A Neurobiologically-inspired Deep Learning Framework for Autonomous Context Learning

By

David Ludwig

A thesis submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

Middle Tennessee State University

December 2020

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to my advisor and mentor, Dr. Joshua L. Phillips, for his guidance and assistance throughout my graduate school career. This work would not have been realized without his knowledge and persistent support. I would also like to give thanks to Dr. Sal Barbosa and Dr. Cen Li for serving on my committee and for taking the time to review this work. Last but not least, I would like to acknowledge and thank all of the faculty in the Computer Science department for their support and inspiration.

ABSTRACT

Neurobiologically-inspired working memory models have managed to accurately demonstrate and explain our ability to rapidly adapt and alter our responses to the environment. However, the applications of these working memory models have been limited to reinforcement learning problems. Furthermore, the incorporation of contextual/switching mechanisms outside of the realm of working memory modeling for general-use cases has also been relatively unexplored. We present a new framework compatible with Tensorflow Keras enabling the straightforward integration of working memory-inspired mechanisms into typical neural network architectures. These mechanisms allow models to autonomously learn multiple tasks, statically or dynamically allocated. We also examine the generalization of the framework across a variety of multi-context supervised learning and reinforcement learning tasks. The resulting experiments successfully integrate these mechanisms with multilayer and convolutional neural network architectures. The diversity of problems solved demonstrates the framework's generalizability across a variety of architectures and tasks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I.
INTRODUCTION

Imagine you enter a dark room for the first time that has multiple light switches. You flip one of the switches and notice that the lights don't turn on. Because of your lack of experience with the switches in this room, you would likely assume that the switch you flipped was incorrect and you would try the next. Sure enough, the lights turn on. Now imagine that you revisit this room some days later. You flip the switch and the lights do not turn on. Were you wrong about the switch? The lights came on every other time you interacted with the switch previously. Rather than ”unlearning” the fact that the switch turns the lights on, your experience tells you that you were not wrong about the switch, but something in the environment has changed.

The previous example is a simple demonstration of how working memory can be used to influence our actions and behavior around certain tasks. Working memory is a form of short-term memory that is used to influence our decision-making process. Instead of requiring the complete unlearning/relearning of a task, working memory facilitates the ability to rapidly and dynamically alter our responses to changes in the current situation. While there are many models that can accurately demonstrate and explain the mechanisms involved with working memory [4, 8, 9, 11, 12, 13, 14, 15, 16, 18], they are not currently utilized much outside of the realm of working memory research. While mechanisms for including contextual information in traditional neural networks has been explored before [5], current solutions often require the inclusion of additional input and prohibit autonomy. However, the incorporation of the mechanisms from working memory models could allow a neural network to autonomously detect and respond to changes in the expected output without any assistance.

In this paper, we present a general-use framework for implementing the mechanisms inspired by working memory models into common neural network architectures. Since the framework is built around Tensorflow Keras, the new framework components can be added to existing model architectures with minimal adjustment. Along with the design and development of the mechanisms and components, we also aim to demonstrate the framework's generalization ability by applying it across a variety of problems in both supervised learning and reinforcement learning settings. These experiments include the utilization of multilayer neural networks, convolutional neural networks and reinforcement learning.

CHAPTER II.
BACKGROUND

Suppose one would like to model a logic gate using a neural network to predict the resulting binary output given two binary inputs. By presenting the network with some input and the corresponding expected output, the network can learn an accurate approximation of the logic gate function. Now suppose one would like to model two or more logic gates using a single neural network. Once presented the second logic gate function, the expected output for the provided input differs from what was previously modeled, and the neural network must adjust to meet this new target. Due to these conflicting functions, this results in catastrophic interference as the network must now unlearn the previously approximated value to learn the new function. Since the neural network on its own has no means of determining the intended function to approximate, it will be attempting to learn to approximate a target which is always moving; thus making convergence impossible.

There are various methods and approaches to work around this behavior. Analogous to learning multiple functions, one common approach to resolving these issues is to provide context to the network. This is typically achieved by providing additional input to the network, commonly in the form of a one-hot encoded vector. This additional contextual input allows the neural network to distinguish the normal input which may be common among multiple functions, enabling the network to learn each function approximation discretely. This of course requires explicit indication of the active function/logic gate to predict or learn under.

In order to fully autonomize the selection and utilization of contexts, a neural network model would require several internal mechanisms. First, the model would necessitate the ability to maintain a form of context for each of the possible functions. The number of contexts maintained could be specified explicitly upon model creation or dynamically allocated as needed. The model must then be capable of applying contextualization information to the inputs using these maintained contexts. Finally, the model would require self-monitoring mechanisms to determine and switch to the appropriate contexts by analyzing the provided feedback used to adjust the target value to properly adapt to the task. Since working memory models have been used to explain the neural basis of such context-switching behavior [4, 6, 8, 9, 11, 12, 13, 14, 15, 16, 18], we investigate these models and examine their context-driven learning methods and context-switching mechanisms. With our findings, we can design and develop new mechanisms and components that can be integrated directly with typical neural network architectures.

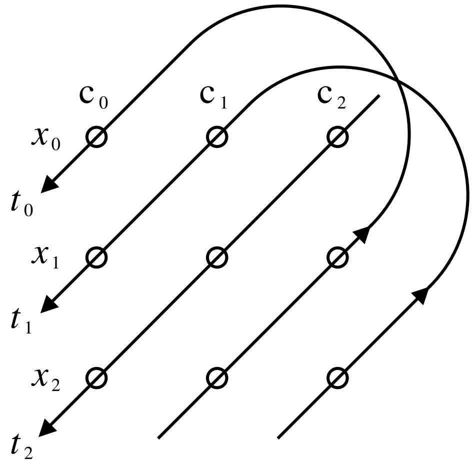
Working Memory

The mechanisms involved in the definition and separation of contextual information stems from the current understanding of working memory and the prefrontal cortex (PFC). A study by O'Reilly et al. [14] provides much insight into these mechanisms and our ability to rapidly update goals and focus on particular tasks. Working memory is an activation-based memory, meaning that memory is retained through persistent neural firing rather than weight-based updates. Learning through weight-based updates is significantly slower and results in perseveration as previously learned information must be unlearned or relearned if task demands have changed. Through activation-based memory and association of contextual information with these neural states, the information contained in working memory can be rapidly updated. It is believed that the PFC maintains a memory representation of the targeted dimension or feature, providing a form of rule-like ”top-down” support, or ”biasing,” to influence the perceptual processing and the selection of actions [14].

While the PFC retains the memory representations, the updates to working memory are handled via the mesolimbic dopamine system. It has been shown that the contents of working memory are updated when neurotransmitter dopamine levels are phasically elevated [1, 4, 8, 11, 18]. Because they are responsible for broadcasting dopamine signals [19], the basal ganglia function as a form of gating mechanism that can update or protect the memory representations currently stored in the PFC [1, 2, 4, 8, 9, 18]. When an expected reward is not delivered, updates to working memory are triggered via a negative error signal. Learning in the presence of these large negative error signals is believed to be limited since the contextual information was likely incorrect [1, 12].

Holographic Reduced Representations

Some more recent models have abstracted away some of the neural details in the above models [15, 16]; in particular, activation-based recurrent layers for handling PFC representations are instead encoded using holographic reduced representations (HRRs) [3, 21, 22]. An HRR is a fixed-width vector that is created by generating real-values from a Gaussian distribution [17]. Any abstract concept can be represented by a single HRR vector; for example, the color blue, a ball, etc.. These HRRs can be combined through the use of circular convolutions, resulting in a new HRR that represents the combination of



$$t_0 = c_0x_0 + c_2x_1 + c_1x_2$$
$$t_1 = c_1x_0 + c_0x_1 + c_2x_2$$
$$t_2 = c_2x_0 + c_1x_1 + c_0x_2$$

Figure 1: A visualization of the circular convolution as a compressed outer product [17].

the overall concept. This new HRR is highly likely to be orthogonal to each of the source HRRs without any increase in dimensionality. While the HRRs themselves are primarily used to automate the distributed encodings of conceptual information, the orthogonality of the resulting circular convolutions allow an artificial neural network to easily distinguish and separate the values of actions based on context.

The computation of the circular convolution, typically denoted as $\circledast$, can be performed on any two vectors of the same length, n. The circular convolution can be defined in terms of the individual components of the resulting vector:

$$t_i \equiv \sum_{j=0}^{n-1} c_j x_{i-j}$$

where $\boldsymbol{c}$ and $\boldsymbol{x}$ are the two input vectors and t_i is the ith component of the resulting vector. Figure 1 provides a simple example and visualization of the circular convolution as a compressed outer product. However, computing circular convolutions in this manner yields a time complexity of $O(n^2)$, making it an expensive operation. Using Fast-Fourier transforms (FFTs), the time complexity of the algorithm can be significantly improved, reducing the overall time complexity down to $O(n \log n)$. This is achieved by transforming each of the input vectors into the frequency domain, multiplying their corresponding components, and transforming the result back to the spacial domain. After incorporating FFTs, we obtain the final equation for computing the circular convolution:

$$\boldsymbol{c} \circledast \boldsymbol{x} \equiv f^{-1}(f(\boldsymbol{c}) \odot f(\boldsymbol{x}))$$

where f is a discrete Fourier transform, f^{-1} is the inverse discrete Fourier transform, and $\odot$ is the component-wise multiplication of two vectors [17].

n-task Learning

Inspired by the models above, n-task learning (nTL) is an approach to autonomous multi-task learning with reinforcement learning problems in mind [6]. The model utilizes abstract task representations (ATRs), each of which internally is a unique HRR, for each of the potential task contexts that are analogous to rule-like representations in the PFC. By encoding the state and action as HRRs and convolving them the active ATR, the resulting HRR is used to contextually separate model inputs in order to discretely learn and predict the value of an action for the active context. The expected reward for each task context is then modeled separately using standard temporal-difference (TD) learning. When negative TD-errors cause the expected reward to drop below a certain threshold, a context-switch is triggered, and the next context ATR is used. If the expected reward for all contexts drops below a threshold, a new context is added and the model weights are re-initialized.

While the models described here are capable of autonomous context learning, there are certain aspects of their implementation that can be quite limiting. For example, the utilization of these mechanisms outside of the reinforcement learning domain has been relatively unexplored. Though the working memory models described above have successfully demonstrated the mechanisms required for context learning/switching, they are rarely utilized outside the realm of working memory research. nTL provides a robust framework for solving a broad set of reinforcement learning problems through the integration of context learning/switching mechanisms. While successful, it requires modeling the reward function and weight re-initialization when dynamically allocating task contexts. We aim to address these issues by creating a new general-use deep learning framework. This framework will enable integration of context switching mechanisms inspired by the working memory models mentioned previously into common/existing neural network architectures with minimal modifications required. Additionally, with the new components the framework will be compatible with more learning approaches outside the realm of reinforcement learning.

CHAPTER III.
MODEL FRAMEWORK

This framework introduces various new components that are compatible with Tensorflow Keras and require only minor modifications to standard training loops to execute. These modifications are described with each of the corresponding components below. In our implementation, the handling of these mechanisms was integrated directly into an extended Keras model. The source code for the framework along with the experiments is available at `https://github.com/DLii-Research/context-learning`

Component Design & Implementation

Context Layer and Abstract Task Representations

One of the fundamental aspects of the framework is its ability to utilize ATRs to differentiate data based on an active context. While there are different potential methods for generating these ATRs [6], the framework continues to implement these ATRs as HRR vectors inline with the working memory models mentioned previously. The generation of these HRR vectors is described in Figure 2 Each ATR is a unique random unitary HRR, and each ATR represents a single context. With a neural network, the input/activation vector can be convolved with the active context's ATR and fed forward through the network. The circular convolution results in a new vector that is roughly orthogonal to each of the source vectors, enabling the model to separate learning of the individual tasks and reducing potential catastrophic interference effects.

To integrate ATRs into the network, we introduce the concept of a *context layer*. This layer is analogous to PFC layers in the previously described working memory models and the ATRs are analogous to PFC stripes within the PFC layers which encode specific contextual activation patterns. The role of the context layer is to generate and store a unique ATR for each of the contexts and compute the convolutions for an active context. The output of the previous layer is convolved with the active context's ATR to produce the new output.

Figure 2 HRR Generation

```
function RandomHRR($hrr\_size$)
    $length$ = floor(($hrr\_size - 1)/2$)
    $x$ = RandomVector($-\pi$, $\pi$, $length$)
    if IsOdd($hrr\_size$) then
        Result = IFFT(Concat([1], Exp(1j * $x$), Exp(-1j * Reversed($x$))))
    else
        Result = IFFT(Concat([1], Exp(1j * $x$), [1], Exp(-1j * Reversed($x$))))
    end if
    return Result
end function
```

Inserting this layer into a neural network will grant the model the ability to learn and predict under these different contexts. Because this layer contains no trainable weights, it depends on additional layers. First, because convolutions require components to be the same size, a 1D layer should precede the context layer. This layer is mainly responsible for projecting the data to match the dimensionality of the ATRs. Since this layer is expected to precede the context layer, the size of the ATRs is determined automatically based on the number of nodes in the preceding dense layer. Lastly, in order for the switching mechanisms to properly calculate the losses generated by the context layer, it is important that the context layer is followed by a 1D layer. This subsequent layer will allow for the application of the delta rule to calculate the context loss as described in the next section.

Context Switching Mechanisms

In order for the context layer to switch contexts autonomously, we developed a loss-based switching mechanism inspired by the working memory models and the mesolimbic dopamine system mentioned previously. It is important to first determine the loss contributed strictly by the given context layer. Using the computed backpropagation gradients, the deltas at the context layer can be extracted via the *delta rule* and the mean-square of the deltas can be computed and accumulated over a full training epoch to obtain a single value, ΔC, representing the overall context loss. If the subsequent layer contains bias weights, the gradients for these weights will be used to compute the context deltas to reduce the time complexity of the calculation. Otherwise the mean-square of the gradients for the non-bias weights is used. The expected loss for each context can then be modeled to measure how well the network has learned the current function as described in the following equation:

$$A(atr) \leftarrow A(atr) + \alpha_A[\Delta C - A(atr)]$$

where α_A is a learning rate parameter. During early training, the values of ΔC are not so important since large context loss is expected; but as the network learns the function, this loss approaches zero and the values of ΔC become more meaningful. To determine when a context switch should occur, the context delta needs to be computed by taking the difference between the expected context loss and the actual context loss as shown in the equation below:

$$\delta = A(atr) - \Delta C$$

The computed value of δ can be directly compared against some negative threshold value t_{switch}. If δ exceeds this threshold, the active context is swapped with the next-in-line and, in compliance with the working memory models, the value of $A(atr)$ is not updated. In a similar fashion, contexts can also be added dynamically when $\max_{atr} A(atr)$ exceeds a secondary threshold t_{add}. These t_{switch} and t_{add} thresholds are implemented as static hyperparameters and must be adjusted based on task performance similarly to learning rate and other common hyperparameters.

For some epochs, it is possible that no context is appropriate for use for the current task (i.e. the context deltas generated under each context exceed the threshold). While training under one context, the manipulation of the weights may interfere with the other contexts, resulting in an unexpected increase in context loss. This loss may be significant enough to trigger an unwanted context switch, or even an infinite context switch loop if δ exceeds the threshold in all known contexts. To combat this, sequential context switches are counted. If all contexts are attempted but all exceed the value of t_{switch}, the context with the lowest ΔC value is chosen and the value of $A(atr)$ is re-initialized to $A(atr) = \Delta C$.

Model & Training Regime

To properly utilize these new components, a new training regiment needs to implemented to integrate all of the required mechanisms. This is accomplished by creating an extended version of the Keras model which we call the `ContextModel` that implements all of the required mechanisms and component interfaces within the `fit` training method. The resulting training procedure aims to achieve compatibility with both supervised and reinforcement learning problems. Figure 3 provides a general overview of the extended training procedure defined within the `ContextModel` and is described in detail below.

Figure 3 Context Model Training Loop

```
1:  absorb: – Indicate if this is an absorbing state
2:  dataset: – The dataset to train on
3:  retry_fit: – Repeat epoch after context switch
4:  for epoch in num_epochs do
5:      BackupWeights()
6:      repeat
7:          gradients = TrainOnDataset(dataset)
8:          switched = False
9:          for ctx_layer in GetContextLayers() do
10:             switched = switched or UpdateAndSwitch(ctx_layer, gradients, absorb)
11:         end for
12:         if switched then
13:             RestoreBackupWeights()
14:         end if
15:     until not (switched and retry_fit)
16: end for
```

First, the switching mechanisms above necessitate some modifications to how weight updates are handled in the model. During a single epoch when a dataset is broken into batches, the weights are being updated on each batch. In this case, if a context switch happens to occur, the weights in the network would have been updated incorrectly. There are two approaches of handling this situation that were considered. The first method accumulates all of the gradients throughout the batches and does not apply the updates until the very end of the epoch after no switch has occurred. While the first approach increases the accuracy of determining when to switch contexts since the model wouldn't be incorrectly learning, the learning rate is slowed significantly and the stability of learning is greatly reduced as the model is now being updated with large gradients. The second approach retains the traditional learning method, applying the weight updates for each batch individually, but requires maintaining a backup of the weights at the beginning of each epoch in the chance a context switch occurs. If a context switch happens to occur during a training epoch, the weights are restored to prevent learning under the incorrect context in compliance with the working memory models and the epoch may be repeated under the new context. This approach results in significantly faster training times since the weights are updated after each batch during an epoch rather than one update at the very end of the epoch. However, since the model would now be learning on an incorrect context, the generated context deltas aren't as significant in magnitude. In the end, the second approach was chosen due to the potential learning issues with the first approach.

Immediately after a context switch, the repetition of the epoch may be desired. In supervised learning problems the model should locate and learn under the best fitting context. Since the context switch is decided at the end of the epoch, all training from that epoch could potentially be reverted; essentially skipping this epoch. For supervised learning problems, it's important to repeat the epoch if a context switch occurs so the model can learn appropriately without wasting any information. However, in reinforcement learning settings, the repetition of epochs may be unwanted as updates to the model are based on previous states that were potentially evaluated under a different context. To handle these situations, the training loop includes the Boolean parameter, $retry\_fit$, that indicates whether an epoch should be repeated upon a context switch.

For temporally-extended reinforcement learning tasks where an agent is required to pass through non-terminal states (typically along the way to reaching a terminal state, such as a goal state for the task), it is important to distinguish these steps to ensure that the model can correctly track sequential switches in a given episode. During these intermediate steps, the sequential switch counter in the switching mechanisms must be retained to allow handling dynamic context allocation and locating best-fit contexts after all contexts have been attempted. In temporally-extended tasks, the terminal states function as special states in which the reward is absorbed whereas the non-terminal states observe a discounted reward. Since this behavior is programmed explicitly in the task, the *absorb* parameter has been included in the framework's training loop to indicate to the model which states are terminal/non-terminal. During the intermediate steps, this parameter should be assigned `false` to prevent the switching mechanisms from resetting the sequential switch counter. This parameter is enough to fully maintain the proper mechanism behavior. While the non-terminal states are usually the most frequent states in temporally-extended tasks, the default value for the *absorb* parameter was chosen to be `true` to provide the highest compatibility among the other learning approaches (supervised learning, 1-step reinforcement learning) without having to explicitly set this parameter's value.

Lastly, it's important to provide a reliable means of initializing the expected context losses, $A(atr)$, for each context. During a new context with an uninitialized expected context loss, the switching mechanisms are disabled to ensure the model can first gain some experience under the active context. By default, the expected context loss is initialized to the observed context loss of the first absorbing training epoch. However, for some tasks, initializing to the first-observed context loss value may be inaccurate as there are other potential input combinations/values that could produce a higher calculated loss. While there are various approaches to solving this issue, the framework currently provides three methods. The first method continues with the normal value initialization, however it disables context switching for x epochs to allow the model to first gain experience on the uninitialized context before attempting to recognize context switches. The second method allows the user to supply explicit values with which to initialize $A(atr)$. For problems where the initial context loss can be unpredictable, this approach presents a reliable means of initialization. The final approach adds a coefficient to the initial context loss value, allowing it to be scaled-up to give additional room for error.

Hyperparameter Tuning & Debugging Utilities

Because the context deltas are generated directly using the gradients and weights of the neural network architecture, the actual context loss and context deltas can be completely different between experiments. As such, the various hyperparameters added by the framework's components (switch/add thresholds, context-loss learning rate, etc.) must be carefully tuned for each experiment. In order to assist in tuning these new hyperparameters, the framework includes various visualization and analyzation utilities. This section will discuss each of these utilities with examples on how a model can be debugged and tuned with high accuracy and efficiency.

The first tool included with the framework is the *context logger*, a Tensorflow Keras callback module that provides the capability of logging all context losses and context deltas recorded during the training period, as well as built-in plot generation. During the training regime, an instance of this logger is passed into the *fit* training call of the model. The modeled context loss for each of the contexts along with all of the context deltas are logged for each training epoch. After training is complete, the logger instance provides methods to generate plots using *matplotlib*. These plots can be used to quickly analyze and debug the switching behavior of any given model and to determine appropriate switch/add thresholds.

An example plot created from the MNIST divisibility task discussed later along with a brief explanation on how to interpret the displayed traces is shown in Figure 4. In this plot, two separate graphs are generated plotting the modeled expected context loss and the generated context delta, δ, at each training epoch. The *Context Delta* plot also includes traces of t_{switch} and t_{add}. Context switches can be identified by the points at which the value of δ fall below t_switch. While there can be multiple context deltas generated in a single epoch due to multiple sequential switches or finding the best-fitting context, the trace indicates the value of δ that triggered the first context switch. Any subsequent context deltas generated in a single given epoch are currently not displayed.

Another tool included in the framework is the *context replay* utility. This utility allows for quick re-modeling of the expected context loss without having to retrain the model from scratch. After training, it may be found that there are hyperparameters related to context-loss modeling that may need tweaking, such as the initial context-loss values, learning rate, etc.. Rather than retraining models from scratch, the context replay utility can track the generated context losses and deltas over time and replay them back using the specified switching mechanisms and hyperparameters. This allows for quick re-modeling of the expected context loss and context deltas so that these hyperparameters can be tuned much more quickly and easily. Since the utility is implemented as an additional Tensorflow Keras callback module, it can be passed into any context model during during the *fit* training call.

Lastly, while the visualization from the context logging can aid in determining good values for the context switch threshold, t_{switch}, and context add threshold, t_{add}, the values of these thresholds can be greatly optimized through an approach we call *emulated context switching*. Emulated context switching works by disabling autonomous context switching while training a model on a dataset or a subset of that dataset and explicitly indicating to the model when a switch should occur. During emulated switching, the expected context loss and the generated context deltas are still logged as normal and can be analyzed through

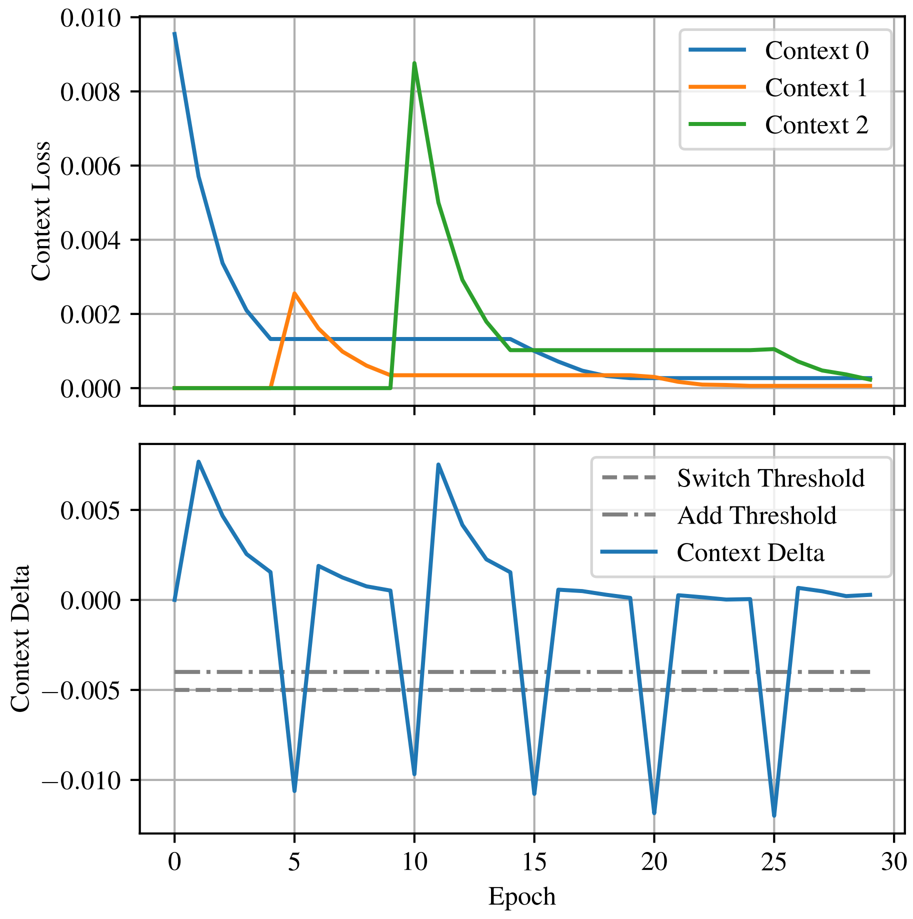


Figure 4: An example plot generated from the context logger containing traces of the context-loss and context-deltas over time. The *Context Loss* plot portrays the modeled expected context loss at the given training epoch, while the *Context Delta* plot displays the calculated Context Deltas at each epoch. The negative-valued spikes in the Context Delta plot that lie below the *switch threshold* are the primary indication of a context switch being triggered. In supervised learning tasks, an epoch may be repeated under different contexts to search for one that fits. In that case, the spike visible is the original delta that triggered the first switch. Subsequent deltas are not currently indicated on the graph. If the context delta is found to be below the *switch threshold* in every case, the context yielding the maximum context delta is used. If dynamically-allocated contexts are available, this threshold is compared to the *add threshold*, potentially adding a new context to the model if the delta is found to exceed it.

the context logger. Additionally, the deltas at the indicated switch positions are logged separately with their corresponding training epoch. Since this emulated switching model is switching optimally, these logged context deltas can be analyzed externally to determine appropriate switching thresholds with much greater precision and accuracy.

Evaluation & Controls

We evaluate this framework over a variety of both supervised learning and reinforcement learning tasks. These tasks serve not only to provide support for successful learning with the framework, but also demonstrate the overall capability and generalizability across the variety of different tasks. While the framework is primarily evaluated with static context allocation, sample runs for each task using dynamic context allocation are included. These tasks and experimental details are described in the following chapters.

It is also important to note that due to the novelty of this framework it is difficult to make any fair comparisons with other models. While our controls focus on the reduction in available ATRs, we cite the controls provided by [6] as they are solving the same types of problems described here. LSTM's were chosen as the state-of-the-art model for one of their controls. They were able to demonstrate and conclude that LSTM's were fundamentally unable to learn these types of tasks. This result was not surprising as error-monitoring mechanisms are not present in standard deep-learning approaches but are critical components of task-switching models in the literature.

CHAPTER IV.
SUPERVISED LEARNING TASKS

To train the supervised learning models, the framework provides an extended training regiment. First, a separate training dataset is created for each of the desired contexts. Each dataset may share the same input values, but will differ in the expected output values/labels for each context. The training regime then trains the model on each dataset indivdiually for *n* epochs each. The model is said to have completed a *training cycle* once it has learned on each of the context datasets. Typically, the regime trains a model over the full list of datasets for *k* complete training cycles, shuffling the order of the datasets at the beginning of each cycle. While the training regime is capable of maintaining a mapping of the contexts to each of the datasets, the original context order can often be preserved by supplying the list of datasets in the appropriate initial order and only shuffling during the subsequent cycles. This training regiment is applied to each of the supervised learning tasks described below.

Logic Gate Task

As a simple proof-of-concept, this supervised-learning task requires a single model to learn and predict multiple different logic gates autonomously without any provided information regarding the active logic gate. The model will be presented a training batch containing two binary inputs and the expected output for a particular logic gate. After some time, the logic gate will be swapped for a different gate, modifying the model's expected output. The model should recognize this and switch contexts to avoid overwriting it's prior knowledge. In the end, the model should be able to accurately predict the output for each logic gate under its own context.

Methods

This experiment constructs a multilayer neural network utilizing the framework as shown in Figure 5 to model a total of eight different logic gates. These gates are comprised of six common gates: *AND, OR, NAND, NOR, XOR, XNOR*; along with two additional custom gates that simply forward one of the given input values. Each logic gate is separated into its own dataset containing all possible pairs of bipolar inputs (-1 is used in-place of 0) along with the expected output value. The order of the logic gate datasets is randomized during each training cycle, and the active dataset is shuffled during each training epoch. The model is trained with a batch size of 1 and updated via stochastic gradient descent (SGD) and binary-crossentropy loss. The resulting hyperparameters for this experiment are described in Table 1.

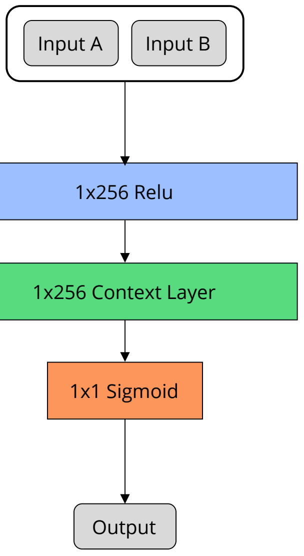


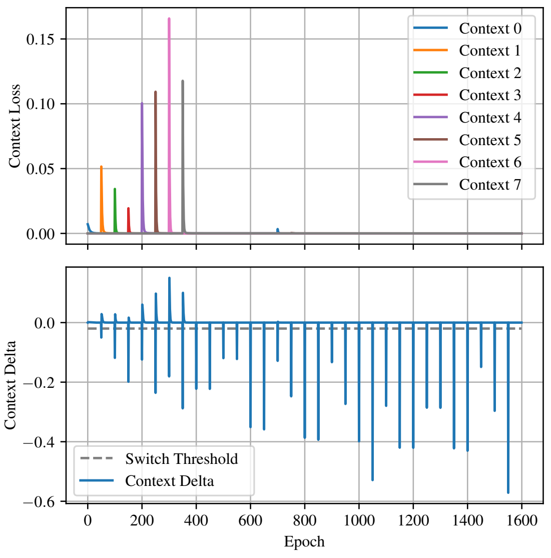
Figure 5: The model architecture for the logic gate task. The model receives two input integers, projects them into a higher dimension using a Relu layer and passes them through the context layer. The resulting output is then projected down to a single sigmoid output value.

Table 1: Logic Gate Task Training Parameters

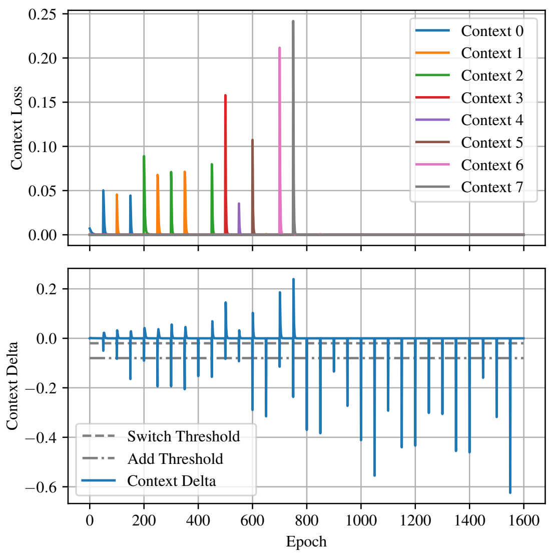
Name	Value	Description
k	4	Number of training cycles
n	50	Number of training epochs/cycle
α	0.1	Learning rate for optimizer
α_A	0.5	Learning rate for $A(atr)$
t_{switch}	-0.02	Context-switch threshold
t_{add}	-0.08	Context-add threshold

Results

First, we can examine the two experimental runs of the logic gate task shown in Figure 9. These two runs demonstrate successful learning of the task as seen by the modeled context losses of each context via statically-allocated and dynamically-allocated contexts. During early training, large context loss is expected which is indicated by the large spikes in the context loss graphs. As the model learns the gate under a context, the expected context loss approaches zero. When the next gate is given, the spike in the context delta (indicated in the context delta plots) triggers a context switch. Since the contexts haven't seen or lack experience with the task, they will have much higher expected context losses early on. While the statically-allocated experiment quickly learned each of the gates under its own context, the dynamically-allocated context model can be seen attempting to learn multiple gates



(A) Statically-allocated Contexts



(B) Dynamically-allocated Contexts

Figure 6: The resulting context loss and context delta traces for each of the experimental runs. The top row of graphs are plots of the value of $A(atr)$ for each context at each epoch. The bottom row of plots are traces of the computed value of δ from each epoch. Context switches can be identified by the δ trace exceeding the switch threshold value.

under the same context as indicated by the repetition of context loss spikes. This is often due to task outputs differing by only a single bit, for example AND and XNOR. As the network continues to partition such conflicting task outputs to a single context ATR, the context losses become more significant and trigger the mechanisms requiring the addition of another context. During development it was found that while the dynamically-allocated context model would frequently attempt to learn multiple gates under a single context, the model consistently allocated the correct number of contexts and learned each gate successfully in the end.

We then perform a benchmark over this task using statically-allocated contexts over

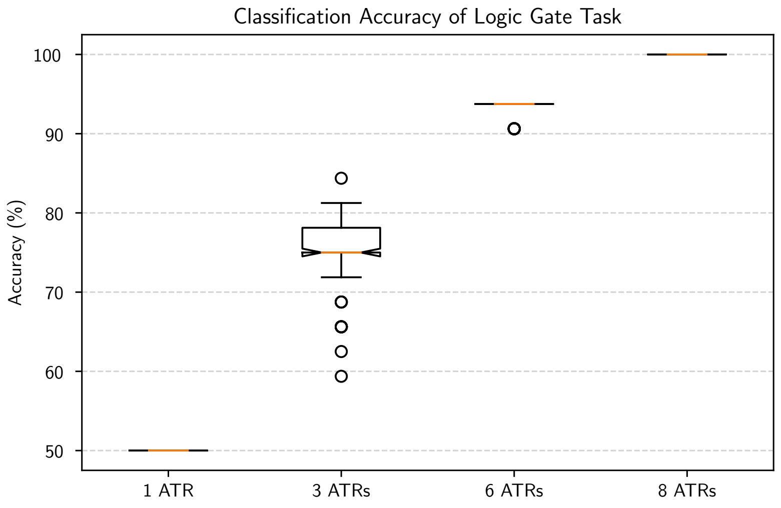


Figure 7: Classification accuracy distributions for the logic gate task across a range of statically-allocated ATRs. For each boxplot, 100 independent models were trained, each with random initial weights and ATRs. Each model was set to use the corresponding number of ATRs, run for a training regime of 8 cycles, 50 epochs each, and the average classification accuracy across all gate functions was recorded at the end of training. The orange line indicates the median accuracy value across the 100 models and the notches above and below the median represent a 95% confidence interval. Additional hyperparameter values are specified in Table 1. In general, using too few ATRs results in poorer performance since conflicting logic functions cannot be represented in the models without independent context representations for each function.

100 runs. This benchmark includes control experiments with a multi-layer neural network (essentially a contextual network limited to using one ATR) along with two contextual neural networks allocated with fewer ATRs than the number of actual gates for comparison. As shown in Figure 7, after a 8 full training cycles the models where the number of allocated ATRs are fewer than the actual number of contexts cannot fully learn the tasks. The standard neural network converged on the expected theoretical 50% accuracy. As the number of ATRs approaches the actual number of contexts, the accuracy increases and approaches 100%. It's not until the number of ATRs is equal to the number of actual contexts before the accuracy converges on 100%. After the full 8 training cycles, all 100 models were able to obtain 100% classification accuracy.

MNIST Divisibility Task

With a simple proof-of-concept in place, this next task aims to provide a more practical supervised learning demonstration by classifying images from the MNIST dataset as either even, odd, or divisible by 3. The MNIST dataset provides handwritten digits in the form of 28x28 greyscale images and comes pre-split with 60,000 images for training and 10,000 images for testing [10]. The model must recognize the digit in the image and output a single value indicating whether or not the image satisfies the rule.

Methods

While this problem could be approached with a multilayer neural network similar to the model described in the previous problem, we instead construct a convolutional neural network integrated with a context layer as described in Figure 8. The model takes a single image as input and outputs a single binary value indicating if the given digit fits the current rule. The images in the dataset are normalized such that each greyscale value is between 0 and 1, and the datasets/images are shuffled in the same manner as described in the logic gate task. The model is trained on the first 5,000 images from the training dataset and evaluated against the first 1,000 images from the testing dataset using a batch size of 32. Weight updates are applied using the Adam optimizer and binary-crossentropy loss. The hyperparameters are listed in Table 2.

Table 2: MNIST Divisibility Task Training Parameters

Name	Value	Description
k	2	Number of training cycles
n	5	Number of training epochs/cycle
α	0.001	Learning rate for optimizer
α_A	0.5	Learning rate for $A(atr)$
t_{switch}	-0.005	Context-switch threshold
t_{add}	-0.004	Context-add threshold

Results

This task plays an important role since the model architecture in this experiment is a convolutional neural network. Analyzing the plots of this task in Figure 8, it can be noted that both static and dynamic context allocation models were very stable in learning. This stabilization was achieved due mainly to the large training dataset of 5,000 images. Along with the stabilization both models were also very consistent in their learning of the task, regardless of their context allocation method. Even with the smaller training dataset, these

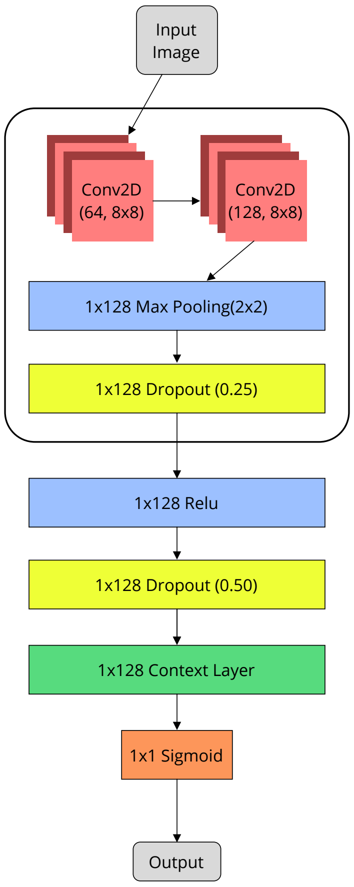
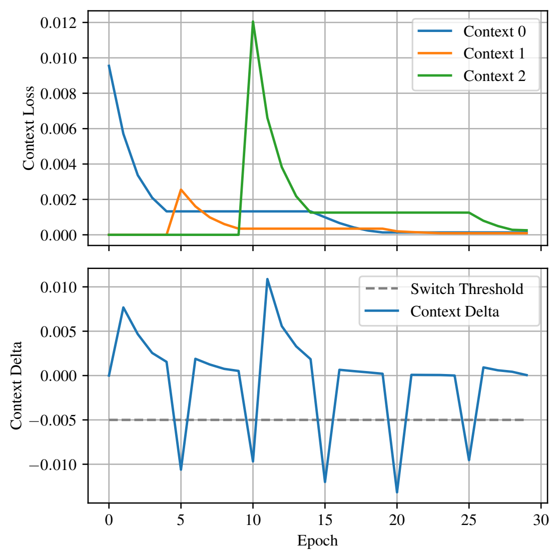
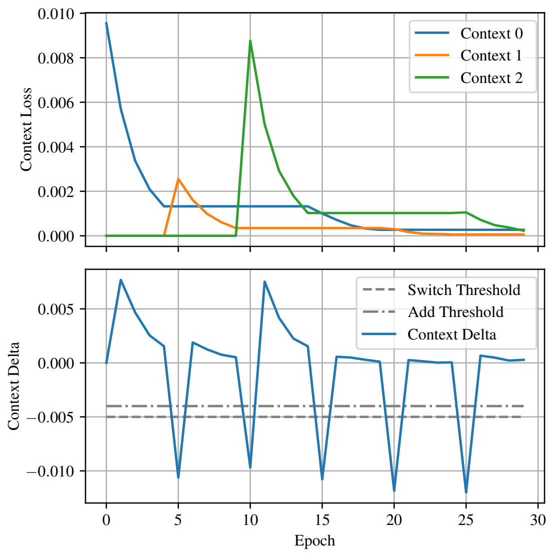


Figure 8: The model architecture for the MNIST divisibility task. The model takes in a 28x28 MNIST normalized image as input and feeds it through a sequence of 2D convolution layers where the result is project and convolved with the active context. The final output is a single sigmoid value indicating if the image satisfies the given rule.

models were able to achieve an accuracy of 98% on never-before-seen images.



(A) Statically-allocated Contexts



(B) Dynamically-allocated Contexts

Figure 9: The resulting context loss and context delta traces for each of the experimental runs. The top row of graphs are plots of the value of $A(atr)$ for each context at each epoch. The bottom row of plots are traces of the computed value of δ from each epoch. Context switches can be identified by the δ trace exceeding the switch threshold value.

CHAPTER V.
REINFORCEMENT LEARNING TASKS

For each of the reinforcement learning experiments in this section, we utilize the Q-learning algorithm, a variation of TD-learning, to model the expected value of future rewards and learn the optimal policy [20]. Q-learning is an off-policy algorithm which prevents exploratory actions from negatively impacting the learning of the policy. Using the Q-function, $Q(s,a)$, the value of each available action, a, for a given state s can be evaluated individually. Typically, a form of greedy policy is employed which selects the action with the highest Q-value. After transitioning to a new state, the Q-function for the previous state/action pair can be updated with observed reward. For terminal states, the update equation is equivalent to that of standard TD-learning:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \cdot [r_{t+1} - Q(s_t,a_t)]$$

where $alpha$ is the learning rate. For n-step problems with non-terminal states, the update to the Q-function factors in the highest-valued action at the new state with some discount factor, *gamma*, preventing negative impact from exploratory actions. The update equation for non-terminal states is defined below.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot max_a Q(s_{t+1},a) - Q(s_t,a_t)]$$

In the following sections, we construct various tasks that utilize the Q-learning algorithm. These tasks consist of both 1-step and n-step Q-learning implementations to demonstrate the complete potential of the framework under reinforcement learning settings.

Wisconsin Card Sorting Test (WCST)

The first task utilizes the framework to solve a 1-step reinforcement learning task based on the Wisconsin Card Sorting Test (WCST). The WCST is a common working memory task that requires a participant to learn and rapidly switch between various rules dictating how to sort a presented card based purely on correct/incorrect feedback. This task consists of a number of rounds and trials. During a trial, the participant is presented a stimulus card containing three features drawn from three dimensions (number, color, shape), each dimension having three possible feature values. For example, a card may consist of two red circles, three green squares, one blue triangle, etc.. The participant is then required to match the card based strictly on a particular dimension governed at the beginning of each round, of which they are not informed. After several trials have been attempted, the round ends and a new round begins with a brand new sorting rule.

Methods

As mentioned in the beginning of this chapter, we implement the 1-step Q-learning algorithm using the framework to model the expected reward and learn the sorting policy for each rule. Once presented a stimulus card, the Q-learning model allows the agent to predict the values of each of the possible actions. Here, an ε-greedy policy is used to govern action selection. This policy prefers the selection of actions that yield the highest expected value, but with some probability, ε, will select an action completely at random to employ a means of exploration. At the end of the trial, the agent can use the observed reward to update its previous prediction. Since this task is a special case of Q-learning in that every action results in transitioning to a terminal state, the full reward is absorbed without ever applying any discount factor.

The next step is to create the neural network model to predict the Q-values based on the given stimulus card. The stimulus card is represented by a 3x3 matrix where each row is a dimension consisting of a one-hot encoded feature, giving three possible dimensions to sort by. The model can accept this card as direct input and predict the expected value each of the possible actions simultaneously. While simultaneous action prediction/fitting is typical with Q-learning models built on top of neural networks, it can be problematic when working with general context models. This is due to situations where the model attempts to locate the best-fitting context using Q-values generated under a different context, leading to inaccurate context switching. While the model could be extended to enable the recalculation of Q-values under new contexts, this issue is instead resolved by providing both the state and the chosen action as input. The model then outputs a single value representing the value of that action choice for the given state. The architecture for the resulting model is described in Figure 10.

Next, we construct the complete WCST experimentation procedure as described in Figure 11. To keep the task simple, the chosen features contained in the stimulus card are

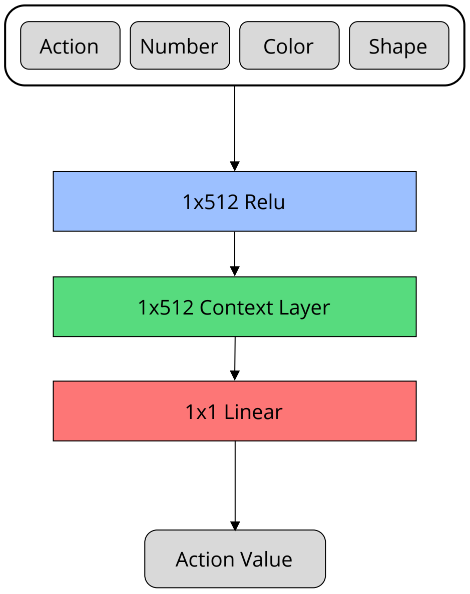


Figure 10: The model architecture for the Wisconsin Card Sorting Test. The model receives four one-hot encoded vectors flattened into a single dimension: action, number, color, and shape. The output is a single Q-value indicating the learned value of the action at the given state.

guaranteed to be free of any ambiguity that could come from the feedback (two dimensions happen to share the same sort). This is achieved by shuffling strictly the rows of a 3x3 identity matrix, resulting in a card consisting of distinct feature values across dimensions. The procedure also ensures that the model experiences each possible rule once before any rule is repeated. This is to reduce any difficulty in learning for models that utilize statically-allocated contexts as they would have to remap rules to different contexts. While this kind of model is very well capable of doing so, a model utilizing dynamically-allocated contexts would be better suited. Once all rules have been in play, subsequent rounds select rules at random so long as they differ from the previous round.

Figure 11 WCST Procedure

```
function WCST($model$, $n\_rounds$, $trials\_per\_round$)
    $rules$ = Shuffle(AllPossibleRules())
    $rule$ = null
    for $round = 0$ to $n\_rounds$ do
        if UsesDynamicContexts($model$) or IsEmpty($rules$) then
            $rule$ = NewRandomRule($rule$)
        else
            $rule$ = Pop($rules$)
        end if
        for $trial = 0$ to $trials\_per\_round$ do
            Trial($model$, $rule$)
        end for
    end for
end function
function TRIAL($model$, $rule$)
    $card$ = SelectRandomCard()
    $action$, $value$ = Policy($model$, $card$, $\varepsilon$)
    $reward$ = Reward($action$, $rule$)
    $target$ = $value$ + $\alpha$*($reward$ - $value$)
    FitModel($model$, $card$, $action$, $target$, $retry\_fit$=false)
end function
```

Finally, we constructed the reward schedule and fine-tuned the hyperparameters using emulated context switching. The employed reward schedule provides the agent with $+1$ for correct sorting and -1 for incorrect sorting. It was found through experimentation that differentiating correct/incorrect actions through positive and negative rewards consistently produces much more distinct context deltas, allowing for more accurate switching. It was also found that removing bias weights from the dense layers of the network resulted in deltas that were more consistent among the contexts for this task. Since every step is independent of the previous, the stochastic gradient descent optimizer was used to train the model. The final list of hyperparameters values specified for the task are presented in Table 3. We evaluate the results of the models in the next section.

Results

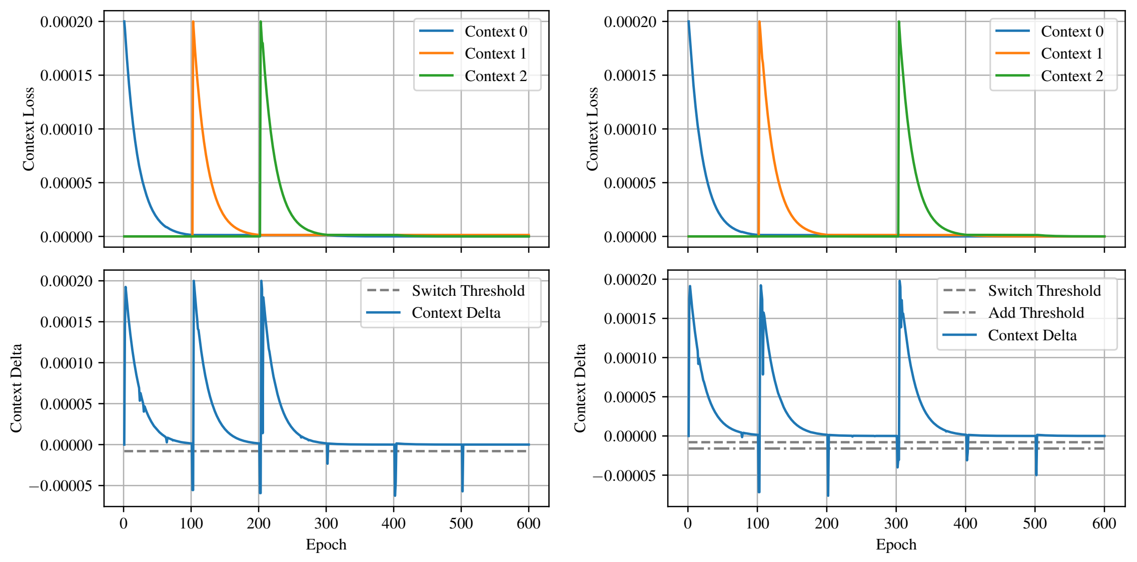
For a proof-of-concept, the results from two successful sample runs are shown in Figure 12 are first analyzed, each using different models: one where contexts are allocated statically, the other dynamically. The statically-allocated context models were presented each possible rule before a rule ever reoccurred. As mentioned previously, this prevents unnecessary unlearning and remapping of the contexts in the early stages of training. However, true random rule selection was utilized for the model using dynamically-allocated contexts as it can adapt only when needed. The large initial context loss gives much needed room for error during the context's first experience with the rule, indicated by the large negative spikes in the context delta traces when expected loss is high. This initial loss ensures these spikes

Table 3: WCST Task Training Parameters

Name	Value	Description
n	20	Total number of rounds
n_{trials}	100	Number of trials per round
ε	0.1	Probability of non-greedy action
α_O	0.1	Learning rate for optimizer
α_Q	1.0	Learning rate for Q-learning
α_A	0.05	Learning rate for $A(atr)$
$A_{initial}$	$2e-4$	Initial loss value of $A(atr)$
t_{switch}	$-8e-6$	Context-switch threshold
t_{add}	$-1.6e-5$	Context-add threshold

remain above the switch threshold to prevent early switching and allow the model time to first gain experience with the context. As the expected loss approaches zero, the model can be much more confident with context switching and the context deltas can be more distinctly recognized.

Next, we examine the results of the benchmark employing 100 different models utilizing statically-allocated contexts with the task. Figure 13 presents the mean of the overall classification accuracy for each model model after each trial. Each model's classification accuracy was determined by applying each rule to the best-fitting context and evaluating the action choices of all possible stimulus cards. The accuracy of each context in each of the models was averaged to determine that model's overall accuracy. In the end, 99/100 models



(A) Statically-allocated Contexts (B) Dynamically-allocated Contexts

Figure 12: The resulting context loss and context delta traces for sample runs of the Wisconsin Card Sorting Test using two different models over 600 trials, one consisting of statically-allocated contexts and the other consisting of dynamically-allocated contexts. These plots demonstrate successful runs in both context allocation methods while also providing visualization of the distinct context deltas generated at the beginning of each round. The hyperparameters can be found in Table 3.

were able to achieve an overall decision accuracy of 100%, leaving only a single outlier in the results.

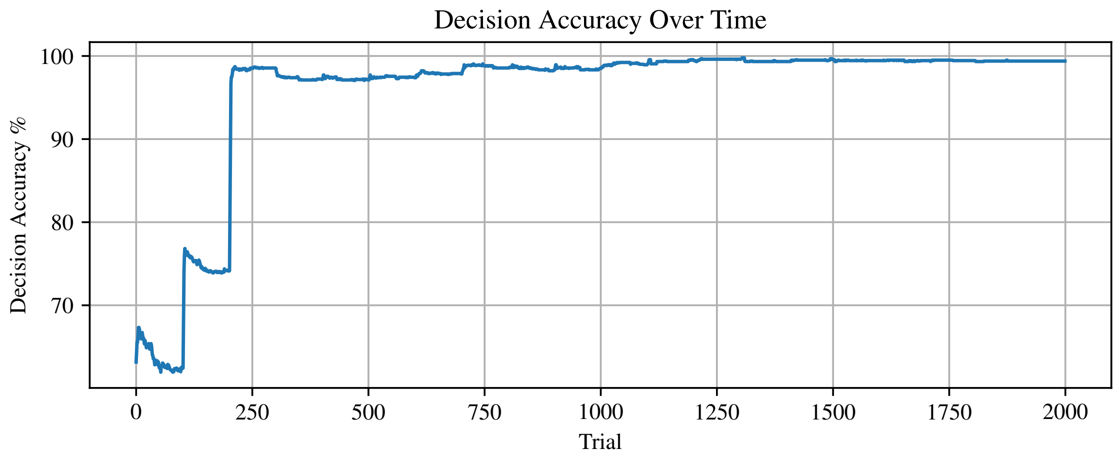


Figure 13: The average decision accuracy of the Wisconsin Card Sorting Test over time across 100 different models. For each model, rule is mapped to its best fitting-context where all possible cards are evaluated. The average prediction accuracy of each model is then averaged and plotted above.

1D-maze Task

In this final experiment, we construct a temporally-extended reinforcement learning task requiring an agent to solve a 1-dimensional maze with varying goal states. In this task, the agent is positioned in a random starting state at the beginning of each episode and must traverse the maze moving either left or right to locate the active goal state. The environment is partially observable in that the agent can observe its current state, but all other information is kept hidden. The maze is also periodic such that if the agent moves beyond one end of the maze it will land on the opposite side of the maze. After a number of episodes, the goal state will switch positions and the agent must switch contexts to properly accommodate the new goal. In the end, the agent should learn the optimal path to each of the goal positions under their own contexts.

Methods

Similar to the WCST experiment, this experiment utilizes the Q-learning algorithm with the framework's context model to approximate the expected future rewards to learn the optimal policy for traversal. As this task is temporally-extended, an episode may consist of the agent taking multiple steps to get to the goal position. During each step, the agent evaluates and determines the action to take based an ε-greedy policy.

We can now construct a context model to predict the Q-values based on the state and action. Each of the possible states in the maze is one-hot encoded to be used as direct input to the model. Due to the same issues described in the WCST task, the concatenation of the one-hot encoded state and chosen action vectors will be required to produce appropriate context deltas. This concatenated vector can then be given as input to the model to predict the value of the given single state/action pair. The final architecture is described in Figure 14.

With the learning approach and model architecture, the 1D-maze task procedure is created as shown in Figure 15. Like the WCST task, each goal is picked at random ensuring for statically-allocated context models that each goal is selected once before a goal is ever re-selected. At the beginning of each episode, the agent is inserted into a random starting state always differing from the goal position. This eliminates immediate episode termination and always gives the agent a chance to update its knowledge for each episode. The agent then predicts and evaluates action choices for its current state and decides which adjacent state to transition to based on the policy. Upon transitioning to the new state, the agent observes the received reward and updates its predicted value for that previous state/action pair. It's important to note that as this is a temporally-extended task it's required to denote absorbing states when fitting the model to a target value to ensure the switching mechanisms are correctly engaged. Since the calculation for the target value used in Q-learning is dependent on whether the state is absorbing or not, indication of this information fits naturally within the procedure.

Through experimentation using emulated context switching, the hyperparameters were tuned and the reward schedule was created. The procedure rewards the agent with $+1$ for reaching the goal and -1 for any intermediate steps. As mentioned in the WCST task, the differentiation in positive and negative rewards assists in producing more distinct context deltas and thus more accurate switching. However, in this task it was determined through experimentation that enabling the bias weights within the dense layers produced context deltas with much greater magnitude than without the bias weights. It was also found during experimentation that large context deltas could occur when exploring new contexts for the first time making it difficult to prevent inaccurate switching. This can be resolved by explicitly increasing the initial context loss value, but the alternative approach used in this experiment was to disable the autonomous switching mechanisms for the first n_{switch} episodes of each new context. This gives the model time to learn the general shape of the function for the current context before attempting to recognize context deltas. The model weights are then updated using the stochastic gradient descent optimizer with a lower learning rate to stabilize learning and avoid exploding gradients. All of the final hyperparameters for this experiment are provided in Table 4.

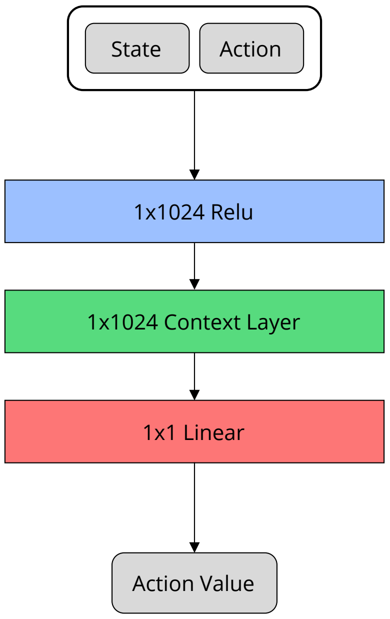


Figure 14: The model architecture for the 1D-maze task. The model receives two one-hot encoded vectors, state and action, flattened into a single dimension. The output is a single Q-value indicating the learned value of the action at the given state.

Table 4: 1D-maze Task Training Parameters

Name	Value	Description
n	3000	Total number of episodes
n_{switch}	500	Number of episodes between goal switches
ε	0.3	Probability of non-greedy action
α_O	0.01	Learning rate for optimizer
α_Q	1.0	Learning rate for Q-learning
α_A	0.003	Learning rate for $A(atr)$
A_{delay}	250	Episodes to delay switching for new contexts
$A_{initial}$	5.0	Initial loss value of $A(atr)$
t_{switch}	-0.06	Context-switch threshold
t_{add}	-0.06	Context-add threshold

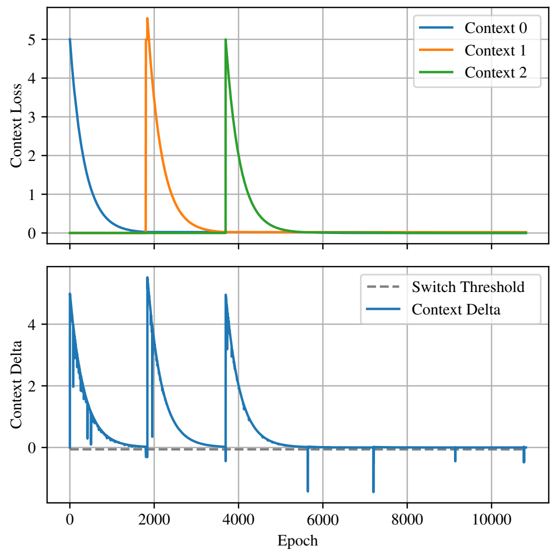
Figure 15 1D-maze Task Procedure

```
function MAZETASK($model$, $maze\_len$, $goals$, $n\_episodes$, $switch\_frequency$)
    $goal\_indices$ = Shuffle(GoalIndices($goals$))
    $goal\_index$ = null
    for $episode = 0$ to $n\_episodes$ do
        if $episode$ mod $switch\_frequency$ == 0 then
            if UsesDynamicContexts($model$) or IsEmpty($goal\_indices$) then
                $goal\_index$ = NewRandomGoal($goal\_index$, $goals$)
            else
                $goal\_index$ = Pop($goal\_indices$)
            end if
        end if
        Episode($model$, $maze\_len$, $goals[goal\_index]$)
    end for
end function
function EPISODE($model$, $maze\_len$, $goal$)
    $state$ = RandomStartingState($maze\_len$, $goal$)
    $new\_state$ = null
    $moves$ = 0
    while $state \neq goal$ and $moves <$ MOVE_LIMIT do
        $action$, $value$ = Policy($model$, $state$, $\varepsilon$)
        $new\_state$ = NextState($state$, $action$, $maze\_len$)
        $reward$ = Reward($new\_state$, $goal$)
        if $new\_state$ == $goal$ then
            $target = value + \alpha \cdot (reward - value)$
            FitModel($model$, $state$, $action$, $target$, $retry\_fit$=false, $absorb$=true)
        else
            $q_{max}$ = MaxActionValue($model$, $new\_state$)
            $target = value + \alpha \cdot (reward + \gamma \cdot q_{max} - value)$
            FitModel($model$, $state$, $action$, $target$, $retry\_fit$=false, $absorb$=false)
        end if
        $state = new\_state$
        $moves = moves + 1$
    end while
end function
```

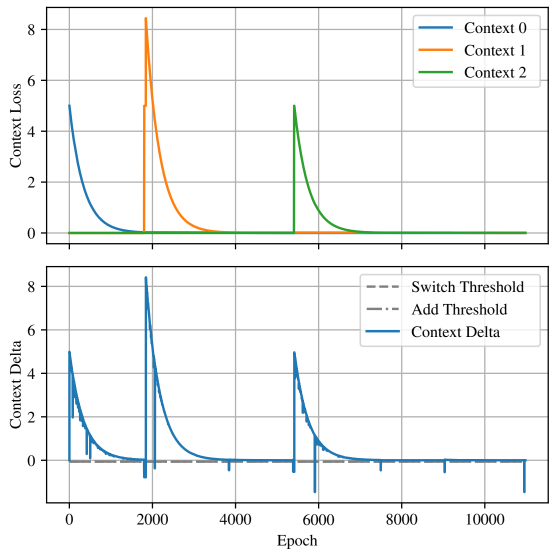
Results

We first provide and analyze two successful runs with both different models using each context allocation method. The resulting traces for these runs are displayed in Figure 16. The context delta's triggering the context switches are quite distinct, but it's also worth noting the large context spikes that happen to occur during the early stages of training. In Figure 16B, some context deltas during the model's first experience with a new goal position can even be seen spiking below the threshold. Since the context deltas generated during early training can be quite large compared to the deltas later in the experiment, we utilize the switch delay mechanism to disable autonomous switching during the first 250 episodes of the task, rather than tweaking the initial loss hyperparameters. For problems where the initial loss can be unpredictable, this alternative approach can prove much more reliable.

After analyzing the individual runs, a benchmark of 20 different models was performed and evaluated by examining the overall accuracy in optimal move selection for each model, as plotted in Figure 17. Of these models, all but one fully learned the optimal policy for each of the goal positions under their own contexts. For each of the models, each goal position in the maze is matched with the best-fitting context (i.e. the context generating the least loss). By calculating the distance to the corresponding goal position at each state under each of the contexts, the number of sub-optimal moves can be accumulated based on the current learned policy. Computing the number of optimal moves vs. the number of non-terminal states yields the model's overall decision accuracy. Another common approach to evaluating similar temporally-extended tasks is to evaluate the number of sub-optimal moves over time by providing intermittent runs throughout training where exploratory moves are disabled. While this approach was considered, unless setting the correct context explicitly, the number of sub-optimal moves would would never approach zero as the model would frequently require context switching to reach the correct goal position. It is for this reason that the alternative approach to examining the number of sub-optimal moves was employed.



(A) Statically-allocated Contexts



(B) Dynamically-allocated Contexts

Figure 16: The resulting context loss and context delta traces for each move for sample runs of the 1D-maze task using two different models over 3,000 episodes, one consisting of statically-allocated contexts and the other consisting of dynamically-allocated contexts. These plots demonstrate successful runs in both context allocation methods while also providing visualization of the distinct context deltas generated at the locations of goal switches. The hyperparameters for these runs are specified in Table 4.

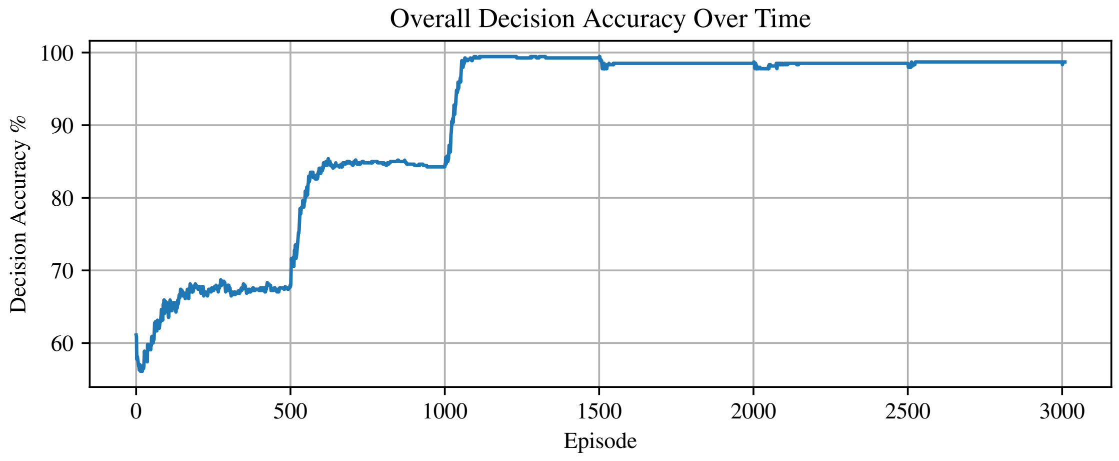


Figure 17: The average accuracy for optimal move selection in the 1D-maze task over time for 20 different models. For each model, each goal is mapped to its best-fitting context and the policy is evaluated for its corresponding goal position by calculating the average accuracy of optimal move selection across all possible states.

CHAPTER VI.
DISCUSSION & FUTURE WORK

Working memory is a critical feature to our decision making process. While current research is able to accurately explain how working memory and its internal mechanisms function, the computational models created have not been used much outside the realm of working memory research. Here, we presented a new deep learning framework inspired by these working memory models that allow traditional neural networks to autonomously learn under a varying number of contexts. The implementation of this framework into Tensorflow Keras allows for integration of context learning into existing neural networks with minimal-to-no modifications to the architecture required.

Much of the framework's capability and generalizability was demonstrated through a variety of tasks. It was shown that the integration of the context layer in multi-layer neural network models can allow for learning under multiple contexts in both supervised learning and reinforcement learning settings with little-to-no task-specific modifications. Beyond multi-layer neural networks, a convolutional neural network integrated with a context layer was also successfully demonstrated to learn multiple classification rules on the MNIST dataset.

Since the few existing models that include these contextual learning mechanisms have been designed with reinforcement learning in mind, our framework is rather novel in the fact that it is also compatible with supervised learning problems. Furthermore, the successful integration of a context layer into convolutional neural networks presents powerful potential. A single network could house a variety of image classifiers or even distinctly search for and detect the presence of certain features within an image/video. While this framework lays the foundation for the integration of context learning mechanisms into common neural network models, there is much that remains to be explored.

It's worth noting that the experiments in this framework focused mainly on static context allocation to allot sufficient time and resources into the design and development of the core mechanisms and components. As a result, these statically-allocated context models require more careful dataset handling. Since each of these models would switch to the next uninitialized context during the early stages of training, it was important that the dataset for each given task was presented to the model once before re-occurring. This prevents the model from having to unlearn and remap tasks to different contexts. With this in mind, the supervised learning tasks were learned on the datasets in terms of *cycles*, where each of the possible tasks are shuffled and presented to the model. This guarantees a uniform distribution of the tasks/datasets. While sample runs were provided in each of the experiments, models utilizing dynamically-allocated contexts would be capable of resolving these issues and allowing for truly randomized task input. Future research with a focus on dynamic context allocation could result in a significant advancement to the overall capability of the framework.

While the context-switching mechanisms worked well for the experiments included here, there are still clear limitations. Due to the stochastic nature of some tasks, or the learning interference affecting inactive contexts, the static threshold may cause difficulty in more complicated tasks potentially reducing the accuracy and increasing the task's learning difficulty. Investigation into dynamic threshold mechanisms could result in significant performance increases as it could avoid unnecessary context switching. These mechanisms could allow much more room for error during the early stages of training when a lot of noise in context loss is expected, and narrow in as the model becomes more confident with its predictions. It is also important to note that the current model ignores the sign of the context deltas. In a reinforcement learning setting, observing the sign of the context delta can lead to more information-driven context switching [7] as positive deltas can often indicate the appropriate context for the current task.

Finally, there are additional concepts/methods of facilitating the context layer that could be investigated. For instance, the framework currently only supports 1-dimensional abstract task representations, but circular convolutions can be performed on higher dimensions. Future work could include the implementation of these higher dimensional ATRs that can be convolved with 2D or 3D activation/input vectors. Furthermore, the experiments in this project focused on the use of a single context layer toward the end of the network, but the implementation of the framework theoretically allows any number of context layers. These additional layers could add levels of contextual support to a model, perhaps increasing the range of problems by introducing macro/micro contexts. These concepts were not experimented with during the development of this research; however, they could prove to be a powerful addition to the framework architecture.

BIBLIOGRAPHY

[1] Christopher H. Chatham and David Badre. Multiple gates on working memory. *Current opinion in behavioral sciences*, 1:23–31, Feb 2015. ISSN 2352-1546. doi: 10.1016/j.cobeha.2014.08.001. 26719851[pmid].

[2] Christopher H. Chatham, Michael J. Frank, and David Badre. Corticostriatal output gating during selection from working memory. *Neuron*, 81(4):930–942, Feb 2014. ISSN 1097-4199. doi: 10.1016/j.neuron.2014.01.002. 24559680[pmid].

[3] G. M. DuBois and J. L. Phillips. Working memory concept encoding using holographic reduced representations. In *Proceedings of the 28th Modern Artificial Intelligence and Cognitive Science Conference*, 2017.

[4] Michael J. Frank, Bryan Loughry, and Randall C. O'Reilly. Interactions between frontal cortex and basal ganglia in working memory: A computational model. *Cognitive, Affective, & Behavioral Neuroscience*, 1(2):137–160, Jun 2001. ISSN 1531-135X. doi: 10.3758/CABN.1.2.137.

[5] Ashish Gupta, Lovekesh Vig, and David C. Noelle. A Cognitive Model for Generalization during Sequential Learning. *Journal of Robotics*, pages 1–12. ISSN 1687-9600. doi: 10.1155/2011/617613.

[6] M. Jovanovich and J. L. Phillips. n-task learning: solving multiple or unknown numbers of reinforcement learning problems. In *In Proceedings of the 40th Annual Meeting of the Cognitive Science Society*, 2018.

[7] N. Khan and J. L. Phillips. Combined model for sensory-based and feedback-based task switching: Solving hierarchical reinforcement learning problems statically and dynamically with transfer learning. In *Proceedings of the 32nd International Conference on Tools with Artificial Intelligence*, 2020.

[8] Trent Kriete and David C. Noelle. Generalisation benefits of output gating in a model of prefrontal cortex. *Connection Science*, 23(2):119–129, 2011. doi: 10.1080/09540091.2011.569881.

[9] Trenton Kriete, David C. Noelle, Jonathan D. Cohen, and Randall C. O'Reilly. Indirection and symbol-like processing in the prefrontal cortex and basal ganglia. 110(41): 16390–16395, 2013. ISSN 0027-8424. doi: 10.1073/pnas.1303547110.

[10] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[11] Yael Niv, Reka Daniel, Andra Geana, Samuel J. Gershman, Yuan Chang Leong, Angela Radulescu, and Robert C. Wilson. Reinforcement learning in multidimensional environments relies on attention mechanisms. 35(21):8145–8157, 2015. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.2978-14.2015.

[12] Randall C. O'Reilly. Biologically based computational models of high-level cognition. 314(5796):91–94, 2006. ISSN 0036-8075. doi: 10.1126/science.1127242.

[13] Randall C. O'Reilly and Michael J. Frank. Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. 18(2), 2006. ISSN 0899-7667.

[14] Randall C O'Reilly, David C Noelle, Todd S Braver, and Jonathan D Cohen. Prefrontal cortex and dynamic categorization tasks: representational organization and neuromodulatory control. *Cerebral cortex (New York, N.Y. : 1991)*, 12(3):246—257, March 2002. ISSN 1047-3211. doi: 10.1093/cercor/12.3.246.

[15] J. L. Phillips and D. C. Noelle. A biologically inspired working memory framework for robots. In *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, pages 599–604, 2005.

[16] J. L. Phillips and D. C. Noelle. Working memory for robots: Inspirations from computational neuroscience. In *Proceedings of the 5th International Conference on Development and Learning*, 01 2006.

[17] T. A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641, 1995.

[18] Nicolas P. Rougier, David C. Noelle, Todd S. Braver, Jonathan D. Cohen, and Randall C. O'Reilly. Prefrontal cortex and flexible cognitive control: Rules without symbols. 102 (20):7338–7343, 2005. ISSN 0027-8424. doi: 10.1073/pnas.0502455102.

[19] W. Schultz, P. Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, Mar 1997.

[20] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3): 279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698.

[21] A. S. Williams and J. L. Phillips. Multilayer context reasoning in a neurobiologically inspired working memory model for cognitive robots. In *Proceedings of the 40th Annual Meeting of the Cognitive Science Society*, 2018.

[22] A. S. Williams and J. L. Phillips. Transfer reinforcement learning using output-gated working memory. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence, New York, NY.*, 2020.