Investigating the Efficacy of the Cascade Model of Synaptic Plasticity in a Biologically Constrained Simulation

Completed for the Certificate in Scientific Computation
Fall 2023

Ellie Kim
Bachelor of Science
Department of Neuroscience
College of Natural Sciences

Michael Mauk
Professor
Department of Neuroscience

1 Abstract

Learning and memory retention in the brain involve systematic modifications in synaptic strengths. Numerous algorithms have been proposed to mathematically represent the mechanisms underpinning synaptic plasticity. The challenge lies in constructing models that capture the brain's adaptability and rapid learning while maintaining resistance to avoid erasure of preexisting memories. The cascade model of plasticity tackles this challenge by utilizing binary synaptic strength values while incorporating diverse resistance levels for each synapse. In this paper, we leverage a biologically constrained computer simulation of the cerebellum to investigate whether the cascade plasticity algorithm emerges as the optimal model, outperforming others, in replicating the observed learning performance in our brain. Through associative learning trials in our experimental design, we assess different aspects of learning where the cascade model demonstrates notable performance. Based on these findings, we delineate the strengths and weaknesses of the cascade plasticity model compared to alternatives, proposing avenues for further research.

2 Introduction

Synaptic Plasticity in Learning and Memory

The question of how memory is encoded within the intricate network of neurons in the brain has intrigued neuroscientists for decades. Both experimental and theoretical studies have contributed to the framework, suggesting that alterations in synaptic strengths during the learning process form the foundation for information storage in the brain (Wang, 1997). This phenomenon, known as synaptic plasticity, enables the dynamic modification of synaptic strengths between neurons, allowing the brain to learn in response to experiences. As the brain processes new information, certain subsets of neurons undergo plasticity (i.e. strengthening or weakening) events and modify their synaptic strengths accordingly. A lasting alteration in these strengths signifies that the memory is encoded, and the recall of that learned memory depends on the detectability of these changes (Martin, 2000). Consequently, the study of synaptic plasticity emerges as a crucial area of research when exploring the processes underlying learning and memory retention. Notably, the precise algorithms governing synaptic strength adjustments pose intriguing theoretical questions. Various models of synaptic plasticity mechanisms have been proposed to elucidate the biological processes underpinning learning and the robust memory retention evident in human brains.

Different Models of Synaptic Plasticity

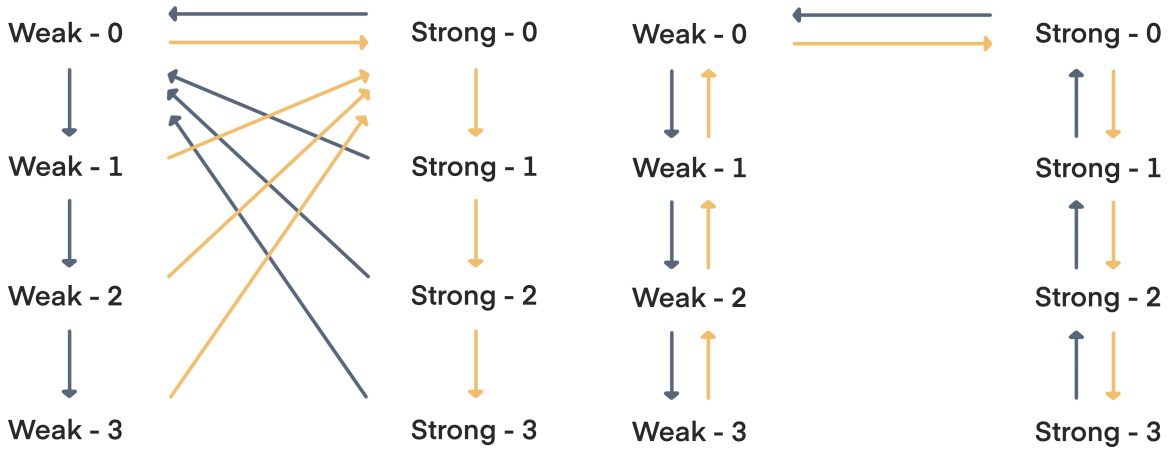
Before delving into the different models of synaptic plasticity, it is important to mention that these algorithms are often studied in the context of mathematical and computational representations of neural circuits. These models are constructed with the aim of better capturing and understanding what occurs in biological settings. In these computational representations, the strength of synapses is often referred to as "weights," represented by values between 0 and 1. A decimal value closer to one indicates a strong synaptic connection, whereas a value closer to zero signifies a weaker connection. Millions of synaptic weight values can be stored, modified, and studied to understand how the brain learns. Testing the different models of synaptic plasticity are crucial, as they govern the algorithms used to modify these weight values.

One of the widely used models of synaptic plasticity is what we will refer to as "graded plasticity." In this model, the possible values a synapse can take form a continuous spectrum of decimal values between zero and one. During a strengthening event (i.e., long-term potentiation (LTP)), a synapse increases its value by a small increment, while during a weakening event (i.e., long-term depression (LTD)), it decreases its value by a small increment. This algorithm results in a slow, gradual modification pattern of synaptic weights, allowing synapses to assume a wide range of values between zero and one. As we will demonstrate later in this paper, this model allows great memory retention but is slower at encoding memory due to the small and gradual unit of modification.

Another proposed model of synaptic plasticity is what we refer to as "binary plasticity." Unlike graded plasticity, synapses in the binary plasticity model can only take one of two predetermined weight values – either a strong value (e.g., 0.7) or a weak value (e.g., 0.3) (O'Connor, 2005). During a strengthening plasticity event, weak weight values are flipped to strong values, and the reverse occurs during a weakening plasticity event. One advantage of this model is that modification occurs quickly and reliably. However, synapses under this model are more susceptible to the erasure of previously learned memories. This susceptibility arises because learning that has occurred due to strengthening or weakening events are more likely to be reset when subject to random noise in brain activity.

Cascade Model of Synaptic Plasticity

The challenges in developing a model of synaptic plasticity are intricately tied to the complexities of the actual biological network in the brain. The chosen algorithm must strike a delicate balance, supporting robust learning by dynamically adapting to new environments, while also enabling long-term memory retention through resilience to environmental noise events (Fusi, 2005). The gradual nature of the graded plasticity model excels in memory retention but falls short in effective learning.



(a) Cascade Plasticity

(b) Modified Cascade Plasticity

Figure 1: **Organization of weight states in cascade plasticity models**. Synaptic weights are associated with either "Weak" or "Strong" strengths and have varying degrees of resistance, which correspond to different states as illustrated above. The arrows indicate potential transition pathways between these weight states. Yellow arrows represent possible transitions during strengthening events, while blue arrows are for weakening events.

Conversely, the volatile nature of the binary plasticity model struggles to preserve encoded memories. In response to these challenges, a novel synaptic plasticity model, known as the "cascade model of synaptic plasticity," was introduced in 2005 by Fusi and Abbott (Fusi, 2005). This algorithm aimed to synergize the strengths of both graded and binary models, presenting a promising solution to the current challenge.

Cascade plasticity incorporates certain features of binary plasticity, notably the constraint of synapses to two predetermined weight values – strong or weak. Nevertheless, the model introduces additional mechanisms to compensate for the limitations of binary plasticity, specifically addressing the ease with which weight values can be reset after the initial learning event. In the cascade model, resistance levels are assigned to each synapse, alongside their actual synaptic weight values. Consequently, weights remain restricted to adopting either strong or weak values, but there is increased diversity in the extent to which they are susceptible to be changed in strength. For instance, between two synapses with the exact same strength, the one that has undergone more strengthening events to reach that strong state is assigned higher resistance levels, making it less likely to be flipped to the weak state when subjected to weakening plasticity events.

This mechanism is implemented by defining eight distinct "weight states" that a particular synapse can assume, as illustrated in **Figure 1(a)**. Synapses in weight states "Weak-0" through

"Weak-3" all possess the same weak weight value (e.g., 0.3), but they exhibit varying resistance levels, with "Weak-3" being the most resistant and “Weak-0” being the least. Similarly, "Strong-0" to "Strong-3" all feature strong weight values (e.g., 0.7), but "Strong-3" has the highest resistance while "Strong-0" has the lowest. This degree of resistance is incorporated into the algorithm by the predetermined transition probabilities within these weight states. For example, the probability of a synapse in the "Weak-0" state transitioning to the "Strong-0" state during a strengthening event is higher compared to one in the "Weak-3" state transitioning to the same "Strong-0" state. This ensures that synapses in the “Weak-3” state have higher resistance to change than “Weak-0” synapses, as they had to undergo more strengthening events to be placed in the “Weak-3” state. This mechanism is facilitated by the transition pathways of synapses between states with the same weight values (i.e. “Weak” states and “Strong” states). A synapse subjected to multiple events of the same kind, like repetitive strengthening events or weakening events, is placed in a state with lower transition probabilities. **Figure 1(a)** provides a visual representation of the organization of different weight states and the associated transition probabilities between them.

Furthermore, adding on to the original cascade model of synaptic plasticity proposed by Fusi and Abbott, we introduce a variant of the model with a slightly different organization of transition pathways between weight states. In response to the counterintuitive nature that a single strengthening or weakening event may flip a synapse in the deepest levels (i.e. "Weight-3" or "Strong-3") to the opposite weight value, this modified version introduces added complexity by eliminating direct pathways stemming from these states. Instead, when a synapse in the "Weight-3" state undergoes a strengthening event, it transitions to "Weight-2." Illustrated in **Figure 1(b)**, this model will be referred to as "modified cascade plasticity” and will be assessed for its performance alongside the original cascade model.

General Approach

Despite the wealth of theoretical support for the remarkable capabilities of the cascade model of synaptic plasticity, this algorithm has yet to undergo testing within a biologically constrained simulation of a brain network. This raises a compelling question: If implemented in a simulation that accurately represents the composition and structure of the brain, would this model still demonstrate improved learning and memory retention abilities? Answering this question holds the potential to refine the utilization of theoretical neuroscience in elucidating biological mechanisms. This paper aims to address this inquiry by systematically evaluating the performance of the two cascade plasticity models within a brain simulation, comparing them with graded and binary models of synaptic plasticity. Our hypothesis posits that its learning capability would be comparable to graded plasticity, while significantly surpassing binary plasticity. Moreover, we predict it will exhibit

superior long-term memory retention compared to either graded or binary plasticity. The underlying reasons for the advantages and disadvantages of each of these models will be uncovered through a close examination of synaptic weight behavior during the learning processes.

3 Materials and Methods

Computer Simulation

The environment used for testing the various models of synaptic plasticity was a computer simulation network of the cerebellum, developed by the Mauk Lab in the Department of Neuroscience at The University of Texas at Austin. Constructed over decades of rigorous studies, this simulation is proven to faithfully represent the physiology and anatomy of the cerebellum through computational modeling (Medina, 2000). The accuracy of these representations was continually refined through systematic comparisons of simulated neural activities with in vivo recordings of rabbit neurons. The resulting model effectively mimics the real-life activity of cerebellar neurons in rabbits throughout the learning process.

Implementation of Plasticity Algorithms

The simulation was initially developed using the graded model of plasticity, focusing on the synaptic junctions between the Granule cell layers and Purkinje cells (Medina, 2000). Granule cells process auditory stimuli, while Purkinje cells integrate information from Granule cells and Climbing Fibers (which process somatosensory input) to identify the association between the two types of stimuli. Consequently, synaptic plasticity at the junction between Granule and Purkinje cells governs the associative learning process within the cerebellum (Moore, 1997). This specific junction, with Granule cells as presynaptic neurons and Purkinje cells as postsynaptic neurons, was chosen as the focal point for implementing different plasticity models to assess their impact on learning. Translating theoretical algorithms into functional code was necessary to integrate these models seamlessly into the overall simulation.

(1) Graded Plasticity

The code presented in the **Appendix** encompasses plasticity functions written in the CUDA C++ language. These functions are invoked in the subsequent sections of the simulation codes to update synaptic weights based on graded, binary, cascade, and modified cascade plasticity algorithms, respectively. Take, for instance, the "updatePFPCGradedSynWeightKernel" function, which calculates synaptic weights at each time step using the graded plasticity algorithm. Its inputs include an

array of previous weight values, the history of Granule cell firing activity, and predefined parameters, such as the incremental steps of weight values. With these inputs, the function initially assesses if a synaptic strengthening or weakening event has occurred by examining whether Granule cell firings occurred within the plasticity time window. Subsequently, it determines how the value will be adjusted in the next time step. If a strengthening event has taken place, the increment is added to the synaptic weights. Conversely, if a weakening event has occurred, the increment is subtracted from the synaptic weights. The result of this function is an updated array of weight values for the synaptic junction between Granule and Purkinje cells.

(2) Binary Plasticity

The function "updatePFPCBinarySynWeightKernel" performs a similar role but utilizes the binary plasticity algorithm. Its inputs resemble those of its graded counterpart, with additional parameters specifying the high/low values that weights can assume, as well as the weight transition probability. This function computes the synaptic weight values for the next time step using the binary algorithm. In the event of a strengthening event, the weight is flipped to the stronger value, but only under a specific transition probability. Conversely, during a weakening event, the weight is flipped to the weaker value. The result is once again an updated array of weight values. Due to the binary nature of this plasticity, both the input and output arrays exclusively contain weight values equal to either the high or the low parameter.

(3) Cascade Plasticity

Other functions, namely "updatePFPCAbbottCascadeLTDPlastKernel" and “updatePFPCAbbottCascadeLTPPlastKernel” are responsible for updating synaptic weights based on the cascade plasticity algorithm. Similar to aforementioned functions, they take inputs such as the array of previous weight values. However, they add a layer of complexity by incorporating an array of previous weight states, since the cascade algorithm considers not only weight values but also their states, signifying the resistance level of each synapse. The computation is initiated by scrutinizing the history of Granule cell firing activities and determining whether a synaptic strengthening or weakening event has transpired. “updatePFPCAbbottCascadeLTPPlastKernel” is used in case of strengthening events, while “updatePFPCAbbottCascadeLTDPlastKernel” is used in case of weakening events. Afterwards, the switch-case statements in each function classify the synapses based on the cascade states they occupied in the previous time step (refer to **Figure 1(a)** for an overview of the different weight states in cascade plasticity). Synapses in each weight state then undergo transitions into different states dictated by their respective transition probabilities/pathways set by the cascade plasticity algorithm.

After this intricate computation, the resulting output comprises an array containing updated weight values and another array containing updated weight states for the subsequent time step.

(4) Modified Cascade Plasticity

Finally, the modified cascade plasticity algorithm is incorporated into the simulation through the functions "updatePFPCMaukCascadeLTDPlastKernel" and "updatePFPCMaukCascadeLTPPlastKernel." The inputs, outputs, and organizational structure of the code remain consistent with the functions for the original cascade plasticity, as detailed in the preceding paragraphs. However, there is a subtle variation in the design of transition pathways between synaptic weight states, as illustrated in **Figure 1(b)**.

Following the integration of the synaptic plasticity models into the computer simulation, an exhaustive tuning process was conducted to select optimal parameters for each function. This involved defining the value for incremental steps in graded plasticity, setting the high and low weight values in binary plasticity, and establishing transition probabilities for binary, cascade, and modified cascade plasticity. A range of values for each parameter was systematically tested, and the most effective ones were selected based on the learning performance metrics, which will be detailed later.

Associative Learning

The synaptic plasticity occurring in the junction between Granule and Purkinje cells plays a critical role in integrating auditory and somatosensory inputs, essential for the brain's ability to learn the association between them. To experimentally investigate this learning process, we employ the associative learning model consisting of two stimuli known as the "unconditioned stimulus (US)" and "conditioned stimulus (CS)." The unconditioned stimulus is an input that automatically elicits a behavioral response, such as the somatosensory stimulus of an eye puff that causes the animal to blink (i.e. unconditioned response). In contrast, the conditioned stimulus is a neutral input not inherently linked to the behavioral response, like an auditory tone that would not normally cause the animal to blink. Through repeated presentation of the US and CS, the synaptic plasticity governing learning gradually associates the auditory tone (CS) with the blinking behavior. Eventually, the tone alone can trigger the eyelid blink (i.e. conditioned response), indicating learned behavior. In experimental settings, neural recordings from the rabbit cerebellum provide data on the inner workings of neurons during this learning process. Computer simulation allows us to generate similar neural activity data, crucial for studying synaptic plasticity in learning (Moore, 1997).

Experimental Design

For the purpose of this study, neural activity data from Purkinje cells were gathered in the simulation across repetitive trials involving the coordinated presentation of the unconditioned stimulus (US) and conditioned stimulus (CS). Each trial in the computer simulation represented the somatosensory (US) and auditory (CS) stimuli through the firing of Climbing Fibers and Mossy Fibers, mirroring the physiological processes in the cerebellum. Employing these computerized representations of associative learning, an experimental design was crafted to assess the learning abilities of each synaptic plasticity model.

First, 540 trials were conducted where both the puff (US) and auditory stimulus (CS) were presented concurrently to evaluate the neural network's capacity to acquire the association. Subsequently, 216 trials were executed with only the CS presented, excluding the US, to examine how well the association between the CS and the behavioral response is extinguished. Additionally, to account for the potential variations in learning performance influenced by different ways of presenting the stimulus, we manipulated the temporal interval between US and CS presentation, known as the interstimulus interval (ISI). The chosen intervals were 250 ms, 500 ms, 750 ms, and 1000 ms. In summary, spike train data were collected from 32 Purkinje cells in the simulation, encompassing experiments with four distinct models of synaptic plasticity across a sequence of acquisition and extinction trials, each with varying ISIs.

Performance Metrics in Associative Learning

The neural spike data collected from the aforementioned experimental design underwent thorough analysis and evaluation to address the central question: Does the cascade model of plasticity demonstrate enhanced learning and memory retention abilities in comparison to existing models? In tackling this inquiry, it became crucial to define metrics that would gauge the efficacy of learning based on the gathered neural activity data from Purkinje cells. Drawing upon insights from previous studies regarding expected Purkinje cell behavior in the presence of conditioned responses (CR), we developed two quantitative criteria. The results from these methods serve as indicators of the extent to which the brain simulation has successfully acquired and retained the association.

(1) Amplitude of Conditioned Responses

The conditioned response (CR), which represents the behavioral reaction triggered exclusively by the conditioned stimulus (CS), is characterized by the inhibition of Purkinje cell firings when the auditory stimuli (i.e., the CS) are presented. Therefore, the assessment of learning using the neural spike data involves examining the degree to which Purkinje cells show decreased firing rates upon the presentation of the tone. In each trial, we initially computed the baseline firing rate by averaging

Purkinje cell spikes across the first 400 ms of the trial. Then, we evaluated the difference between the established baseline and the minimum spike count reached by the Purkinje cell during the trial. A greater disparity between these values indicated a higher amplitude of the CR, signifying a more robust learning of the association in the simulation.

(2) Constancy of Asymptotic Performance

Another critical aspect of learning is the simulation's ability to sustain the conditioned response (CR) across multiple trials once it has initially acquired the association. This is referred to as the constancy of asymptotic performance. To measure this, the amplitude of the CR is first identified for each trial. Across the entire set of acquisition/extinction trials, the highest amplitude is pinpointed and serves as the benchmark for asymptotic performance. Then, the trial where initial acquisition happens is determined by identifying the first 10 trials in which the average CR falls within considerable boundaries of the "asymptote." Finally, among the subsequent trials following the initial acquisition, the proportion whose CR amplitude is within the bounds of the asymptote serves as an indicator of how consistently the simulation can maintain its learned response.

Data Analysis Pipeline

The neural activity output from the simulation is in a spike train format. This type of data has a sequence of zeros and ones where the ones denote a time point when the neuron has fired. The initial step of data analysis involves consolidating the spike dataset from 32 individual Purkinje cells by computing the number of neurons that fired at each time point. This unified dataset represents the firing rate of the Purkinje cell layer across time steps and trials. We accomplished this through a Python data analysis script, the details of which are provided in the attached **Appendix**.

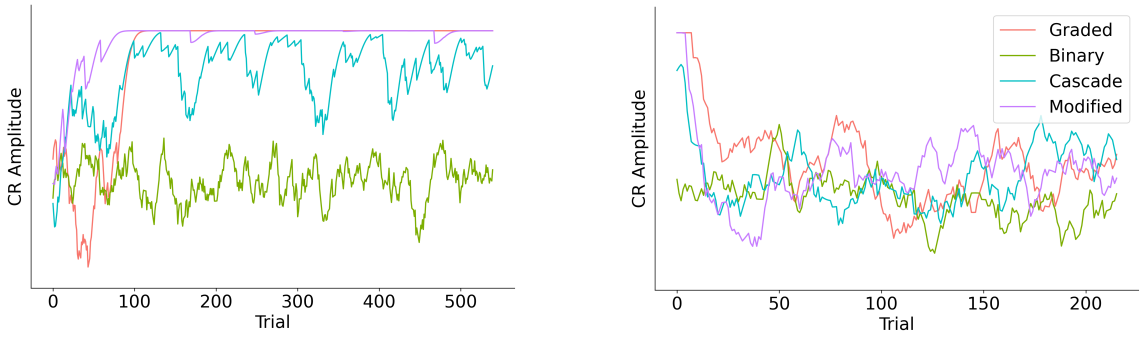
Through the integration of these datasets, we were well-prepared to construct a peristimulus time histogram of Purkinje cell activity. This widely employed plotting format effectively illustrates the firing rates of neurons during a single trial, offering a visual representation of how Purkinje cell firing patterns change over time. To address the substantial variability in spike activities at each time step (in milliseconds), a half-Gaussian smoothing filter was applied both when plotting the peristimulus time histogram and before conducting subsequent analyses. The final processed datasets were then evaluated using coding scripts (see **Appendix**) that apply the aforementioned performance metrics.

4 Results

Learning Performance

Examining the amplitude of the conditioned response across acquisition and extinction trials provides insights into how effectively each synaptic plasticity model supports associative learning. As depicted in **Figure 2(a)**, all plasticity algorithms, with the exception of the binary model, exhibit a gradual increase in the conditioned response after repetitive acquisition trials. This suggests that graded plasticity, cascade plasticity, and modified cascade plasticity are effective synaptic weight modification algorithms conducive to expected learning patterns. In contrast, binary plasticity proved to be ineffective in supporting learning. Furthermore, among the three plasticity algorithms that displayed successful acquisition, the number of trials required for learning and the consistency in exhibiting learned behavioral responses (i.e., CR) varied. These aspects are further explored through the measurement of asymptotic constancy, as illustrated later.

The extinction of the learned response when the association of stimuli is no longer present is another crucial aspect of learning that synaptic plasticity models should support. **Figure 2(b)** illustrates the capacity of each plasticity algorithm to extinguish the conditioned response (CR) when subjected to respective extinction trials. Extinction trials refer to experiments where the tone is not followed by the air puff and is thus no longer associated with the eye blink response. Once again, the binary plasticity model displays insensitivity to learning. In contrast, the remaining synaptic plasticity models successfully extinguish the conditioned responses after undergoing a certain number of extinction trials. The speed of extinction and the ability to maintain the extinguished



(a) Acquisition Trials

(b) Extinction Trials

Figure 2: **Change in conditioned responses throughout the learning process**. The plots depict the trend in conditioned response (CR) amplitude for four distinct synaptic plasticity models. **(a)** Across 540 acquisition trials, the simulation with binary plasticity failed to demonstrate learning, while the other three algorithms facilitated significant CR acquisition. **(b)** Throughout 216 extinction trials, graded, cascade, and modified cascade plasticity algorithms effectively extinguished their learned responses.

neural responses also varied to some extent.

The systematic investigation of algorithm performance in both acquisition and extinction trials was conducted across a range of different interstimulus intervals (ISI). The variation in results became apparent as the ISI was increased. In relatively short ISIs of 250 ms, graded, cascade, and modified cascade plasticity algorithms all exhibited very robust and comparable conditioned responses. However, as the ISI increased to 500 ms, 750 ms, and then 1000 ms, the performance of all three models deteriorated, with the cascade model showing the most notable decrease. Modified cascade plasticity supported better learning in longer interstimulus intervals compared to the original cascade plasticity. Graded plasticity demonstrated the best learning across all interstimulus intervals, while binary plasticity showed the least favorable outcomes.

In addition to comparing CR amplitudes, the asymptotic constancy of each plasticity model across diverse ISI was examined. When the binary plasticity algorithm was applied to the synapses in the simulation, it resulted in poor learning performance, evident not only in CR amplitudes but also in asymptotic constancy (see **Figure 3**). The two cascade models demonstrated comparable CR amplitudes and asymptotic constancy to the graded plasticity algorithm at lower ISIs but exhibited a decline in performance as the interstimulus interval increased. As depicted in **Figure 3**, the

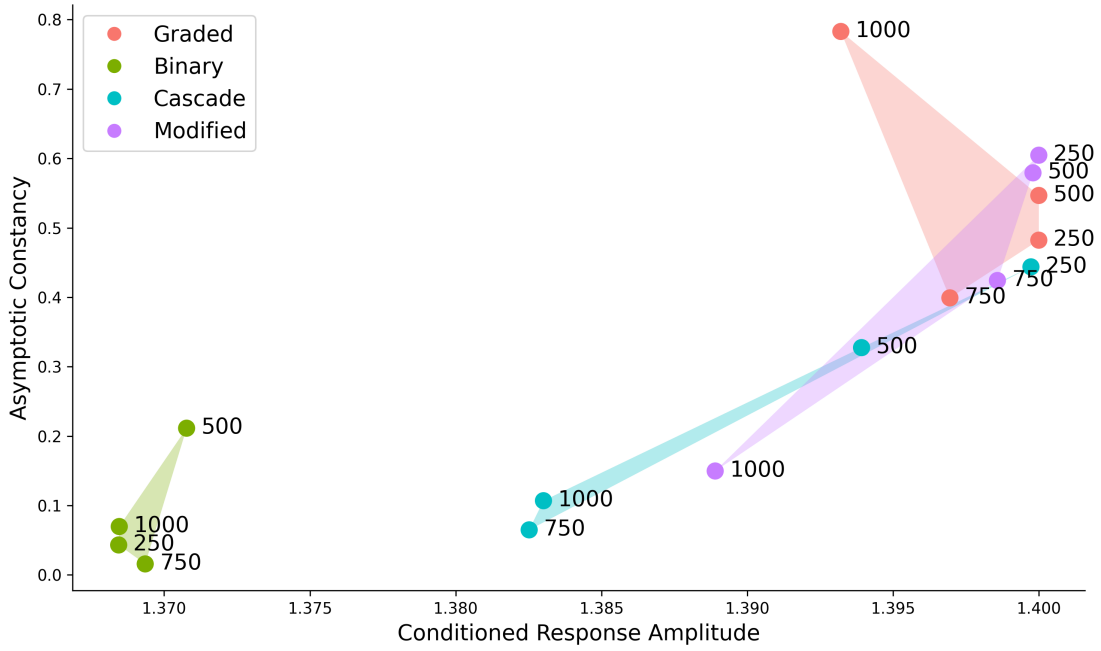


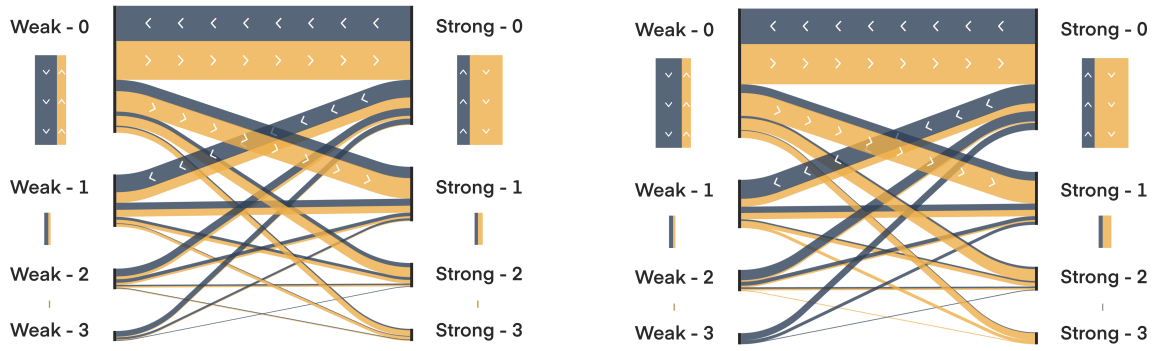
Figure 3: **Comparative analysis of learning performance across plasticity algorithms**. The figure illustrates the assessment of learning performance in the simulation based on conditioned response (CR) amplitude and asymptotic constancy. Point colors indicate the type of algorithm used for modifying synaptic weights during the learning process. Text labels specify the interstimulus interval between the two stimuli for which the simulation was intended to learn the association.

modified cascade algorithm exhibited superior learning performance compared to the original cascade algorithm in both CR amplitude and asymptotic constancy. This improvement could be attributed to the fact that the transition pathway organization in the modified algorithm further diversified the strengths of resistance among synapses. In conclusion, while the cascade model of synaptic plasticity demonstrated significantly better learning performance compared to binary plasticity and showed similar capability to graded plasticity at lower ISIs, limitations still existed in longer ISIs. Further studies can explore whether this deteriorated performance in longer ISIs provides a more biologically accurate representation of neural responses.

Weight Behavior Analysis

For a more in-depth analysis of the underlying processes of cascade plasticity, we investigated the changes in each synaptic weight throughout the trials, as depicted in **Figure 4**. The substantial number of synapses oscillating between "Weak-0" and "Strong-0" states aligns with expectations, given the higher transition probabilities in these levels. In contrast, fewer synapses transitioned to and from subsequent levels of synaptic weight states (**Figure 4**). The varied transition probability, indicating differing levels of resistance, serves as the compensating mechanism unique to cascade plasticity, distinguishing it from binary plasticity. Associating this outcome with the learning performance analysis, it is apparent that introducing these resistance levels to the plasticity algorithm enables the simulation to produce learned responses.

Moreover, a comparison of the weight transition diagrams between cascade and modified cascade plasticity models revealed intriguing insights. The two cascade algorithms exhibit different structures of transition pathways between synaptic states, as illustrated in **Figure 1**. Consequently, a distinct overall pattern of synaptic weight behavior during learning emerged. In contrast to the original algorithm, modified cascade plasticity displayed fewer synapses transitioning from "Weak-1," "Weak-2," "Weak-3" to "Strong-0" states during strengthening events (i.e., yellow arrows). Similarly, there were fewer synapses transitioning from "Strong-1," "Strong-2," "Strong-3" to "Weak-0" during weakening events. This is because the synapses in deeper levels have increased resistance to be flipped to the opposite weight value (e.g., "Weak" to "Strong" or "Strong" to "Weak") in modified cascade plasticity. This distinction potentially explains why modified cascade plasticity outperformed the original cascade model in learning (**Figure 3**). Specifically, the enhanced retention of learned behavioral response, as measured by asymptotic constancy, is likely due to the increased resistance against change.



(a) Cascade Plasticity

(b) Modified Cascade Plasticity

Figure 4: **Overview of synaptic weight transitions during learning with the cascade plasticity model**. The weight states of every synapse between Purkinje cells and the Granule cell layer were collected before and after the execution of 540 acquisition trials. The thickness of arrows between each weight state visualizes the number of synapses transitioning from one state to another throughout the associative learning process

5 Discussion

The pursuit of the optimal theoretical algorithm simulating synaptic weight modification in the brain is a central focus in computational neuroscience. Widely used models feature either a graded range of synaptic weight values or a binary system. Recent proposals introduce models with varied resistance levels coupled with weight values, demonstrating an advantage in capturing the biological intricacies of synapses (Fusi, 2005). Recognizing the pivotal role of synaptic plasticity in learning, this paper evaluates the capability of the newly proposed cascade model of plasticity to support learning and memory retention in a biologically constrained simulation. Through a comparative analysis of learning performance among different plasticity algorithms, we discerned the strengths and limitations of each model, aiming to contribute to the identification of the optimal model that faithfully replicates the brain's synaptic plasticity mechanisms.

In this project, we employed computational techniques to implement various proposed plasticity algorithms into the simulation of cerebellar neural circuits. Subsequently, we assessed the associative learning abilities of the simulated cerebellum, collecting neural firing activity data for further analysis. The results underscored the limited efficacy of binary plasticity models in both acquiring and retaining associations. As hypothesized, cascade models exhibited robust learning abilities comparable to the graded model, with enhanced performance observed when modifying the synaptic state organization. An analysis of synaptic weight transition patterns provided insight into why the modified cascade algorithm better supported learning. Nevertheless, a comprehensive evaluation of response amplitude and asymptotic constancy revealed that the cascade models' performance

was yet to surpass that of the graded plasticity algorithm, especially in associating stimuli when presented with long temporal intervals.

Expanding upon these findings, there remain unexplored avenues for investigation, including the algorithm's potential in resisting the forgetting process in the brain. This represents a promising area where cascade plasticity may demonstrate a unique advantage over other algorithms. Therefore, future studies with comprehensive assessments of long term memory retention could prove valuable. Additionally, there is room for a more intricate examination of synapses undergoing weight changes during the learning process. A comparative evaluation of the number of synapses contributing to the conditioned response for each plasticity algorithm could offer insights into the efficiency of learning within the simulation.

The diverse mechanisms by which synapses in the brain alter their strength, such as morphological modifications or changes in the number of ion channel receptors, offer compelling support for the cascade model of plasticity (Bliss, 1993). Despite its limitations, the cascade model presents a unique and valuable approach to modeling synaptic plasticity. Further investigation and comparison utilizing this model have the potential to unveil insights into the roles and underlying mechanisms that synaptic plasticity plays in various facets of learning.

6 Acknowledgements

I extend my sincere gratitude to Dr. Michael Mauk for his invaluable guidance and mentorship throughout this research. I'm also grateful to lab members Sean Gallogly, Satvik Mojnidar, and Raiyyan Siddiqui for their contributions and support. Their expertise significantly enriched this work.

7 Literature Cited

Bliss, T. V., & Collingridge, G. L. (1993). A synaptic model of memory: long-term potentiation in the hippocampus. Nature, 361(6407), 31-39.

Fusi, S., Drew, P. J., & Abbott, L. F. (2005). Cascade models of synaptically stored memories. Neuron, 45(4), 599-611.

Martin, S. J., Grimwood, P. D., & Morris, R. G. (2000). Synaptic plasticity and memory: an evaluation of the hypothesis. Annual review of neuroscience, 23(1), 649-711.

Medina, J. F., & Mauk, M. D. (2000). Computer simulation of cerebellar information processing.

nature neuroscience, 3(11), 1205-1211.

Medina, J. F., Nores, W. L., Ohyama, T., & Mauk, M. D. (2000). Mechanisms of cerebellar learning suggested by eyelid conditioning. Current opinion in neurobiology, 10(6), 717-724.

Moore, J. W., & Choi, J. S. (1997). Conditioned response timing and integration in the cerebellum. Learning & Memory, 4(1), 116-129.

O'Connor, D. H., Wittenberg, G. M., & Wang, S. S. H. (2005). Graded bidirectional synaptic plasticity is composed of switch-like unitary events. Proceedings of the National Academy of Sciences, 102(27), 9679-9684.

Wang, J. H., Ko, G. Y., & Kelly, P. T. (1997). Cellular and molecular bases of memory: synaptic and neuronal plasticity. Journal of clinical neurophysiology, 14(4), 264-293.

Appendix A

```
//**---------------IO kernels-----------------**

__global__ void updatePFPCGradedSynWeightKernel(float *synWPFPC, uint64_t
*historyGR, uint64_t plastCheckMask,
	unsigned int offset, float plastStep)
{
int i=blockIdx.x*blockDim.x+threadIdx.x+offset;
synWPFPC[i]=synWPFPC[i]+((historyGR[i]&plastCheckMask)>0)*plastStep;

synWPFPC[i]=(synWPFPC[i]>0)*synWPFPC[i];
synWPFPC[i]=(synWPFPC[i]>1)+(synWPFPC[i]<=1)*synWPFPC[i];
}

template <typename randState>
__global__ void updatePFPCBinarySynWeightKernel(float *synWPFPC, uint64_t
*historyGR, uint64_t plastCheckMask,
		unsigned int offset, float plastStep, float synWLow, float synWHigh,
float trans_prob, float *randoms)
{
	int i = blockIdx.x * blockDim.x + threadIdx.x + offset;
	if (randoms[i] < trans_prob)
	{
		synWPFPC[i] += ((historyGR[i] & plastCheckMask) > 0) * plastStep;

		synWPFPC[i] = (synWPFPC[i] > synWLow) * synWPFPC[i] + (synWPFPC[i]
<=synWLow) * synWLow;
		synWPFPC[i] = (synWPFPC[i] > synWHigh) * synWHigh + (synWPFPC[i] <=
synWHigh) * synWPFPC[i];
	}
}

template <typename randState>
__global__ void updatePFPCAbbottCascadeLTDPlastKernel(float *synWPFPC, uint8_t
*synStatesPFPC, uint64_t *historyGPU,
		uint64_t plastCheckMask, unsigned int offset, float synWLow, float
trans_prob_base, float *randoms)
{
	int i = blockIdx.x * blockDim.x + threadIdx.x + offset;
	if ((historyGPU[i] & plastCheckMask) > 0)
	{
		switch (synStatesPFPC[i])
		{
			case 1:
				if (randoms[i] < trans_prob_base / 8.0)
```

```
                        {s
                                synStatesPFPC[i] = 0;
                        }
                        break;
                case 2:
                        if (randoms[i] < trans_prob_base / 4.0)
                        {
                                synStatesPFPC[i] = 1;
                        }
                        break;
                case 3:
                        if (randoms[i] < trans_prob_base / 2.0)
                        {
                                synStatesPFPC[i] = 2;
                        }
                        break;
                case 4:
                        if (randoms[i] < trans_prob_base)
                        {
                                synStatesPFPC[i] = 3;
                                synWPFPC[i] = synWLow;
                        }
                        break;
                case 5:
                        if (randoms[i] < trans_prob_base / 2.0)
                        {
                                synStatesPFPC[i] = 3;
                                synWPFPC[i] = synWLow;
                        }
                        break;
                case 6:
                        if (randoms[i] < trans_prob_base / 4.0)
                        {
                                synStatesPFPC[i] = 3;
                                synWPFPC[i] = synWLow;
                        }
                        break;
                case 7:
                        if (randoms[i] < trans_prob_base / 8.0)
                        {
                                synStatesPFPC[i] = 3;
                                synWPFPC[i] = synWLow;
                        }
                        break;
        }
    }
}
```

```
template <typename randState>
__global__ void updatePFPCAbbottCascadeLTPPlastKernel(float *synWPFPC, uint8_t
*synStatesPFPC, uint64_t *historyGPU,
        uint64_t plastCheckMask, unsigned int offset, float synWHigh, float
trans_prob_base, float *randoms)
{
      int i = blockIdx.x * blockDim.x + threadIdx.x + offset;
      if ((historyGPU[i] & plastCheckMask) > 0)
      {
            switch (synStatesPFPC[i])
            {
                  case 0:
                        if (randoms[i] < trans_prob_base / 8.0)
                        {
                              synStatesPFPC[i] = 4;
                              synWPFPC[i] = synWHigh;
                        }
                        break;
                  case 1:
                        if (randoms[i] < trans_prob_base / 4.0)
                        {
                              synStatesPFPC[i] = 4;
                              synWPFPC[i] = synWHigh;
                        }
                        break;
                  case 2:
                        if (randoms[i] < trans_prob_base / 2.0)
                        {
                              synStatesPFPC[i] = 4;
                              synWPFPC[i] = synWHigh;
                        }
                        break;
                  case 3:
                        if (randoms[i] < trans_prob_base)
                        {
                              synStatesPFPC[i] = 4;
                              synWPFPC[i] = synWHigh;
                        }
                        break;
                  case 4:
                        if (randoms[i] < trans_prob_base / 2.0)
                        {
                              synStatesPFPC[i] = 5;
                        }
                        break;
                  case 5:
```

```
                        if (randoms[i] < trans_prob_base / 4.0)
                        {
                                synStatesPFPC[i] = 6;
                        }
                        break;
                case 6:
                        if (randoms[i] < trans_prob_base / 8.0)
                        {
                                synStatesPFPC[i] = 7;
                        }
                        break;
        }
    }
}

template <typename randState>
__global__ void updatePFPCMaukCascadeLTDPlastKernel(float *synWPFPC, uint8_t
*synStatesPFPC, uint64_t *historyGPU,
            uint64_t plastCheckMask, unsigned int offset, float synWLow, float
trans_prob_base, float *randoms)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x + offset;
    if ((historyGPU[i] & plastCheckMask) > 0)
    {
        switch (synStatesPFPC[i])
        {
            case 1:
                if (randoms[i] < trans_prob_base / 8.0)
                {
                    synStatesPFPC[i] = 0;
                }
                break;
            case 2:
                if (randoms[i] < trans_prob_base / 4.0)
                {
                    synStatesPFPC[i] = 1;
                }
                break;
            case 3:
                if (randoms[i] < trans_prob_base / 2.0)
                {
                    synStatesPFPC[i] = 2;
                }
                break;
            case 4:
                if (randoms[i] < trans_prob_base)
                {
```

```
                                synStatesPFPC[i] = 3;
                                synWPFPC[i] = synWLow;
                        }
                        break;
                case 5:
                        if (randoms[i] < trans_prob_base / 2.0)
                        {
                                synStatesPFPC[i] = 4;
                        }
                        break;
                case 6:
                        if (randoms[i] < trans_prob_base / 4.0)
                        {
                                synStatesPFPC[i] = 5;
                        }
                        break;
                case 7:
                        if (randoms[i] < trans_prob_base / 8.0)
                        {
                                synStatesPFPC[i] = 6;
                        }
                        break;
        }
    }
}

template <typename randState>
__global__ void updatePFPCMaukCascadeLTPPlastKernel(float *synWPFPC, uint8_t
*synStatesPFPC, uint64_t *historyGPU,
        uint64_t plastCheckMask, unsigned int offset, float synWHigh, float
trans_prob_base, float *randoms)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x + offset;
    if ((historyGPU[i] & plastCheckMask) > 0)
    {
        switch (synStatesPFPC[i])
        {
                case 0:
                        if (randoms[i] < trans_prob_base / 8.0)
                        {
                                synStatesPFPC[i] = 1;
                        }
                        break;
                case 1:
                        if (randoms[i] < trans_prob_base / 4.0)
                        {
                                synStatesPFPC[i] = 2;
```

```
                        }
                        break;
                case 2:
                        if (randoms[i] < trans_prob_base / 2.0)
                        {
                                synStatesPFPC[i] = 3;
                        }
                        break;
                case 3:
                        if (randoms[i] < trans_prob_base)
                        {
                                synStatesPFPC[i] = 4;
                                synWPFPC[i] = synWHigh;
                        }
                        break;
                case 4:
                        if (randoms[i] < trans_prob_base / 2.0)
                        {
                                synStatesPFPC[i] = 5;
                        }
                        break;
                case 5:
                        if (randoms[i] < trans_prob_base / 4.0)
                        {
                                synStatesPFPC[i] = 6;
                        }
                        break;
                case 6:
                        if (randoms[i] < trans_prob_base / 8.0)
                        {
                                synStatesPFPC[i] = 7;
                        }
                        break;
        }
    }
}

//**---------------end IO kernels-------------**
```

Appendix B

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from matplotlib.lines import Line2D
from matplotlib.patches import Polygon

plasts = ['AB', 'BN', 'GR', 'MK']
isis = [250, 500, 750, 1000]

# save raster data for all plasticity models and isis
raster = dict()

for isi in isis:
    for plast in plasts:
        file = np.fromfile(f'data/ISI{isi}_{plast}_PC_RASTER.bin', dtype=np.ubyte)
        file = file.reshape(956, isi+800, 32) # trial ts cell
        raster[f'{isi}_{plast}'] = file

# functions for half gaussian filter
def halfgaussian_kernel1d(sigma, radius):
    sigma2 = sigma * sigma
    x = np.arange(0, radius+1)
    phi_x = np.exp(-0.5 / sigma2 * x ** 2)
    phi_x = phi_x / phi_x.sum()
    return phi_x
def halfgaussian_filter1d(input, sigma, axis=-1, output=None,
                      mode="constant", cval=0.0, truncate=4.0):
    sd = float(sigma)
    lw = int(truncate * sd + 0.5)
    weights = halfgaussian_kernel1d(sigma, lw)
    origin = -lw // 2
    return scipy.ndimage.convolve1d(input, weights, axis, output, mode, cval,
origin)

# this function takes raster file from above and converts to smoothed psth file
def raster_to_psth(data):
    xyz = data.shape
    psthdata = np.zeros((xyz[0],xyz[1]))   # trial, time step
    for i in range(0,xyz[2]):   # loop through every cell
        for j in range(0,xyz[0]):   # loop through every trial
            for k in range(0,xyz[1]):   # loop through every time step
                psthdata[j, k] = psthdata[j, k] + data[j, k, i]
```

```
    psthdata_flt = np.zeros((xyz[0],xyz[1]))
    for i in range(0,xyz[0]):
        psthdata_flt[i] = halfgaussian_filter1d(psthdata[i], sigma=50)
    return psthdata_flt

# save psth data for all plasticity models and isis
psth = dict()

for key, rasterdata in raster.items():
    psthdata = raster_to_psth(rasterdata)
    psth[key] = psthdata

# calculate and store CR value for each trial
CRs = dict()

for key, data in psth.items():
    xy = data.shape   # x is trial; y is ts
    diffs = []   # list length of num trials
    for trial in range(xy[0]):
        diff = 1.4 - data[trial, :]
        diffs.append(np.max(diff))   # save max diff from all ts in trial
    CRs[key] = diffs

# apply half gaussian smoothing to CR values for plotting
CRs_flt = dict()
for key, array in CRs.items():
    CRs_flt[key] = halfgaussian_filter1d(array, sigma=10)

# find mean CR for last 100 trials
CRs_mean = dict()
for key, array in CRs_flt.items():
    CRs_mean[key] = np.mean(CRs_flt[key][640:740])

# calculate peak CR value for all plasticity models and isis
CRs_peak = dict()
for key, array in CRs_flt.items():
    CRs_peak[key] = max(array[200:740])

# evaluate asymptotic constancy
asym_const = dict()

for key, array in CRs_flt.items():
    val = CRs_peak[key]

    # find trial where initial acq happens
    for i in range(540):
        if np.mean(array[200 + i : 210 + i]) > val - 0.005:
```

```
            first = i
            break

    # count num of trials whose CR is within peak boundaries
    # start counting after initial acq trial
    total = 540 - first
    counter = 0
    for j in range(first, 540):
        if array[j] > val - 0.005:
            counter += 1
    percent = counter / total
    asym_const[key] = percent

# process synaptic weight data
weights = dict()
trials = [199, 739, 955]   # background, acquisition, extinction

# graded and binary plasticity
for isi in isis:
    for plast in ['GR', 'BN']:
        for trial in trials:
            file =
np.fromfile(f'weights/ISI{isi}_{plast}_PFPC_WEIGHTS_TRIAL_{trial}.bin',
dtype=np.float32)
            weights[f'{isi}_{plast}_{trial}'] = list(file)

# cascade and modified cascade plasticity
for isi in isis:
    for plast in ['AB', 'MK']:
        for trial in trials:
            file =
np.fromfile(f'weights/ISI{isi}_{plast}_PFPC_WEIGHTS_TRIAL_{trial}.bin',
dtype=np.ubyte)
            weights[f'{isi}_{plast}_{trial}'] = list(file)
```