparsification

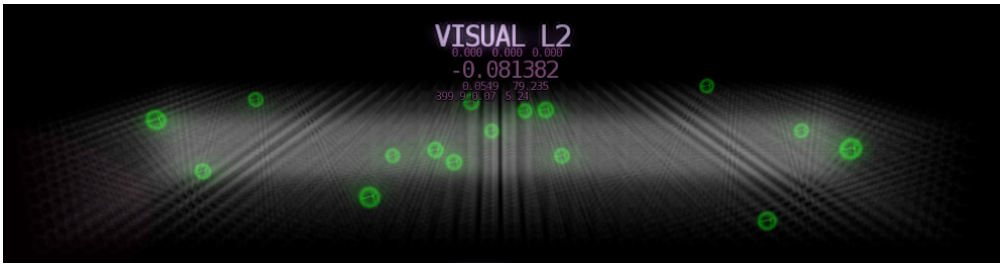
The first phase would activate all the cells of a column if that column did not have any depolarized cells from the previous iteration (bursting). If there are any depolarized cells of an active column, only those cells get activated, and the rest of the column is inhibited. Algorithm 4 presents the TM - Phase 1 pseudocode and Figure 2.12 shows the bursting and sparse neural activations.

Algorithm 4: TM - Phase 1

```
1 foreach cell ∈ layer.predictiveCells do
2     if cell.parentCol.active then
3         cell.active ← true;
4         cell.parentCol.predicted ← true;
5 foreach col ∈ layer.activeColumns do
6     if !col.predicted then
7         foreach cell ∈ col.neurons do
8             cell.active ← true;
```



(a) Bursting columns.



(b) Activation of depolarized columns.

Figure 2.12: Visualization of bursting and sparse neural activations.

2.3.3.2 Phase 2 - Learning Segments Assignment

The main objective of this phase is to assign learning segments to active cells. A learning segment is a flagged segment which is allowed for synaptic adaptation on TM - Phase 3. All the previously active segments of successfully predicted neurons are flagged for learning. For the bursting cells, the algorithm checks whether they have any matching segments. If the input overlap of the potential synapses (all synapses including unconnected ones) of a distal segment is over a set threshold with the previous neural activation, it is referred as a matching segment. This check is there to prevent creating new segments to learn on when there are segments with unconnected synapses that match with the input. For a given cell, the function *bestMatchingSegment* finds the matching segment that has the highest overlap with the previous activation. The resulting segment is chosen as the learning segment for that bursting cell. If there are no matching segments, a new empty segment is created via *createSegment* function and set as the learning segment. Algorithm 5 presents the TM - Phase 2 pseudocode and Figure 2.13 shows a matching segment flagged for learning which does not have any connected synapses.

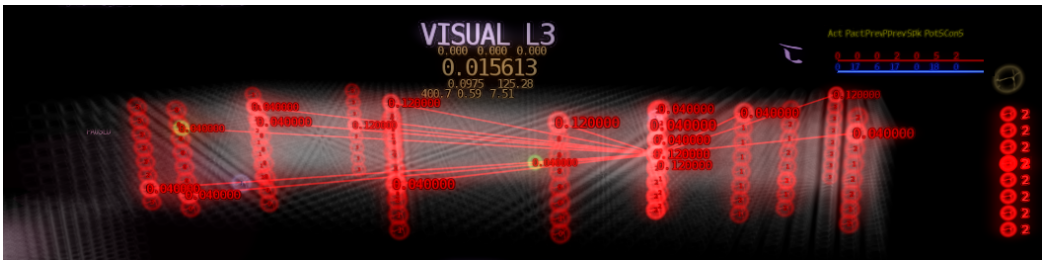


Figure 2.13: A learning matching segment that does not have any connected synapses.

Algorithm 5: TM - Phase 2

```
foreach cell ∈ layer.predictedCells do
    foreach seg ∈ cell.segments do
        if seg.prevActive
            seg.learning ← true;
foreach cell ∈ layer.burstingCells do
    bestSeg ← bestMatchingSegment(cell);
    if bestSeg
        bestSeg.learning ← true;
    else
        newSeg ← cell.createSegment();
        newSeg.learning ← true;
```

2.3.3.3 Phase 3 - Segment Adaptation

The synapses of the learning segments are adapted using the same logic in SP - Phase 3 with a single difference; the synapses sample the previous input activity, not the current activity. Therefore, a synapse is strengthened if the source was active in the previous iteration and weakened otherwise. Temporal Memory has its own permanence increment and decrement parameters that are different from Spatial Pooler. This phase also creates new synapses on the learning segments depending on the already sampling previously active inputs (*alreadySampled*) and the allowed number of new synapses per Temporal Memory iteration (*maxNewSynapses*). Among the unsampled previous active sources (*candidates*), random sources are picked via *popRandom* function and new synapses are formed on the segment sampling from these sources. Algorithm 6 presents the TM - Phase 3 pseudocode.

Algorithm 6: TM - Phase 3

```
foreach seg ∈ layer.learningSegments do
    foreach syn ∈ seg.potentialSynapses do
        if syn.prevActive then
            syn.permanence ←
              min(maxPermanence, syn.permanence + increment);
        else
            syn.permanence ← max(0, syn.permanence − increment);
    candidates ← difference(previouslyActive, alreadySampled);
    creationAmount ← min(candidateSynapses.size(), maxNewSynapses);
    for i ← 0 to creationAmount do
        seg.createSynapse(candidates.popRandom());
```

2.3.3.4 Phase 4 - Depolarization

This is the phase where the HTM layer outputs its prediction. The active neurons resulting from TM - Phase 1 stimulate all their postsynaptic targets. Every neuron of the HTM layer is iterated to check whether it can predict its own activity on the following iteration using the current neural activation. A neuron is depolarized if it has a distal segment with an input overlap that exceeds the *activationThreshold*. The depolarized cells represent the union of all neural activations that may happen at the next iteration based on the temporal context. Algorithm 7 presents the TM - Phase 4 pseudocode and Figure 2.3 visualizes the active cells and the resulting depolarized cells stimulated by them.

Algorithm 7: TM - Phase 4

```
1 foreach col ∈ layer.columns do
2     foreach cell ∈ col.neurons do
3         foreach seg ∈ cell.segments do
4             foreach syn ∈ seg.connectedSynapses do
5                 seg.overlap ← seg.overlap + syn.active;
6             if seg.overlap >= activationThreshold
7                 seg.active ← true;
8                 seg.parentCell.predictive ← true;
```

2.4 Temporal Difference Learning

In order to produce an autonomous HTM agent, the predictions of a layer need to be utilized in choosing the next behavior among many options. The majority of the basal ganglia computational models incorporate reinforcement learning methods at some level [34], [44]. Work by Samson et al. [45] presents the neurobiological basis of TD reinforcement learning methods through a review of neuro-computational models. The error between the expected long term reward and the real long term reward calculated by TD learning shows correlations with dopamine secretion of basal ganglia. In addition, according to [38], striatum has the ability to compare estimated reward and the actual reward when the action is performed.

TD can be described as an umbrella of algorithms that utilizes temporal difference to rewards when modifying the existing model. Sutton describes it as the combination of Monte Carlo and Dynamic programming ideas [46]. It does not need an exact model of the environment dynamics as Dynamic Programming and it can learn online without waiting for the episode to end as Monte Carlo methods. In the popular variations of TD, namely Q-Learning and Sarsa, the rewards are computed through the state-action couples which implicate states and actions are separate. However, research on cortical layers indicates that the output of a region is the state (neural activation) of layer 5 [22]. Therefore, it can be argued that the state is the action itself. It is highly unlikely for cortical regions to decouple actions from states. In addition, neurobiological studies linking reinforcement learning and dopamine modulation are focused

on the correlations with on-policy methods. As a result, the reinforcement learning algorithm decision is narrowed down to the on-policy versions where the states and actions are not decoupled, namely Temporal Difference (λ) which utilizes eligibility traces.

In TD(λ), each state has an eligibility trace based on its occurence recency. The decay of this trace is controlled by the λ parameter. The difference between the expected long term reward at time step t and the actual long term reward calculated at time $t+1$ represents the error signal in TD learning. This error is used for corrections on the state values based on their activation recency - eligibility traces. There have been a variety of improvements on online TD learning approaches. Initially, the traces were accumulative meaning that the trace of a state grew in accordance with its occurrence frequency. A further refinement [47] proposed the eligibility traces to be replacing. Whenever a state occurs, the algorithm sets its trace to 1; essentially replacing it. The results suggest that this refinement provides a faster and more reliable learning. Among the many proposed improvements, Van Seijen and Sutton [48] submitted their own in 2014 containing small modifications to the update rules and eligibility traces. They called this updated version True Online TD(λ). According to the paper, it provided better results in all cases compared to classical TD(λ) while still having the same computational complexity. Moreover, they presented further empirical results in their 2016 publication [49]. However, True Online TD(λ) is based on the theoretical forward view which is not practical for real learning applications compared to the backward view where updates are made on each iteration [48]. Thus, this study implements backward (mechanistic) view of TD(λ) with replacing traces.

2.4.1 TD(λ) Backward View with Replacing Traces

According to TD learning, every state has a value which represents a long term reward prediction. These state values are updated using the error between reward prediction at time t and the calculated reward value on $t+1$. Equation 2.1 shows the error δ calculation for time step $t+1$ where $V_t(s)$ is the value of state s at time t, r_{t+1} is the actual stimulated reward at $t+1$ and γ is the discount factor on the rewards. The discount factor makes the temporally further rewards less important and indirectly limits the maximum temporal distance that the agent considers.

$$\delta_{t+1} = (r_{t+1} + V_t(s) - \gamma V_{t+1}(s)) \tag{2.1}$$

If only the previous state value is updated, it would take multiple episodes for the reward to propagate backward in time. TD(λ) introduces an eligibility trace for every state in order to update all the recent states in a single iteration. The trace of a state is incremented every time it occurs and slowly decays over time otherwise. The λ parameter controls the eligibility trace decay. For example, TD(0) means that there is no decay on state traces. The updates are carried out for all of the states that occurred in the current learning episode. In other words, TD(0) behaves like Monte Carlo on every iteration. On the other hand, TD(1) only updates the previous state value because the traces decay completely in a single iteration as if there were no eligibility

traces. Equation 2.2 shows the update rule for the state traces $e_{t+1}(s)$ where γ is the discount factor. Trace of a state is replaced with 1 if that state was the previous state and it is decayed otherwise.

$$e_{t+1}(s) = \begin{cases} \gamma\lambda e_t(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases} \tag{2.2}$$

State updates are scaled with their respective eligibility traces $e(s)$, the learning rate α and the error signal δ_t as in Equation 2.3.

$$V_t(s) = V_t(s) + \alpha\delta_t e(s) \tag{2.3}$$

It is important to note that this section presented TD(λ) with the backward view in mind, which is also referred as the mechanistic view. The forward view describes it from a theoretical standpoint which is not practical for most implementation purposes.

CHAPTER 3

THE PLATFORM

3.1 The Problem

The objective of this study is to build a non-player character that could learn the surrounding three-dimensional environment by exploring and experiencing positive or negative rewards, similar to what a player does. Again similar to a player, online data from the environment is streamed into the limited sensors of the agent. The environment can be described as a Hidden Markov Model (HMM) from the perspective of the agent. The agent models the environment from the resulting visual output, not from the exact environment state itself. However, the agent has a limited control over the environment, therefore it runs similar to a partially observable Markov decision process (POMDP). On the other hand, HTM models sequences as variable order Markov (VOM) chains by encoding the temporal context in each neural activation. So the environment model of the agent constitutes of high order sequences. The limited sensor is utilized to observe the environment and to decide on the state transitions that optimize temporal distance to rewards. The learning scenario in this study is an example of navigating from point A to point B where point A is randomized each time the agent arrives at point B. There is a portal in the learning environment that the agent needs to find. Figure 3.1 showcases the scenario environment including the terrain topology and the target portal. The agent is stimulated with a positive reward signal when it goes through the portal and negative reward when it goes out of the pre-defined area. As this is an NPC architecture for a video game, player interaction is a critical component. We especially want the agent model to allow for player guidance in learning specific behavior sequences. For the learning to persist through multiple gaming sessions, it is necessary for the agent to be stored and loaded through serialization. The approach of Hierarchical Temporal Memory emphasizes the guidance of related neurobiological research. This study also shares this perspective and neurobiological research is a constraint for the problem at hand.

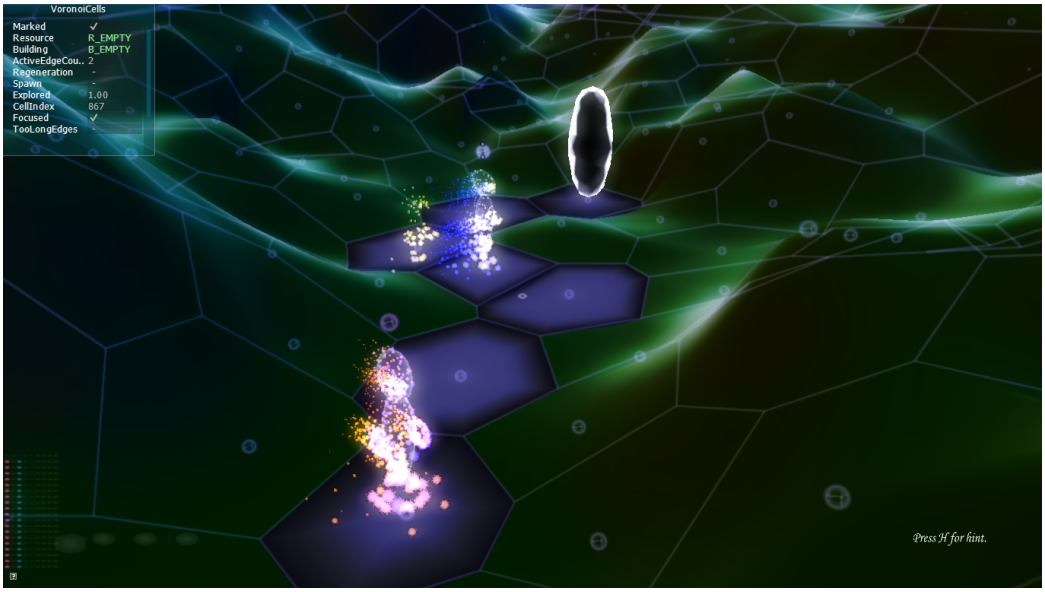


Figure 3.1: A learning scenario where the agent needs to get to the portal and respawns on a blue colored cell on success. The agent learns rewarding behavior sequences via modeling the environment through its visual sensor. Motor actions are produced based on the predictions of the model.

To summarize, below are the priorities of this thesis in building the NPC architecture:

- Online, continuous and lifelong learning
- Serializable architecture
- Real-time performance
- Supervisable by the player
- Directed by relevant neurobiological research

3.2 The Engine

The learning capabilities of the agent is observed in real-time in a three-dimensional virtual environment. The learning platform is generated through an in-house game engine that supports real-time physics, 3D graphics and has been designed around its artificial intelligence component from the ground up. The engine is written in C++ with Boost Library 1.63 and it uses Nvidia PhysX for its physical computations including particle physics. The rendering system utilizes DirectX11 and is entirely handwritten. AntTweakBar GUI library is used for real-time modification of general gameplay, terrain, artificial intelligence, animation, camera, rendering, and physics variables. A custom user interface is designed on top of rendering to precisely visualize a functioning HTM in real-time, and it is referenced as Core in this study. The Core communicates the state of HTM, shows synaptic connectivity, has behavioral

statistics for the agent, allows player interaction, has playback options and most importantly every single column, cell, segment and synapse can be visually debugged through it. Therefore, it is a real-time debugger for HTM and it is a core part of this study which is shown in Figure 3.2.

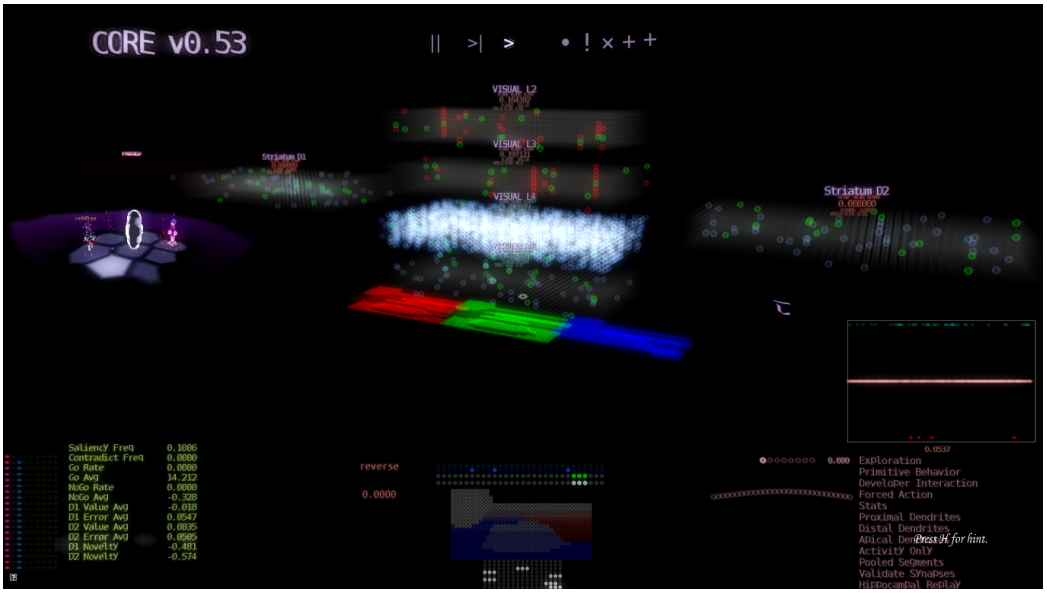


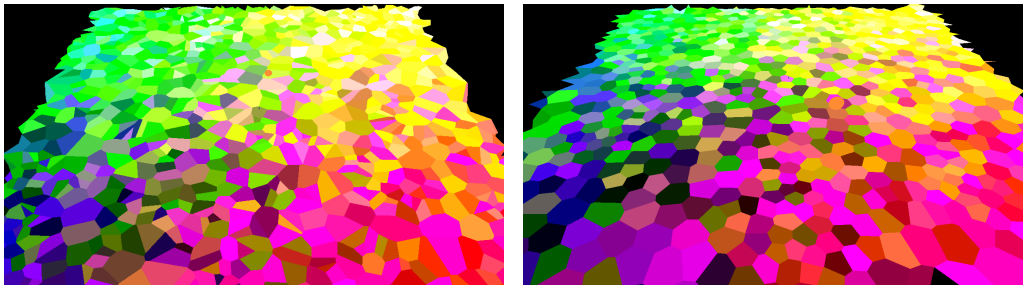
Figure 3.2: Visualization interface of the architecture - Core.

3.2.1 Virtual Environment

The agent traverses a terrain mesh constructed from random polygons which are generated procedurally from a Voronoi diagram. A Voronoi diagram creates a polygonal mesh from a set of random points [50] on a plane which is shown in Figure 3.3a. C++ Boost library is capable of generating a diagram out of given random points and allows the developer to access all the Voronoi cells and their edges. The resulting cells and edges are then converted into custom structures for compatibility with engine processing, game loop, rendering, and physics. The diagram is post-processed by Lloyd's Relaxation [51] to get rid of irregular polygons and relax the cells to create a more uniform look. The method moves the initial random points towards the center of mass of their corresponding Voronoi polygons. The Voronoi diagram is updated by these new points, and the relaxation can be done iteratively until the results are acceptable. In our study, the relaxation is applied iteratively four times on a newly generated Voronoi diagram. The relaxed diagram is shown in Figure 3.3b.

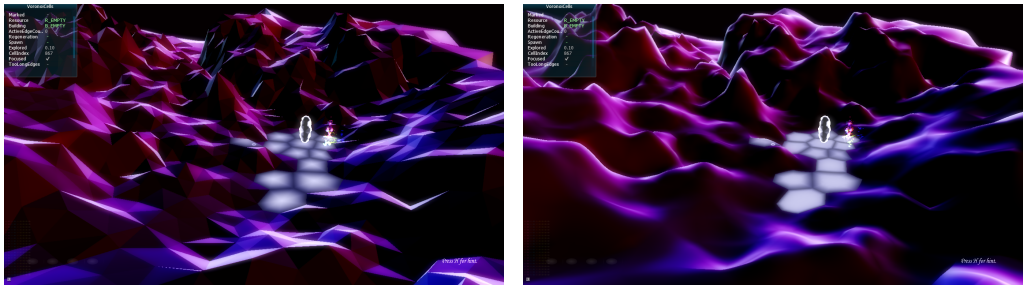
The polygons of this diagram can be sculpted to create any terrain structure, and Figure 3.4a shows an example. Nvidia PhysX takes the resulting polygons and creates a terrain mesh that collides with the agents. The physics geometry is a replica of the relaxed and sculpted Voronoi diagram. The terrain can even be tessellated afterwards for further smoothness as in Figure 3.4b.

The engine allows additional procedurally generated structures and resources on top of the Voronoi cells depending on the scenario. For example, in the environment shown in Figure 3.5, the agent can use the additional structures as visual landmarks



(a) Voronoi Diagram. (b) Lloyd's Relaxation.

Figure 3.3: A Voronoi Diagram is generated on the left. Polygons are colored randomly for easier separation. It is processed by Lloyd's Relaxation aftewards which is presented on the right.



(a) Sculpted Terrain. (b) Tessellation.

Figure 3.4: The height data of the Voronoi polygons are modified to sculpt the terrain which is presented on the left. The terrain is tessellated afterwards for further smoothness as shown in the right figure.

while modeling the environment or as dedicated spawn points after task success or failure.

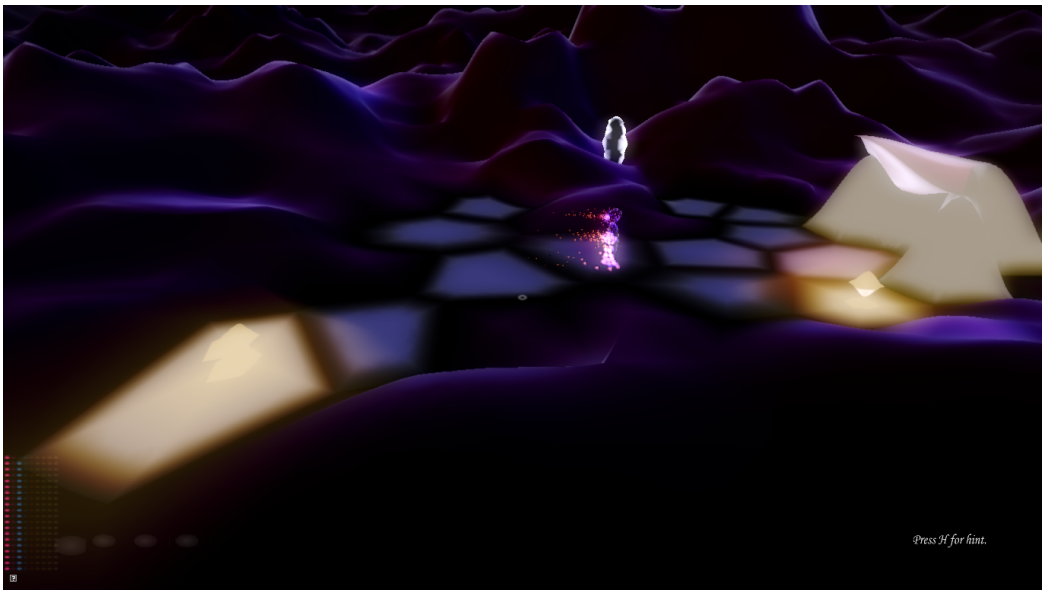
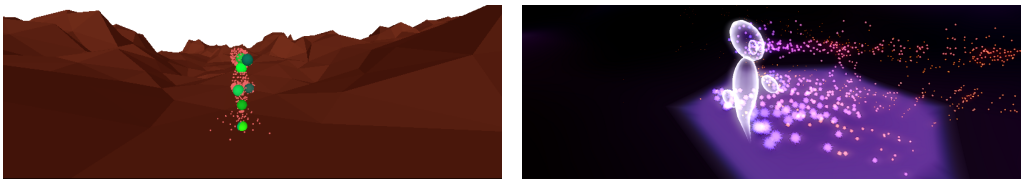


Figure 3.5: Tesellated terrain with additional structures on top.

3.2.2 Agent Body and Motion

The physical body of the agent consists of spheres with spring joints. Although the body formation is not directly related to the scenarios, five spheres correspond to the body parts of the agent as in Figure 3.6a. The spheres have spring joints between them to hold up a pre-defined shape while allowing the body shape to get affected by the motion of the agent. These spheres collide with the terrain. The agent navigates the terrain as if it is floating on top of it. The transparent 3D model of the agent deforms according to the movement of these spheres via a technique called vertex skinning shown in Figure 3.6b. The publication by Lewis et al. [52] forms the basis of vertex skinning which they call Skeleton-Subspace Deformation in their work. The agent also emits particles in real-time from its 3D model. The direction and velocity of the particles depend on a virtual wind fluctuation calculated by Perlin noise which was introduced by Perlin [53].



(a) Physical Agent Body. (b) Visual Agent Model.

Figure 3.6: On the left, the physical body is presented which consists of spheres with spring joints. On the right, the transparent visual model of the agent is deformed according to these spheres via the vertex skinning technique.

As the agent moves through the terrain mesh, it is stimulated with positive or negative rewards according to the collided geometry. If the agent collides with the spherical rigid body of the portal, it gets a positive reward and respawns on top of a random or pre-defined traversable Voronoi cell. The traversable cells and the goal portal are presented in Figure 3.1 as an example. If the agent leaves the area consisting of blue Voronoi cells, it is simulated by negative reward and respawns on a traversable cell.

The actual motor output of the agent is interpreted in 10 different ways by the physics engine: saccade left, saccade right, saccade up, saccade down, stay, move forward, turn behind, pickup, drop and use. The first seven motor actions allow the agent to navigate the environment. The last three enable the agent to interact with its inventory and the terrain. It can pick resources if there are any available on the focused Voronoi cell. It can also drop what is at hand or use what is at hand. Three motor neurons represent every action resulting in 30 motor neurons in total. The physics engine samples the activity of these neurons to decide on the motor action for that game frame.

In Figure 3.7 The motor neurons are represented as a vector, and the rows represent different states of the same motor neurons via coloring. The first row indicates the voluntary activity, the second row indicates the winning motor neurons (the neurons resulting in physical activity), and the third row shows the motor action and agent state mapping. A learned agent has the same activation on the third row and bottom row which indicates the neurons that were previously activated by the physics engine.

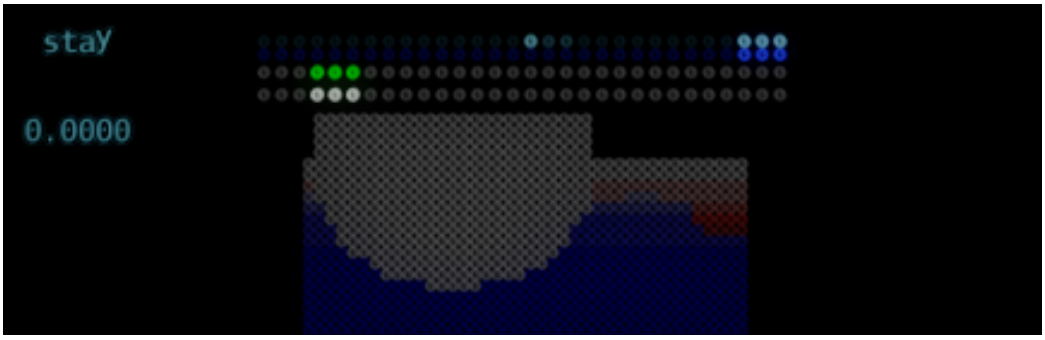


Figure 3.7: Motor neuron array on top and visual sensor image below. The top 4 rows of circles above the visual image represent the states of motor neurons via coloring. Circles on the same column (neurons on the same index of their row) are different colorings of the same neuron.

In this study, the agent is only capable of jumping from cell center to cell center depending on the facing direction. This type of movement in a Voronoi diagram results in variable navigational directions depending on the edge count of the current cell as opposed to fixed directions on a grid. So, this architecture can function on any type of terrain formation such as hexagonal, octagonal formations or polygonal meshes. The formation is there to limit the agent motion and to restrict the variety of incoming visual data that the agent learns on. There are no further constraints on agent navigation other than the terrain formation. The agent can rotate in a discrete fashion iterating over the neighboring Voronoi cells in both directions. It can go forward to the center of the adjacent cell in the facing direction. In Figure 3.8, the agent goes forward on frame 1, and it rotates left on frame 2. Then, it traverses to the center of the faced cell on frame 3 that it turned to on frame 2. Frame 4 shows the resulting position of the agent.

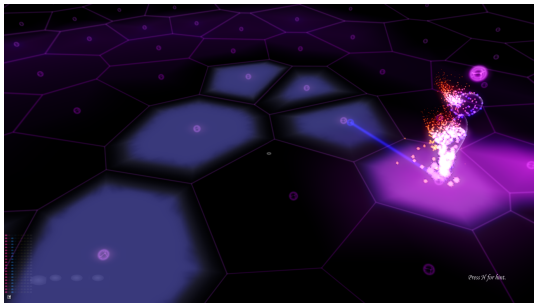
3.2.3 Online Data Stream

One of the advantages of HTM is that it learns continuously from an ongoing data stream, again functionally similar to neocortex [54]. The biological plausibility of the encoded information on the stream is currently out of the scope of HTM. For practical purposes, the majority of HTM research is conducted on n dimensional binary data streams.

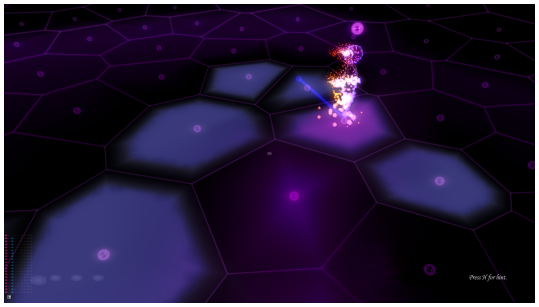
3.2.3.1 Visual Sensor

The partial observations of the agent come through its real-time visual sensor. Continuous and online learning is central for neocortex [54], so the visual sensor updates itself in every game loop iteration. The visual information is generated through ray-casting the virtual environment from the point of view (POV) of the agent as seen in Figure 3.9.

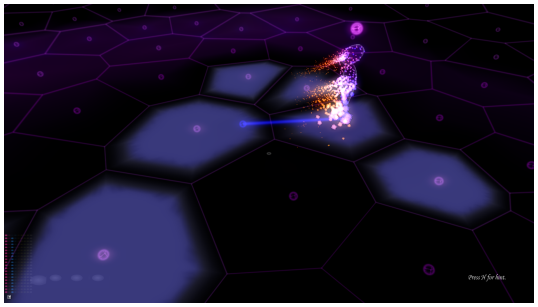
By default, there are 20 vertical and 40 horizontal rays covering 60 degrees of verti-



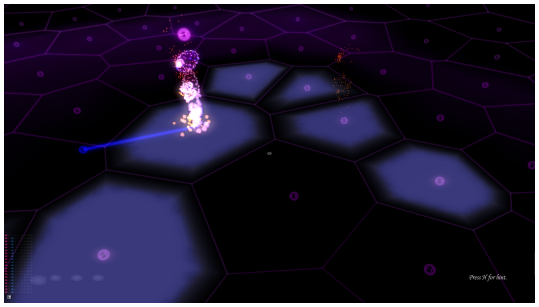
(a) Frame 1, forward.



(b) Frame 2, rotate left.

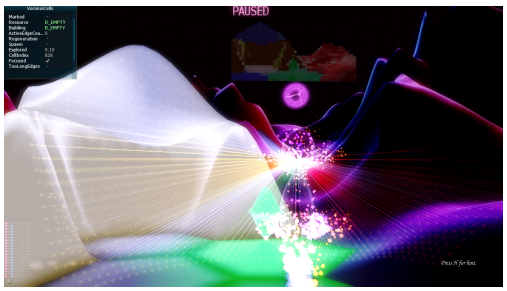


(c) Frame 3, forward.

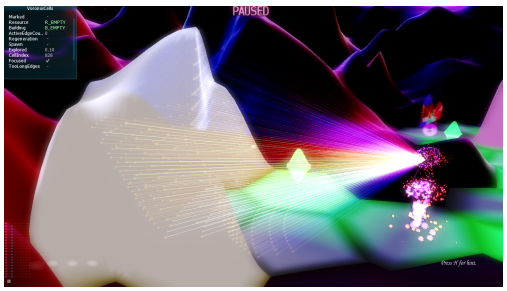


(d) Frame 4.

Figure 3.8: Sequential frames showcasing cellular movement. The agent jumps from cell center to cell center depending on the facing direction denoted by the blue line.



(a) Raycasting seen from head on.

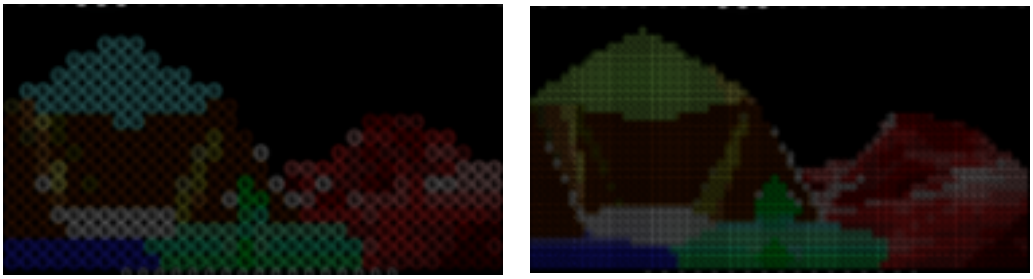


(b) Raycasting seen from side.

Figure 3.9: Raycasting from agent's point of view. 48x24 rays covering 120 degrees of horizontal and 60 degrees of vertical field of view. A transparent version of the constructed image is on top of the agent icon.

cal field of view (VFOV) and 120 degrees of horizontal field of view (HFOV). These parameters are fully adjustable, and the default values are chosen according to the default column size of an HTM layer which is 20x40. Nvidia PhysX raycasting function provides the color information on the points of intersection between the rays and the environment. The color values are then mapped onto a two-dimensional plane for the image construction. Every pixel of this image corresponds to a point of intersection between a ray and a triangle belonging to the game geometry. The exact color information is the average color of the three vertices owned by the triangle that the ray is intersecting with. In Figure 3.10, the same scene is raycast with varying resolutions, and resulting images are presented.

The resulting picture is a low-resolution rendering taken from the POV of the agent.



(a) Raycasted 32x16 image.

(b) Raycasted 64x32 image.

Figure 3.10: Constructed visual sensor images from raycasting with varying resolutions.

Although this method has major aliasing problems when coupled with low ray density, it is sufficient for the learning scenario covered in this study. The resolution of this sensor is increased by casting more rays if the task at hand requires it. However, HTM uses the information encoded from this image, so the dimensions of the visual sensor affect the speed of learning and size growth of the synaptic connectome. The default image dimensions map to the size of an HTM layer for simplicity, but there is no dependency between the sensor dimensions and HTM layer size.

HTM layers take their inputs from proximal dendrites of their columns. The Spatial Pooler algorithm carries out this process as explained in Section 2.3.2. The algorithm accepts input in n dimensional binary encoding for implementation practicality. Therefore, the color values of the pixels are converted into binary volumetric color channels as seen in Figure 3.11. For general compatibility with digital environments, there are three color channels - red, green and blue. For example, red color channel layer visualizes the red intensity for every pixel in the sensor. The constructed image of visual sensor and color channel layers on top are visualized topologically for clarity. The height of the pixels in color channels represents the intensity of the color value at the respective position in constructed image.

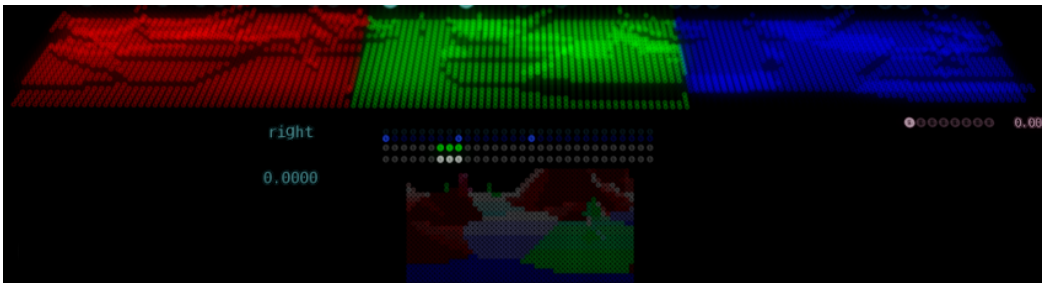


Figure 3.11: Volumetric red, green and blue color channel representations of the raycasted image.

The volumetric channel layers encode the color values into a binary format with discrete steps. Currently, intensity values are within the interval [0.0, 1.0] and have a step size of 0.2. Therefore, there are five intensity steps, three color channels and every color channel has 20*40 pixels resulting in 24000 possible color patterns on the visual sensor. Capacity and fidelity of the visual sensor can be adjusted for the tasks that require it. However, the topological properties of the potential fields belonging to

the layer 4 proximal dendrites are also crucial for interpreting the information on the visual sensor. Figure 3.12 shows an example topological connection between color channels and layer 4. To summarize, the color values of the sensor get updated in real time, encoded into binary and used as input for the feedforward proximal dendrites of layer 4 in every iteration. The information provided by the visual sensor is the primary cause for any columnar activity and the resulting neural activity. Therefore, all the activity in HTM diminishes in time after the stream at the visual sensor stops depending on the pooling parameters of HTM layers. The layer 4 does not have any input pooling, so the columnar activity is directly linked to the activity of the stream. In time, proximal dendrites of layer 4 specialize in specific patterns that occur in color channels via Spatial Pooler algorithm.

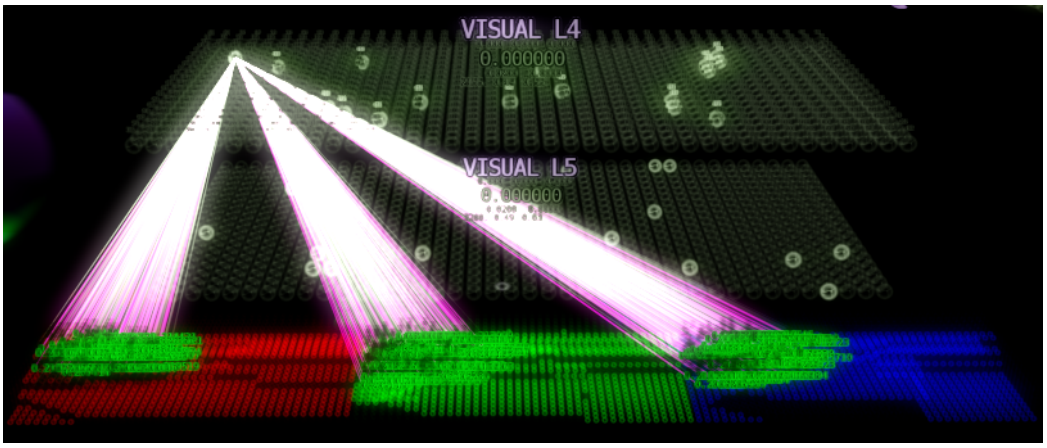


Figure 3.12: The topology between layer 4 and color channels. The layer 4 mini-columns sample the visual data according to their position among the layer.

3.2.3.2 Haptic Sensor

One of the experimental scenarios shown in Figure 3.13 involves interacting with the virtual environment by extracting the resource from a particular Voronoi cell and consuming it to get a reward. This task requires the agent to have an inventory and a simple strategy to sense what is at hand and what is on the ground. For this reason, various types of haptic sensors are implemented to sense the resources and relevant entities nearby to interact with them.

Experiments are conducted through various haptic sensor structures. We attempted to topologically stream information from neighboring cells (Figure 3.14a), cascading the information based on proximity and direction, and collision based data access. While each sensor type has its advantages, the fundamental problem with the scenario is coupling the separate HTM regions dedicated to haptic and visual sensors. This introduced further complexities on voluntary actions since the addition of a new sensor allows additional actions from the combinatorial sensor. It is problematic to make both of the separate regions shown in Figure 3.14b function simultaneously with the reward circuitry - the mechanism deciding which representations to learn and which ones to invoke given an HTM state.

In its current state, the architecture is not mature enough to handle a separate haptic

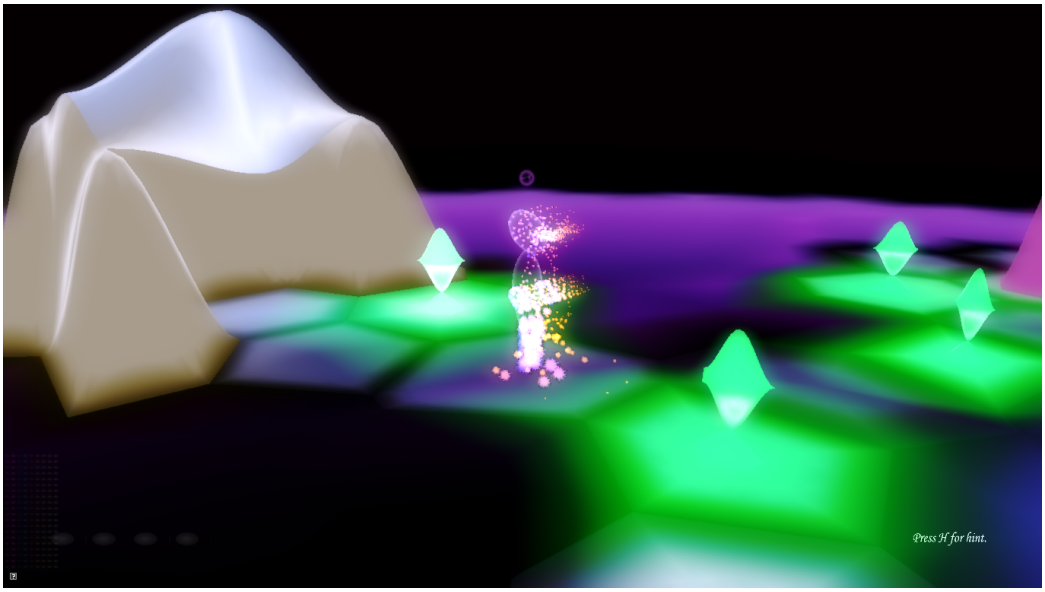
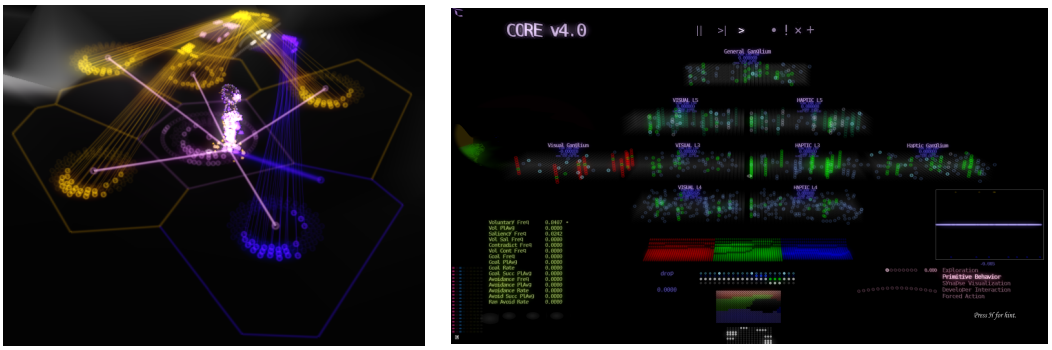


Figure 3.13: The experimental scenario that requires the agent to gather green resources.



(a) Haptic Sensor.

(b) Dedicated Sensory Regions.

Figure 3.14: A custom haptic sensor streaming data from adjacent cells is presented on the left. The dedicated haptic and visual HTM layers are shown on the right.

region. The dedicated region solution is suspended until it is suitable for simultaneous regions. A better strategy for the current architecture might be to embed visual information about the agent inventory to the existing visual sensor, similar to a video game heads-up display (HUD).

3.3 The Core

The scope of the study requires a real-time visual debugger for the agent architecture and in particular the Hierarchical Temporal Memory state. The debugger is built as a user interface on top of the rendering system, and it is the core of this study, hence the name. Implementation-wise it consists of a bitmap font system, circle sprites, and lines. The primary goal of the Core is to visualize the state of the HTM in real-time and in 3D where the layers can be rotated, zoomed and translated via mouse

commands. Figure 3.15 visualizes the proximal synapses that activate their targets along with the distal and apical synapses that depolarize theirs. The Core can show the synaptic connectome of all the layers or a selected layer. There are options to filter specific dendrites among apical, proximal and distal types for clarity. The actual game world is still rendered onto a small circular space at the left side of the interface to keep track of what the agent is doing.

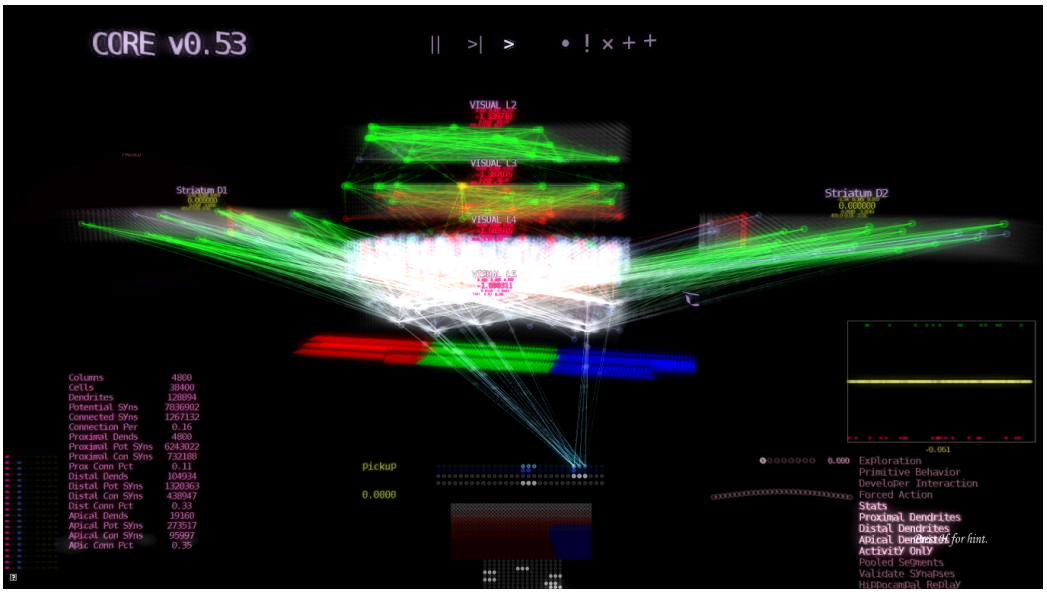
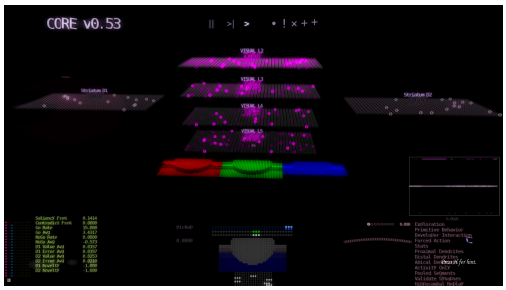
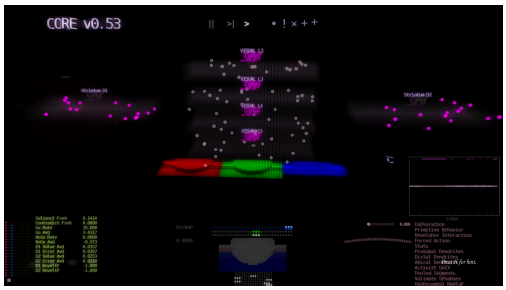


Figure 3.15: The synaptic connectome visualization via Core.

The activity visualization has two main composition types shown in Figure 3.16: columnar and neural. The columnar view represents a higher level and cleaner view of the layers by just drawing a single circle for each minicolumn of neurons. On the other hand, neural composition allows communicating the finer details through debugging activity at a neural level.



(a) Columnar view.

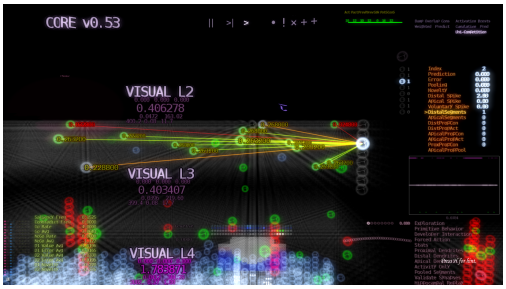


(b) Neural view.

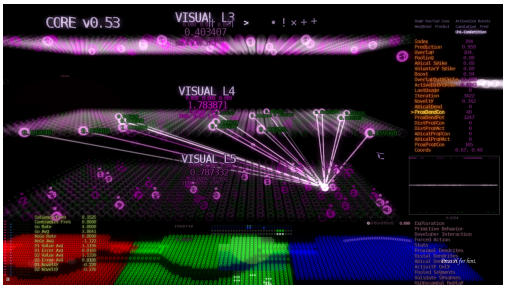
Figure 3.16: Visual composition types of the core. Columnar view represents all the neurons of a minicolumn as a single circle while neural view shows every neuron.

Core is capable of visualizing the properties of every synaptic node (column or neuron) in real-time by a selection mechanism as in Figure 3.17. It can represent all the dendrites belonging to the selected synaptic node. In Figure 3.18, there are various coloring templates to showcase activity, distal depolarization, apical depolarization, voluntary depolarization, and prediction results including false positives, false nega-

tives, true positives. Furthermore, it can visualize not only the synapses of a selected node but also the synapses that sample from that particular node. In other words, the Core is capable of visualizing all the incoming and outgoing synapses to every synaptic node including to display their permanence (connection strength).

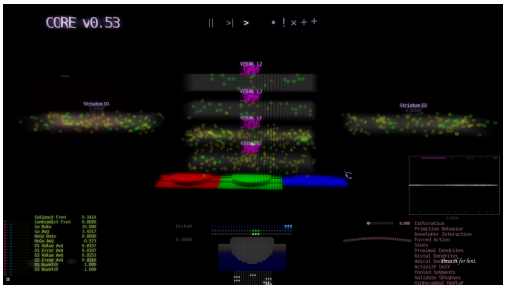


(a) An active distal segment of a layer 2 neuron sampling from nearby neurons.

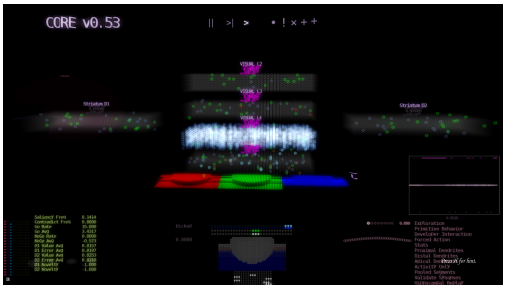


(b) The proximal dendrite of a layer 5 column sampling from layer 4.

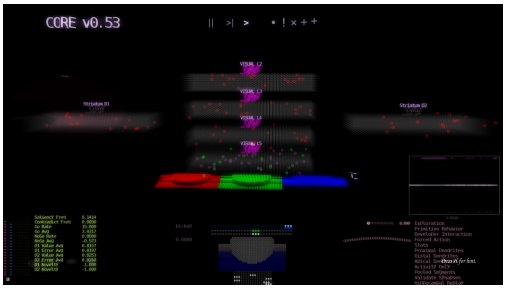
Figure 3.17: Synaptic node and dendrite visualizations.



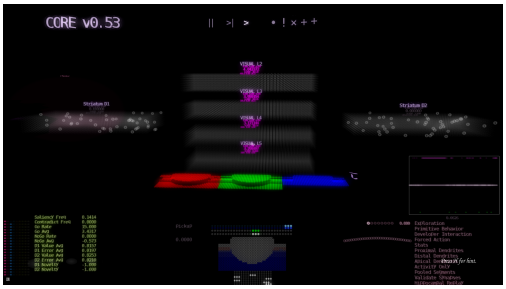
(a) False predictions scheme.



(b) Depolarizations and voluntary activations.



(c) Apical depolarization scheme.



(d) Pooled (recent) activation coloring.

Figure 3.18: Core information colorizations. Green = successful prediction, yellow = false prediction, red = unpredicted activation, blue = predictive, purple = apically predictive, cyan = go voluntary active, orange = nogo voluntary active, white = neurons with nonzero eligibility traces.

The debugger also has playback capabilities that allow pausing, single iteration, continuous learning and breaking on triggered events such as rewards. The Core also computes real-time information about the architecture regarding segment and synapse counts including their connectivities which are displayed at the left bottom corner of the interface as in Figure 3.2. Additionally, it has a testing mechanism to identify duplicate and invalid synapses via the validate button by crossreferencing all the incoming connections with outgoing mirror connections. This testing is crucial to validate the architecture while it is running and also after serialization. There are also

behavioral statistics about the agent presented with data related to the characteristics of the recent voluntary behavior and with a moving average reward graph. At the bottom of the Core interface, the motor neurons, the visual and haptic sensors are displayed.

3.4 Serialization

In video games, non-player characters are either generated newly or loaded from the stored save files when initializing a level. For an NPC to be persistent between gaming sessions, it has to be serialized into files. The agents in our study can learn behavior sequences in an unsupervised fashion. Players can also teach them specific actions. Including both cases, the agent adapts its synaptic connectome throughout its lifetime. Serialization capability allows the agent to be stored and restored with that same learned synaptic connectome at any time which removes the limitations imposed by program lifetime. As a result, lifelong learning without time limitations is made possible.

Boost Serialization library with the binary stream configuration is utilized to serialize the HTM at any given time. The connectome of HTM may be stored or the agent may be loaded with a different HTM at any time and anywhere through the AntTweakBar UI, as in Figure 3.19.

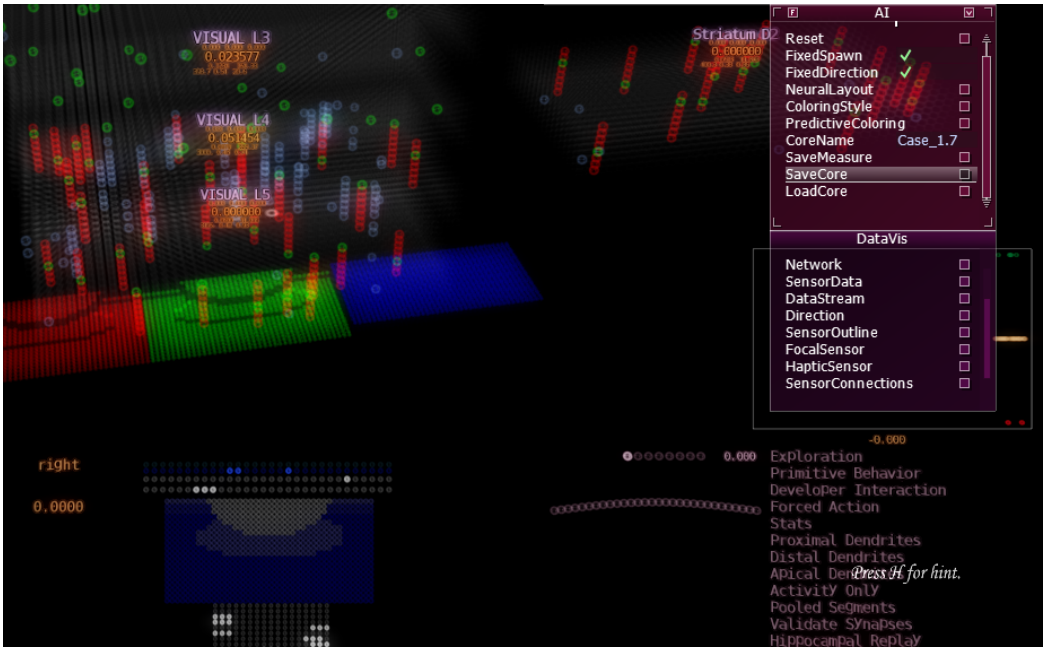


Figure 3.19: Architecture save and load user interface.

It is important to note that the serialized file does not store the columnar and neural activations. Any temporal activation would be invalid unless the agent is restored at the same spot and looking at the same direction while ensuring the environment is also in the same state. Not including any activity in the file also improves the save/load speeds. Moreover, the file becomes more compact because of the reduction in data

which is necessary to encode synaptic nodes and synapses. Synaptic information takes up the majority of the file size. This information is compressed rather than being stored as an exact copy to reduce the file size. Below is the compressed synapse structure with a size of 4 MBytes:

- **unsigned short sourceIndex** : The index to the presynaptic target (source).
- **unsigned short shortPerm** : The permanence of the connection which is converted to a floating point value when loading.

The compression ruleset is the following:

- Serialization discards any temporal and spatial activity.
- Serialization stores only the independent variables that cannot be generated from other data.
- The source and target addresses of a synapse that are stored in synaptic pointers are converted to numerical indices with a custom logic. This results in a pointer free synapse serialization which prevents further performance overhead and minimizes memory footprint.
- Some floating point variables such as synaptic permanences are stored on 1-byte char or 2 byte short data types rather than as is. The library is significantly slower on serializing floating point values.

The serialization measurements on file size and execution times with respect to synapse counts and varying HTM layer sizes are presented on Chapter 6.

CHAPTER 4

AUTONOMOUS AGENT

4.1 General Overview

The previous chapters introduced the main algorithmic components of the agent and the platform it interacts with. This chapter focuses on the internal design of the agent utilizing its sensors to navigate and model the environment. Information is presented in a top-down fashion expanding the details of the agent incrementally. The exact implementational details are laid out in the next chapter as this one is more about the mechanistic explanation. The agent framework is a simple computational cycle at its core which is presented in Figure 4.1. Partial environment state is fed into the sensor and then a suitable action according to the sensory information is executed. As a result, the incoming data into the sensor changes to something that the agent prefers more in terms of temporal distance to rewards.

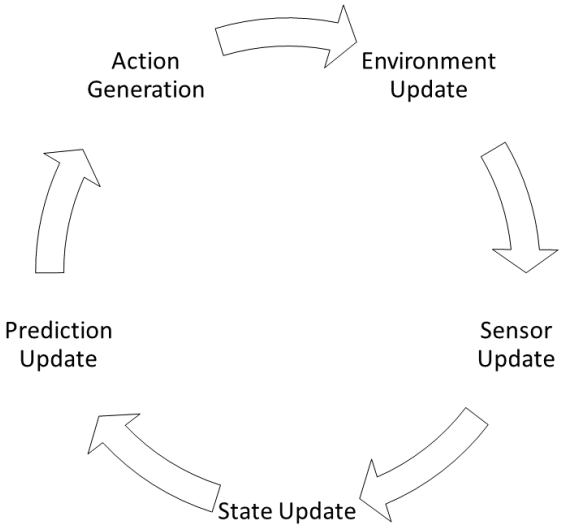


Figure 4.1: Computational cycle of the agent.

The following sections describe the incremental stages of building an autonomous agent. The outline is below:

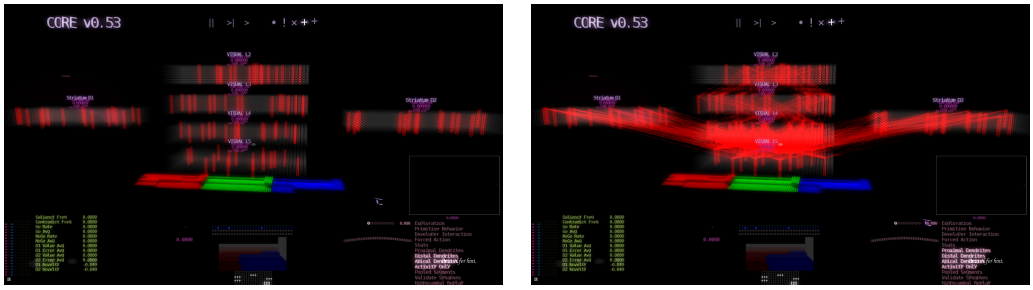
- Introducing sensory motion to facilitate learning

- Representing states
- Generating predictions based on the environmental model constructed from the sensory stream
- Associating actions with states
- Assigning values to states
- Using the state values to bias predictions
- Generating the required actions to realize the biased predictions

4.2 Sensory Motion

The first stage is to create sensory motion to facilitate learning. Learning in our study refers to the agent altering its synaptic connections to capture and control the change in its sensors. Unless there is motion in the environment other than the agent itself, the data on the visual sensor are static, and there is nothing to learn. Therefore, the agent needs to have some motion to encounter any sensorial change and to learn the changes it prefers. Biological mechanism of homeostatic plasticity allows neurons to stay at a useful activity level by modulating the activation thresholds [55]. This modulation provides spontaneous activations that may result in noisy motor activity. Thus, we introduced random movement for the exploration to start and sensory data to change. The mammalian neocortex starts learning sensory patterns even before birth [56]. Sensory changes due to the environment and due to the motion of the mother allow learning to happen at the neocortex. Therefore, the biological starting point for the neocortex learning is fuzzy, but the learning is facilitated by sensory change. The agent behaves randomly unless overridden by any voluntary actions in our study. The excited motor neurons that produce action are represented with dark blue (motor neurons, second row in Figure 4.2). In the absence of a voluntary excitation or inhibition of motor neurons (second row in Figure 4.2), random ones are excited to generate motion.

Regarding the initial connectome, the layers start with random topological proximal synapses between them to facilitate feedforward columnar activation. On the other hand, the agent has no distal and apical synapses which modulate the neural activity at the start. Therefore, all the neurons of active columns burst in the first learning cycle as in Figure 4.2.



(a) Bursting cells of the first frame.

(b) Initialized proximal synapses for columns.

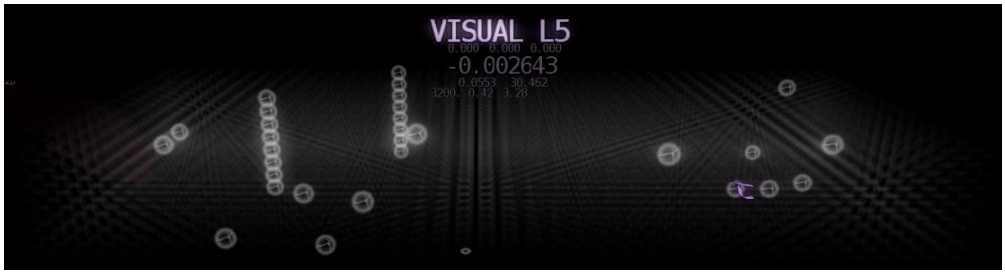
Figure 4.2: On the left image, we enabled the visualization of apical and distal synapses but there aren't any in the first learning frame. On the right, all the synapses are visualized including the proximal ones. The connection points of proximal dendrites are at the top of the columns.

4.3 Agent State

The state of the agent is represented by the cellular and columnar activations of the layers at a given time. Although the activation of every layer contributes to the state of the agent, the layer 5 alone defines the state as in Figure 4.3. A single layer has a finite capacity for state representation. The equation 4.1 represents the capacity of a layer R, based on the column count c, neurons per column n and the activation sparsity s.

$$R(c, n, s) = \begin{pmatrix} c \times n \\ c \times n \times s \end{pmatrix} \tag{4.1}$$

(a) Columnar activation. Each minicolumn of neurons is represented with a single circle.



(b) Neural activation showing all the neurons.

Figure 4.3: Activation representing the current state of layer 5.

In this framework, layer 5 encapsulates all the information from the other cortical and striatum layers through its proximal, distal and apical connections [22]. Layer 5 is the

primary output layer of a cortical region sending information to other subcortical and motor regions [20], [23]. Moreover, there is a learning mechanism in our architecture which associates the activity of layer 5 with motor neurons by growing synapses between them. This process indirectly maps layer 5 states onto agent actions. Therefore, layer 5 activity represents both the state and the action [24]. The agent architecture is presented in Figure 4.4.

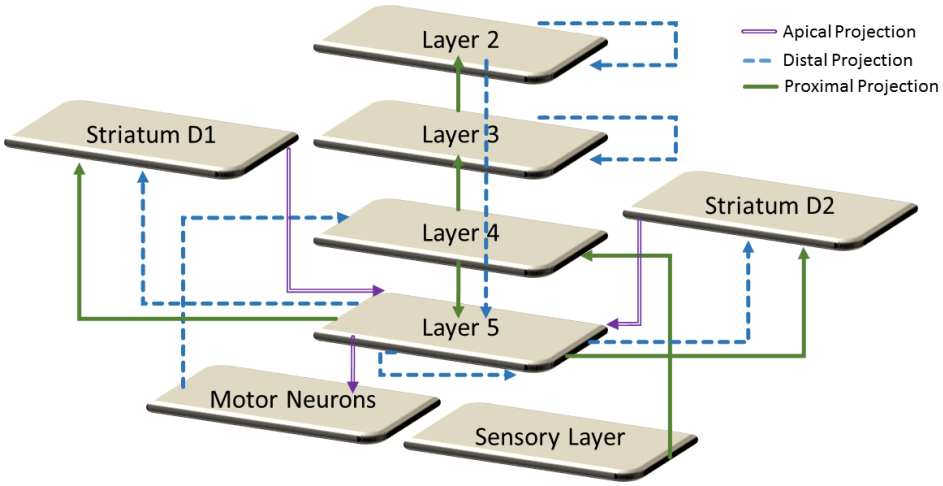


Figure 4.4: Information flow of our agent architecture. Layer 5 integrates information from most of the cortical layers.

4.4 Predictions On Sensory Stream

To change the current state to a preferable one, the agent needs to have a model of the environment. From the perspective of the agent, temporal and spatial patterns on the sensor are the environment itself. Control over these patterns is only possible through predicting which sensory patterns may occur in the following operation cycles. In other words, it is necessary for the agent to predict its potential next states and this is the premise of Hierarchical Temporal Memory. For every state, HTM can flag the next possible states by depolarizing the cells representing those states as explained in Section 2.3.3 Temporal Memory algorithm. HTM layer builds up a sequential model of the streaming data by sampling the previous activation and connecting it with the current activation through forming synapses. Activation at time step t depolarizes the potential activations at time step $t + 1$ and these depolarizations constitute the pool of the possible states that can happen in the following iteration as shown in Figure 4.5.

Figure 4.5: Current activation along with the predictive (depolarized) neurons of layer 5. Green = active, blue = depolarized.

4.5 Motor Learning by Association

Modeling the environment also includes capturing the relationship between motor actions and the sensory change. In our framework, layer 5 is the output of a region and it represents both the state and the action. Biologically, this output reaches to motor areas, even down to spinal cord [23] as well as the basal ganglia via corticostriatal connections and other subcortical structures. Therefore, at a high level, layer 5 maps the states to actions and it does so by learning by association. When a particular layer 5 activation happens, it forms connections with the motor neurons that led to this particular activation. In other words, the state maps itself to the action that results in itself. This mechanism allows indirectly invoking the necessary motor behavior via activating the target state. The agent can just invoke that particular state which causes the necessary motor activation. Motor neurons associating themselves to layer 5 state through apical connections are presented in Figure 4.6.

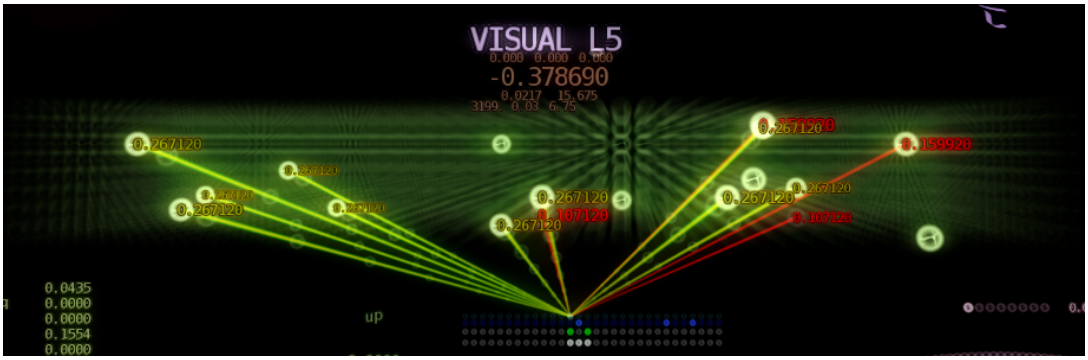


Figure 4.6: Through its apical denrites, the motor neuron at the 13th index of the row associates its activation with the current layer 5 activation above.

In Figure 4.6, the motor activity that is mapped to the current layer 5 state is represented in the third row (green coloring) and the motor neuron activity required for the previous action is represented in the fourth row (white coloring). The previous action of the agent (white) and the layer 5 associated action (green) overlap with two motor neurons which show the association completeness. In other words, the overlap represents how the learning by association fits the actual action. The individual neurons of layer 5 are connected to the individual motor neurons. This is a many-to-many connection type because different actions may be mapped to similar states and similar

actions may be mapped to the same states during learning. A single neuron of layer 5 is part of multiple agent states, so a single layer 5 activation may be associated with multiple motor actions as seen in the Figure 4.7. It is important to note that as the learning continues and the agent experiences more, representations of layer 5 become more stable and specialized leading to a better separation of associated actions.

Figure 4.7: Current layer 5 activation is associated with 9 different motor neurons represented by white in the 3rd row.

4.6 Reward Circuitry

Producing behavior is possible through manipulating the activity at layer 5 in our framework, which is directly connected to motor neurons. The reward circuitry layers D1 and D2 in Figure 4.4 have the ability to depolarize neuron groups and indirectly cause activations on layer 5, which lead to the corresponding motor behavior. Biologically, the activity of layer 5 is modulated through its apical tufts reaching layer 1. A cortical region is influenced by basal ganglia through its layer 1 via direct (Go), indirect (No-Go) and hyperdirect pathways [25]. Figure 2.2 shows the pathways interconnecting thalamus, frontal cortex and basal ganglia.

The studies indicate that basal ganglia specialize in action selection, conflict resolution [32] and it is the main component of the reward circuitry through dopamine receptive cells which control learning of behaviors [33]. Basal ganglia work in conjunction with thalamus and cortical layers to regulate the motor activity. The direct pathway described as *Go* increases the motor activity while the indirect pathway described as $No - Go$ suppresses it. There is a third pathway outlined on some of the computational models [34] called the hyperdirect pathway, which suppresses all the motor activity until appropriate action is resolved [40], [39], [38]. During this suppression, the striatum is influenced by information from other cortical regions [41]. Since there is a single cortical region in our architecture, the functionality of hyperdirect pathway is currently not included. The studies show that the imbalance between these two pathways lead to inconsistencies in behavior such as Parkinson's Disease and attention-deficit hyperactivity disorder [37], [38] because motor actions result from the competition between these pathways.

In our framework, there are two primary layers influencing layer 5. These are called D1, D2 layers and they are named after the striatal neurons that have specialized

receptors detecting D1 and D2 dopamine levels [22]. D1 dopamine receptive cells take part in the Go circuitry reinforcing the actions while the D2 dopamine receptive cells take part in the No-Go circuitry suppressing actions. D1 and D2 layers sample from layer 5 through their distal and proximal dendrites shown in Figure 4.4. In other words, both these layers do spatial and temporal classification on layer 5 activations to generate predictions. Therefore, the depolarized cells of layers D1 and D2 correspond to activations of layer 5 that might occur in the next cycle.

The computational models suggest that the Go pathway inhibits the inhibitory neurons at thalamus which increases motor excitation indirectly. On the other hand, No-Go pathway excites the inhibitory neurons of thalamus acting on the cortical regions and motor areas to suppress behavior [34]. In our framework, layers representing the D1 and D2 receptive neurons function in competition with opposite learning rules as suggested by Prescott et. al. in [36]. As explained in Temporal Difference Learning Section 2.4, the difference between the expected state value from time t and the actual long term reward value at time $t+1$ is the error signal. This signal controls the adaptation of apical dendrites of layer 5 sampling from D1 and D2. In practice, this means that layer 5 apical dendrites sampling from D1 are strengthened when the error signal is positive (unexpected positive reward difference) and weakened when negative. For D2, synapses are strengthened when the error is negative (unexpected negative reward difference). In short, layers D1 and D2 have opposite error signs. Equations 4.2 and 4.3 denote the increment and decrement formulas for layer 5 apical synapses sampling from D1 and D2.

$$ApicalAdaptInc(type) = \begin{cases} error \times TD\_LEARNING\_RATE & \text{if } type \text{ is D1} \\ -error \times TD\_LEARNING\_RATE & \text{if } type \text{ is D2} \end{cases} \tag{4.2}$$

$$ApicalAdaptDec = |error| \times TD\_LEARNING\_RATE \times 2.0 \tag{4.3}$$

There is a cyclic information flow between layer 5 and layers D1, D2 in our architecture as seen in Figure 4.8. Outline of this flow is given below:

- D1 and D2 are stimulated through distal and proximal dendrites with the current layer 5 activation.
- Through Spatial Pooler and Temporal memory algorithms outlined in HTM Section 2.3, proximal and distal dendrites are adapted accordingly to predict and classify layer 5 activations better.
- The apical dendrites of layer 5 adjusts to the previous D1 and D2 activations according to the error signal. This adaptation allows the layer 5 apical dendrites to learn the D1 and D2 activations which result in a difference between expected and the actual reward.

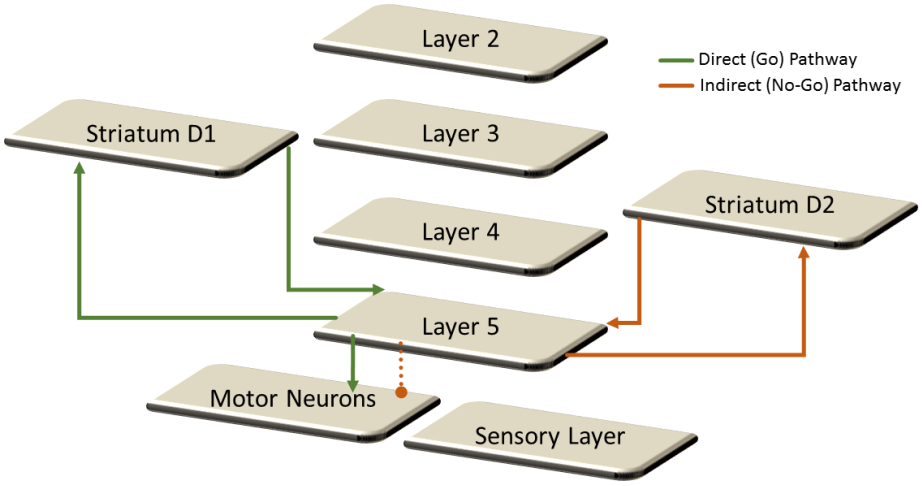


Figure 4.8: Abstract reward circuitry pathways in our framework. Note that all the connections between layer 5 and striatum layers are excitatory. Layer 5 inhibits and excites motor neurons depending on those excitations.

First, layer 5 stimulates D1 and D2, then D1 and D2 modulate the depolarization of layer 5 through its apical dendrites. At the end of each cycle, there are two types of depolarized neurons at layer 5 showcased in Figure 4.9. The first group is depolarized by distal dendrites originating from layers 2/3 and 4. This group represents the possible layer 5 activations at the next cycle due to current layer 2/3 and 4 activations. The second group is depolarized by apical dendrites sampling from layers D1 and D2. This group represents the possible salient layer 5 activations that are learned by reward circuitry layers D1 and D2 based on the error signal. In other words, the latter group represents the union of wanted and to be avoided layer 5 activations which are utilized in producing voluntary behavior explained in the following section.

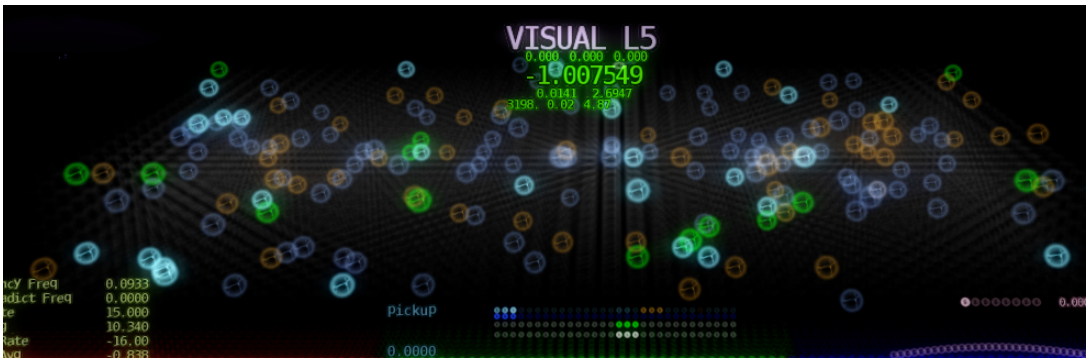


Figure 4.9: Orange and cyan colored layer 5 cells are activated by D1 and D2 layers. Activation of orange cells originate from D2 and these cells are inhibitory on the motor neurons. The cyan cells are excitatory on motor neurons and their activation originates from D1.

Layer 1 of a cortical region is the central hub for information coming from higher cortical regions of the hierarchy and other subcortical structures including basal gan-

glia [25]. There is only a single region in this study and a single structure managing this region, so layer 1 and the apical dendrites of layer 5 sampling from layer 1 [27] are not modeled explicitly. The framework assumes that apical dendrites of layer 5 have direct connections with striatum - as a result, D1 and D2 layers. This direct connection imitates the influence of basal ganglia over layer 5 and also neglects the role of thalamus which is hypothesized to be doing a simple one-way relay to the cortical regions [42].

4.7 State Value

At the lowest level, the state representation allows the agent to differentiate between sensory patterns. Some of these states are more rewarding for the agent than others depending on the environment at hand. Preference among states is necessary for the agent to produce behavior; otherwise, all the states have the same value for the agent and there is nothing to learn. The temporal distance to the positive and negative rewards dictates this preference. The value of a state is calculated by Temporal Difference Learning using the long term rewards. The neural activation represents the state of a layer. Therefore, the state value is the combination of values computed separately for every neuron as shown in Figure 4.10. Although layer 5 activation defines the agent state and the resulting state value, higher level layers D1 and D2 are in place to sample from and classify the activations of layer 5 spatially and temporally. Values of the layer 5 activations are stored in layers D1 and D2 which control learning and producing voluntary behaviors.

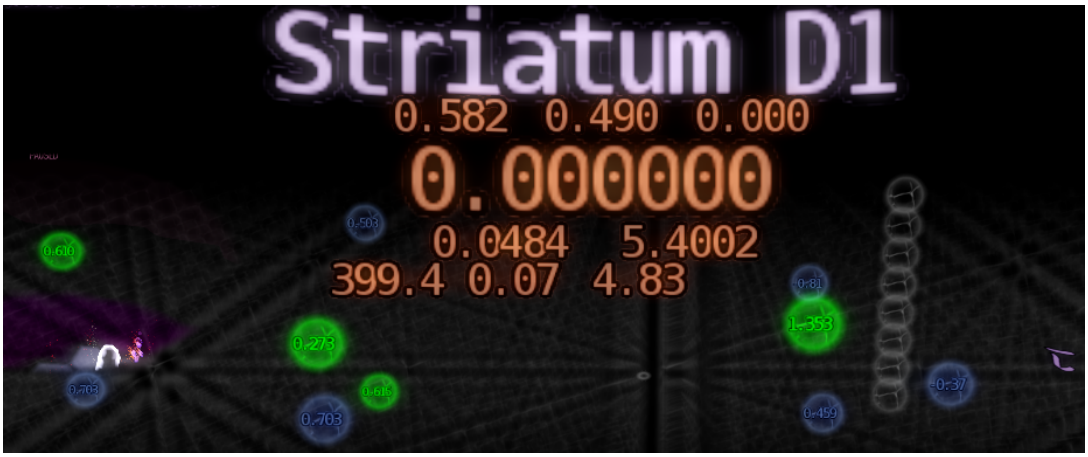


Figure 4.10: State values of the active and depolarized neurons of layer D1. Combined state value is 0.582 while the error with the previous value expectation is 0.490.

4.8 Producing Voluntary Action

The framework stages explained until this point are capable of modeling the environment, mapping motor actions to states and learning salient state transitions for a given state. Layers 4 and 5 classify sensory patterns based on their temporal context

via HTM algorithms. The layer 5 states are then mapped to the motor commands leading to themselves via learning by association mechanism between layer 5 and motor neurons. Layers D1 and D2 assign state values to the layer 5 activations. D1 forms connections to layer 5 states that produce a positive error signal. D2 makes the exact opposite and forms connections with layer 5 states that produce a negative error signal. Up to this point, the agent knows the state it is in, the potential next states and the state transitions that cause an error in reward expectations. The last step to produce voluntary behavior involves an actual activation modulation in layer 5. The distally depolarized cells of layer 5 denote the union of potential next activations while the apically depolarized ones represent the salient next activations. In our framework, combined distal and apical depolarizations cause voluntary activations.

The work by Larkum et al. [57] states that distal apical depolarizations can cause neural activations on layer 5 neurons without needing feedforward proximal stimulation. In practice, this means that a cell can fire if it exceeds an amount of distal apical depolarization. In our framework, layer 5 columns are activated by the proximal input from layer 4 activation which concurs with [21] and conflicts with [20] which states that layer 5 is activated by thalamic input. This decision is made because of organizational simplicity since layer 6 is neglected and there is a single region which takes sensory input directly. Neurons among these columns are activated based on the distal input from layer 2/3 and 4 cells along with the apical input from layers D1 and D2. In our architecture, if a cell is depolarized by both distal and apical presynaptic targets, that neuron stimulates the motor neurons connected to it. There are two types of voluntary motor neuron stimulations caused by layer 5; Go and No-Go stimuli originating from D1 and D2 layers respectively. The motor neurons have a base level of constant excitation which produces random behavior if there is no other influence. Go stimulations increase a particular neuron's excitation level while the No-Go stimulation inhibits the neuron. Therefore, the motor behavior is modulated by these voluntary stimulations. To simplify the action selection process, only the top three most excited neurons are allowed to fire and the rest is inhibited in our architecture. This winner-take-all mechanism is functionally the same as the inhibition phase of Spatial Pooler algorithm in Section 2.3.2.2. In Figure 4.9, the orange coloring of the neurons in the third row indicates No-Go stimulation while the cyan coloring indicates Go stimulation. The intensity of the coloring scales with the intensity of the stimulation. The figure also shows the synaptic pathways leading to these stimulations originating at reward circuitry layers D1 and D2.

The actual agent behavior is produced by the physics engine based on the motor neuron activation. Every agent action is mapped to three sequential motor neurons as outlined in Section 3.2.2. The physics engine activates the action that has the most active motor neurons. The motor activity frames of a simple seven step behavior sequence are given in Figure 4.12. The produced voluntary behavior results in the agent reaching sensory patterns that have better state values. In other words, the agent navigates within the possible states while preferring the ones with better long term rewards. A video demonstration for this process is also available in Appendix B.

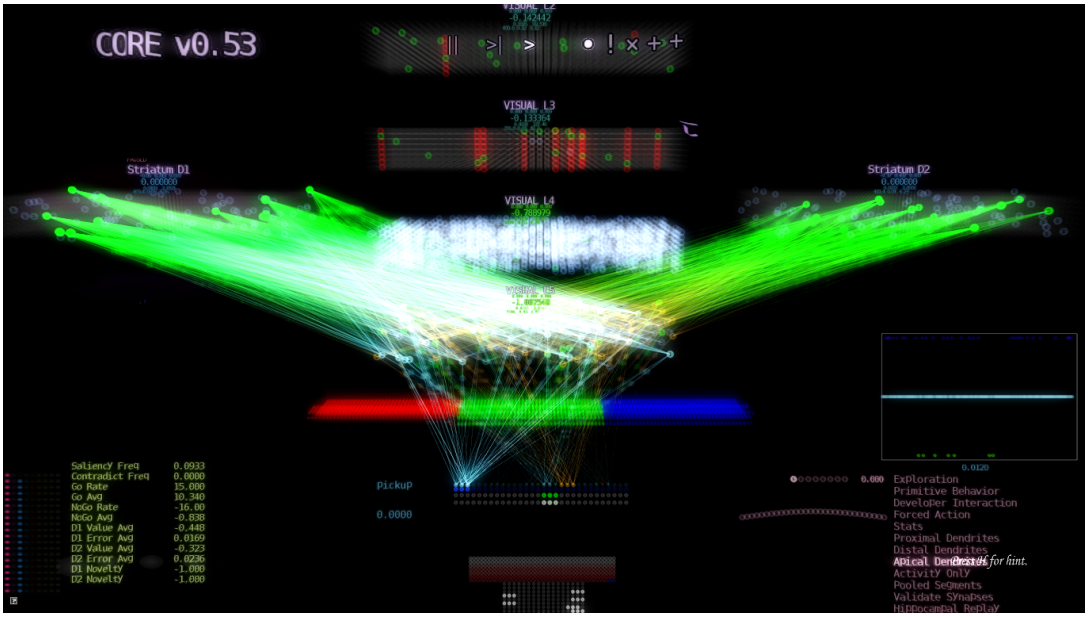
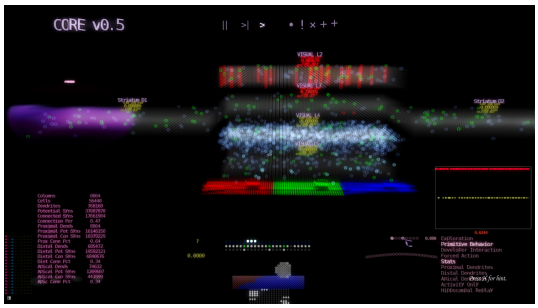
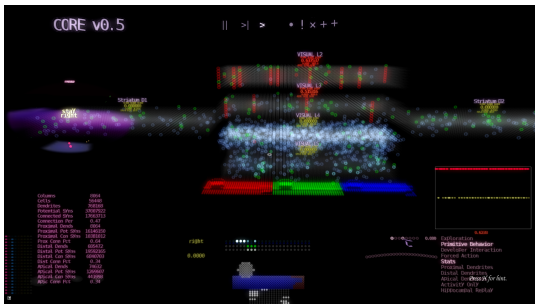


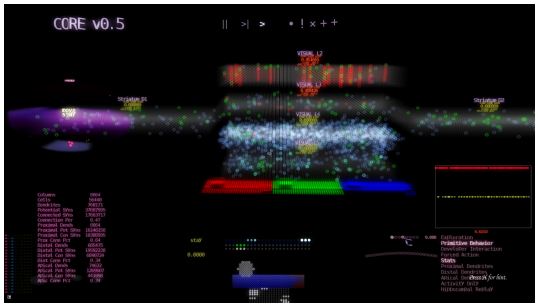
Figure 4.11: Visualized Go and No-Go pathways stimulating motor neurons.



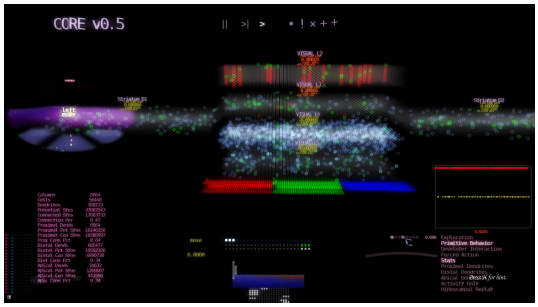
(a) Frame 1, rotate right.



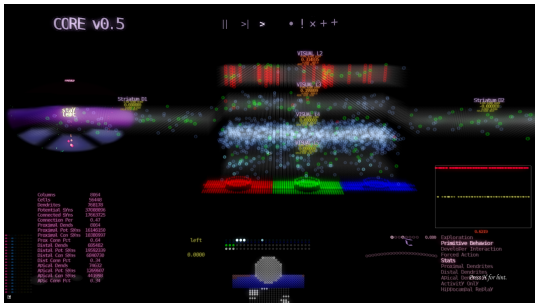
(b) Frame 2, stay.



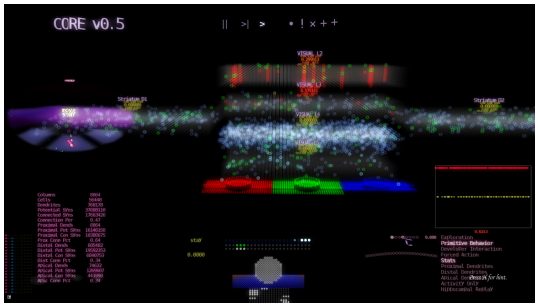
(c) Frame 3, go forward.



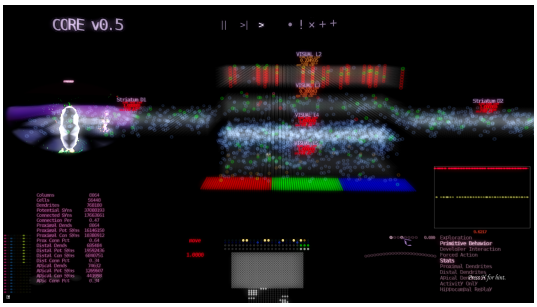
(d) Frame 4, rotate left.



(e) Frame 5, stay.



(f) Frame 6, go forward.



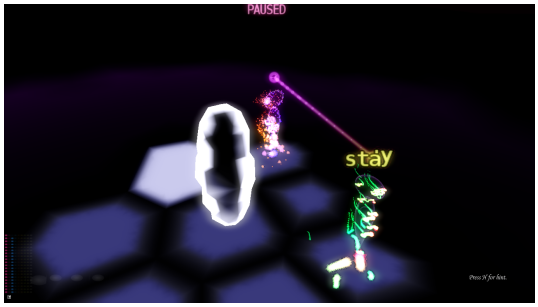
(g) Frame 7, success.

Figure 4.12: A learned behavior sequence involving 7 motor actions.

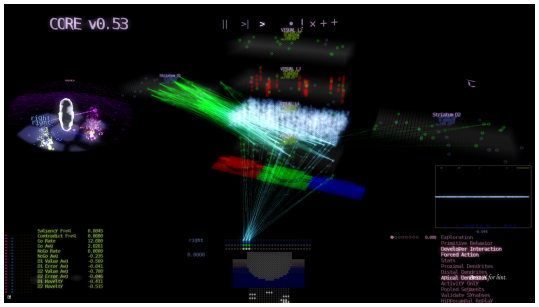
4.9 Player Interaction

The mechanisms explained above allow the agent to learn rewarding behaviors without any supervision. Our architecture also provides a player guidance mechanism for the agent. The player selects an agent via its icon in the game world. This selection establishes a sensorimotor feed from the player towards the agent. The architecture of the agent adapts its synapses based on the visual information the player gets and the motor actions that the player provides. It is just as if the agent is sensing the world through the player and navigating the world via the body of the player. During this process, the agent is motionless and the player can demonstrate any sequence of action. The HTM layers learn the environment model involving the demonstrated sequence of action. However, the agent only learns behaviors based on the rewards stimulated to the player which the sensorimotor feed also includes. The reward circuitry connectome of the agent is modified by the reward signals of the player. Therefore, the agent models all the relevant sensory patterns, but only learns the behaviors that are rewarding. In other words, the agent may learn the visual patterns but not the associated behavior depending on the previously learned rewarding behaviors.

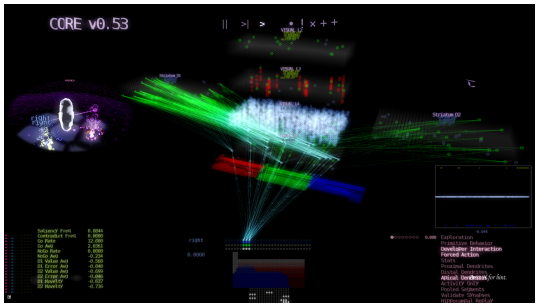
For a sequence to be completely learned, multiple episodes of that movement are required for the synapses to form connections. The necessity of multiple trials is due to the tweaked learning speed that sustains a level of stability among the synaptic connectome. Figure 4.13 showcases a seven step player taught behavior which is unlikely for the unsupervised agent to learn because of its inefficiency concerning rewards. The sequence involves the agent rotating around itself and then moving to the portal rather than going towards it onsight. Learning of the sequence is complete when the agent can predict the states of all the involving steps and when it can produce all the necessary voluntary activations to realize the behavior. As soon as the sensorimotor feed disconnects, the architecture switches to taking input from the agent's own sensor and the body. The synaptic modifications related to the interaction process persist and the agent proceeds the unsupervised learning on the new connectome. This mechanism allows the player to interfere with the agent at any time during the unsupervised learning and guide the process. However, a taught sequence involves only a fraction of the agent's environment model. It is very vulnerable to representational change caused by the following unsupervised learning of the agent. Therefore, the impact of a taught behavior on the agent's learning process is dependent on the representational stability of the HTM layers at the time of interaction and the significance of that particular sequence regarding rewards.



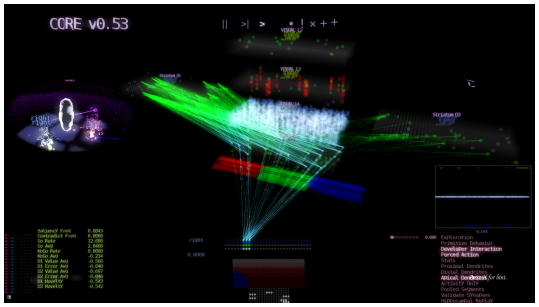
(a) Enable sensorimotor feed.



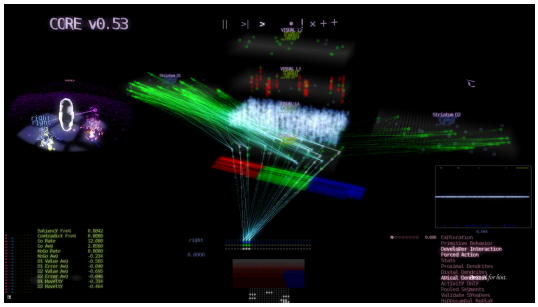
(b) Frame 1, rotate right.



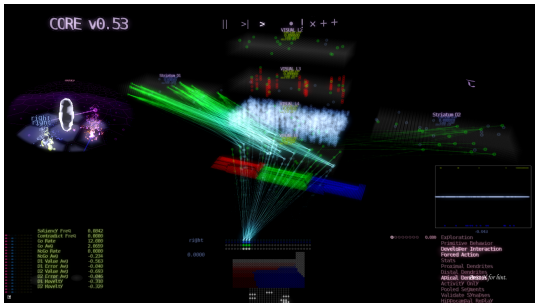
(c) Frame 2, rotate right.



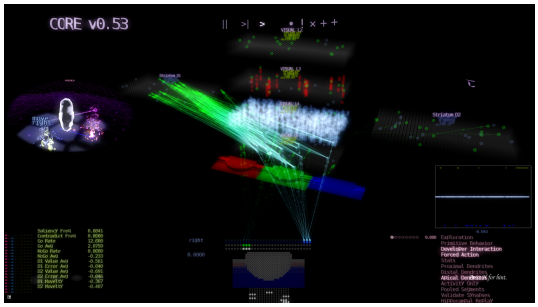
(d) Frame 3, rotate right.



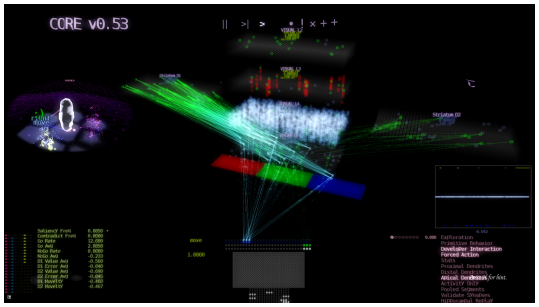
(e) Frame 4, rotate right.



(f) Frame 5, rotate right.



(g) Frame 6, go forward.



(h) Frame 7, success.

Figure 4.13: Above is a taught behavior sequence involving 7 motor actions where the player icon is purple and the agent icon is yellow. At the first encounter with the portal, the agent does a full rotation towards right and faces the portal again. The agent then proceeds to the portal at this second encounter. This behavior is unlikely to form via unsupervised learning because of its inefficiency in terms of reward. After 10 demonstrations, the HTM layers are able to predict the state of every following step of the behavior. Successfully predicted neurons are represented by green neurons. In addition, the agent is able to excite the correct voluntary activations for every step represented by cyan neuron coloring. Notice the green smile icon at the top of the agent on Frame 7 caused by the positive reward stimulated to the player.

CHAPTER 5

IMPLEMENTATION AND ARCHITECTURE FEATURES

5.1 Overview

The game engine that is used as the platform is designed around the autonomous agent. During this study, it became incrementally refined both algorithmically and performance-wise. Embedded inside the engine are the Hierarchical Temporal Memory and Temporal Difference Learning implementations with the extended features that realize the architecture. In Chapter 2, the up-to-date version of the vanilla HTM theory along with TD(λ) is presented. Chapter 3 describes the surrounding environment which constitutes constraints on the agent such as terrain formation, sensors, agent body, and motion. The proposed agent architecture for this environment is outlined mechanistically in Chapter 4. Although this study shares the same vanilla HTM algorithms at its core, there are implementational differences due to platform and performance constraints in addition to the necessary additions required by the player interaction and serialization systems. This chapter aims to address implementation specifications to ensure that this study is reproducible. Moreover, additional architectural features are presented at the end of the chapter.

There are four subsystems of the engine that are directly related to the HTM implementation: Cortical System, Old Brain System, User Interface System and PhysX System. Figure 5.1 shows the game engine loop for the exact order of computation. Cortical System is the one where the interplay of HTM and TD(λ) implementation subsists. Old Brain System calculates the visual and haptic sensor data. It also handles player interaction functionality that is explained in Section 4.9. The architecture visualization, Core, is a major part of the User Interface System. The actual motion computations of the agent are carried out in PhysX System. Old Brain System and Cortical System provide the main functionality together which take up to 7500 lines of code.

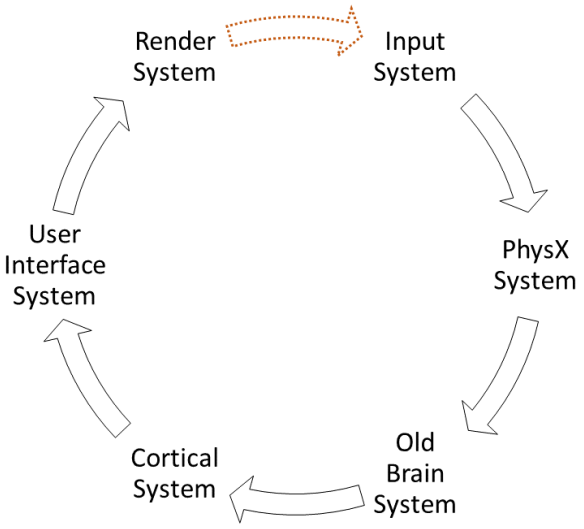


Figure 5.1: The computational ordering of engine systems related to this study starting with Input System.

5.2 Old Brain System

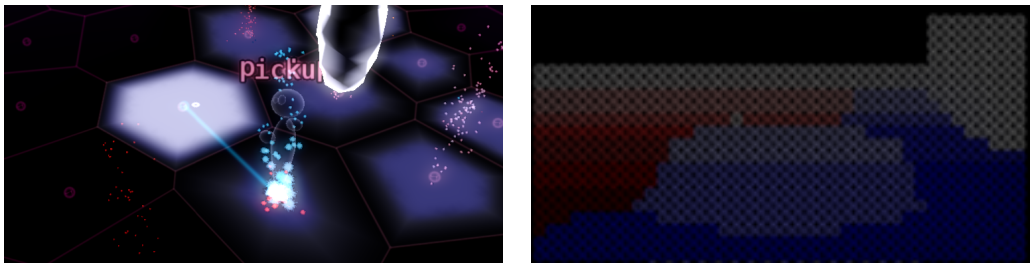
Visual sensor update constitutes the majority of this system. The general mechanism of this update is laid out in Section 3.2.3.1, so this section aims to clarify the implementation details. Below is the pseudocode of the Old Brain System.

Algorithm 8: Old Brain System Update Pseudocode

```
1  StimulateRetina(agent, player);
2  StimulateHaptics(agent);
3  SensoriMotorOverride(agent, player);
5  Function StimulateRetina(agent):
6      horAngleStep ← HFOV / SENSOR_WIDTH;
7      verAngleStep ← VFOV / SENSOR_HEIGHT;
8      for i ← 0 to SENSOR_WIDTH do
9          for j ← 0 to SENSOR_HEIGHT do
10             visualSensor[i · j + j] ←
                 Raycast(agentDir, i · horAngleStep, j · verAngleStep);
11 Function SensoriMotorOverride(agent, player):
12     agent.visualSensor ← player.visualSensor;
13     agent.hapticSensor ← player.hapticSensor;
14     agent.motorAction ← player.motorAction;
```

As explained earlier, there are three color channels that the agent can sense - red, green, blue. The image sensed by the agent is stored in separate bit vectors per color. The visual data is cleared on every frame at the start of computation. Then, depending on the horizontal and vertical facing direction of the agent, the system calculates the

corresponding Voronoi cell that the agent looks at. This step allows the agent to interact with a particular Voronoi cell which is of particular importance for future experiments involving terrain interactions. This cell is brightened in the visual sensor along with the resources on top of it to achieve identifiable change in the image as in Figure 5.2.



(a) Focused cell on terrain.

(b) Focused cell on visual sensor.

Figure 5.2: Focused cell visualization. This is the Voronoi cell that the agent is facing depending on the agent's vertical and horizontal looking direction.

The image is constructed by casting rays from the agent towards the environment geometry. The direction of the central ray - the exact facing direction - is calculated using the agent's eye position as the origin and the looking direction which combines the vertical saccade offset with the horizontal direction of the agent. The other rays are created by rotating the central ray around the origin using fixed amount of degrees on both vertical and horizontal axes via a nested computation loop as in line 10. The resulting group of rays cover a preset horizontal and vertical field of view. For every ray, the intersection color, distance, and hit status are retrieved from the environment geometry using the functionality provided by PhysX. The color value is black if that particular ray does not intersect with the environment. Pixels are then brightened according to the normals of intersecting triangles to represent the lighting. Then, the color value is divided into five intervals to store the image in binary. The last step is to fill out the color channel bit vectors representing the image that the rays are mapped onto. Every ray corresponds to a position in the 2D image and has discrete color values as seen in Figure reffig:rayMap. Core visualizes the color bit vectors via volumetric layers per color.

Old Brain System also handles player interaction by having a mode called Sensorimotor Feed. In this mode, the motor activity and sensory information of the player are directly fed to a specified agent in line 3. This mechanism allows the agent to learn from the behavior of the player. In each frame, the agent's visual sensor data and motor activity data are replaced with the player data before the HTM iteration. The agent learns directly from the visual sensor and motion of the player which allows teaching specific movement patterns that may or may not lead to reward. Once the sensorimotor feed disconnects, the agent resumes sensing the environment through its own visual sensor and navigating via its own body, but the synaptic connections of the HTM layers have been altered by the experience provided by the player. The player interaction requirement has a major influence on the implementation.

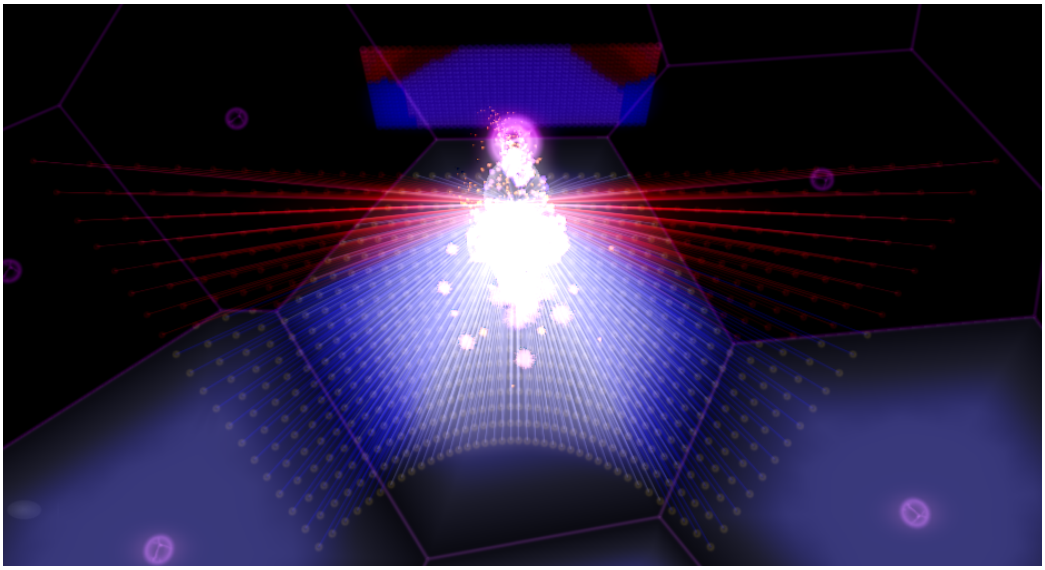


Figure 5.3: Raycasting visualization where the agent looks down. Constructed image is at the top of the agent icon.

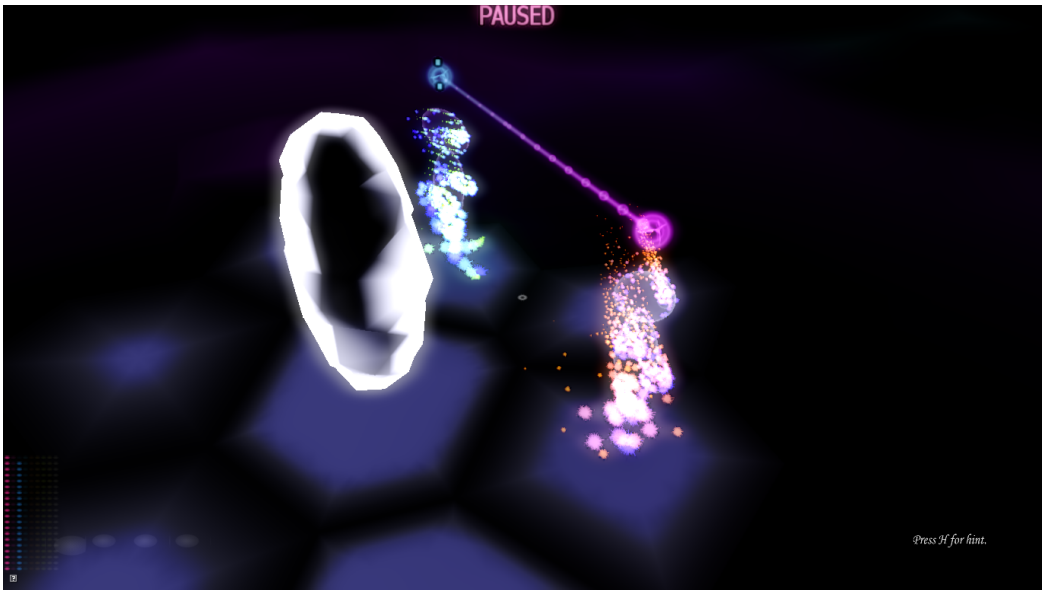


Figure 5.4: Player (purple icon) is streaming its sensorimotor information for the agent (blue icon) to learn on.

5.3 Cortical System

The functionality of Cortical System is the backbone of this study as it facilitates real-time HTM in conjunction with TD(λ). This section expands on the architecture implementation to provide the mechanisms presented in Chapter 4. Algorithm 9 shows the pseudocode for the Cortical System update per agent.

Algorithm 9: Cortical System Update Pseudocode

```
if !initialized
    InitializeLayers(allLayers);
    InitializeConnectome(allLayers);
if reset
    ResetActivity(allLayers);
UpdateSensoryLayer(visualSensor);
RefreshDepolarization(layer4, motorLayer);
foreach layer ∈ layers{layer4, layer3, layer2, layer5} do
    SpatialPooler(layer);
    TemporalMemory(layer);
    UpdateStatistics(layer);
foreach layer ∈ striatum{layerD1, layerD2} do
    SpatialPooler(layer);
    TemporalMemory(layer);
    DecayTraces(layer.pooledNeurons);
    CalculateAverageValue(layer);
    CalculateAverageError(layer);
    RefreshTraces(layer.pooledNeurons);
    UpdateStateValues(layer.pooledNeurons);
    UpdateStatistics(layer);
if !segmentTraces
    DecayTraces(layer5.pooledSegments);
    AdaptPooledSegments(layer5.pooledSegments);
ApicalTemporalMemory(layer5);
if !segmentTraces
    RefreshTraces(layer5.pooledSegments);
ApicalTemporalMemory(motorLayer);
VoluntaryExcitation(layer5);
GenerateBehavior(motorLayer);
foreach layer ∈
  allLayers{layer4, layer3, layer2, layer5, layerD1, layerD2, motorLayer} do
    SegmentDecay(layer);
    CleanUpSegments(layer);
UpdateArchitectureStatisticsAndValidation();
```

5.3.1 Layer Creation and Connection

The algorithm checks whether the layers are created and connected as in line 1. It creates the layer by taking column count and neurons per column as the constructor arguments. The proximal, distal and if there are, apical sources of a layer must also be defined for the connection stage. The other required information is whether the layer uses neural or columnar activity of its sources as the input. After the creation of all layers, the connectivity is initialized by mapping layers to their sources. Below is the layer creation parameters with some discussion as to why. The layers have dedicated learning parameters depending on the type as presented in Appendix A.

Layer 4

- Proximal Sources: Sensory Layer
- Proximal Input Type: Neural Activation
- Distal Sources: Motor Layer
- Distal Input Type: Neural Activations

Research on layer 4 states that it is the input layer of a region among the six layers [18], [19], [16]. According to the studies, input originates from the interplay between thalamus, other cortical regions, and subcortical structures. The thalamus acts as the gating mechanism feeding information to layer 4 while also possibly transforming the input. Therefore, layer 4 is proximally connected to the sensory layer. It is also known that motor activity information is utilized by the layers in motor cortex (M1). The sensory thalamic input constitutes only a portion of the information that the layer 4 of M1 and S1 regions get [17]. Therefore, the motor activity is also fed through the layer 4 in our framework. The information can be supplied through proximal dendrite along the sensory information or from the distal dendrite. For simplicity, sensory information and motor information are decoupled. Distal dendrites of layer 4 only sample from motor activation. In practice, this means that layer 4 does sensory pattern classification through columnar activity and puts it into motor activity context through its neural activation. In other words, the resulting neural activity represents the sensory pattern in the context of motor activity which helps mapping required motor actions to respective sensory patterns as argued in Section 4.5. On the other hand, mixing motor and sensory information through both proximal and distal dendrites is still functionally feasible. The structural composition of sensory and motor layers consists only of neurons rather than columns of neurons which simplifies the implementation. As a result, the input type of layer 4 is neural activation which samples from sensory and motor layers.

Layer 3

- Proximal Sources: Layer 4
- Proximal Input Type: Columnar Activation

- Distal Sources: Layer 3
- Distal Input Type: Neural Activation

Layer 2

- Proximal Sources: Layer 3
- Proximal Input Type: Columnar Activation
- Distal Sources: Layer 2
- Distal Input Type: Neural Activation

Layer 3 and 2 are the primary focus of HTM theory, and they are hypothesized to be doing variable order sequence modeling [14]. For example, if a sequence ABCD is learned and the sequence EBCD occurs, The B appearing after A is represented with different neurons than the B appearing after E. As a result, the representations of C and D are different for both sequences since they are dependent on the context which is also different. This is the higher level functionality of Temporal Memory algorithm explained in Section 2.3.3. In our architecture, layers 3 and 2 are increasingly higher level, slower changing, more stable temporal and spatial abstractions of layer 4. Temporal pooling mechanism is enabled for these layers to represent temporal abstractions as explained in Section 5.4.2 at the end of this chapter.

Layer 5

- Proximal Sources: Layer 4
- Proximal Input Type: Neural Activation
- Distal Sources: Layers 4 and 2
- Distal Input Type: Neural Activation
- Apical Sources: Layers D1 and D2
- Apical Input Type: Neural Activation

As explained in Reward Circuitry Section 4.6, the apical tufts of layer 5 integrates input originating at basal ganglia through layer 1 [25], [27]. Therefore, the apical sources of layer 5 are reward circuitry layers D1 and D2 in our framework.

The canonical microcircuit theory of cortical layers indicates that information flows in L4→L3/2→L5/6 [18], [19]. However, the work by Constantinople and Bruno [20] challenge this classical assumption with their findings stating that layers 5/6 are not being driven by layers 2/3. In fact, they show that layers 5/6 may even be getting activated slightly before layers 2/3 due to thalamic input. Work by Ramaswamy and Makram [22] states that layer 5 integrates information from all cortical layers. Moreover, publication by Schubert et. al. suggests that layer 4 has excitatory influence on

layer 5. According to another study, layer 6 modulates and even activates some of the representations at layer 5 [31] which is the primary output pathway of the region. However, layer 6 is not explicitly modeled in our framework because its mechanistic functionality is not yet clear in neurobiological studies [58], [29].

In our framework, we integrate input to layer 5 from both layer 4 which gets sensorimotor input and layer 2 which has higher level spatiotemporal representations. The proximal input of layer 5 comes from layer 4 neural activation because the neural activation of layer 4 represents the sensory information in the context of the motor action. Layer 5 takes its distal input from both layer 4 and layer 2. Neuroscience studies show that the layer 3/2 also projects to layer 5 [18], [19], [20]. Temporally and spatially abstract representations of layer 2/3 are hypothesized to help with layer 5 predictions in our framework.

Striatum D1

- Proximal Sources: Layer 5
- Proximal Input Type: Columnar Activation
- Distal Sources: Layer 5
- Distal Input Type: Neural Activation

Striatum D2

- Proximal Sources: Layer 5
- Proximal Input Type: Columnar Activation
- Distal Sources: Layer 5
- Distal Input Type: Neural Activation

Based on the relevant neurobiological research presented in Section 4.6, the reward circuitry layers proximally sample the columnar activation of layer 5 which represents the agent state via integrating information from layers 4 and 2. In practice, layer D1 and D2 make first order predictions on layer 5 activations. Therefore, these layers predict their next activation by distally sampling from the current activation of layer 5 in our framework. The distal input type is a neural activation because it also includes the contextual information of layer 5 which is denoted by its neurons.

Motor Layer

- Motor Layer
- Apical Sources: Layer 5
- Apical Input Type: Neural Activation

The output of a region originates at layer 5 and propagates indirectly to motor areas, mid brain and even down to spinal cords [23]. In our framework, the only input of motor layer comes directly from layer 5. The motor neurons take this input through their apical dendrites because the learning by association mechanism explained in Section 4.5 is carried out via the Temporal Memory algorithm on apical dendrites.

5.3.2 Reset

Uninterrupted continuous learning is one of the differentiating factors of this implementation from vanilla HTM. In default HTM, there is a reset tag on the input stream that signifies the end of a sequence to keep the length of the learned sequences in check. This implementation supports both non-reset learning and resets on important events such as deaths or task completions as in line 4. The architecture clears all the temporal activity after a reset, treating the next input as the first one in a sequence without any context. The first input after a reset results in all the columns bursting. Bursting means that a layer activates all the contextual neurons representing that input which are the neurons inside the active columns. The bursting also results in the depolarization of all the next activations among all the sequences involving the current input.

Resetting also prevents the layers from modeling multiple task completions as a single sequence. Treating multiple trials as a single sequence means representing the patterns of the second task in terms of the first task. Therefore, if the patterns of the second task appear after a different first task, it needs to learn the second task in the context of the newly encountered first task. In short, not resetting leads to a deeper contextual learning but slower learning and lower generalizing capability. Resetting provides a faster learning if the tasks are independent. On the other hand, having a reset mechanism is difficult to justify concerning neurobiology. It can be argued that it is a higher level abstraction for novelty detection functionality that separates sequences via an end signal. However, additional neurobiological evidence is required to justify this ad-hoc reset mechanism of vanilla HTM.

5.3.3 Sensory Layer Update and Layer 4 Depolarization Refresh

The implementation treats the columns and neurons as synaptic nodes that can either be presynaptic or postsynaptic targets of synapses. However, the sensory data consists of bit vectors which are not synaptic targets. So, the sensory bit vectors are fed to a hidden sensory layer in this stage where every bit is treated as a synaptic node. As a result, layer 4 can function as if sampling from the activity of another layer. The hidden sensory layer update in line 6 serves as an interface for the cortical layers to take inputs as if they are sampling from synaptic nodes.

By default, distal synapses sample the previous activity of their presynaptic targets. Neurons that represent the prediction of the next step are depolarized through these distal synapses. The distal source of layer 4 is the previous activity of the motor neurons. However, the depolarized cells of the current time step are based on the current motor activation which is decided at the end of the architecture iteration. Therefore,

the depolarization stage of layer 4 belonging to time step t is redone at the start of next iteration at $t + 1$ as in line 7, when the motor activation of time step t is known. This is an implementational workaround for an architectural inadequacy which will be improved in the future. At the moment, the predictive neurons of layer 4 represent all the potential states caused by all the possible motor actions in Core. Figure 5.5 shows a large amount of depolarized cells displayed in visualization. These depolarized cells are sparsified according to the produced motor action at the start of the next iteration.

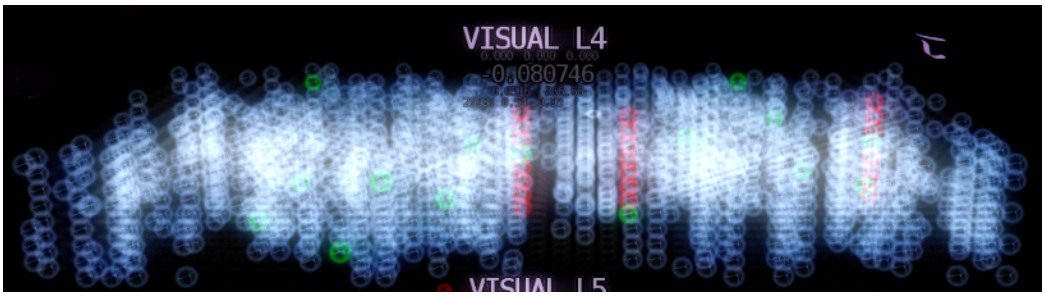


Figure 5.5: Layer 4 depolarization showing all the possible activations at the end of an HTM iteration. The depolarization gets sparsified at the start of the following HTM iteration using the last motor action as context.

5.3.4 Cortical Layers Actuation

This part of the algorithm in line 8 carries out the HTM calculations for cortical layers in the order: layer 4, layer 3, layer 2 and layer 5. For each layer, Spatial Pooler and Temporal Memory algorithms are sequentially actuated as explained in Section 2.3. The order of layers is due to the input dependencies between the cortical layers. Decoupling the dependency between the layers is a future research direction that would allow for a better parallelization if attempted. Every layer takes its proximal input and activates the columns that have the best overlap with that proximal input via Spatial Pooler algorithm. Proximal dendrites of these active columns are then adapted to the input that caused the activation. Among the neurons of these columns, the ones that have been depolarized/predictive from the previous iteration are chosen as the active and learning cells via Temporal Memory algorithm. The distal segments that caused the depolarization are then adapted to the input that caused the depolarization. If there are no depolarizations among an active column, the column bursts and every neuron becomes active to learn.

The Spatial Pooler implementation in this study has a modification to the bumping and boosting mechanisms which facilitate competition among the columns as explained in Section 2.3.2.3. At the time of implementation, this modification was proposed to HTM community and got positive encouragement to be researched on. Vanilla Spatial Pooler has a synapse strength bumping mechanism for the columns that have lower input overlap frequencies among the layer. Periodically, an overlap duty cycle variable is calculated for every column which represents its recent input overlap frequency. The proximal dendrite of a column is then strengthened if the overlap duty cycle is lower than the majority of the columns. Boosting mechanism, on the other

hand, calculates a boost value for every column depending on its activation frequency. The overlap of columns are scaled with this value and this results in a more uniform usage among the layer because lesser used columns have higher boost values.

In vanilla Spatial Pooler, boosting and bumping are carried out with separate calculations. In this study, bumping mechanism is carried out by the boost value rather than calculating a separate overlap duty cycle variable. In our implementation, there is a base synaptic increment value for all the columns, so the proximal dendrites of all the columns are continuously strengthened rather than the ones with lower input overlap frequency. This base increment value is then scaled with the individual boost value calculated for every column. This method simplifies and merges the boosting and bumping mechanism to use the same boost value rather than an additional one for overlap frequencies. In our experiments, some columns did not ever get used even with the bumping mechanism of vanilla Spatial Pooler because the input overlaps of those columns were not below the threshold for strengthening the proximal dendrite.

5.3.5 Reward Circuitry Layers Actuation

The actuation of layers D1 and D2 in line 12 provide the functionality of Temporal Difference Learning for the architecture. It is important to note that all the steps of this subsection are applied to both D1 and D2 layers. The actuation starts off with HTM calculations: Spatial Pooler and Temporal Memory algorithms for both layers. The resulting activation on both layers is the spatiotemporal classification of the activation in layer 5. The active neurons have a pooling value that represents the respective eligibility traces for TDλ as in Figure 5.6. The eligibility traces of all the pooled neurons are decayed just after the HTM calculations with respect to the decay and discount parameters as in Equation 5.1.

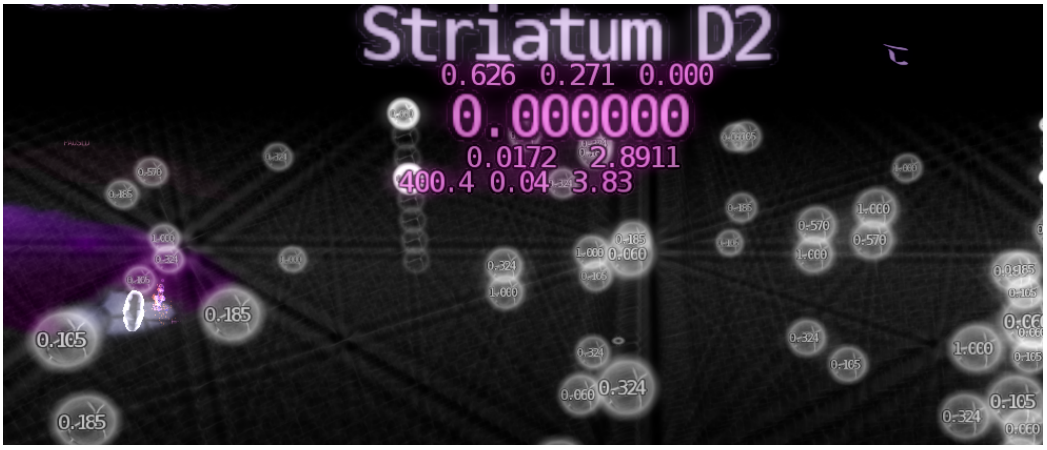


Figure 5.6: Pooled (nonzero eligibility traces) neurons of layer D2 with visualized trace values.

$$Trace_i = Trace_i \times TD\_DISCOUNT\_RATE \times TD\_TRACE\_DECAY \quad (5.1)$$

As in TD(λ), the next step is to calculate the current state value to compare it with

the expected value from the previous time step. Current state value is the weighted average of the state values of all active neurons depending on prediction. If the cell is active because it is burst (failed prediction), the weight is 1. Otherwise, if the cell is a successfully predicted cell, the weight is 10. The reasoning behind the weighting is to sustain stability when the resulting activations of D1 and D2 consist of a mix of burst and predicted cells. The computation is as in Equation 5.2 where n is the size of the current neural activation, $Value_i$ is the state value of neuron i and $Weight_i$ is the weight for neuron i.

$$AvgValue = \frac{\sum_{i=1}^{n} Value_i \times Weight_i}{n} \tag{5.2}$$

The difference between the actual state value at time $t + 1$ and the expected value from time t guides the learning in TDλ. To calculate this error, the state value that is already computed is used in conjunction with the expected value of the previous activation, discount parameter, and the actual reward. The exact calculation is shown in Equation 5.3 where R is the actual reward stimulation, n is the size of the previous neural activation, $Value_i$ is the state value of neuron i and $AvgValue$ is the calculated average value.

$$AvgError = \frac{\sum_{i=1}^{n} R + TD\_DISCOUNT\_RATE \times AvgValue - Value_i}{n} \tag{5.3}$$

The implementation then increases the eligibility traces of the neurons according to TD(λ) with replacing traces [47]. The traces of the previously active cells are set to 1 as in Equation 5.4.

$$Trace_i(Neuron_i) = \begin{cases} 1 & \text{if } Neuron_i \text{ was active in previous iteration.} \\ Trace_i & \text{else} \end{cases} \tag{5.4}$$

The error signal computed by the layers D1 and D2 are used for adapting the layer 5 apical dendrites sampling from these layers. As explained in Reward Circuitry Section 4.6, D1 and D2 functions in competition with opposite learning rules presented in Equation 4.2. The apical dendrites of layer 5 adapt to the neural activation of D1 if the error is positive. In practice, this means that layer 5 learns the behaviors that provide a positive error through adapting to layer D1. On the other hand, the apical dendrites adapt to D2 activation when the error is negative. Through this mechanism, whenever a learned layer 5 activation occurs, layers D1 and D2 depolarize the associated Go and No-Go activations on layer 5 through its apical dendrites. These competing depolarizations form the basis of generating the voluntary behavior.

The last stage involves updating the state values of neurons according to the error

$AvgError$, eligibility traces $Trace_i$ and the learning rate $TD\_LEARNING\_RATE$ as in Equation 5.5.

$$Value_i = Value_i + TD\_LEARNING\_RATE \times AvgError \times Trace_i \tag{5.5}$$

5.3.6 Layer 5 Apical Temporal Memory

This phase is required for layer 5 activations to learn the D1 and D2 activations. Algorithmically, the computation in line 24 is identical to Temporal Memory, but the adaptation happens on the apical dendrites and the adaptation itself is dependent on reward signals. The connections formed in this phase are the backbone of voluntary motor actions because layers D1 and D2 depolarize layer 5 neurons through these apical synapses. If the segment eligibility traces feature in Section 5.4.1 are enabled, the traces of active apical segments are replaced with 1 in this phase.

5.3.7 Learning by Association For Motor Neurons

The activations of layer 5 are mapped to the necessary motor commands leading to their own activation. The phase in line 27 provides this mapping with an identical algorithm to Temporal Memory with a small difference: the synapses are apical and they learn the current activation of the source, not the previous activation. Learning by association can be achieved with a variety of strategies. Implementation-wise, a Temporal Memory modification was the simplest solution for our framework. Additional modifications, as well as a computational reordering of Temporal Memory algorithm, are required. This mechanism allows exciting or inhibiting motor neurons via the stimulation caused by voluntary activations of layer 5.

5.3.8 Voluntary Excitation by Combined Depolarization

Motor neurons need to be stimulated by the layer 5 activation to produce voluntary actions and override random behavior. The depolarization of layer 5 happens through both distal and apical dendrites. The former is caused by the layer 5 distal input layers 4 and 2 while the latter is caused by apical input layers D1 and D2. If a cell is both distally and apically depolarized, it gets activated without proximal input in our framework based on the relevant neurobiological research presented in Producing Voluntary Action Section 4.8. Activation is either suppressive on motor neurons if it belongs to No-Go circuitry originating from D2 or it is generative if it belongs to the Go circuitry originating from D1. This phase in line 28 calculates voluntary spikes for all cells by summing distal and apical depolarization levels.

5.3.9 Generating Behavior

Through their apical dendrites, motor neurons learn the layer 5 states that are active via learning by association. The voluntary activity of layer 5 stimulates the corresponding motor neurons to fire. The firing type is dependent on the stimulation type resulting in either inhibition of the neuron or the excitation of the neuron. Layer 5 activations may cause the same motor neurons to get excited and inhibited simultaneously. The total stimulation from the competing negative (inhibitory) and positive (excitatory) sources determines the resulting motor neuron state in line 29. Among all the calculated stimulation values, the algorithm picks the highest three motor neurons. These top neurons are then forwarded to the PhysX System of the engine at the next game cycle.

5.3.10 Global Apical and Distal Segment Decay

The size of the synaptic connectome grows as the agent learns continuously. In time, the same patterns may get represented by different activations which render some groups of segments obsolete. In some cases, these obsolete segments never get activated again and as a result, never get updated. Architecture slowly fills up with these outdated and unused segments that do not participate in learning. This situation becomes worse over time leading to unnecessary memory consumption and performance overhead. Therefore, a global synapse decay mechanism is implemented in line 31 to get rid of never used segments by slowly decrementing their permanences. For a complete decay to happen, a particular segment should never get activated for around 20000 iterations which take around 10 minutes in real-time. The decay is faster for apical dendrites of motor and layer 5 neurons because those require a higher level of plasticity and more efficient use of capacity. These apical dendrites are under constant change based on the error signal produced by layers D1 and D2.

5.3.11 Segment CleanUp, Validation and Architecture Statistics

As an interesting observation during the study, some segments that are created with different synapses converge on the same connectivity. A significant amount of duplicate segments emerge when the simulation runs for a while. These duplicates have the same functionality but fill up both memory and the segment capacity of neurons. This phase does routine checks on segments to detect duplicates and flag them. Also, segments do not get deleted during an iteration because of pointer references scattered on some synaptic nodes. For runtime safety, duplicate and deleted segments are removed at the end of an architecture cycle in line 32. There is also a feature of the architecture that allows validating synapses by crosschecking pre and postsynaptic targets to ensure that all synapses are valid. This feature is crucial for serialization operations. The data about synapse counts and connection percentages are updated in real-time in line 33 and presented as the architecture statistics in Core.

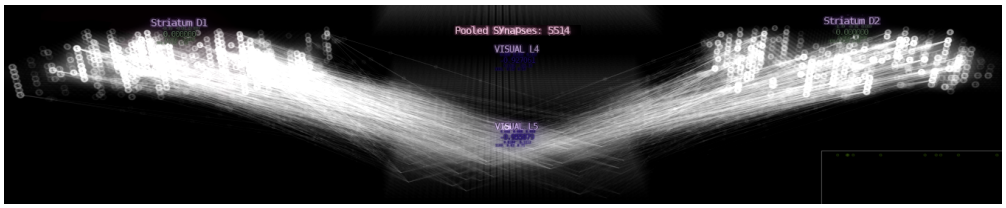
5.4 Architecture Features

5.4.1 Segment Eligibility Trace

This phase in line 21 actuates only when the segment eligibility traces feature is enabled. The initial implementation of HTM and TD(λ) combination updated the neural state values based on the neural segment traces and only adapted the active apical synapses of layer 5 from the previous iteration as in Figure 5.7a, according to the error signal. Introducing eligibility traces for the segments allows the architecture to adapt all the recently active synapses as visualized in Figure 5.7b based on the activation recency. With this feature, apical segments of layer 5 neurons have their own eligibility traces used in the adaptation based on the error. This is an additional feature that is not required, but it improves the plasticity of the architecture in theory. It enables the adaptation of up to 10 times more apical synapses in one iteration as shown in Figure 5.7b.

When this feature is activated, the first step is to decay the traces of the segments. The formula for segment trace decay is the same with the trace decay of neurons as in Equation 5.1, but it is calculated using the activation of the segments. Then, all the synapses of recently active segments are adapted according to their eligibility traces using the synaptic increment and decrement values computed by reward circuitry layers D1 and D2. The last step involves replacing the eligibility traces of the active segments and it is done after Apical Temporal Memory computation of layer 5.

(a) Active layer 5 apical synapses from the previous iteration: 459



(b) Recently active layer 5 apical synapses with nonzero segment eligibility traces: 5514

Figure 5.7: Comparison of the number of adapting apical synapses with respect to the segment eligibility traces feature. It is disabled in the upper figure and enabled in the lower one.

5.4.2 Temporal Pooling

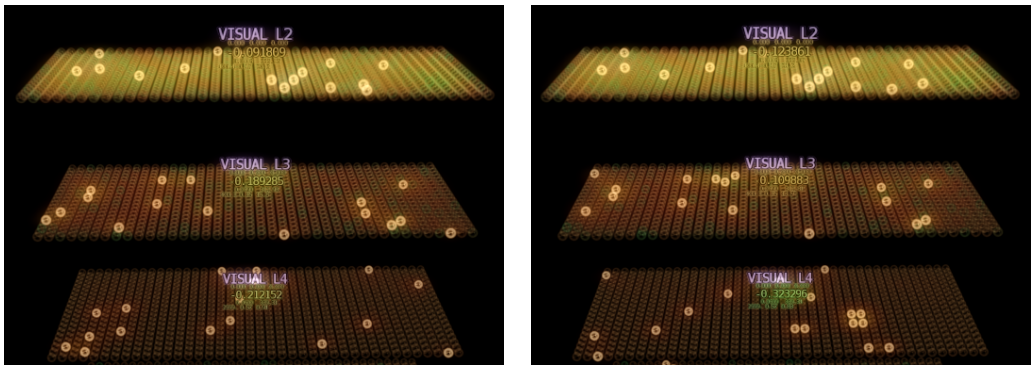
Layers 2 and 3 in our architecture represent input patterns with temporal abstraction via temporal pooling. A single activation on these layers encodes multiple temporal input patterns. Therefore, their activations are not only spatially but also temporally

abstract representations of their input. Along with the distal depolarization from layer 4, layer 5 is also depolarized distally by layers 2/3 which have more stable activations that result in a better prediction. The slower changing and temporally abstract context is utilized in the predictions of the following agent states especially when the layer 4 depolarization is not sufficient for a prediction in layer 5.

The term temporal pooling is also referred as union pooling in Nupic research implementations. It allows the spatial activation of an HTM layer to represent sequential input data. A stable activation sampling from temporally changing inputs allows that activation to encode multiple sequential inputs. In other words, the stable activation represents the temporal abstraction of the changing inputs. This mechanism can be implemented via a variety of strategies involving modifications to the Spatial Pooler algorithm. This implementation achieves temporal pooling by introducing an overlap accumulation mechanism to the input overlap calculations of Spatial Pooler algorithm. The overlap of a column with the input accumulates and decays over time. Equation 5.6 shows the accumulation rule where $Overlap_i$ is the newly calculated overlap, $PrevOverlap_i$ is the previous overlap of the column i and the $Overlap\_Decay$ is the overlap decay factor which is set to 0.9 in our study.

$$Overlap_i = (PrevOverlap_i + Overlap_i) \times Overlap\_Decay \tag{5.6}$$

The slower change of overlaps allows the active columns to stay active longer than their overlapping inputs. As a result, these columns adapt not only to the input that caused the activation but also to the following input patterns. This creates a more stable layer with slowly changing activations which also provides smoother state transitions. Figure 5.8 shows the more stable activations of layers 2 and 3 compared to the layer 4 on sequential frames.



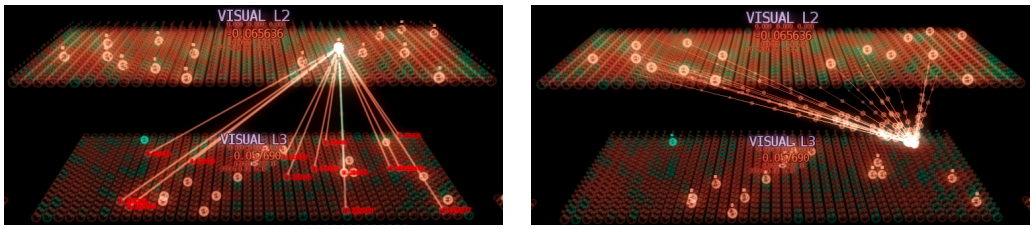
(a) Frame 1. (b) Frame 2.

Figure 5.8: Layer 2 and 3 representations that are more stable and with smoother transitions compared to layer 4 on sequential frames.

5.4.3 Mirror Synapses Optimization

The open source implementation of HTM - Nupic - checks the input overlap of the proximal, distal and apical dendrites by iterating the whole layer. For example, the

proximal dendrite of every column is iterated when calculating the input overlap of those columns. However, only some of the input sources are active at a given time. Vanilla computation method iterates over all the proximal synapses of the layer checking whether the source is active or not, which is a top-down check as in Figure 5.9a. This creates a substantial amount of redundant computation because the active input sources are known beforehand. So, if the algorithm has a way to iterate only the synapses which have active sources, it reduces the amount of computation to calculate the input overlaps. To do this, there are mirror synapses in this implementation that originate from the input sources which hold references to the actual proximal synapses. The mirror synapses provide a bottom up strategy as in Figure 5.9b by allowing the iteration of synapses that sample from active input bits.

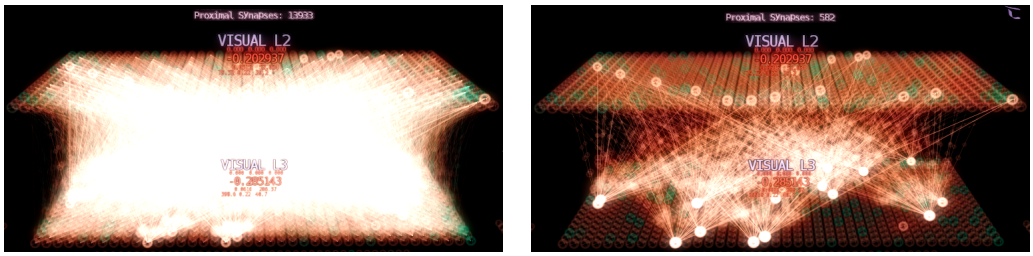


(a) Default proximal dendrite visualization of an L2 column sampling from L3 columns.

(b) Mirror dendrite showing proximal propogation of an L3 column to L2 columns.

Figure 5.9: Top-down and bottom up synaptic visualization.

The same optimization strategy is utilized for all the dendrites including distal and apical types. For some cases, this optimization results in up to 20 times shorter execution times on the HTM implementation. Figure 5.10 showcases the significant difference between the iterated synapses on vanilla Spatial Pooler algorithm and the mirror synapses optimization. The vanilla Spatial Pooler iterates 13933 synapses for the overlap input calculation on the proximal dendrites of layer 2. On the other hand, mirror synapses optimization only iterate 582 proximal synapses for the same computation. These results underline the importance of this optimization regarding the real-time performance.



(a) 13933 synapses iterated on vanilla Spatial Pooler.

(b) 582 synapses iterated on mirror synapses optimization.

Figure 5.10: Impact of the optimization on the number of proximal synapses iterated for input overlap calculations. L3 columnar activation is the input for the proximal dendrites of L2 columns in this figure.

CHAPTER 6

RESULTS

This chapter presents the results of the testing scenario along with the parameters and hardware.

6.1 Testing Platform and Parameter Initialization

Parameter initializations and their explanations are in Appendix A. The implementation is tested on a mid-level laptop with the following specifications:

- Intel Core i7-3632QM CPU @ 2.20GHz with 3.20GHz Boost Mode
- 12 GB 1600MHz CL 11-11-11-28 DDR3 RAM
- Nvidia GT645M GPU
- Samsung 850 Pro 256 GB SSD

6.2 Scenario Results

The results are obtained from the scenario shown in Figure 6.1. To recap the scenario, the agent respawns on a random cell and with a random direction whenever it reaches the portal or dies because it went out of bounds. A positive reward is given whenever the agent accomplishes the task. If the agent traverses out of the valid area outlined by the blue Voronoi cell coloring, it is stimulated with negative reward.

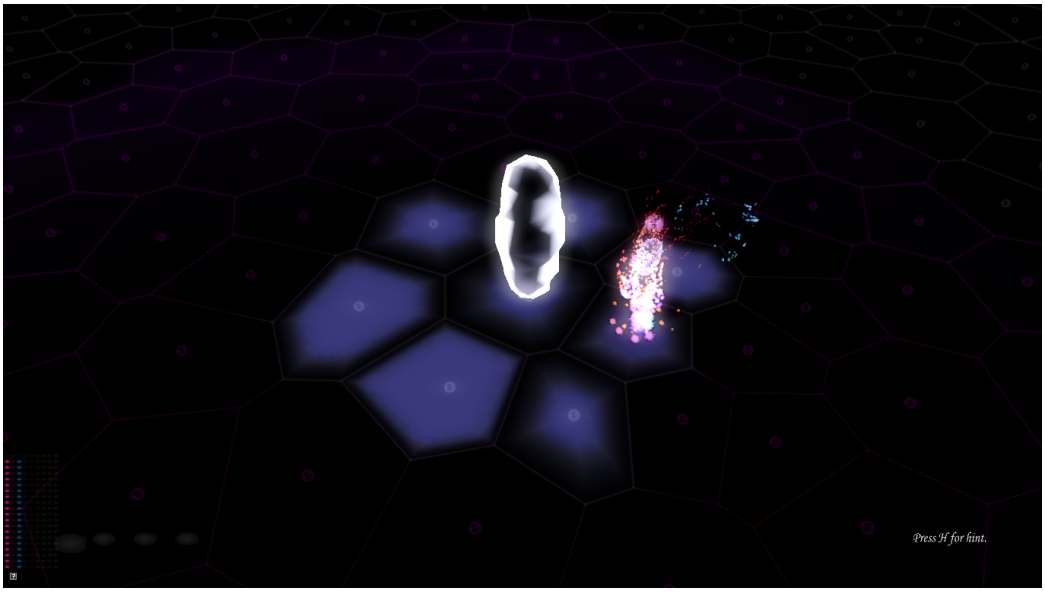


Figure 6.1: This figure shows the learning scenario. The agent is stimulated by positive reward when it navigates to the portal. Negative reward is given if it traverses out of bounds. After each trial, the agent respawns at a random Voronoi cell. The agent starts with random movement and learns better behaviors in terms of reward during the scenario.

6.2.1 Serialization Results

Figures 6.2, 6.3, 6.4 present the serialization performance with respect to the synapse count and variable layer sizes. All the figures displayed in this section belong to the same test run and as a result share the same iterations. Exact data on serialization is presented on Table 6.1. The architecture is in a non-reset, non-pooled segments configuration with parameters $lambda = 0.6$, $learningrate = 0.5$, $discountrate = 0.95$ and $800x8layersizes$; 800 columns per layer and 8 neurons per column.

Table 6.1: The exact results from serialization measurements.

Iteration	Proximal	Distal	Apical	Total	Save	Load	Size
0	$6.24 \cdot 10^6$	0	0	$6.24 \cdot 10^6$	1.46	1.47	27.75
1,000	$6.24 \cdot 10^6$	$5.3 \cdot 10^5$	$1.67 \cdot 10^5$	$6.93 \cdot 10^6$	1.75	1.73	30.92
2,000	$6.24 \cdot 10^6$	$9.74 \cdot 10^5$	$2.72 \cdot 10^5$	$7.48 \cdot 10^6$	1.87	1.94	33.43
3,000	$6.24 \cdot 10^6$	$1.35 \cdot 10^6$	$3.42 \cdot 10^5$	$7.93 \cdot 10^6$	2	2.1	35.47
4,000	$6.24 \cdot 10^6$	$1.72 \cdot 10^6$	$4.17 \cdot 10^5$	$8.38 \cdot 10^6$	2.08	2.28	37.49
5,000	$6.24 \cdot 10^6$	$2.06 \cdot 10^6$	$4.74 \cdot 10^5$	$8.77 \cdot 10^6$	2.18	2.41	39.27
6,000	$6.24 \cdot 10^6$	$2.35 \cdot 10^6$	$5.23 \cdot 10^5$	$9.11 \cdot 10^6$	2.29	2.63	40.82
7,000	$6.24 \cdot 10^6$	$2.68 \cdot 10^6$	$5.71 \cdot 10^5$	$9.49 \cdot 10^6$	2.39	2.71	42.55
8,000	$6.24 \cdot 10^6$	$2.99 \cdot 10^6$	$6.08 \cdot 10^5$	$9.84 \cdot 10^6$	2.49	2.79	44.13
9,000	$6.24 \cdot 10^6$	$3.27 \cdot 10^6$	$6.34 \cdot 10^5$	$1.01 \cdot 10^7$	2.56	2.91	45.52
10,000	$6.24 \cdot 10^6$	$3.55 \cdot 10^6$	$6.69 \cdot 10^5$	$1.05 \cdot 10^7$	2.64	3.03	46.95
11,000	$6.24 \cdot 10^6$	$3.82 \cdot 10^6$	$7.03 \cdot 10^5$	$1.08 \cdot 10^7$	2.71	3.14	48.38
12,000	$6.24 \cdot 10^6$	$3.96 \cdot 10^6$	$7.29 \cdot 10^5$	$1.09 \cdot 10^7$	2.76	3.21	49.2
13,000	$6.24 \cdot 10^6$	$3.95 \cdot 10^6$	$7.43 \cdot 10^5$	$1.09 \cdot 10^7$	2.77	3.24	49.21
14,000	$6.24 \cdot 10^6$	$3.93 \cdot 10^6$	$7.68 \cdot 10^5$	$1.09 \cdot 10^7$	2.79	3.28	49.27
15,000	$6.24 \cdot 10^6$	$3.96 \cdot 10^6$	$7.85 \cdot 10^5$	$1.1 \cdot 10^7$	2.79	3.25	49.47
16,000	$6.24 \cdot 10^6$	$3.96 \cdot 10^6$	$7.98 \cdot 10^5$	$1.1 \cdot 10^7$	2.78	3.28	49.57
17,000	$6.24 \cdot 10^6$	$3.99 \cdot 10^6$	$8.05 \cdot 10^5$	$1.1 \cdot 10^7$	2.81	3.29	49.76
18,000	$6.24 \cdot 10^6$	$4 \cdot 10^6$	$8.21 \cdot 10^5$	$1.11 \cdot 10^7$	2.81	3.32	49.89
19,000	$6.24 \cdot 10^6$	$4.02 \cdot 10^6$	$8.31 \cdot 10^5$	$1.11 \cdot 10^7$	2.8	3.32	50.02
20,000	$6.24 \cdot 10^6$	$4.04 \cdot 10^6$	$8.41 \cdot 10^5$	$1.11 \cdot 10^7$	2.88	3.33	50.18

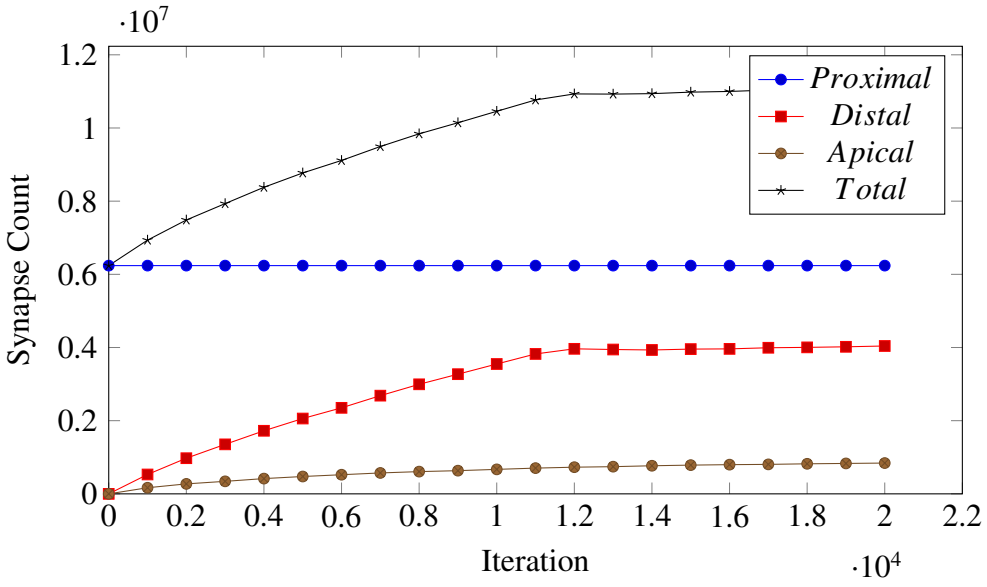


Figure 6.2: Synapse count increase with respect to iterations.

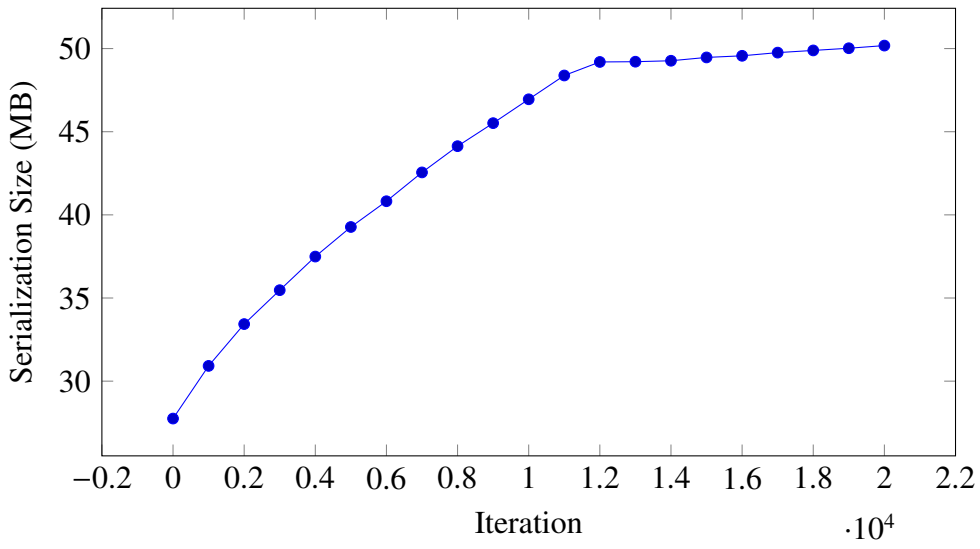


Figure 6.3: Serialization size growth with respect to iterations.

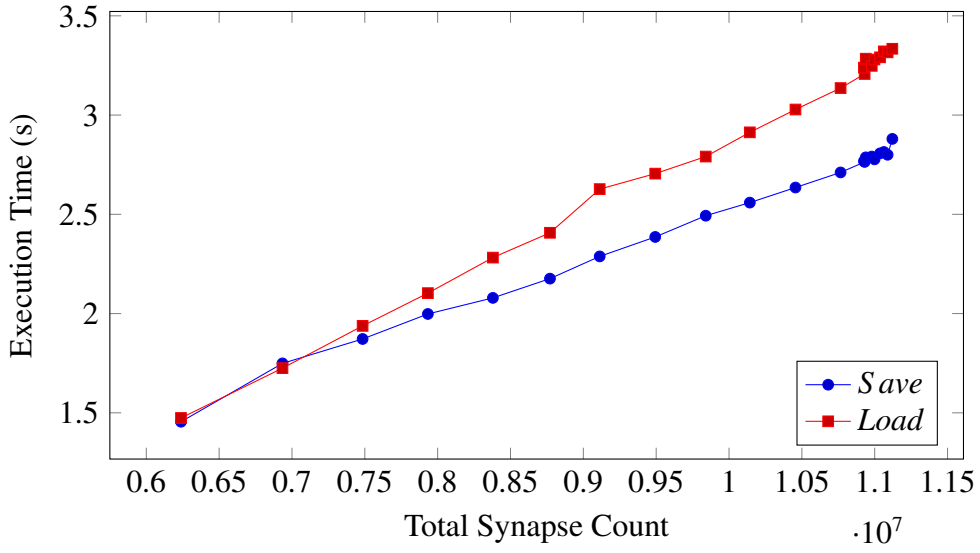


Figure 6.4: Serialization timings due to synapse count.

6.2.2 Runtime Results

The execution times of the individual algorithmic phases of Cortical System are presented in Figure 6.5. The impact of layer sizes on real-time performance is in Figure 6.6.

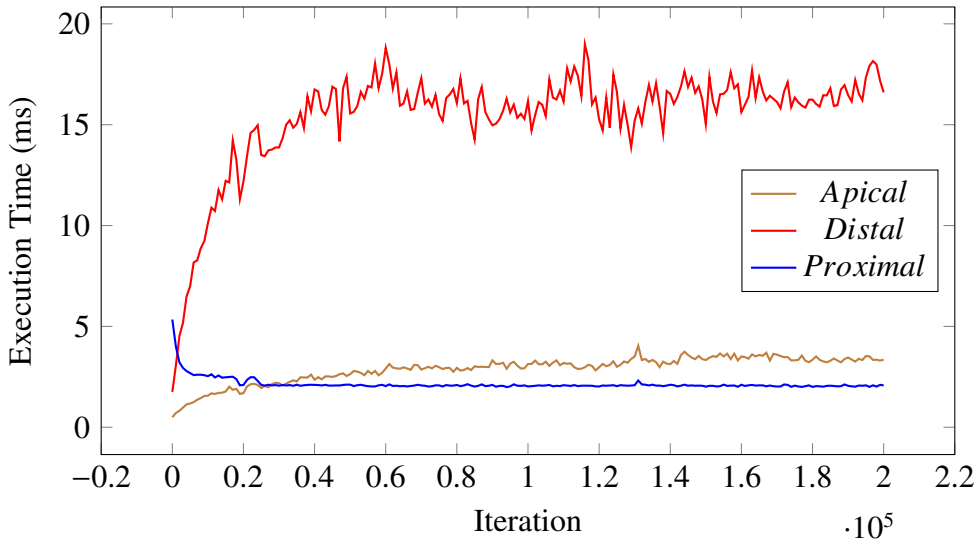


Figure 6.5: Execution timings of the modifications to proximal synapses (Spatial Pooler), distal synapses (Temporal Memory) and apical synapses (Apical Temporal Memory). The timings show the total cost for all HTM layers of the architecture. Non-reset, non-pooled segments configuration with parameters *lambda* = 0.6, *learningrate* = 0.5, *discountrate* = 0.95 and 800*x*8*layersizes*.

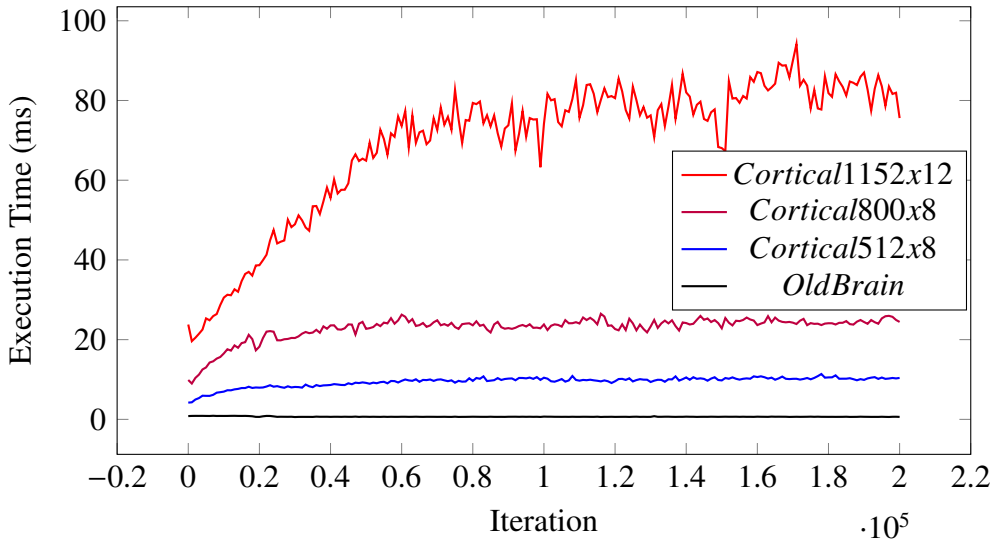
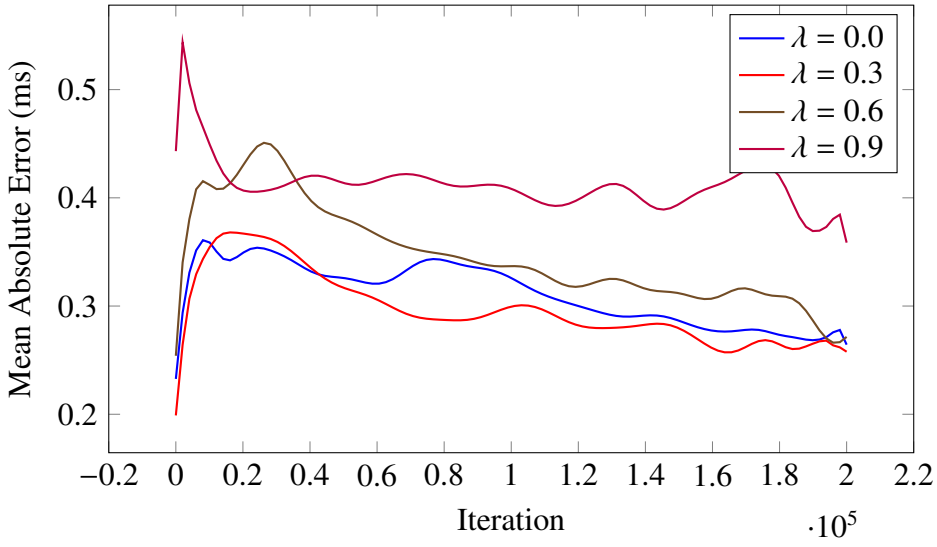


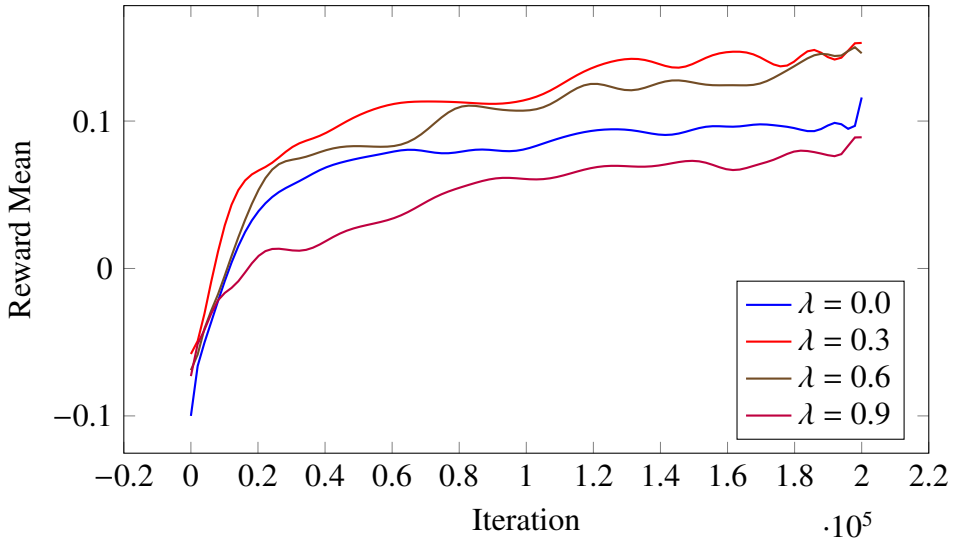
Figure 6.6: Total Cortical System measurements due to layer sizes of the architecture. Non-reset, non-pooled segments configuration with parameters *lambda* = 0.6, *learningrate* = 0.5 and *discountrate* = 0.95.

6.2.3 Learning Results

Learning convergence results depending on the lambda parameter of TD(λ) are presented in Figure 6.7 using Mean Absolute Value and actual reward averages as metrics. The average reward graph represents the task completion efficiency because the positive reward is only stimulated on task completion and the agent is stimulated by negative reward when it traverses to an invalid Voronoi cell. The impact of layer size is shown in Figure 6.8. Data points of the graphs are smoothed out by Bezier curves for more comprehensible plots. Plots generated by raw data is in Appendix B.

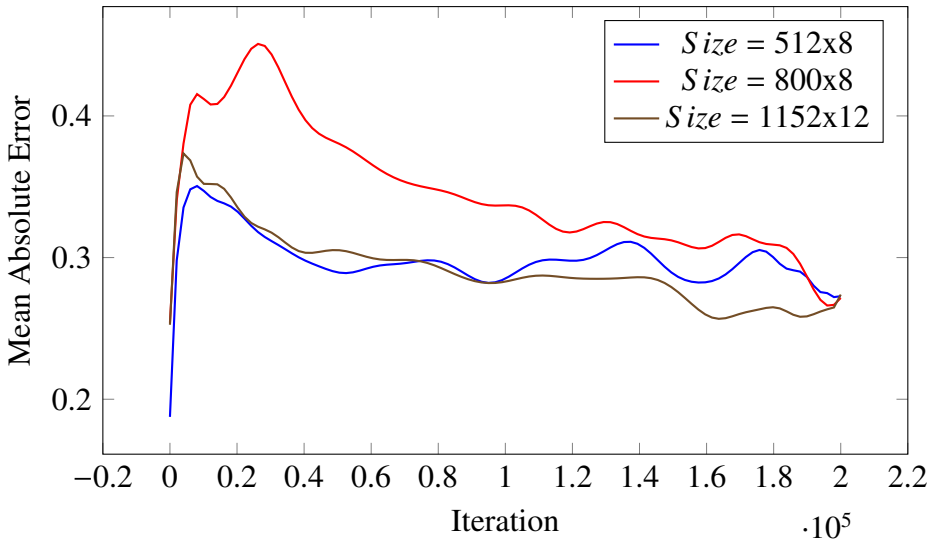


(a) The impact of lambda on MAE.

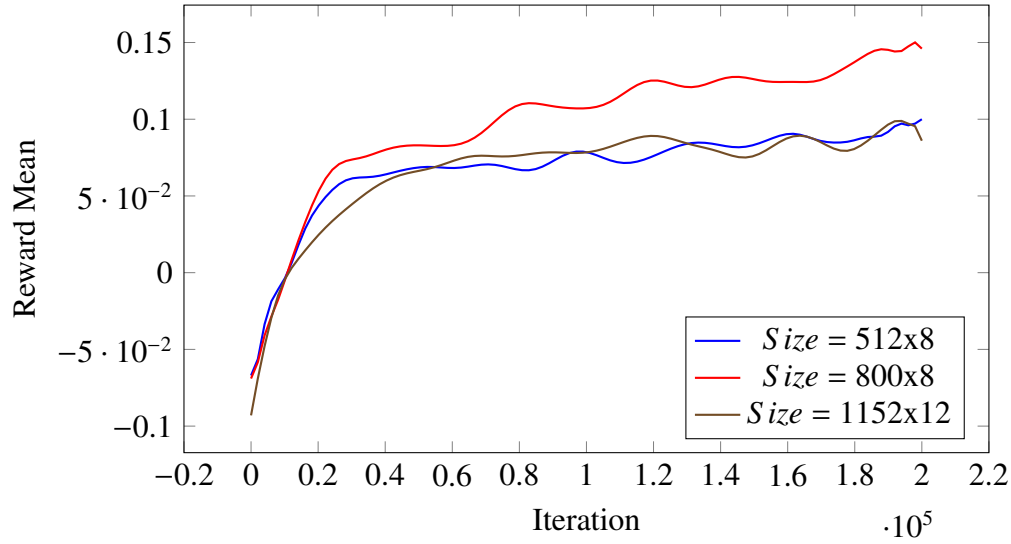


(b) The impact of lambda on reward mean.

Figure 6.7: The impact of lambda parameter on learning measurements. Non-reset, non-pooled segments configuration with parameters $learningrate$ = 0.5, $discountrate$ = 0.95 and 800x8$layersizes$.



(a) The impact of layer size on MAE measurements.

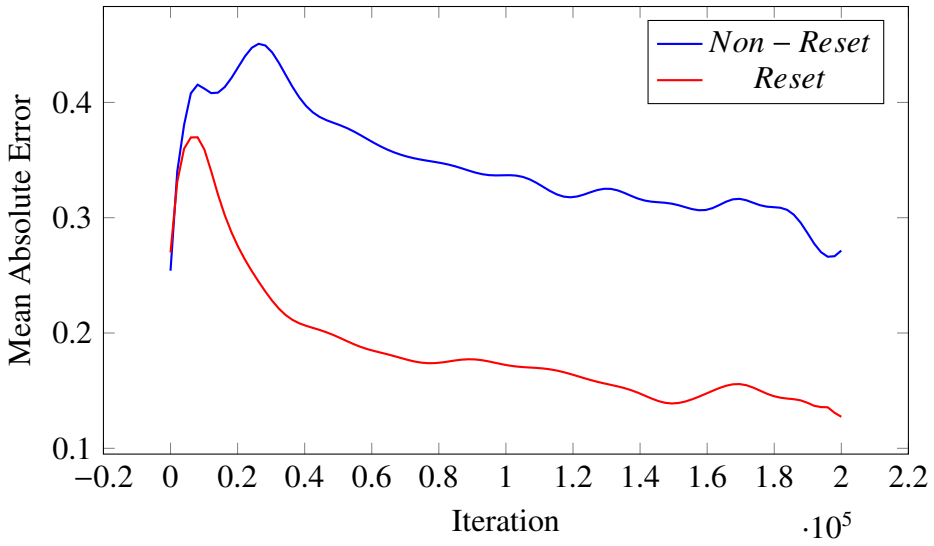


(b) The impact of layer size on reward measurements.

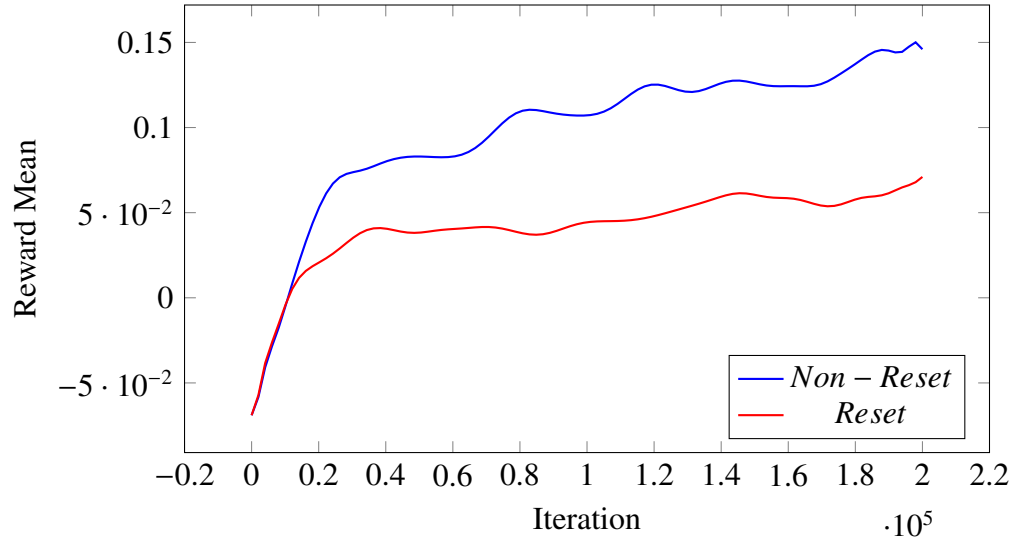
Figure 6.8: The impact of layer size on learning performance. Non-reset, non-pooled segments configuration with parameters *lambda* = 0.6, *learningrate* = 0.5 and *discountrate* = 0.95.

6.2.4 Results for Reset and Segment Eligibility Trace Features

The spatial and temporal activities of the architecture can be reset when the agent respawns which indirectly ends the currently learned sequence. The practical difference of the reset mechanism and the eligibility trace based adaptation for segments feature (Section 5.4.1) are presented in the Figures 6.9 and 6.10.

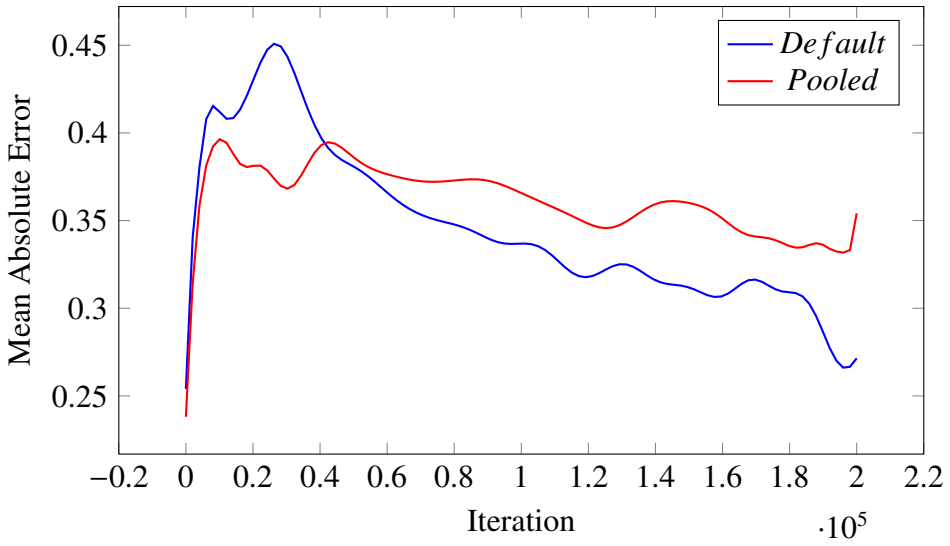


(a) Reset mechanism impact on MAE.

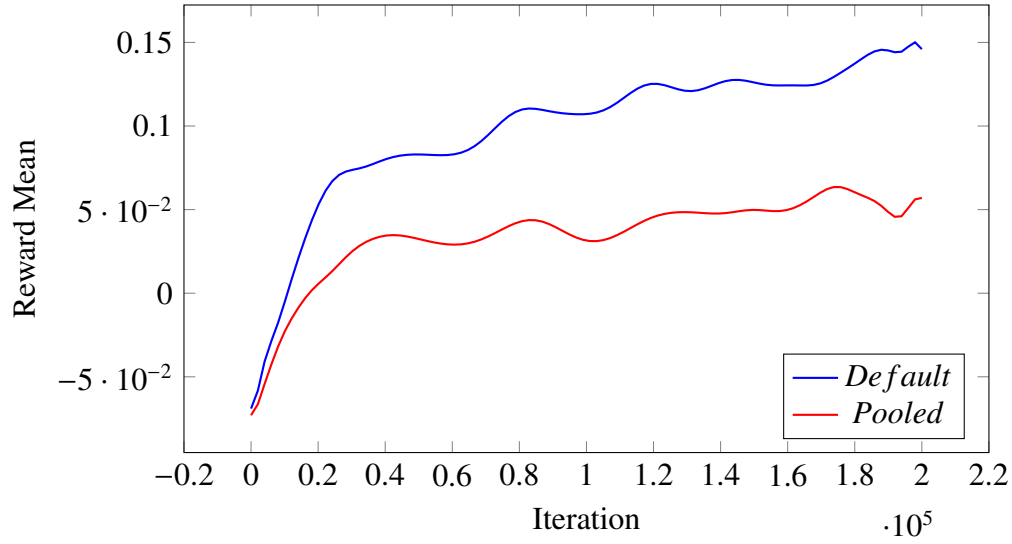


(b) Reset mechanism impact on reward averages.

Figure 6.9: Reset mechanism comparison for non-pooled segments configuration with parameters *lambda* = 0.6, *learningrate* = 0.5, *discountrate* = 0.95 and 800*x*8*layersizes*.



(a) The impact of segment eligibility traces feature on MAE.



(b) The impact of segment eligibility traces feature on reward average.

Figure 6.10: Pooled segment adaptation mechanism comparison for non-reset configuration with parameters *lambda* = 0.6, *learningrate* = 0.5, *discountrate* = 0.95 and 800*x*8*layersizes*.

CHAPTER 7

DISCUSSION

7.1 Parameter Tweak Remarks

The default architecture parameters are presented in Appendix A. This section provides the reader with remarks on important parameters of the architecture.

7.1.1 Spatial Pooler

The most important Spatial Pooler parameters are as follows:

- Local Area Density (Sparsity)
- Boost Strength
- Permanence Increment
- Permanence Decrement

An HTM layer has a fixed level of activation *sparsity* representing the input patterns. This *sparsity* is usually set to 2% in the open source implementation of HTM - Nupic. Moreover, in a publication by Hawkins et al. [14] arguing about the theory of sequence memory in the neocortex, *sparsity* is set to 2% for some of the HTM layer examples. Numenta claims that values around 0.05% and 4% provide the best capacity utilization while not undermining the ability to show similarities and differences between different representations. Layers can learn quickly on lower *sparsity* such as 10%, but the system suffers from spatiotemporal capacity according to Numenta.

Booststrength controls how strong the lesser used neurons are reinforced via artificial overlap scaling and the permanence increments applied every frame. A value of 4 provides adequate reinforcement to allow a more uniform usage among the columns. Higher values provide better uniformity but also cause greater instability while learning because of the frequent representational changes.

The *increment* and *decrement* values control the specialization degree of columns onto spatial patterns. If the *increment* is lower than the *decrement*, the columns tend

to specialize on one or two input patterns depending on the ratio between these values. If the ratio between *increment* and *decrement* increases, the columns can preserve synapses representing different input patterns. As a result, these values also control the degree of columnar representation overlap on different input patterns. The representational overlap between differing inputs is proportional to the ratio between synaptic *increment* and *decrement*. Therefore, setting the *increment* value higher than *decrement* value allows the columns to participate in the representation of multiple patterns resulting in the competition mechanism of Spatial Pooler to work as intended which activates the best overlapping columns.

7.1.2 Temporal Memory and Apical Temporal Memory

The most critical Temporal Memory parameters are as follows:

- Activation Threshold
- Matching Threshold
- Maximum New Synapses
- Permanence Increment
- Permanence Decrement

Parameters of Temporal Memory are adjusted based on the minimum and maximum active cells for a given HTM layer. Spatial Pooler dictates the number of active columns at a given time for an HTM layer according to *sparsity*. The number of active cells at a given time is at least equal to the active column count (sparse activation) and at most equal to the active column count times neurons per column (bursting activation). As an example, a layer with 2% *sparsity* and 2048 columns with 16 neurons per column would result in 40 active cells on sparse activation and 640 active cells on bursting activation. For a distal segment to learn and adapt its synapses, it has to be a matching segment at least. In other words, the activity on its synapses should exceed the *matchingthreshold*. Therefore, the *matchingthreshold* cannot be set higher than active cells at a sparse activation because any created segment would not get activated on a sparse activation. Setting the *matchingthreshold* to 40% of the sparse activation provides the results in this study.

If the activity on a distal segment exceeds *activationthreshold*, the segment becomes active and as a result, the neuron owning that segment becomes depolarized. Therefore, this value also needs to be lower than a sparse activation cell count for a segment to become active on a sparse activation. On the other hand, this parameter should be set above the *matchingthreshold* which flags a segment when it has the potential to match an activation. In that case, the segment adapts to that activation without the need for creating a new one. As a result, *activationthreshold* should be set higher than *matchingthreshold* to work as intended. This study sets *activationthreshold* around 60% of the sparse activation.

Distal segments are meant to subsample their sources to create synaptic variety. In other words, the segments grow synapses to only a group of active cells from the previous activation. This mechanism prevents all the learning segments of the same time step from forming the same synapses. Moreover, it introduces better synaptic variety when learning temporal patterns of activation. Therefore, $maximumnewsynapses$ should be lower than a sparse activation to result in subsampling. This study sets the value around 70% of sparse activation.

Synaptic *increment* and *decrement* values control the specialization degree of a segment to its receptive input pattern. Similar to Spatial Pooler parameters, setting the *increment* value higher than the *decrement* value results in a single segment encapsulating multiple input activations. However, Temporal Memory does not have a competition mechanism like Spatial Pooler, so having higher increment values leads to underfitting to the temporal patterns. This study sets *increment* value lower than the *decrement* value to result in segments that have high specialization on their receptive input patterns.

7.2 Results Evaluation

7.2.1 Serialization

The autonomous agent in this study is intended to be applied to virtual environments in real-time, particularly as a non-player character in a video game. Thus, serialization capabilities are required for the agent to store and callback itself between simulation sessions. The performance and size of the serialization are also crucial for providing an interactive experience and for managing the storage constraints of a video game. The results are promising in these departments for the agent to function as an NPC.

The results presented in Figure 6.2 show the synapse counts with respect to learning iterations. As seen from the figure, the proximal synapses have a fixed quantity throughout the learning due to the Spatial Pooler algorithm. They make up almost half of the total synapses even at the later stages of learning, around iteration 200000. On the other hand, the distal and apical synapses show a logarithmic increase in quantity and they are almost equal in numbers to proximal synapses at the end of the test run. Distal synapses encode the temporal patterns of the neural activations via the Temporal Memory algorithm. So, the architecture forms more distal synapses as it encounters new activation transitions. This is why there is a significant increase on the number of distal synapses during learning.

Figure 6.3 presents the size footprint of the serialization. Naturally, the size increase is proportional to synapse count increase as the synapse data take the most amount of space in the serialization file. The size increase slows over time and stabilizes around 20000 iterations to 50 Mbytes which is a reasonable storage size for an NPC. The serialization ruleset and the stored synapse structure is explained in Section 3.4.

Figure 6.4 shows the relation between save/load times and synapse counts. After starting with similar execution times, loading becomes slower to carry out. They are

both around 3 seconds even at 10 million synapses. It is important to note that the test hardware utilizes a Solid State Disk which is considerably faster than a traditional Hard Disk. Still, the results on save/load execution times are promising for an NPC with 10 million synapses.

7.2.2 Runtime

Functionality in real-time is crucial for an interactive system, especially an NPC. Figure 6.5 measures the execution times of three main algorithmic components of the architecture: Spatial Pooler, Temporal Memory, and Apical Temporal Memory. The results show that the majority of processing power is taken by the distal synapses, Temporal Memory. It runs just under $20ms$ on layer sizes of 800 columns and 8 neurons per column which is acceptable at best for real-time performance. The execution time of Apical Temporal Memory increases as the connectome grows more apical synapses, but compared to Temporal Memory, it does not have a large impact with $4ms$. One interesting finding is that the execution time of Spatial Pooler actually decreases over time and converges to a stable value of $2ms$. Proximal synapses have fixed quantities unlike the apical and distal synapses which increase over time. Proximal dendrites of columns represent and specialize in multiple input patterns. Fewer modifications on proximal synapses are required as the HTM layers adapt to their proximal inputs which explains the decrease in execution times.

Figure 6.6 compares the execution times of different layer sizes. The obvious takeaway from the graph is that the performance does not scale linearly with respect to the column and neuron counts. At the last iteration of the test run, 800x8 layer size takes $24ms$ to execute while the 512x8 layer takes $10ms$. 1512x12 layer takes around $75ms$ to execute which is around 7.5 times the computational cost of a 512x8 layer. In a book about the organization of the brain [59], Llinas and Pare state that the processing of sensory information is in discrete time steps with a lower bound of $12ms$ based on their experimental observations. The runtime performance of our architecture is not far with a range of $10ms$ to $75ms$ depending on the layer size. The $24ms$ execution time of the default layer size is relatively close to their observations. The black line in the graph indicates the execution time of Old Brain System which updates the visual sensor of the agent by raycasting the environment. However, it is negligible with around $1ms$ of execution time with 40x20 rays, and it is constant as expected.

The runtime results show that an agent with 512x8 layer size is viable for real-time performance at this stage of the study. On the other hand, the default layer size of the architecture is 800x8 because it can still run in the range of interactive frames per second (FPS). This layer size configuration also provides a better capacity for learning patterns as shown in the Learning Section sec:learningResults of the results.

7.2.3 Learning

The primary goal of the agent is to model its environment via its visual sensor to learn rewarding behaviors. The figures presenting the learning characteristics suggest that the architecture is capable of modeling the environment while minimizing the

prediction errors and increasing the task completion rates. The agent becomes more efficient at reaching the portal as it learns. The number of actions it does for task completion decreases in time because actions that result in quicker reward are encouraged by the architecture. So, the agent finds more optimal action sequences that lead to the reward. In addition, the agent moves in a random-walk fashion prior to any learning. As a result, the initial values of all the graphs represent the random-walk measurements for comparison. It is important to remember that the graphs presented in this Section 6.2.3 are smoothed out for comprehension. Appendix B includes the raw plots of the data points.

Mean Absolute Value is a standard metric to measure the accuracy of models that incorporate Temporal Difference Learning (λ). It shows how much the predictions about the upcoming state values differ from the actual state values. Among all the parameter combinations tested, MAE decreases as the agent models its environment. Figure 6.7 presents the learning capability with respect to the λ parameter by measuring the MAE and average actual reward with a variety of parameter values. For all cases, MAE initially increases because the agent explores the states that were previously not encountered. For all λ values, the MAE measurements start to decrease at some point and the decrease slows over time. The best λ value regarding MAE is 0.3 from the graph with 0.6 and 0.0 resulting in a similar MAE at the end of the run. A λ value of 0.9 provided significantly worse MAE compared to the rest. On the other hand, average reward graph representing the task efficiency declares 0.6 and 0.3 values to be the best in our test runs. Considering both graphs, it is safe to say that a λ of 0.6 and 0.3 performs reasonably similar to each other. They also provide better learning than the 0.0 and 0.9 values.

Figure 6.8 shows the impact of layer sizes on learning. In terms of MAE, a layer with size 800x8 starts of much worse compared to the others. However, all layer sizes end up with similar MAE after 200000 iterations. In terms of average reward, 800x8 layer is the best configuration which performs 50% better than 512x8 and 1152x12 layer sizes. It should be noted that although the architecture parameters are initialized according to layer sizes, much of the experiments are conducted on 800x8 layers. The performance difference in reward averages may be due to tweaking the system at 800x8 layer configuration most of the time.

7.2.4 Reset and Segment Eligibility Trace Features

Vanilla HTM runs on input data that include manual reset points to help with the sequence learning. This study runs HTM in a non-reset configuration, but comparison with a reset configuration is provided nonetheless. In the reset configuration, all the spatiotemporal activity of the HTM layers are cleared after the agent respawns on a random Voronoi cell caused by task completion or going out of bounds.

Figure 6.9 presents the results for comparison between the reset and non-reset configurations. There is a substantial difference between the MAE measurements. Reset configuration yields 600% better performance compared to non-reset configuration. This makes sense considering that the states belonging to different task sequences are modeled in isolation using manual resets. Without manual resets, the states of

a previous task trial may get updated with the errors from the current task behavior. However, non-reset configuration provides a significantly better reward mean in the long run compared to the reset configuration. This may be due to the agent having a deeper temporal model which also encapsulates the transitions between different trials. Non-reset agent models sequences that include multiple trials and random respawns. However, this results in more expected value error. Therefore, it is difficult to decide on a clear winner configuration in the current state of the study. Non-reset configuration has a reward mean of almost three times the reset configuration. On another note, the default configuration of the architecture is non-reset because it provides a more rewarding agent life cycle supported by Figure 6.9.

The last Figure 6.10 compares the default architecture with a feature that allows segment adaptations based on eligibility traces. The errors on state values propagate back in time to recently activated neurons in TD(λ). This feature allows the reward to adapt the synapses that caused those recent activations. Moreover, the number of adapted synapses per iteration is over 10 times more with this feature enabled. Neurons essentially have a history of active segments that are pooled and adapted using the errors calculated by TD(λ). However, the results strongly suggest that this feature needs improvements to provide learning at the level of default settings. Both MAE and reward average results show inferior performance with the pooled segments feature enabled. Improvement on this feature may provide better results, but the initial tests suggest that learning without this feature is objectively better in its current form.

7.3 Mechanisms With Inadequate Neurobiological Evidence

The proposed architecture constitutes of mechanisms that have supporting neurobiological research, and these are presented along with the mechanism explanations. However, to ensure functionality, there is a high level of abstraction when interpreting the related research. There are still some components that are lacking relevant biological evidence. This section aims to present some of the known issues regarding these.

7.3.1 Inhibition in HTM

One of the fundamental but lacking features of HTM is the inhibitory neurons. Without any inhibition the biological neural regions would always be encouraged for activation, leading to a chain reaction of all the neurons firing if the stimulation resources allow it. Inhibitory circuits play a crucial role in cortical layers. For example, the work by Naka [23] suggests that inhibitory circuits within layer 5 play a fundamental role in controlling the cortical output.

HTM layers do not have any explicit inhibitory neurons that affect other groups of neurons. However, there are inhibition mechanisms both at the columnar and neural level to ensure sparsity in activations. There is an inhibition phase in Spatial Pooler algorithm in Section 2.3.2.2 that guarantees a fixed number of active columns at all times. This inhibition process among the mini columns is attributed to the func-

tionality of chandelier cells which provide feedforward proximal inhibition for the pyramidal cells [60]. The inhibition process among the neurons of the individual minicolumns is carried out via the Temporal Memory. According to the algorithm, if a mini column is activated and if there is an already depolarized cell among the mini column, this depolarized cell gets activated while inhibiting the rest of the non-depolarized neurons. This functionality is based on the fast-spiking basket cells that have inhibitory influence among the neurons of a mini column [61], [14]. Despite the relevant neurobiological evidence, the inhibition aspect of cortical microcircuits is still an abstraction in HTM theory at this state.

7.3.2 Excitatory and Inhibitory Influence on Different Postsynaptic Targets

There is no biological evidence suggesting that the same presynaptic cortical neuron can have both excitatory (GABAergic) and inhibitory (glutamatergic) influence on different postsynaptic targets. The inhibitory neurons provide same inhibitory influence to all of its targets while excitatory neurons excite all of its postsynaptic targets. In 1954, Eccles et al. proposed this pattern in their work [62] attributing it to the neuroscientist Henry Hallet Dale. Since then, this pattern is referred to as Dale's principle. There are exceptions to this pattern in the vertebrate nervous system [63] and on D1, D2 modulatory dopamine cells [64] which positively and negatively affect their targets. However, there are no known exceptions about cortical neurons in neurobiological literature, especially layer 5 pyramidal neurons. In spite of this, for the sake of simplicity, a single layer 5 neuron in our architecture can do both depending on the origin of its activation: Go and No-Go pathways as shown in Figure 4.8. It can have an inhibitory and excitatory influence on its postsynaptic targets which are motor neurons.

7.3.3 Abstraction of Voluntary Activations

There are voluntary activations excited on layer 5 through its apical dendrites that are stimulated by layers D1 and D2. For the intended functionality of our architecture, this voluntary activation type is ignored by all the layers except motor neurons. So, there are two different activations types in layer 5 happening simultaneously:

- **Normal Activation** : These are activations caused by proximal feedforward input along with the distal and apical depolarization.
- **Voluntary Activation** : These are activations caused by the combined distal apical depolarization reaching a certain threshold. This activation does not need proximal feedforward input and is detected only by the motor neurons.

According to the neural firing rate models, it is known that the regularity and the rate of a neuron's responsive firing are dependent on its synaptic input [65]. It can be argued that the voluntary activated neurons which are stimulated by Go and No-Go pathways have firing characteristics which only the motor neurons are receptive to. In addition, the findings of Fields [66] suggest that the neurons are capable of modifying

conductance delays for computational purposes. This may explain how a layer can selectively pick specific activations among its source for computation while delaying its response to others. There is currently little neurobiological evidence that supports the simultaneous existence of normal and voluntary activations. However, this functionality is an abstraction for the interplay between layer 5 and motor neurons which involve additional components such as layer 6 neurons, thalamus, cerebellum and spinal cord. Ideally, the voluntary activation should inhibit any feedforward proximal activation and as a result, other layers should sample from the overridden voluntary activation. However, this is a profound change that requires further research and modifications to HTM while ensuring a similar motor functionality.

CHAPTER 8

CONCLUSION

This thesis presented the current state of the Hierarchical Temporal Memory theory along with Temporal Difference Learning (λ) with replacing traces. It provided the relevant neurobiological research on the cortical layers and the computational models of basal ganglia. We mechanistically described an autonomous agent architecture based on these methods along with supplementary neurobiological evidence. In our framework, HTM is combined with TD(λ) in a way that abstracts the functionality of layer 5 and its interplay with basal ganglia to generate voluntary behavior. A custom game engine is used as the platform for testing the unsupervised, online and continuous learning architecture on a video game learning task. The agent navigates a three-dimensional and procedurally generated environment where it is punished for going out of the specified area and rewarded for reaching the target portal within the viewing distance of its online visual sensor. Results on this test scenario are presented via Mean Absolute Error (MAE) and moving average of reward with respect to various architecture parameters. The proposed methodology is capable of modeling and navigating spatiotemporal patterns with low complexity which is the current limitation of the architecture. Improvements in hardware may enable real-time functionality of larger HTM layer sizes which would provide better learning, but key features such as attention and temporal abstractions are still lacking. On the other hand, for tasks with low complexity, the agent successfully learns rewarding behavior sequences as demonstrated via the audio-visual material in Appendix B. Moreover, the proposed architecture meets the criteria of this study consisting; real-time performance, serialization capability, player interaction capability, continuous, online and lifelong learning capabilities. Results suggest that the proposed method is promising to function as an autonomous agent intelligence, specifically for a non-player character.

8.1 Future Works

The agent is currently capable of learning navigational tasks that utilize its visual sensor. One of the main future directions is to expand its capability to include terrain interactions. It possibly involves crude haptic sensors that are experimented with during this study. The current movement system of the agent is a rather limited motion type that jumps from point to point. Extending movement capability to result in a more refined motion type and in the long run, learning from the continuous movement are other research directions. In addition, comparison with vanilla HTM on the same

domain is also one of the priorities for reference. The end goal is to complete tasks that require the agent to learn temporal abstractions. This mechanism is referred as temporal pooling/union pooling in HTM theory and is a hot research topic.

REFERENCES

[1] Marek Otahal. "Architecture of Autonomous Agent Based on Cortical Learning". PhD thesis. Jan. 2013. DOI: 10.13140/2.1.3321.0567.

[2] Antonio Sánchez Gómez and Jonathan Shapiro. *Hierarchical Temporal Memory as a reinforcement learning method*. Tech. rep. 2016.

[3] Ali Kaan Sungur and Elif Surer. "Voluntary behavior on cortical learning algorithm based agents". In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, Sept. 2016, pp. 1–7. ISBN: 978-1-5090-1883-3. DOI: 10.1109/CIG.2016.7860428.

[4] Pentti Kanerva. "Sparse distributed memory and related models". In: *Associative Neural Memories: Theory and Implementation*. 1993, pp. 50–76. ISBN: 0-19-507682-6.

[5] Jeff Hawkins and Sandra Blakeslee. *On intelligence*. New York: Henry Holt and Company., 2004, p. 272. ISBN: 978-0805074567.

[6] Dileep George and Jeff Hawkins. "Towards a mathematical theory of cortical micro-circuits". In: *PLoS Computational Biology* 5.10 (2009). ISSN: 1553734X. DOI: 10.1371/journal.pcbi.1000532.

[7] Dileep George. "How the brain might work: a hierarchical and temporal model for learning and recognition". In: (2008).

[8] Gerard J. Rinkus. "A cortical sparse distributed coding model linking mini- and macrocolumn-scale functionality". In: *Frontiers in Neuroanatomy* 4 (June 2010), p. 17. ISSN: 16625129. DOI: 10.3389/fnana.2010.00017.

[9] D. Mumford. "On the computational architecture of the neocortex - I. The role of the thalamo-cortical loop". In: *Biological Cybernetics* 65.2 (1991), pp. 135–145. ISSN: 03401200. DOI: 10.1007/BF00202389. arXiv: arXiv:1408.1149.

[10] D. Mumford. "On the computational architecture of the neocortex - II The role of cortico-cortical loops". In: *Biological Cybernetics* 66.3 (1992), pp. 241–251. ISSN: 03401200. DOI: 10.1007/BF00198477. arXiv: arXiv:1408.1149.

[11] Adam H Marblestone, Greg Wayne, and Konrad P Kording. "Toward an Integration of Deep Learning and Neuroscience". In: *Frontiers in Computational Neuroscience* 10 (2016), p. 94. ISSN: 1662-5188. DOI: 10.3389/fncom.2016.00094.

[12] Pietro Mazzoni, Richard A. Andersen, and Michael I. Jordan. "A more biologically plausible learning rule than backpropagation applied to a network

model of cortical area 7a". In: *Cerebral Cortex* 1.4 (1991), pp. 293–307. ISSN: 14602199. DOI: 10.1093/cercor/1.4.293.

[13] Yuwei Cui, Subutai Ahmad, and Jeff Hawkins. "Continuous Online Sequence Learning with an Unsupervised Neural Network Model". In: *Neural Computation* 28.11 (Nov. 2016), pp. 2474–2504. ISSN: 0899-7667. DOI: 10.1162/NECO_a_00893.

[14] Jeff Hawkins and Subutai Ahmad. "Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex". In: *Frontiers in Neural Circuits* 10 (Mar. 2016), p. 23. ISSN: 1662-5110. DOI: 10.3389/fncir.2016.00023.

[15] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. "Unsupervised real-time anomaly detection for streaming data". In: *Neurocomputing* 262 (2017), pp. 134–147. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.04.070.

[16] Andre M Bastos, W Martin Usrey, Rick A Adams, George R Mangun, Pascal Fries, and Karl J Friston. "Canonical microcircuits for predictive coding." In: *Neuron* 76.4 (Nov. 2012), pp. 695–711. ISSN: 1097-4199. DOI: 10.1016/j.neuron.2012.10.038.

[17] Rita Bopp, Simone Holler-Rickauer, Kevan A.C. Martin, and Gregor F.P. Schuhknecht. "An Ultrastructural Study of the Thalamic Input to Layer 4 of Primary Motor and Primary Somatosensory Cortex in the Mouse". In: *The Journal of Neuroscience* 37.9 (2017), pp. 2435–2448. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.2557-16.2017.

[18] Neel T Dhruv. "Rethinking canonical cortical circuits". In: *Nature Neuroscience* 18.11 (Oct. 2015), pp. 1538–1538. ISSN: 1097-6256. DOI: 10.1038/nn1115-1538.

[19] Laura A DeNardo, Dominic S Berns, Katherine DeLoach, and Liqun Luo. "Connectivity of mouse somatosensory and prefrontal cortex examined with trans-synaptic tracing." In: *Nature neuroscience* 18.11 (Nov. 2015), pp. 1687–1697. ISSN: 1546-1726. DOI: 10.1038/nn.4131.

[20] Christine M Constantinople and Randy M Bruno. "Deep cortical layers are activated directly by thalamus". In: *Science* 1591.June (2013), pp. 1591–1594. ISSN: 1095-9203. DOI: 10.1126/science.1236425.

[21] D. Schubert, R. Kötter, H. J. Luhmann, and J. F. Staiger. "Morphology, electrophysiology and functional input connectivity of pyramidal neurons characterizes a genuine layer Va in the primary somatosensory cortex". In: *Cerebral Cortex* 16.2 (2006), pp. 223–236. ISSN: 10473211. DOI: 10.1093/cercor/bhi100.

[22] Srikanth Ramaswamy and Henry Markram. "Anatomy and physiology of the thick-tufted layer 5 pyramidal neuron." In: *Frontiers in cellular neuroscience* 9.June (2015), p. 233. ISSN: 1662-5102. DOI: 10.3389/fncel.2015.00233.

[23] Alexander Naka and Hillel Adesnik. “Inhibitory Circuits in Cortical Layer 5.” In: *Frontiers in neural circuits* 10.May (2016), p. 35. ISSN: 1662-5110. DOI: 10.3389/fncir.2016.00035.

[24] Jonas A. Hosp and Andreas R. Luft. *Cortical plasticity during motor learning and recovery after ischemic stroke*. 2011. DOI: 10.1155/2011/871296. arXiv: arXiv:1011.1669v3.

[25] Marianela Garcia-Munoz and Gordon W. Arbuthnott. “Basal ganglia-thalamus and the ”crowning enigma””. In: *Frontiers in Neural Circuits* 9 (2015), p. 71. ISSN: 1662-5110. DOI: 10.3389/fncir.2015.00071.

[26] Wen Jun Gao and Ze Hui Zheng. “Target-specific differences in somatodendritic morphology of layer V pyramidal neurons in rat motor cortex”. In: *Journal of Comparative Neurology* 476.2 (2004), pp. 174–185. ISSN: 00219967. DOI: 10.1002/cne.20224.

[27] H G Kim and B W Connors. “Apical dendrites of the neocortex: correlation between sodium- and calcium-dependent spiking and pyramidal cell morphology.” In: *The Journal of neuroscience : the official journal of the Society for Neuroscience* 13.12 (1993), pp. 5301–5311. ISSN: 0270-6474.

[28] Gerald Edelman and Joseph Gally. “Reentry: a key mechanism for integration of brain function”. In: *Frontiers in Integrative Neuroscience* 7 (2013), p. 63. ISSN: 1662-5145. DOI: 10.3389/fnint.2013.00063.

[29] Charles C Lee and S Murray Sherman. “Modulator property of the intrinsic cortical projection from layer 6 to layer 4.” In: *Frontiers in systems neuroscience* 3.February (2009), p. 3. ISSN: 1662-5137. DOI: 10.3389/neuro.06.003.2009.

[30] M. Morishima. “Recurrent Connection Patterns of Corticostriatal Pyramidal Cells in Frontal Cortex”. In: *Journal of Neuroscience* 26.16 (2006), pp. 4394–4405. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.0252-06.2006.

[31] Juhyun Kim, Chanel J Matney, Aaron Blankenship, Shaul Hestrin, and Solange P Brown. “Layer 6 corticothalamic neurons activate a cortical output layer, layer 5a.” In: *The Journal of neuroscience : the official journal of the Society for Neuroscience* 34.29 (July 2014), pp. 9656–64. ISSN: 1529-2401. DOI: 10.1523/JNEUROSCI.1325-14.2014.

[32] José L. Lanciego, Natasha Luquin, and José A. Obeso. “Functional neuroanatomy of the basal ganglia”. In: *Cold Spring Harbor Perspectives in Medicine* 2.12 (2012). ISSN: 21571422. DOI: 10.1101/cshperspect.a009621.

[33] W Schultz, P Dayan, and P R Montague. “A neural substrate of prediction and reward.” In: *Science* 275.June 1994 (1997), pp. 1593–1599. ISSN: 0036-8075. DOI: 10.1126/science.275.5306.1593. arXiv: NIHMS150003.

[34] Sebastien Helie, Srinivasa Chakravarthy, and Ahmed A. Moustafa. “Exploring the cognitive and motor functions of the basal ganglia: an integrative review of computational cognitive neuroscience models”. In: *Frontiers in Computational*

Neuroscience 7.December (2013), p. 174. ISSN: 1662-5188. DOI: 10.3389/fncom.2013.00174.

[35] Joshua W. Brown, Daniel Bullock, and Stephen Grossberg. "How laminar frontal cortex and basal ganglia circuits interact to control planned and reactive saccades". In: *Neural Networks* 17.4 (2004), pp. 471–510. ISSN: 08936080. DOI: 10.1016/j.neunet.2003.08.006.

[36] Tony J Prescott, Joanna J Bryson, and Anil K Seth. "Introduction. Modelling natural action selection." In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 362.1485 (2007), pp. 1521–9. ISSN: 0962-8436. DOI: 10.1098/rstb.2007.2050.

[37] Katherine H. Taber. "Neuroanatomy of Dopamine: Reward and Addiction". In: *Journal of Neuropsychiatry* 24.1 (2012), p. 1. ISSN: 0895-0172. DOI: 10.1176/appi.neuropsych.24.1.1.

[38] Alia Tewari, Rachna Jog, and Mandar S Jog. "The Striatum and Subthalamic Nucleus as Independent and Collaborative Structures in Motor Control." In: *Frontiers in systems neuroscience* 10.March (2016), p. 17. ISSN: 1662-5137. DOI: 10.3389/fnsys.2016.00017.

[39] Baltazar Zavala, Kareem Zaghloul, and Peter Brown. "The subthalamic nucleus, oscillations, and conflict". In: *Movement Disorders* 30.3 (2015), pp. 328–338. ISSN: 15318257. DOI: 10.1002/mds.26072. arXiv: 15334406.

[40] Michael J. Frank. "Hold your horses: A dynamic computational role for the subthalamic nucleus in decision making". In: *Neural Networks* 19.8 (2006), pp. 1120–1136. ISSN: 08936080. DOI: 10.1016/j.neunet.2006.03.006.

[41] Atsushi Nambu, Hironobu Tokuno, and Masahiko Takada. *Functional significance of the cortico-subthalamo-pallidal 'hyperdirect' pathway*. 2002. DOI: 10.1016/S0168-0102(02)00027-5.

[42] Suzanne N. Haber. *Neuroanatomy of Reward: A View from the Ventral Striatum*. Vol. 5. Md. 2011, pp. 1–24. ISBN: 9781420067262. DOI: http://www.ncbi.nlm.nih.gov/books/NBK92797/.

[43] Stewart Shipp. "Structure and function of the cerebral cortex". In: *Current Biology* 17.12 (June 2007), R443–R449. ISSN: 09609822. DOI: 10.1016/j.cub.2007.03.044.

[44] Randall C. O'Reilly and Michael J. Frank. "Making Working Memory Work: A Computational Model of Learning in the Prefrontal Cortex and Basal Ganglia". In: *Neural Computation* 18.2 (Feb. 2006), pp. 283–328. ISSN: 0899-7667. DOI: 10.1162/089976606775093909.

[45] R D Samson, M J Frank, and Jean-Marc Fellous. "Computational models of reinforcement learning: the role of dopamine as a reward signal." In: *Cognitive neurodynamics* 4.2 (June 2010), pp. 91–105. ISSN: 1871-4099. DOI: 10.1007/s11571-010-9109-x.

[46] Richard S. Sutton. “Learning to Predict by the Methods of Temporal Differences”. In: *Machine Learning* 3.1 (1988), pp. 9–44. ISSN: 15730565. DOI: 10.1023/A:1022633531479.

[47] S P SINGH and R S Sutton. “Reinforcement learning with replacing eligibility traces”. In: *Machine Learning* 22.1-3 (1996), pp. 123–158. ISSN: 0885-6125. DOI: 10.1007/BF00114726.

[48] Harm van Seijen and Richard Sutton. “True Online TD(λ)”. In: *Proceedings of the 31th International Conference on Machine Learning, {ICML} 2014, Beijing, China, 21-26 June 2014* 32 (2014), pp. 692–700. ISSN: 1938-7228. DOI: 10.13140/2.1.1456.2568.

[49] Harm van Seijen, A Rupam Mahmood, Patrick M Pilarski, Marlos C Machado, Richard S Sutton, and Richard S Sutton van Seijen. “True Online Temporal-Difference Learning”. In: *Journal of Machine Learning Research* 17 (2016), pp. 1–40. ISSN: 15337928. arXiv: 1512.04087.

[50] Franz Aurenhammer. “Voronoi diagrams—a survey of a fundamental geometric data structure”. In: *ACM Computing Surveys* 23.3 (1991), pp. 345–405. ISSN: 03600300. DOI: 10.1145/116873.116880.

[51] Stuart P. Lloyd. “Least Squares Quantization in PCM”. In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. ISSN: 15579654. DOI: 10.1109/TIT.1982.1056489.

[52] J.P. Lewis, Matt Cordner, and Nickson Fong. “Pose Space Deformation : A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’00* (2000), pp. 165–172. DOI: 10.1145/344779.344862.

[53] Ken Perlin. “An image synthesizer”. In: *ACM SIGGRAPH Computer Graphics* 19.3 (1985), pp. 287–296. ISSN: 00978930. DOI: 10.1145/325165.325247.

[54] Jeff Hawkins. “Online learning from streaming data”. In: *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management - CIKM ’13*. New York, New York, USA: ACM Press, 2013, pp. 1915–1916. ISBN: 9781450322638. DOI: 10.1145/2505515.2514695.

[55] Gina G. Turrigiano and Sacha B. Nelson. “Homeostatic Plasticity in the Developing Nervous System”. In: *Nature reviews Neuroscience* 5.9 (2004), pp. 97–107. ISSN: 1471-003X. DOI: 10.1038/nrn1327.

[56] Eino Partanen, Teija Kujala, Risto Näätänen, Auli Liitola, Anke Sambeth, and Minna Huotilainen. “Learning-induced neural plasticity of speech processing before birth.” In: *Proceedings of the National Academy of Sciences of the United States of America* 110.37 (Sept. 2013), pp. 15145–50. ISSN: 1091-6490. DOI: 10.1073/pnas.1302159110.

[57] M E Larkum, K M Kaiser, and B Sakmann. “Calcium electrogenesis in distal apical dendrites of layer 5 pyramidal cells at a critical frequency of back-propagating action potentials.” In: *Proceedings of the National Academy of*

Sciences of the United States of America 96.25 (1999), pp. 14600–14604. ISSN: 0027-8424. DOI: 10.1073/pnas.96.25.14600.

[58] Dante S Bortone, Shawn R Olsen, and Massimo Scanziani. "Translaminar inhibitory cells recruited by layer 6 corticothalamic neurons suppress visual cortex." In: *Neuron* 82.2 (Apr. 2014), pp. 474–85. ISSN: 1097-4199. DOI: 10.1016/j.neuron.2014.02.021.

[59] Rodolfo R. (Rodolfo Riascos) Llinas and Patricia Smith. Churchland. *The mind-brain continuum : sensory processes*. MIT Press, 1996, p. 315. ISBN: 9780262121989.

[60] Melis Inan and Stewart A Anderson. "The chandelier cell, form and function." In: *Current opinion in neurobiology* 26 (June 2014), pp. 142–8. ISSN: 1873-6882. DOI: 10.1016/j.conb.2014.01.009.

[61] H. Hu, J. Gan, and P. Jonas. "Fast-spiking, parvalbumin+ GABAergic interneurons: From cellular design to microcircuit function". In: *Science* 345.6196 (Aug. 2014), pp. 1255263–1255263. ISSN: 0036-8075. DOI: 10.1126/science.1255263.

[62] J C ECCLES, P FATT, and K KOKETSU. "Cholinergic and inhibitory synapses in a pathway from motor-axon collaterals to motoneurones." In: *The Journal of physiology* 126.3 (Dec. 1954), pp. 524–62. ISSN: 0022-3751.

[63] Naoshige Uchida. "Bilingual neurons release glutamate and GABA". In: *Nature Neuroscience* 17.11 (2014), pp. 1432–1434. ISSN: 1097-6256. DOI: 10.1038/nn.3840.

[64] D. James Surmeier, Jun Ding, Michelle Day, Zhongfeng Wang, and Weixing Shen. "D1 and D2 dopamine-receptor modulation of striatal glutamatergic signaling in striatal medium spiny neurons". In: *Trends in Neurosciences* 30.5 (May 2007), pp. 228–235. ISSN: 01662236. DOI: 10.1016/j.tins.2007.03.008.

[65] Emilio Salinas and T J Sejnowski. "Impact of correlated synaptic input on output firing rate and variability in simple neuronal models." In: *The Journal of neuroscience : the official journal of the Society for Neuroscience* 20.16 (2000), pp. 6193–6209. ISSN: 0270-6474. DOI: 20/16/6193[pii].

[66] R Douglas Fields. "A new mechanism of nervous system plasticity: activity-dependent myelination." In: *Nature reviews. Neuroscience* 16.12 (2015), pp. 756–67. ISSN: 1471-0048. DOI: 10.1038/nrn4023.

APPENDIX A

Appendix A

A.1 Parameter Initialization

This section presents the important initialization parameters for all layers actuating Spatial Pooler, Temporal Memory and Apical Temporal Memory phases of HTM-TD(λ) implementation. The specialized parameters for different layers are given after the default initialization values. Parameter descriptions are at the end of this section.

Spatial Pooler

- spike type = activation
- synapse maximum permanence = 1
- synapse connection permanence = 0.2
- synapse permanence increment = 0.04
- synapse permanence decrement = 0.004
- base permanence increment = 0.0004
- global potential = true
- potential field percentage = 0.5
- connection percentage = 0.15
- boost strength = 4
- active boost = false

Temporal Memory

- spike type = activation
- synapse maximum permanence = 10
- synapse connection permanence = 0.2

- synapse initial permanence = 0.2
- synapse permanence increment = 0.04
- synapse permanence decrement = 0.08
- inactive decrement = 0.0008
- synaptic decay = 0.000001
- maximum segment count = 128
- maximum new synapses = 12
- activation threshold = 9
- matching threshold = 6
- random distal connections = true
- single cell learning = true

Apical Temporal Memory

- spike type = activation
- synapse maximum permanence = 10
- synapse connection permanence = 0.2
- synapse initial permanence = 0.2
- synapse permanence increment = 0.04
- synapse permanence decrement = 0.08
- inactive decrement = 0.004
- synaptic decay = 0.000001
- maximum segment count = 128
- maximum new synapses = 12
- activation threshold = 9
- matching threshold = 6
- random distal connections = true
- single cell learning = true

A.2 Layer Specific Values

Below are the overridden parameters for specific layers:

Layer 2, Layer 3, Layer 5, D1 and D2

- **spatial pooler** — spike type = weighted

Layer 4

- **spatial pooler** — permanence increment = 0.02
- **spatial pooler** — permanence decrement = 0.02
- **spatial pooler** — potential percentage = 0.25
- **temporal memory** — maximum new synapses = 3
- **temporal memory** — activation threshold = 2
- **temporal memory** — matching threshold = 1
- **temporal memory** — random distal connections = false

Motor Layer

- **apical temporal memory** — inactive decrement = 0.02

A.3 Parameter Descriptions

- **spike type:** Controls which state of the sources to sample. Options include active, predicted and weighted. In weighted setting, predicted sources have a higher weight than active sources
- **maximum synapse permanence:** Limit of synaptic connection strength
- **initial permanence:** The starting connection strength of created synapses
- **connection permanence:** Threshold controlling whether a synapse is connected or not
- **permanence increment:** Permanence increase amount when the source is active
- **permanence decrement:** Permanence decrease amount when the source is inactive
- **inactive decrement:** This variable controls the decay amount of the distal and apical synapses that caused a false positive; a predictive neuron not becoming active on the next iteration

- **synaptic decay:** The amount of global decay applied to all distal and apical synapses
- **maximum new synapses:** The maximum number of new connections that can form onto a segment per iteration
- **maximum segment count:** A cap to the number of segments a neuron can have, for performance reasons
- **activation threshold:** The amount of synaptic activity required for a segment to become active. Similar to stimulus threshold of Spatial Pooler
- **matching threshold:** The amount of potential activity required for a segment to become matching. Unconnected synapses are taken into account to calculate if the segment would become active when they are connected
- **random connections:** Depending on this variable, neurons prefer forming connections onto closer sources or random sources among the previous active sources
- **single cell learning:** Only a single neuron is chosen for learning among the active neurons of a column
- **base permanence increment:** Global growth amount for proximal synapses. It is scaled with the boost value of columns
- **local area density:** This is the sparsity of the activation; ratio of active columns to total columns
- **stimulus threshold:** Controls the threshold for treating the input coming from the proximal dendrite as noise. Any overlap below this threshold is neglected
- **boost strength:** The base value controlling the intensity of columnar competition mechanisms: bumping and boosting. An increase in this variable results in a more uniform activation distribution among the columns but less stability
- **active boost:** It controls whether there is artificial overlap boost on columns based on their usage
- **potential percentage:** Percentage of the total input field that determines the size of a column's receptive field
- **connection percentage:** The percentage of randomly connected synapses at start among the column's receptive field
- **global potential:** Variable controlling whether the columnar receptive fields among the input are topological with respect to the column positions within the layer. Global means no topology

APPENDIX B

Appendix B

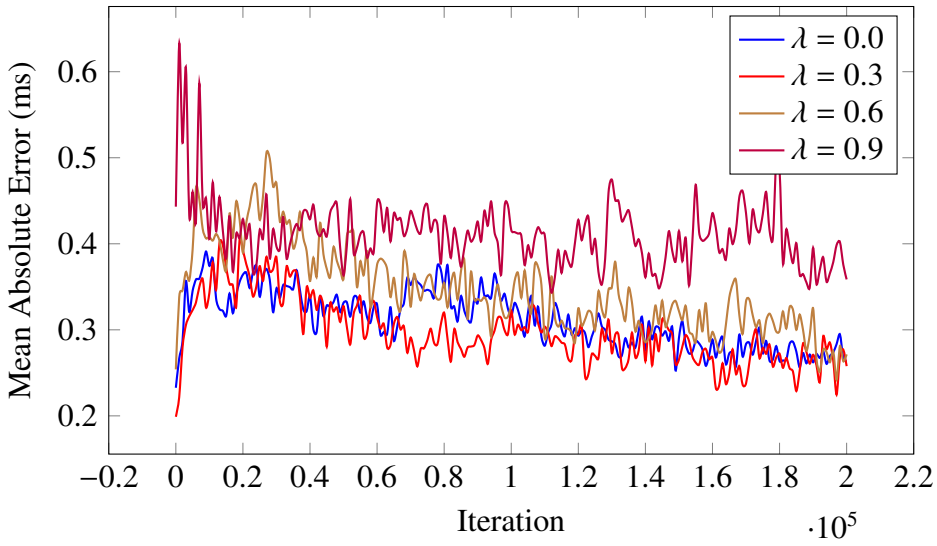
B.1 Video Demonstration

Below is the link for an in-game footage consisting of agent learning demonstration and real-time architecture visualization:
https://youtu.be/pX7DVm50qLY

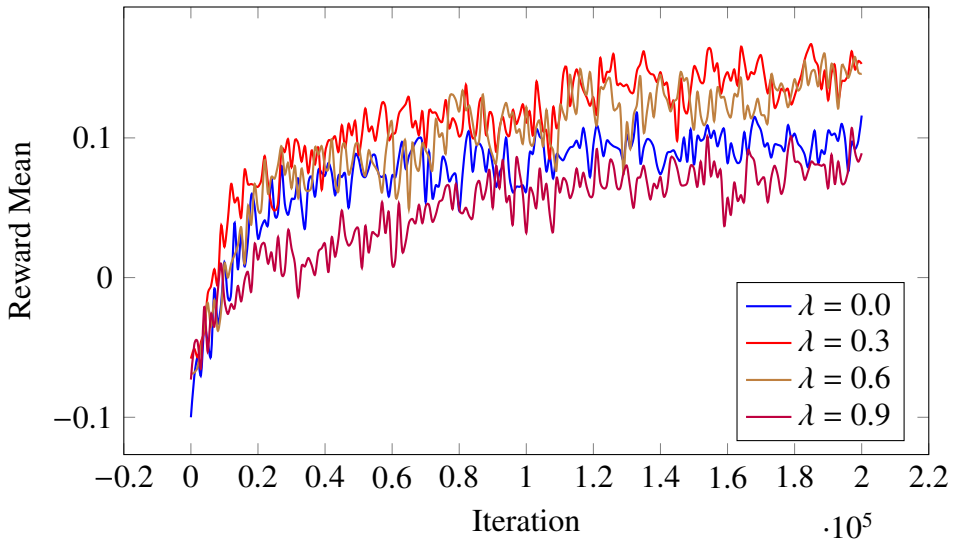
B.2 Results with Raw Plots

The following are the plots generated from the raw data points of the learning sessions.

B.2.1 Learning Results

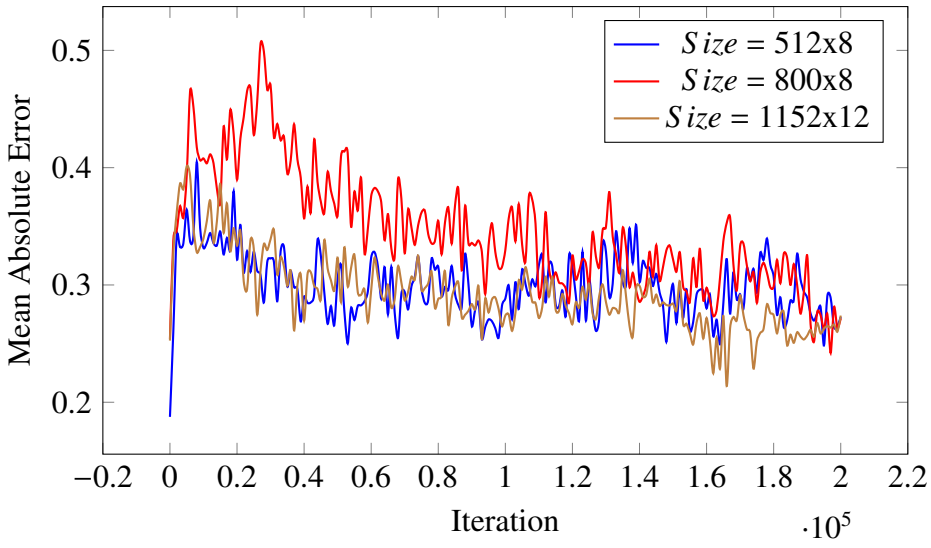


(a) The impact of lambda on MAE.

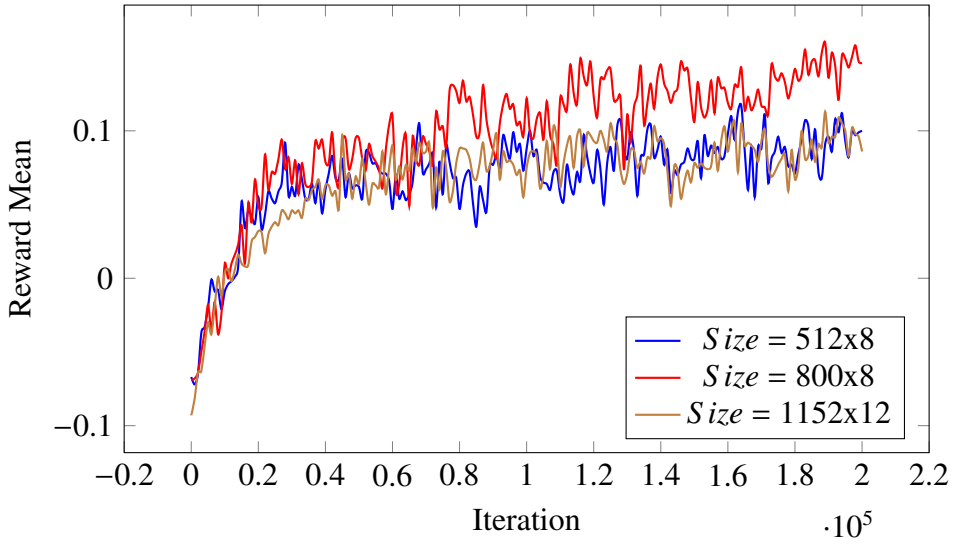


(b) The impact of lambda on reward mean.

Figure B.1: The impact of lambda parameter on learning measurements. Non-reset, non-pooled segments configuration. Learning Rate = 0.5, discount rate = 0.95 and 800x8 layer sizes.



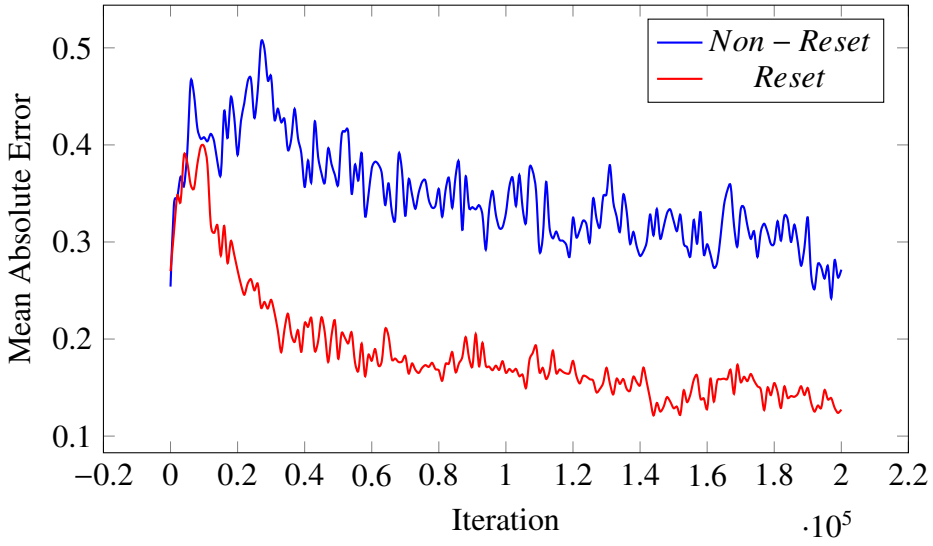
(a) The impact of layer size on MAE measurements.



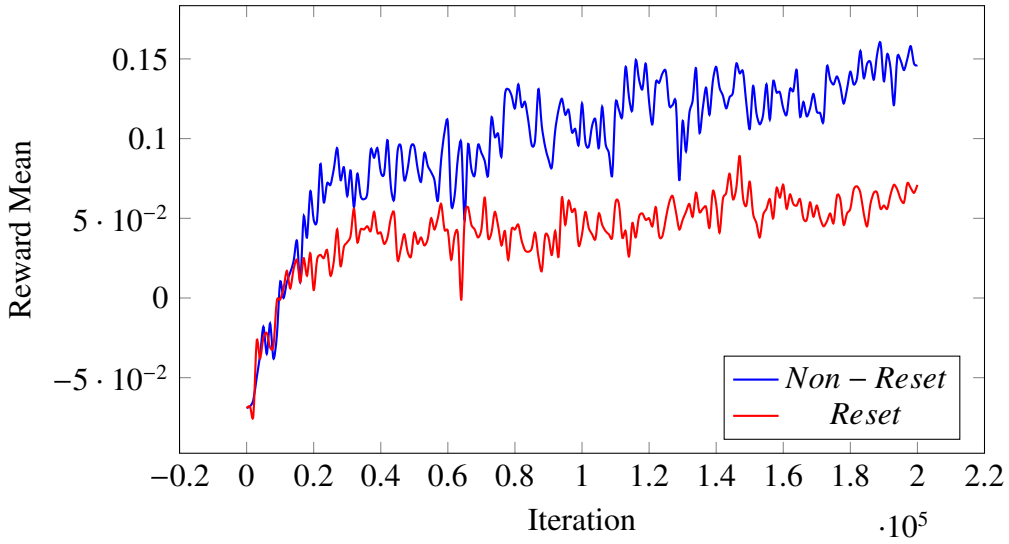
(b) The impact of layer size on reward measurements.

Figure B.2: The impact of layer size on learning performance. Non-reset, non-pooled segments configuration. Lambda = 0.6, learning rate = 0.5 and discount rate = 0.95.

B.2.2 Results for Reset and Segment Eligibility Trace Features

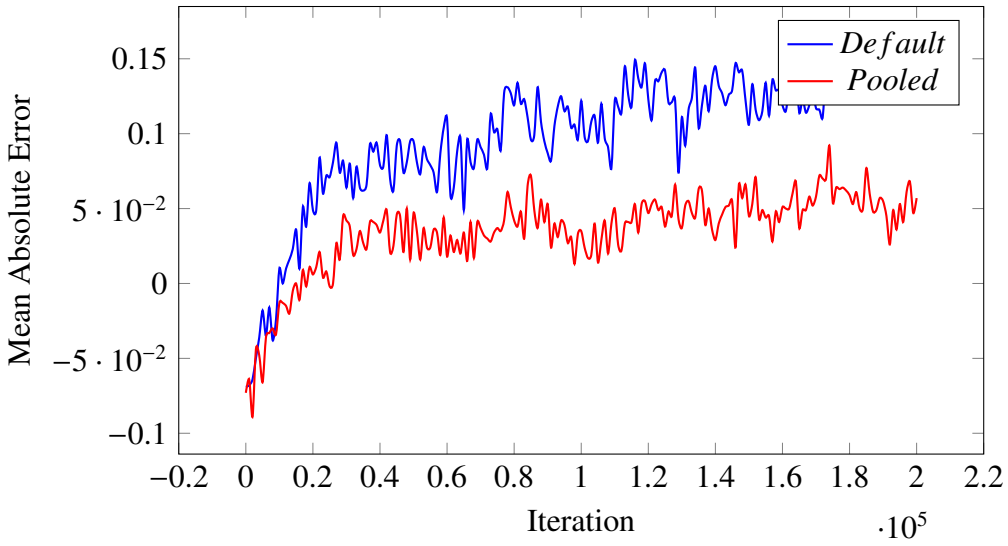


(a) Reset mechanism impact on MAE.

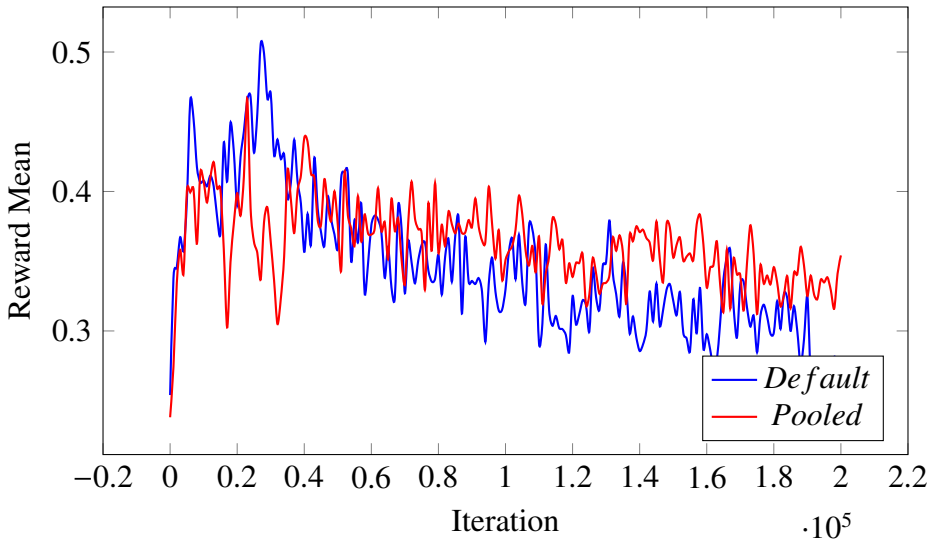


(b) Reset mechanism impact on reward averages.

Figure B.3: Reset mechanism comparison for non-pooled segments configuration. Lambda = 0.6, learning rate = 0.5, discount rate = 0.95 and 800x8 layer sizes.



(a) The impact of segment eligibility traces feature on MAE.



(b) The impact of segment eligibility traces feature on reward average.

Figure B.4: Pooled segment adaptation mechanism comparison for non-reset configuration. Lambda = 0.6, learning rate = 0.5, discount rate = 0.95 and 800x8 layer sizes.