Reinforcement Learning with Constant Size Frequency Encodings

By

Jackson Goble

A thesis submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

Middle Tennessee State University

Nov 2021

Thesis Committee:

Dr. Joshua Phillips

Dr. Zhijiang Dong

Dr. Cen Li

ACKNOWLEDGEMENTS

I want to thank Dr. Joshua Phillips for his endless support, guidance, and insight throughout the creation of this manuscript. I lost my way a few times, and he, along with my family and close friends, offered me valuable advice both academically and in my journey through life. Sometimes I follow the advice I am given, often I do not, but I take all of it to heart. This manuscript was written during some strange times, and I initially thought I was taking it all in stride only to realize over time that the effects were building up slowly and affecting me deeply all along. So, more than anything I was to thank everyone who has been there and supported me emotionally, you all know who you are.

ABSTRACT

In Reinforcement Learning, Markov Decision Processes (MDPs) enable agents to learn complex behavior by following simple algorithms and receiving sparse feedback from the environment. MDPs have a drawback, which is that due to their sequential nature, they lock an agent into operating at a particular time scale. Environments may then have signals that they can only express across a different time scale requiring the agent to have some mechanism, such as an episodic memory, to extract this information over multiple steps of an MDP. We humans do this easily, and it is believed that the hippocampus in our brains and those of living things is responsible for managing such information. In this work we propose and analyze a method to create a constant-length episodic memory trace we call a Holographic Frequency Trace (HFT) that can be calculated and used in real time during Reinforcement Learning processes.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I.

INTRODUCTION

Trying and failing is an important part of any learning process. The closer we get to success, the more meaningful (and sometimes difficult) each failure becomes. And often, as we get closer to succeeding, the attempts we make become more and more precise. Each attempt we make is more like the last, but we know it is also inherently distinct from every previous attempt, otherwise we would have no reason to try. We, as humans, have a clear sense of *now* and we know that it is different from every other moment we have ever experienced, even if it feels like we have done something a thousand times before. In this paper, we introduce a similar sense of novelty into a machine learning environment by folding up the history of a sequence of steps into a constant-length trace that is very sensitive to small changes in the past.

We take distributed representations of states in an environment and rotate them in the frequency domain in such a way that when they are used in the AND-like vector operation of circular convolution, they have a reduced impact on the result. By repeating this at each step throughout a machine learning process we maintain a fading trace of past history which can be used by an agent to treat separate but highly-similar series of events as distinct. It has been proposed [2] that the hippocampal region of the brain may perform similar functions of creating context-dependent memories and/or acting as a novelty detector.

To formalize the learning process, we use Reinforcement Leaning. In the field of Reinforcement Learning, Markov Decision Processes are a powerful tool that allow agents to learn complex behavior in a large variety of situations [7]. These processes are sequential in nature, and force the agent following the process to be restricted to a specific time scale. By being forced into a specific time scale, however, any information that is transmitted over a different time scale must either be encoded or lost. In our daily lives we seem to be able

to process information at multiple time scales, which would require processing multiple different MDPs simultaneously. To overcome this limitation, in [7], Sutton proposed using an MDP operating at a very small time scale, and enabling an agent to plan at a higher level incorporating a varying number of these small time steps [7]. In this paper, we explore a different and novel method to take a non-Markovian process and encode the history of an episode in such a way as to make it appear Markovian to the agent.

CHAPTER II.

BACKGROUND

Neural Networks

An artificial neural network is a type of function approximator that is inspired by the highly parallel activity of neurons in the nervous systems of living things [3, 7]. Given some input data, it performs some nonlinear transformation on it to yield a meaningful output. This nonlinear transformation is performed in a series of steps through neural layers, each layer being a linear transformation followed by some nonlinear activation function [1]. A large portion of modern machine learning research is dedicated to investigating how to create artificial neural networks for different applications, but the general purpose remains constant: ”Given some knowledge we have about the world X, tell us Y.” The more relevant and informative the input X is, the easier it should be for the network to make a claim about Y. For example, if someone was told ”the bouncing ball is red,” and then was asked what color the bouncing ball is, they should feel pretty confident that the correct answer is ”red.” But if instead they were asked what color blimp the ball was dropped from, they could only guess. The provided information is simply not rich enough. Similarly, when given a hundred details about a situation with only one or two that are relevant, one has to sort through all of the unnecessary distractions in order to give the right answer. A neural network, too, can suffer from the lack of relevant information or excess of unnecessary information [5], and as such we want the input for a neural network to be as relevant as possible.

A neural network learns how to mimic a function through an iterative training process. The process is useful because it does not require information about the function itself past knowing the inputs and outputs. Specifically, a neural network can learn to approximate a function from sample data alone, which is often either in abundance or can be generated easily.

The learning process proceeds as follows. The network is provided with some input X and produces some predicted output Y_{pred}. This output is compared to the known (from the sample data) actual value of the function Y_{real} for that input. This produces an error value which is used to slightly update the parameters of the network, a process called back-propagation, in an attempt to minimizes the error over time. The idea is that by minimizing the error, it becomes less *wrong* and therefore more *right.* In order to train, and hence create, an artificial neural network there must be a function to approximate and some error that represents how far the network's predictions are from this function. Because neural networks use nonlinear transformations that map from one finite-dimensional space to another, the input and output sizes for a neural network are of fixed length. Care must then be taken when working with neural networks that all inputs generated must be of the exact same size.

Reinforcement Learning

In Reinforcement Learning, an agent, often incorporating a neural network, makes decisions at discrete time steps about the action it should take given the current state of its environment [7]. The environment is then updated based on the action that was taken, if any. Changes to the environment can be deterministic, stochastic, or a mix of both. Perhaps the agent moved and is in a new position, or something came into the agent's view, or some other random event occurred. In most cases, much like the world we live in, the agent does not know everything about its environment and must learn about it over time through experiencing small pieces of it. Reinforcement Learning enables learning in such environments by allowing the agent to make predictions at each time step about the reward it expects to receive for each action available to it [7]. The agent can then take the action that it expects will maximize its long-term future rewards, and compare it to the reward it actually receives. The difference between this actual reward and the expected reward is an error value which is something that a neural network can try to minimize [7].

When calculating these predicted long-term rewards over a possibly infinite number of steps, it is necessary to guarantee that the value of a state is not an infinite sum, because assigning values to states can be viewed as a function that maps states to values. If the value of a state is infinite it is then impossible to make a meaningful comparison between it and another state. To solve this, we apply a discount factor gamma, $\gamma = \{x \,|\, x \in \mathbb{R}, 0.0 \leq x < 1.0\}$, to the reward at each time step [7]. This has the added benefit of making a reward received sooner appear more valuable than a the same reward received later. A value of $\gamma = 0.0$ would mean that an agent is only concerned with rewards it receives at the next time step, and a value of $\gamma = 1.0$ would mean that an agent is concerned with all rewards it receives in the future without care for how long it takes, which again, is an undefined value that a function cannot produce and cannot be meaningfully compared. [7]. These functions give rise to different algorithms for determining which state an agent should try to move to based on the value of the states. The algorithm that the agent follows throughout a Reinforcement Learning process is called the policy π, and the policy that the agent is trying to learn is called the optimal policy.

Markov Decision Processes

An important concept in Reinforcement Learning is the Markov Decision Process (MDP). Figure 1 shows the flow of information in an MDP as an agent sequentially takes actions and receives feedback from the environment which it then uses to choose a new action at the next time step. The process repeats until the agent arrives in a special terminating state that ends the episode. In an MDP, when an agent interacts with its environment it moves between states that are independent of all previous states. This way, at any time t the state S_{t+1} that results from the agent taking an action A_t depends only on the action taken and any stochastic processes of the environment, not on previous states. An environment that has this quality is said to have the Markov Property [7].

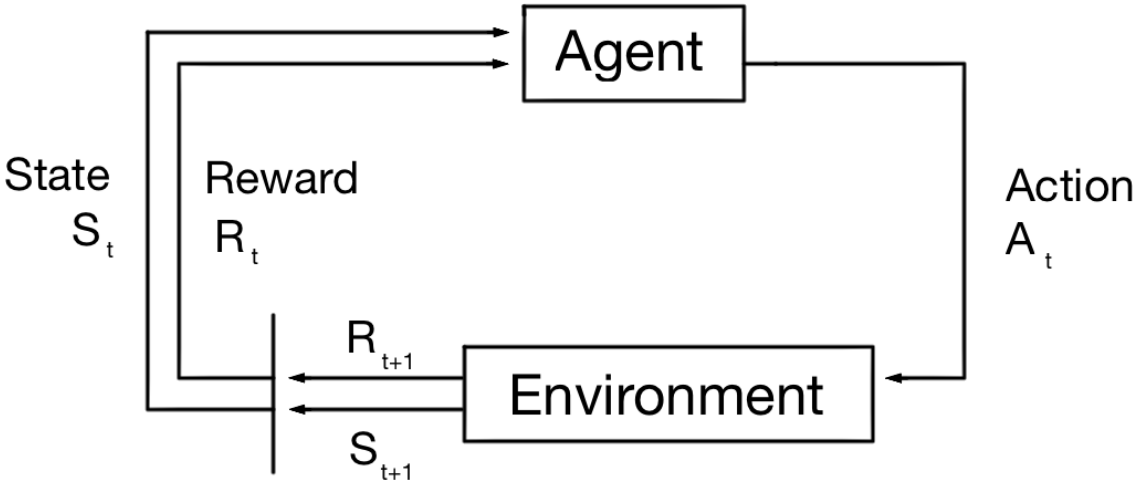

Figure 1: A Markov Decision Process where the agent interacts with it's environment by taking actions and then receiving a new state and reward at each time step.

In the real world, our environment is not a true MDP because we experience the world in continuously and do not move between states that are independent from all previous states. Another reason is that our world is partially observable, meaning we do not see everything at any given time, and so the things we experience can be due to past events we have yet to see the effects of. That is, the relevant information from the environment might only be deduced from past states, and is therefore not Markovian.

Q-Learning

Q-Learning is an *offline* Reinforcement Learning algorithm, meaning the policy function π that the agent tries to approximate (i.e. the optimal behavior in an environment) is not the same as the one that it actually follows during training. Often, (and as is the case in this work) the policy that the agent follows is an ε-*greedy* policy that allows for a small chance $\varepsilon = \{x \,|\, x \in \mathbb{R}, 0.0 \leq x \leq 1.0\}$ of making a random exploratory step instead of greedily choosing the action it thinks is best at each time step. This is useful because without any

form of exploration, the agent will learn a likely-suboptimal policy and treat it as ”good enough” with the possibility of never exploring better alternatives.

In Q-Learning, the agent learns a function q, often called the *Q-function*, which is a function of both the state the agent is in and the action being considered. This Q-function returns the agent's perceived value of taking a certain action a in a certain state s if it were to follow a certain policy π from then on. So, $q_\pi(s,a)$ is ”the value of taking action a in state s under policy π” [7].

Eligibility Traces

An eligibility trace is a measure calculated during each step of training that assigns past states a fading amount of credit for the reward currently received. The inspiration for this is that if state S_A led to state S_B led to a reward at state S_C, all three states are partially responsible for the reward. State S_C more so than S_B which again more so than S_A. An eligibility trace can be calculated by storing the trace (which starts as a vector of zeros) from the previous time step, multiplying it by some value lambda, $\lambda = \{x \,|\, x \in \mathbb{R}, 0.0 \leq x \leq 1.0\}$, and then adding the representation of the current state, as shown in Figure 2. Because λ is applied at every time step, past states receive exponentially decreasing amounts of credit for the current reward. These resulting representations, shown on the right side of each box in Figure 2, are used as the input vectors during training of a neural network, so that multiple past states are given credit when the weights are updated. These representations are OR-like because, due to addition, either constituent representation can be partially seen in the output, as opposed to an AND-like representation in which the output would represent the unique case in which both constituent representations are present. This method requires constant space and constant time at each time step to calculate an arbitrarily long trace.

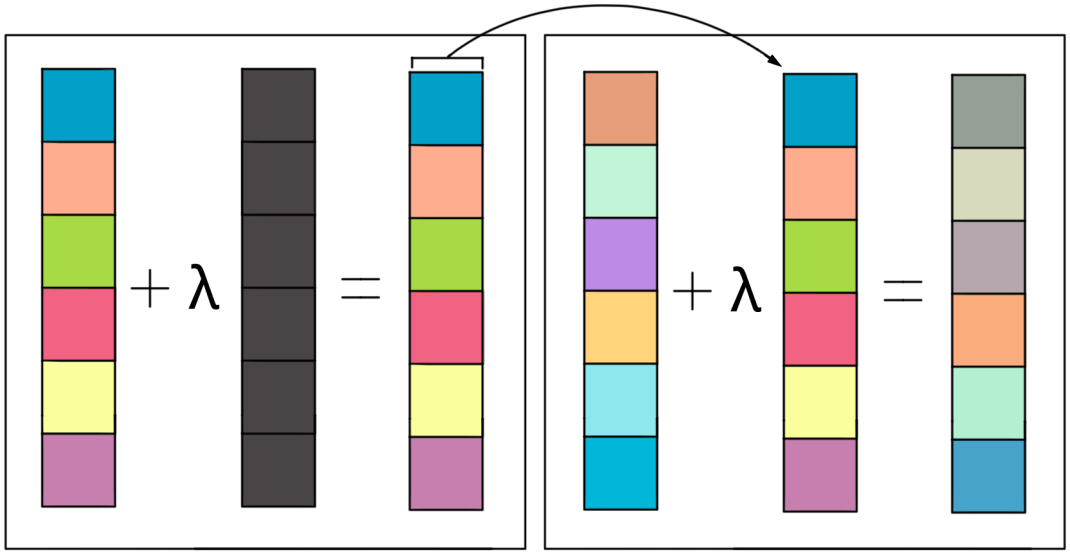

Figure 2: Visual representation of an eligibility trace. The trace starts as the zero vector (left), is multiplied by some small value λ and added to the representation of the current state. This trace is then used on the next time step (right), again being reduced and added to the representation of the new state, yielding a new vector

Distributed Representation

When working with neural networks, the inputs are represented as a vector where each value in the vector corresponds to a *feature* of that input. This is useful because it allows us to use a fixed number of features to describe an arbitrarily large number of things by saying how much of each feature that thing has. For example, if something is spherical, bouncy, and black and white, there is a good chance that it is a soccer ball. If we have even 100 well-selected features, we can describe far more than 100 different objects. An alternative method for representing different concepts as vectors, which is often used for categorization tasks, is a *one-hot* encoding, in which the input vector is filled with 0s except for a single element which has the value 1. In this way, a vector of length 10 can represent up to 10 different things. Encoding information in this way is useful when there is no overlap between features because each of these vectors is orthogonal to each other, which allows a

neural network to treat them as completely different entities.

Orthogonality and Dot Products

In the case of one-hot encoding it is easy to see how a neural network will be able to treat two different vectors differently because the network will use an entirely different set of weights at the first layer. It is important though that it's the *orthogonality* of these vectors that is actually responsible for this ability. In the remainder of this paper, the term orthogonality will not be used to refer to whether two vectors are perfectly orthogonal (i.e have a dot product of 0), but instead to refer to how close two vectors are to being orthogonal (e.g. two vectors with a very low dot product are highly orthogonal).

In relation to an MDP a highly orthogonal encoding is useful to represent different states because each state in an MDP does not depend on any other state, and should be viewed as a completely different entity.

Holographic Reduced Representations

One-hot encoding has some undesirable properties, one of which is that in order to represent N things, the length of the vector must be at least N. This can cause problems, for example, if it is ever necessary to represent a variable number of things the vectors and network must be designed to overestimate that number, which is also wasteful of all the unused weights. Further, if it is ever necessary to represent *combinations* of features in an AND-like way, which is an integral part of this research, the number of necessary one-hot encodings explodes multiplicatively, as does their length.

A better method of encoding combinations of information into a fixed-length array is to use Holographic Reduced Representations (HRRs) [6]. HRRs are highly orthogonal vectors of very small values centered around zero that are distributed in such a way that performing

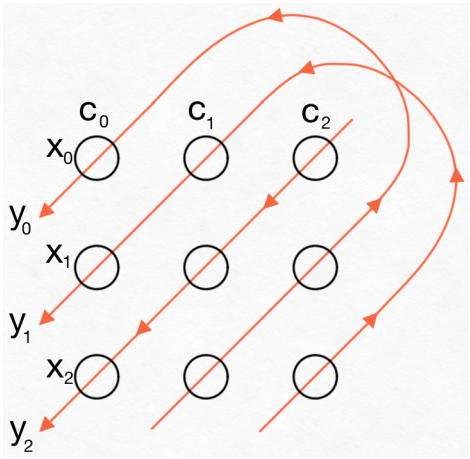

Figure 3: Circular convolution as a compressed outer product of two vectors $\vec{c}$ and $\vec{x}$ of length 3. The operation yields a new vector $\vec{y}$ that is also length 3. When $\vec{c}$ and $\vec{x}$ are HRRs, $\vec{y}$ will also be an HRR.

a circular convolution of two HRRs yields a new HRR that has a near-zero dot product with either of its constituent HRRs. Figure 3 shows how a circular convolution is calculated. It is effectively a compressed outer-product of two vectors in which each element of the outer product contributes equally to the result. The process shown in Figure 3 is an expensive operation, but it can be computed efficiently in *O(nlogn)* using a fast Fourier transformation. Because the resulting composite HRR is also an HRR, this process can be repeated with any number of HRRs, each one yielding a new HRR that will be highly orthogonal to the others. Further, by normalizing the base HRRs to have a unit length, we can guarantee that the repeated convolution will also yield unit-length HRRs and thus, no HRRs will dominate the others and unfairly drive learning.

With respect to circular convolution of HRRs, there is a notion of a convolutive power which is the result of performing circular convolution between an HRR and itself. Raising an HRR to the convolutive power of 2, we would effectively *square* the HRR, as seen in Figure 4. By raising it to the convolutive power of 1, it remains unchanged, and by raising it

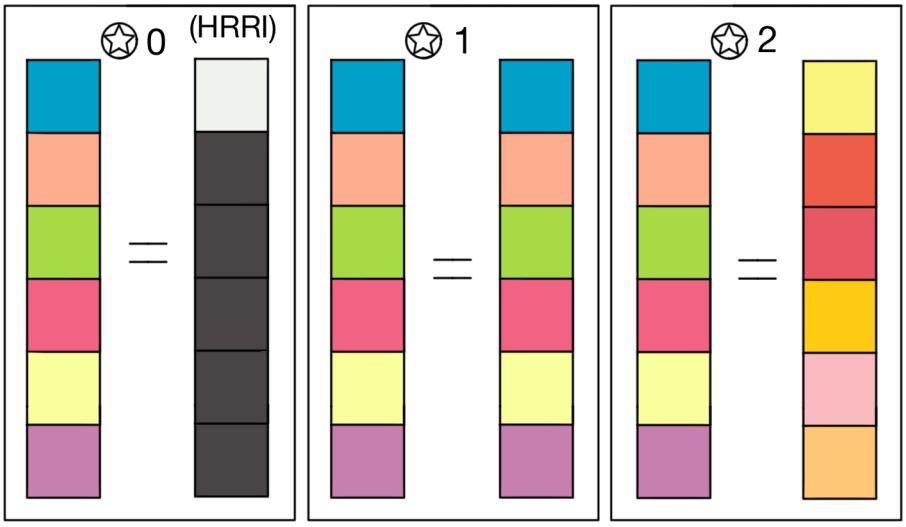

Figure 4: The colors represent small random values distributed around 0. An HRR raised to the convolutive power 0 yields the identity HRR (left), an HRR raised to the convolutive power 1 yields itself (middle), and an HRR raised to the convolutive power 2 yields a new HRR that is highly orthogonal to the original HRR (right).

to the convolutive power of 0, it becomes the identity HRR, which is a vector with 1 as the first element, and then 0 for all the rest. The identity HRR has the special property that any HRR convolved with the identity HRR yields the original HRR, unchanged.

Episodic Memory

Episodic memories are the kind of memories that store a particular past event or episode. If you can remember two different times you went to the same place and did the same thing, those are two separate episodic memories that, despite being very similar, you can recall distinctly given the proper prompting. Another example of episodic memory is remembering where you parked your car in the morning. There is a lot of overlapping information from parking each day, especially if you park in the same parking lot at the same time of day, and yet we have the ability to ignore the past memories based on small differences. Research

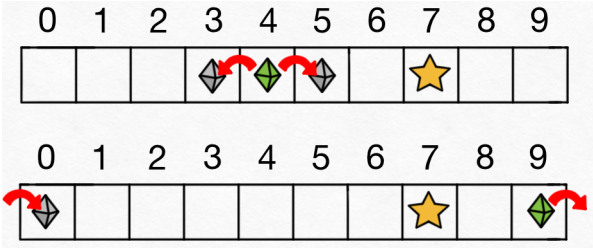

Figure 5: Two states of a linear maze of length 10. The green marker is the current location of the agent, and the gold star is the goal state. The gray markers show the the resulting location if the agent were to take the action indicated by the arrow.

has shown that the hippocampus is highly involved in the creation and retrieval of episodic memories and is also closely tied with spatial reasoning [2], of which we use both concepts in this work. We use episodic memory in the context of training models over many episodes, and spatial reasoning within the context of Reinforcement Learning inside a linear maze.

Linear Maze Problem

A useful example of a Reinforcement Learning task is the linear maze problem, shown in Figure 5, in which an agent is placed at a random location in an array, and has to walk to the goal location, by moving either left or right at every time step. The maze is cyclic, such that if the agent moves off the end of the array it reappears at the position on the opposite end. In the simplest version of this task there is a single goal location that does not change across episodes, and the function is considered learned when the agent takes the shortest path to the goal from any starting state. This base task has been used, along with several variants, in other research [8, 4].

Limitations of Markov Decision Processes

As previously outlined, MDPs have properties that are useful in a Reinforcement Learning environment, but they have a drawback that in many situations, like the real world, information is received at different rates and so it becomes unclear where one state should end and another begin. In a task that requires the agent to make many frequent actions, it could easily be the case that the environment can only provide a certain piece of important information at a slower rate, spanning many time steps. One way to approach this problem would be to keep an infinite record of past events, but this would be costly in terms of space complexity. It would be useful to have a different way to encode past information that is not so space-reliant.

CHAPTER III.

METHODS

Eligibility Traces in the Frequency Domain

If we take inspiration from the concept of an eligibility trace, which due to using vector addition creates a *fading OR-like representation*, and its ability to fold up past information using constant space, perhaps we can find a similar method that provides a *fading AND-like representation* of past states. Figure 6 shows the analogy between OR-like operations and AND-like operations in different domains, and should illustrate how deeply important both types of operations are in each domain. By having an AND-like representation of the past, we would have a way to treat a moment in time completely differently depending on the current state *AND* the past sequence of events.

If we use HRRs for our state and action representations, we can use a convolutive power zeta between 0 and 1 to decay an HRR towards the identity HRR. This can be seen as an AND-like decay (towards the multiplicative identity 1) in the same way that multiplication by λ is an OR-like decay in an eligibility trace (towards the additive identity 0). This novel combination of ideas has not been explored in the literature. The full process can be seen in Figure 7. State information (which in the case of the flashing light maze problem is the convolved position and light HRRs) is convolved with the decayed trace, which creates a new trace for the next time step. The trace is then convolved with the HRR representing the action being considered, creating the input HRR for the neural network. In this paper, we explore it's potential use as a method to encode AND-like episodic memories with constant space complexity.

We call this eligibility trace that holds a fading representation in the frequency domain a Holographic Frequency Trace (HFT). To compute the convolutive power of an HRR representing a state, we calculate the fast Fourier transform of it, rotate it by some ratio, zeta,

Domain	OR-like Operations	AND-like Operations
Binary	OR	AND
Scalar	+	×
HRR	Vector Addition +	Circular Convolution ⊛

Figure 6: The binary notions of AND and OR have different but equivalent operations in different domains. For instance, the bit-wise operation of OR is equivalent to the scalar operation of addition, while the bit-wise operation AND is equivalent to the scalar operation of multiplication. If we extend these notions of OR and AND into the domain of HRRs, we get vector addition and circular convolution respectively.

towards 1 in the frequency domain, so that it decays to the identity HRR and then convert it back to the time domain. Circular convolution is then performed on this decayed HRR and the HRR of the new state. Doing this gives us a deterministic way to encode a series of steps into a single vector that will be orthogonal to other similar history vectors given a large enough HRR length and a value of zeta that is not too close to 0.

The first experiment we perform is a simple test of viability in a linear maze. We train models on a simple linear maze of length 20, with a single goal position. We test both one-hot encoded vectors and HRRs, and we test them with both eligibility traces calculated in the time domain and eligibility traces calculated in the frequency domain to confirm that an agent can make use of the information in each of these simple cases. If this appears viable we want to further explore the qualities of these frequency domain traces.

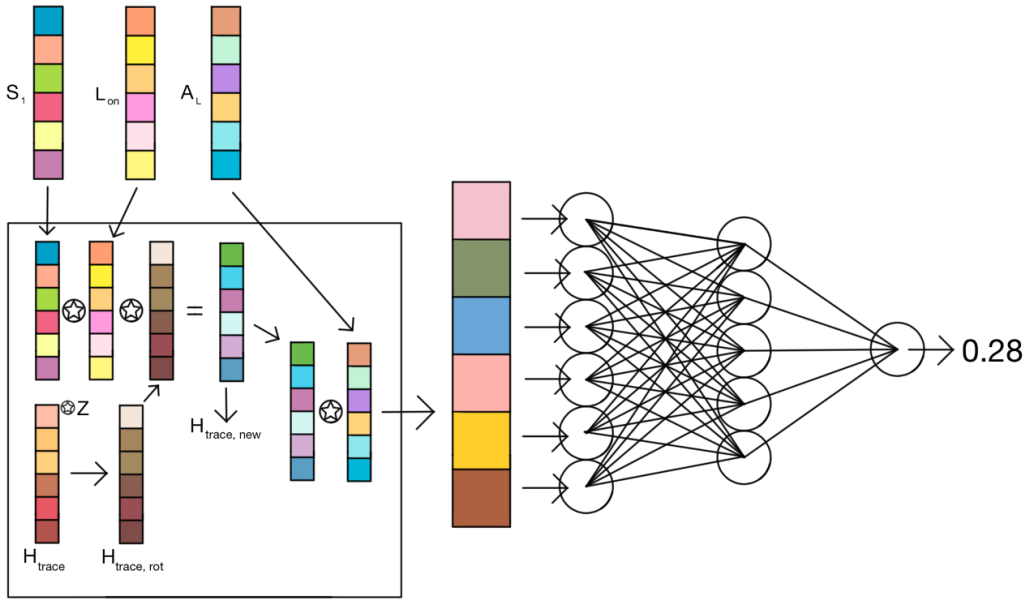

Figure 7: Full HFT encoder. State information (convolved position and light HRRs) is convolved with the decayed trace, creating the new trace for the next time step. That trace is then convolved with the HRR representing the action being considered, creating the input HRR for the neural network, which has an AND-like, fading episodic memory of past states.

Flashing Light Maze Problem

This research is interested in exploring reinforcement learning tasks in partially observable environments in which the state seen by the agent at any given time step is insufficient to determine the optimal behavior. Said another way, we are interested in tasks which do not have the Markov Property so that we can test whether encoding episodic memories in the form of HFTs is robust whether or not the Markov assumption holds.

This work introduces a modified version of the linear maze problem that satisfies this constraint, called the flashing light maze problem, in which the frequency of a flashing light is what indicates the goal location within the linear maze. The light is represented by an HRR, and is either on or off at every time step. In this problem, there is not sufficient information at any single moment in time for the agent to determine the location of the goal state. Past information about the episode, specifically the frequency at which the light is

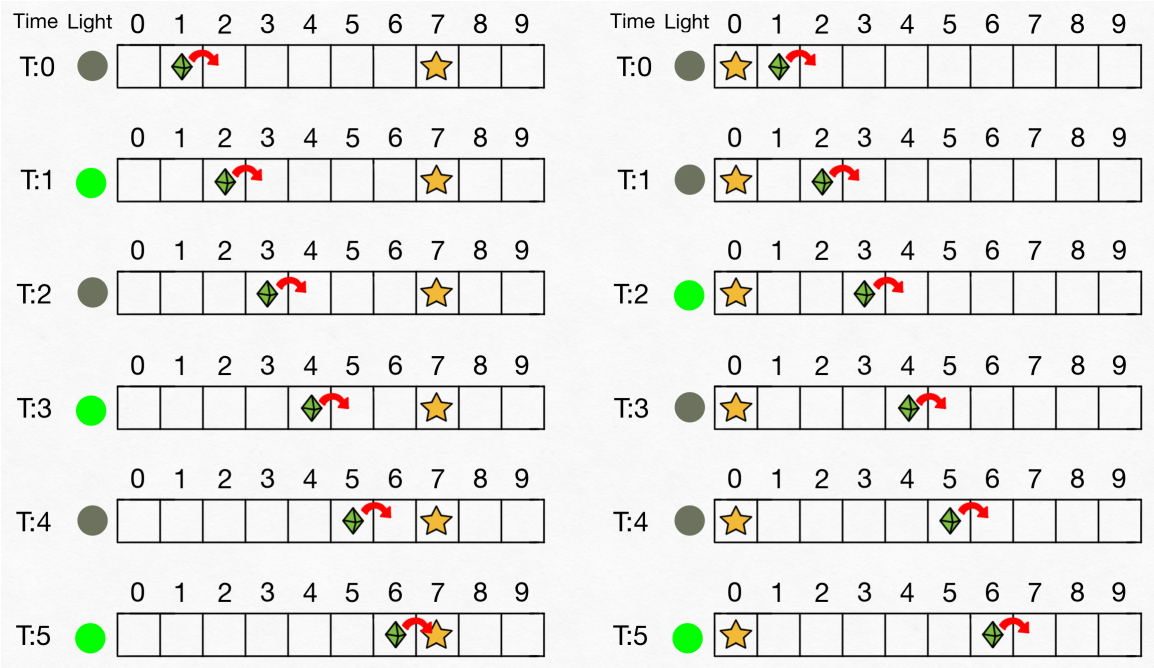

Figure 8: Two examples of the agent as it traverses the maze in the flashing light task. Left: the light blinks with a period of 2, indicating to the agent that the goal is be in state 7. Right: the light blinks with a period of 3, indicating that the goal is in state 0. The behavior of the agent in neither episode is optimal.

flashing, must be incorporated into the encoding for the agent to solve the problem. In Figure 8 we follow an agent over two episodes (left and right) attempting to solve the flashing light maze. At each time step, the agent sees the state it is in and the current state of the light, and makes a decision about which action to take (L/R). This example illustrates the non-Markovian nature of the task. We can see that at T_0, T_4 and T_5 of both episodes, the environment looks identical to the agent. At T_0 of both episodes, the agent is in position P_1 and the light is off. Because it is the start of the episode and there is no past information to use, the agent cannot know where the goal is located. Conversely, at T_4 of both episodes, the agent is in position P_5 and the light is off. Even though the agent receives the exact same information from the environment in both situations, if the agent has saved past information, it can differentiate where the goal is located.

Encoders

Different methods for encoding past state information are explored in this work, and they satisfy the following constraints. Firstly, the encodings generated should be constant-size HRRs, so that they retain their useful qualities. Secondly, the encoding process should be biologically plausible so we may be able to glean insight into how living things achieve similar tasks, though a direct comparison between the methods and actual biological processes is outside the scope of this work. Thirdly, they should be calculable in real time, in an on-line learning environment. And lastly, the encoding methods should be calculation driven not storage driven, they should aim to minimize the space required to generate the encodings and rely on computation over explicit storage for scalability.

The 5 encoders tested can be seen in Algorithms 1 to 5. In Algorithm 1, we see the basic HFT encoder. The trace it stores folds in all state information, specifically the light and position, at each time step. This is a flexible encoder because it treats all input from the

environment identically. It does not require additional information about the domain of the problem in order to be used. This encoder as well as those outlined in Algorithms 2 and 3 use the *zeta* parameter, Z, explained previously, as a means to decay the trace over time towards the identity HRR.

In Algorithm 2 the basic HFT encoder is adapted by using the knowledge that the frequency of the light is what indicates the location of the goal. This encoder keeps only the light information in its trace, not the position information. This causes fewer possible trace values which may make it more useful because the input space used by the agent may provide richer information.

For Algorithm 3 not only all state information is folded into the trace but also the HRRs that represent the actions taken. This additional information could be useful because it makes arriving in a state from one direction appear completely different than arriving from the other direction. If the agent moves continuously in the same direction, the action HRR is effectively convolved in multiple times and a useful pattern could emerge that is useful to the agent.

Algorithm 4 has its own internal HRR representation for each task. It uses a naive approach of simply convolving state and action information together without maintaining a trace. Once the light has been seen by the agent twice, it switches modes and additionally convolves the task representation into its output. This encoder should perform as best as we can expect out of the other encoders but because it requires significant outside knowledge, we use it as a metric of comparison for the others, it is not of much interest itself.

The approach in Algorithm 5 always naively returns the convolved position action and light HRRs. This encoder should perform the worst of all encoders because it does not create an episodic memory and cannot provide the information necessary to solve the problem.

Each encoder is analyzed in two primary ways. First, the qualities of the encodings produced, such as the dot-product similarity, are analyzed with no actual training being performed. Second the encodings are empirically tested in a Reinforcement Learning environment by using them as the input during training of agents in the flashing light maze task.

Algorithm 1 HFT Encoder

Initialization:

$H_{trace_{j-1}} \leftarrow H_I$ ▷ Trace starts as the identity HRR

$Z \leftarrow \{x \,|\, x \in \mathbb{R}, 0 \leq x \leq 1\}$

Every Step:

$H_{pos}, H_{light}, H_{act} \leftarrow Environment$ ▷ HRRs for state at time T_j

$\theta_{j-1} \leftarrow \text{ANGLE}(\text{FFT}(H_{trace_{j-1}}))$ ▷ Angle in frequency domain

$H_{trace_{j-1},rot} \leftarrow \text{REAL}(\text{IFFT}(\text{EXP}(i * \theta_{j-1} * Z)))$ ▷ Trace after zeta decay

$H_{trace_j} \leftarrow H_{pos} \circledast H_{light} \circledast H_{trace_{j-1},rot}$

return $H_{act} \circledast H_{trace_j}$

Algorithm 2 HFT Encoder (Light Only)

Initialization:

$H_{trace_{j-1}} \leftarrow H_I$ ▷ Trace starts as the identity HRR

$Z \leftarrow \{x \,|\, x \in \mathbb{R}, 0 \leq x \leq 1\}$

Every Step:

$H_{pos}, H_{light}, H_{act} \leftarrow Environment$ ▷ HRRs for state at time T_j

$\theta_{j-1} \leftarrow \text{ANGLE}(\text{FFT}(H_{trace_{j-1}}))$ ▷ Angle in frequency domain

$H_{trace_{j-1},rot} \leftarrow \text{REAL}(\text{IFFT}(\text{EXP}(i * \theta_{j-1} * Z)))$ ▷ Trace after zeta decay

$H_{trace_j} \leftarrow H_{light} \circledast H_{trace_{j-1},rot}$

return $H_{pos} \circledast H_{act} \circledast H_{trace_j}$

Encoder Dot-Product Similarity

In order to evaluate the effectiveness of an encoder outside of empirical trials its performance is tested under two situations. The first is when the agent moves at random, which is

Algorithm 3 HFT Encoder (Action Included)

Initialization:

$H_{trace_{j-1}} \leftarrow H_I$ ▷ Trace starts as the identity HRR

$Z \leftarrow \{x \,|\, x \in \mathbb{R}, 0 \leq x \leq 1\}$

Every Step:

$H_{pos}, H_{light}, H_{act} \leftarrow Environment$ ▷ HRRs for state at time T_j

$\theta_{j-1} \leftarrow \text{ANGLE}(\text{FFT}(H_{trace_{j-1}}))$ ▷ Angle in frequency domain

$H_{trace_{j-1},rot} \leftarrow \text{REAL}(\text{IFFT}(\text{EXP}(i * \theta_{j-1} * Z)))$ ▷ Trace after zeta decay

$H_{trace_j} \leftarrow H_{pos} \circledast H_{light} \circledast H_{act} \circledast H_{trace_{j-1},rot}$

return H_{trace_j}

Algorithm 4 Optimal Encoder

Prior to start of training

for each task **do**

$H_{task} \leftarrow new\,HRR$ ▷ HRR to represent each task

end for

Every Step:

$H_{pos}, H_{light}, H_{act} \leftarrow Environment$ ▷ HRRs for state at time T_j

if light has been seen two or more times **then**

return $H_{pos} \circledast H_{light} \circledast H_{act} \circledast H_{task}$

else

return $H_{pos} \circledast H_{light} \circledast H_{act}$

end if

Algorithm 5 Naive Encoder

Every Step:

$H_{pos}, H_{light}, H_{act} \leftarrow Environment$ ▷ HRRs for state at time T_j

return $H_{pos} \circledast H_{light} \circledast H_{act}$

the expected behavior of the agent at the beginning of training. The second situation is when the agent moves optimally, which is the expected behavior at the end of training. In both cases, the ideal outputs produced by an encoder would be orthogonal to each other based on the current state of the environment, the action the agent is considering, and the frequency the light is flashing at. To collect these encodings, the agent is simulated moving through multiple episodes of the flashing light maze.

For both the random behavior and optimal behavior, the output of the encoder is tracked for every position-action-task triplet over many episodes, until every position-action-task has been considered at least 100 times. The most recent 100 outputs are then taken from every triplet and their dot products are compared in three ways.

Firstly, the 100 outputs from every position-action in one task are compared to the 100 outputs from the same position-action in the second task. This information reveals if the agent is able to consider the same position-action differently for the two tasks. This is called the *inter-task* evaluation, and ideally, these dots products will be low. High dot products for this measurement would indicate an inability for the agent to contextualize based on task information, and therefore an inability to solve the task. This is because the optimal behavior may conflict for the same location in different tasks, and the network would need to map the same input to two different outputs.

Secondly, the outputs from every position-action in one task are compared to every other position-action in the same task. Ideally, this metric will also be low, because the position-actions within a specific task should not interfere with each other. This will be referred to as the *intra-task* evaluation. A high dot product here would indicate an inability for the agent to differentiate between locations. For example, being in location 4 would have almost the same representation as being in location 3 even though they should represent unrelated states.

Lastly, the 100 values of each state-action-task are compared with the other 99 from the

same position-action-task. This is the *intra-state* comparison, and the agent could benefit if these dot products are higher than the other metrics because the behavior of the agent for the same position-task should be the same. A high dot product here indicates that the agent perceives being in the exact same position (and solving the same task) very similarly, and a low dot product indicates that being in the exact same position (and solving the same task) appears novel. Even if this is low, the task can still be learned, but the agent will need to map multiple different inputs to the same area of the output space.

All of these dot products are collected for varying values of zeta for every encoder and then plotted as both a pdf, to show the distribution of the dot products as zeta is varied, and as a cdf to compare the relationship between the three *inter-task*, *intra-task*, and *intra-state*, metrics above.

Encoding for Q-Learning in a Flashing Light Maze

In order to evaluate how well the encoders enable learning in a non-Markovian environment, their performance is empirically tested in a flashing light maze. A task (i.e. a frequency for the light and corresponding goal state) and starting position in the maze are randomly chosen. Then, the Q-Learning algorithm is used in conjunction with the encoder being tested to train a neural network to solve the task. At every time step, the state, action, and light information is passed through an encoder to yield the input vector (an HRR) for the neural network to use to approximate the Q-function. We expect the different encoders to enable learning to varying degrees and at different rates, and the naive encoder and the optimal encoder are used as guides for the expected upper and lower estimates of performance.

In order to measure performance, the number of suboptimal steps made by the agent during each episode are tracked over the course of training. The number of suboptimal steps is calculated as simply the total number of steps made by the agent minus the optimal number of steps the agent could have made. This metric was chosen over some more sophisticated measurements that might account for the earliest moment at which the agent could possibly disambiguate which task it should perform, or for the fact that any single step in the wrong direction must be undone, causing a total of two suboptimal steps. The optimal number of steps is the minimum distance between the starting state and the goal state. The minimum is specified because the maze is cyclic and there are always two distances to the goal, one distance by moving only left and one by moving only right.

For each encoder multiple models are trained in this way. For each model the number of suboptimal steps per episode varies greatly because each episode is randomly initialized. To smooth this data, a sliding window of length 100 is used to find the average number of suboptimal steps localized around an episode number. Once this smoothed data for every model is generated, the confidence intervals of $\pm 1.96\sigma$ are calculated for the suboptimal steps of each encoder for each episode. This provides a graph that will show how well the encoders perform over time on average as well as whether or not the differences are significant.

The same test can be performed again to see the effects of tweaking different hyperparameters, specifically the zeta value for the HFTs or the HRR lengths for all encoders, on the learning speed and performance of the models. The zeta value of the HFT encoders are particularly interesting because it is a new hyperparameter introduced by the HFT algorithm. Because the zeta value represents the ratio of how much of the history is kept at every time step, it would be interesting to know which encoders are more robust to changes in zeta. In order to find this, the average performance of the models towards the end of training is

recorded for different values of zeta and it is observed which of the encoders performed better over large spans of zeta values.

Maze and Model Specifications

For the experiments conducted, unless otherwise noted, the following specifications and hyperparameters are used. For the maze, a flashing light maze of length 30 is used with two goal locations at 11 and 20 corresponding to the two light frequencies of once every 2 steps and once every 3 steps. An optimistic, mixed reward scheme is used with a reward of 1.0 for finding the goal state and a reward of -0.1 for every time step otherwise. This reward scheme for the pragmatic reason that it seemed to converge quickly in the simple linear maze task.

In the experiments neural networks with an input layer of length 1024 were used, corresponding to the length of the HRRs used, and a single hidden layer of size 1024. Biases were not allowed in the weight updates for fear that, while they may help jump-start convergence, they might not play well with the natural distributions of HRRs, as each value in an input vector is not an independent feature. The hyperbolic tangent activation function was used in the hidden layer to allow for nonlinear positive and negative values. The output layers had length 1 and used a linear activation function, a mean-squared-error loss function, and stochastic gradient descent with learning rate 0.01. For the Q learning algorithm $\varepsilon = 0.1$ and $\gamma = 0.5$ was used.

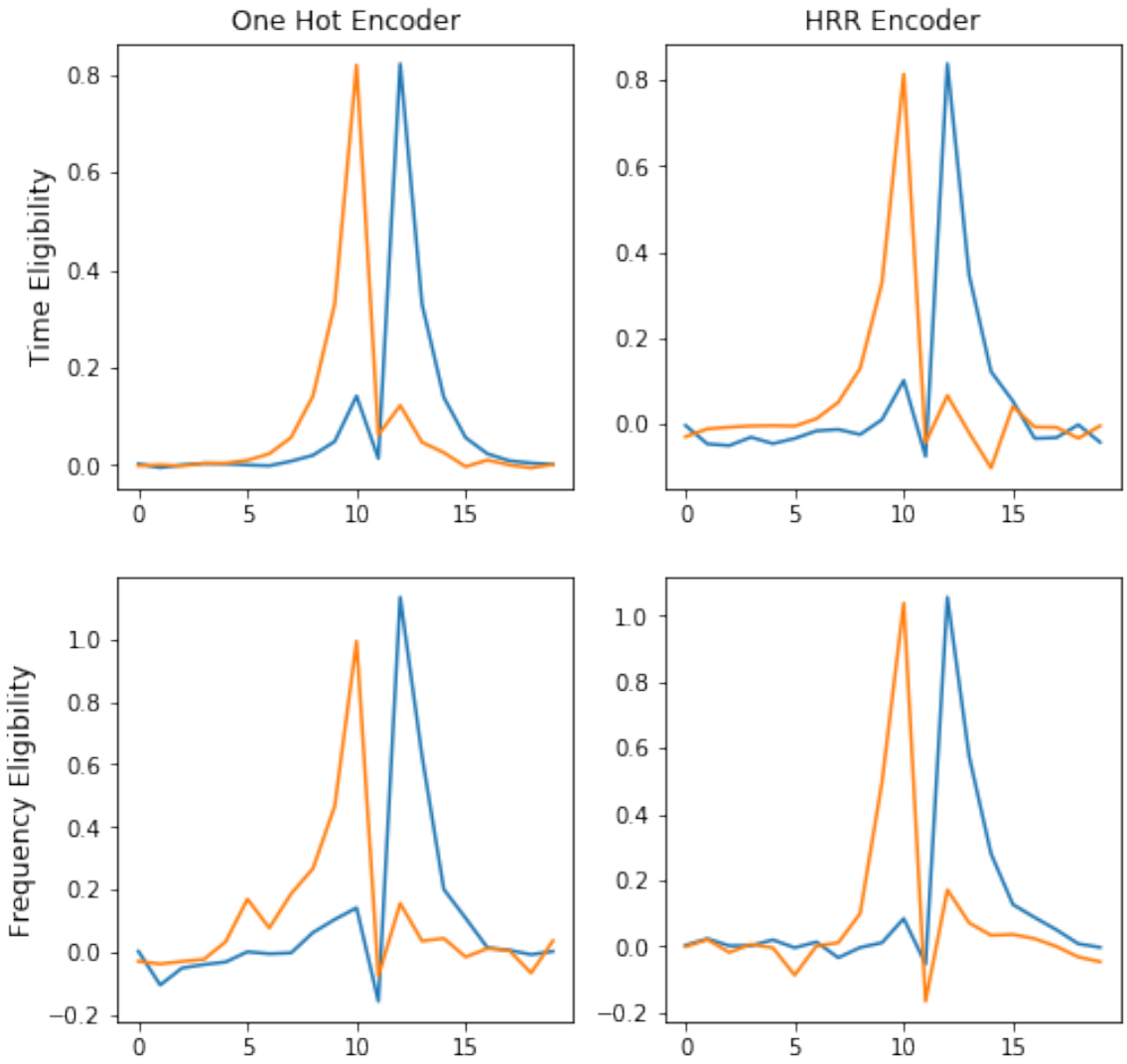

Figure 9: The behaviors learned by agents using eligibility traces calculated in the frequency and time domains and either HRRs or one-hot encodings. Agents were trained for 400 episodes in a maze of length 20 (x-axis), with a single goal at position 11. The reward scheme in this test was 1.0 for reaching the goal state and 0 for arriving in any other state. The orange line represents the agents perceived value of taking the ”right” action and the blue line represents the perceived value of taking the ”left” action.

CHAPTER IV.

RESULTS

We first test to see if the notion of an eligibility trace in the frequency domain, which again, we call a Holographic Frequency Trace (HFT), can be used by an agent to solve a simple linear maze. In Figure 9, an agent was trained in each of 4 situations, a regular time-domain eligibility trace with one-hot encoding or HRR encoding, and the novel HFT with one-hot or HRR encoding. We can see that all 4 models were easily able to learn the optimal behavior around the goal state, but struggled in states more distant from the actual reward. We can also see that the agent was able to create a smooth curve (which is closer to the actual Q-function) more quickly for the time-domain eligibility trace using one-hot encoding than the other methods.

Dot-Product Comparisons

Now that we've confirmed that using HFTs is a viable encoding scheme, we start to examine the qualities of the outputs of potential encoders. As previously stated, we would like for the encoders to generate representations that are unique for each state-action-task the agent will be considering. In Figures 10 and 11 we see the CDF for the distribution of dot products between different states, actions, and tasks. The light blue is the intra-task dot products, and it shows that the PDF has a normal distribution around 0.0 because of the shape of the CDF as it quickly reaches 1.0. This makes sense because as the HFT is calculated over the course of an episode, it will be nearly orthogonal to the representation at the previous time step due to the nature of circular convolution and the unlikelihood that any two random HRRs will be highly similar. The light green (inter-task) is the dot-product comparison of each state-action in one task to the same state-action in the other task. We can see that in the random movement simulation of Figure 10 that represents the beginning of training, the inter-task comparison hovers slightly above the intra-state comparisons (shows

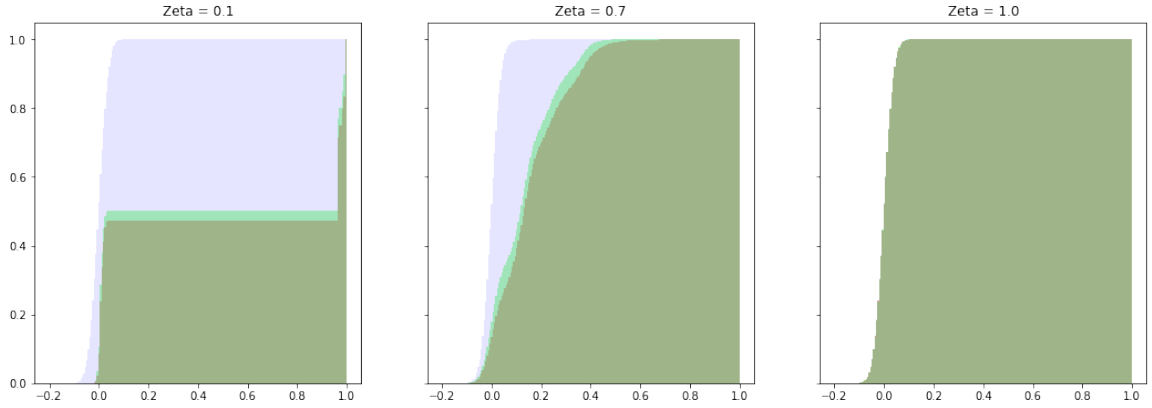

Figure 10: Cumulative density functions of the HFT encoder at 3 values of zeta during random movement throughout a flashing light maze. Random movement is the expected behavior of an agent towards the start of the training process. The light blue are the intra-task comparisons, the light green are the inter-task comparisons, and the red (appearing dark green because alpha values are applied) are the intra-state comparisons.

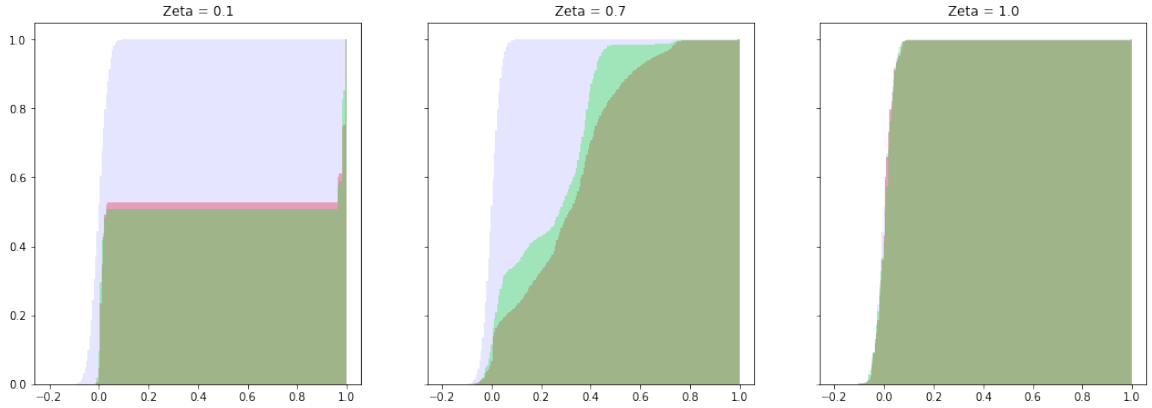

Figure 11: Cumulative density functions of the HFT encoder at 3 values of zeta during optimal movement throughout a flashing light maze. Optimal movement is the expected behavior of an agent toward the end of training, assuming it is able to learn the function. The light blue are the intra-task comparisons, the light green are the inter-task comparisons, and the red (appears dark green because alpha values are applied) are the intra-state comparisons.

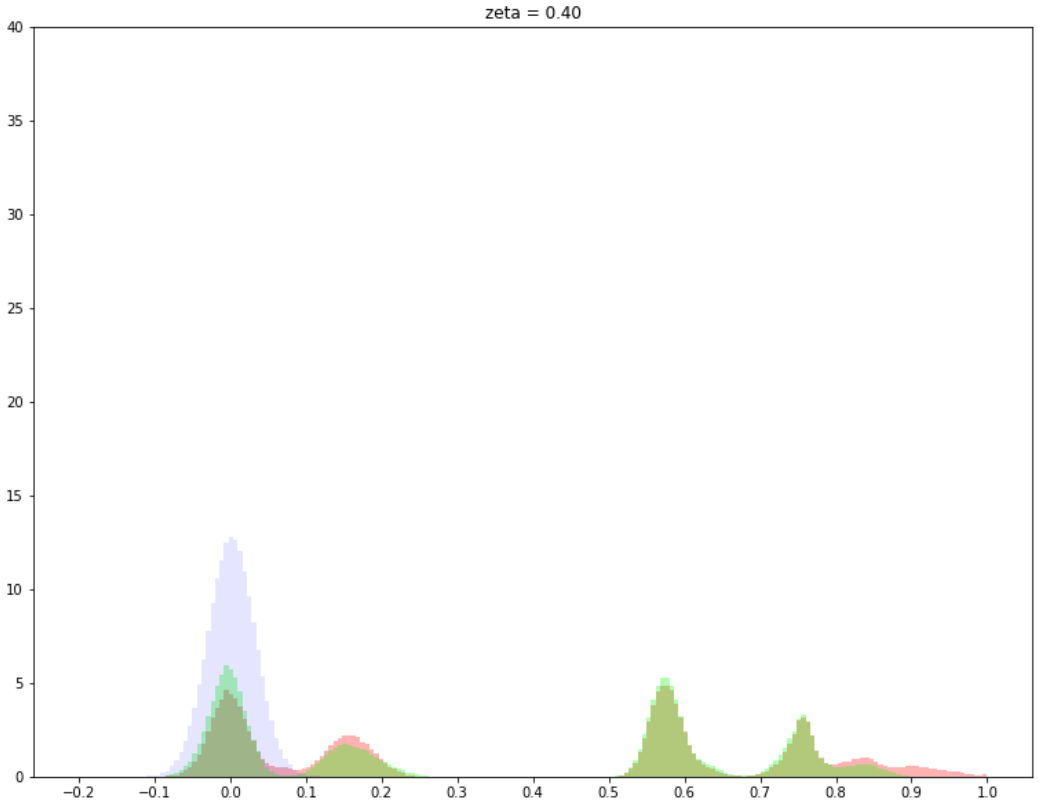

Figure 12: Multimodal PDF for the HFT Encoder (with actions). The light blue are the intra-task comparisons, the light green are the inter-task comparisons, and the red are the intra-state comparisons (dark green is overlapping inter-task and intra-state.)

in red, with alpha applied, so it appears dark green) and, for reasonable values of zeta, the gap increases for the optimal behavior simulation in Figure 11. This gap shows us that the dot products for the same state-action are generally lower between tasks than the same state-action in the same task. This tells us that with proper zeta values as the agent learns the correct behavior the current task becomes easier to distinguish, as well.

As zeta approaches 1.0 all PDFs approach the same normal distribution around 0.0 that the intra-task distribution has. This is because a zeta value of 1.0 corresponds to not decaying the trace and holding onto all past information at every time step. As zeta approaches 0.0 more

and more past information is thrown away, and the encoder behaves more and more like the naive encoder. A zeta value slightly higher than zero will throw away almost all information, but because a small amount of information is kept, many of the dot products are near 1.0 because the vectors are very close to the identity HRR. As zeta varies throughout the middle values, the inter-task and intra-state dot product distributions have varied multi-modal PDFs which can be seen clearly in Figure 12.

Q-Learning Performance

One of the more interesting things we can test in the HFTs is the effects of the value of the *zeta*, Z, parameter. Remember that zeta is the ratio of past history we should keep in the frequency domain at each time step. It effectively controls how much of the past we want to use in orthogonalizing our current state. A large value of zeta, near 1.0 will be highly orthogonal, while a value of zeta near 0.0 will yield a vector very close to the identity HRR. In Figure 13 we trained many models for each zeta value and see the average suboptimal steps towards the end of training for each. We can see that the HFT and HFT-with-actions encoders had similarly wide troughs in comparison with the light-only encoder. These troughs indicate a wider acceptable value for zeta. All three encoders reached a similar peak performance level around 13 suboptimal steps, and without a more fine-grained search across zeta it is hard to say which has the highest potential for performance. Still, in many ways, the trough mentioned before is a more practical measure of the quality of an encoder. The light-only encoder had a peak performance at a much higher zeta value than the other two, and we were not able to discern why this might be the case. Another notable feature of this graph is that toward $Z = 0.0$ all three encoders experienced a second improvement in performance which may not be trivial.

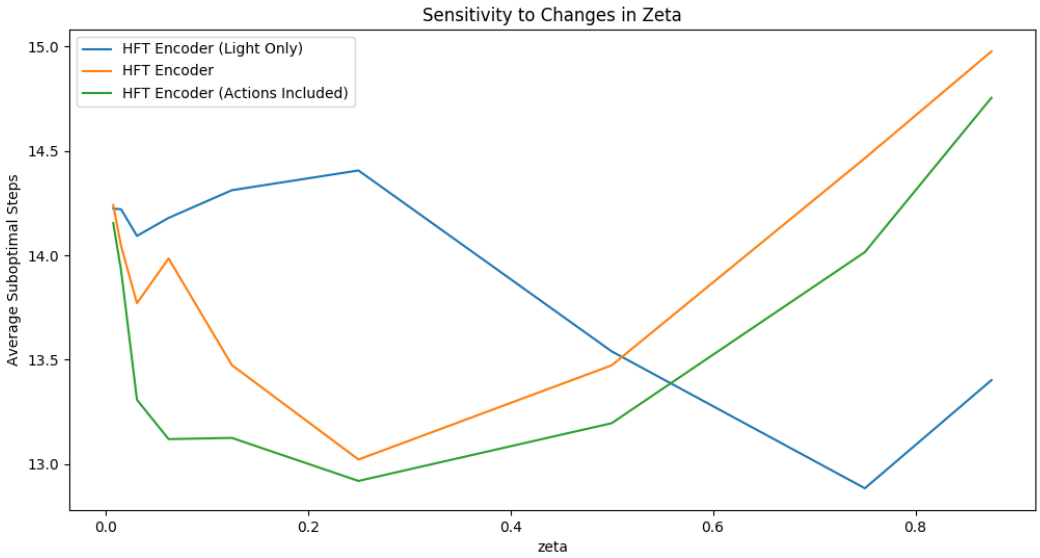

Figure 13: 64 models were trained at several zeta values for each of the three HFT encoders. Each model was trained for 4000 episodes and the average number of suboptimal steps over the last 100 episodes was calculated. We can see that each of the HFT encoders reached a similar peak level of performance at around 13 suboptimal steps per episode, but they did so at different values of zeta. The HFT encoder with actions has the widest trough, showing that it is the most robust to changes in zeta.

After calculating that $Z = 0.25$ was a fitting value for the HFT and HFT-with-actions encoders and that $Z = 0.75$ was fit for the HFT-light-only encoder, we use those values to compare these three encoders with the naive and optimal implementations in Figure 14. We can see that while all three HFT encoders perform significantly better than the naive encoder, they do not approach the performance of the optimal encoder. Additionally, while the HFT-with-actions encoder seemed to perform best, all three encoders have overlapping confidence intervals, represented in gray, at most points of training.

By far, (with the obvious exceptions of using the oracle or optimal encoder) HRR length has the most direct impact on performance of these models, as we can see in Figure 15 and Figure 16. At the end of training, the 3 HFT encoders had slowed down learning considerably, but seem to still be improving for all but the smallest length HRRs. The

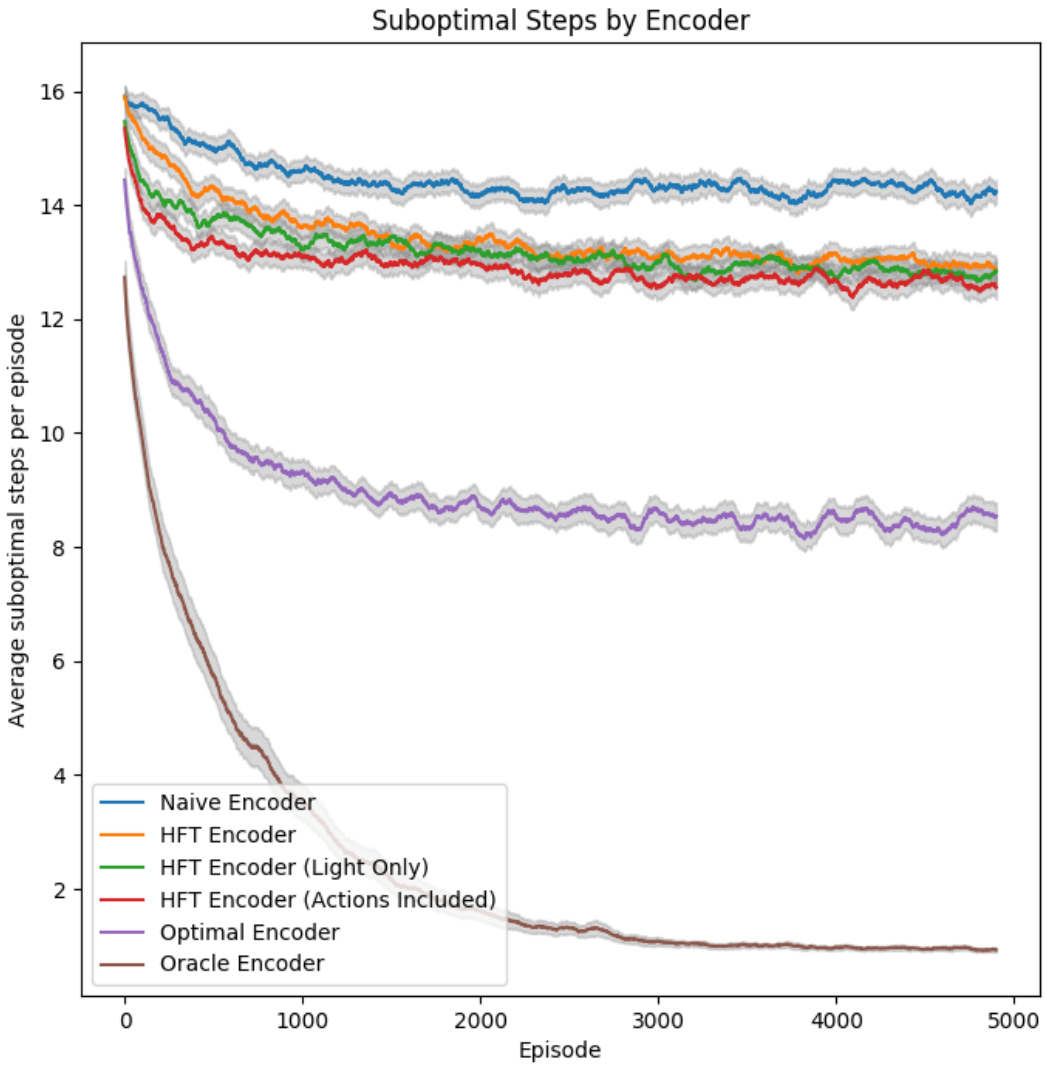

Figure 14: 128 models were trained for each encoder, with the HFT encoders using their approximate respective best zeta values (found experimentally in Figure 13). Each model was trained in a maze of length 30 with two possible goal locations. The gray error bars represent the 95% confidence interval for each encoder around that episode of training. We can see that the models using the oracle encoder were able to completely learn the function. However, the optimal encoder levels off at about 9 suboptimal steps.

optimal encoder achieved a peak performance of about 7 suboptimal steps per episode. Because of how we choose to calculate the number of suboptimal steps, if the agent takes a wrong step at the start of training, it needs to switch direction once the light disambiguates the goal location, causing a total of two suboptimal steps. In this case, the number of suboptimal steps will always be even. Additionally, because the maze is cyclic and has an even length, if the agent takes the wrong path around the maze, the total steps and optimal steps will either both be even or both be odd, meaning that, again the suboptimal steps will always be even. This means that on average, the agent using the optimal encoder takes 3 or 4 steps in the wrong direction every episode. With periods of 2 and 3 for the flashing light, the task is disambiguated at some point between $t = 3$ and $t = 6$. This means that the optimal encoder itself did not achieve truly optimal performance because we can expect that on average it would choose the correct direction 50% of the time if it guessed randomly, and perhaps more than 50% if it strategized its early moves based on potential goal locations. We are uncertain the cause of this result because the agent needs simply to learn three representations of each position-action pair, one for task 0, one for task 1, and one for no task, which should be attainable intuitively, and requires further investigation.

For the HFT encoders, exponentially increasing the HRR length yielded roughly linear improvements in the average number of suboptimal steps per episode. This makes sense, intuitively, because doubling the length of the HRR roughly doubles the number of states it can represent, and each time step within an episode doubles the number of possible representations there can be.

Because the agent has two options at each of up to 30 time steps, and there are roughly 2^5 starting positions, the total number of representations that could exist are roughly 2^{35}. In practice however, the number the agent needs to consider should be much smaller because when the agent has learned the proper behavior it will only be experiencing a tiny subset of

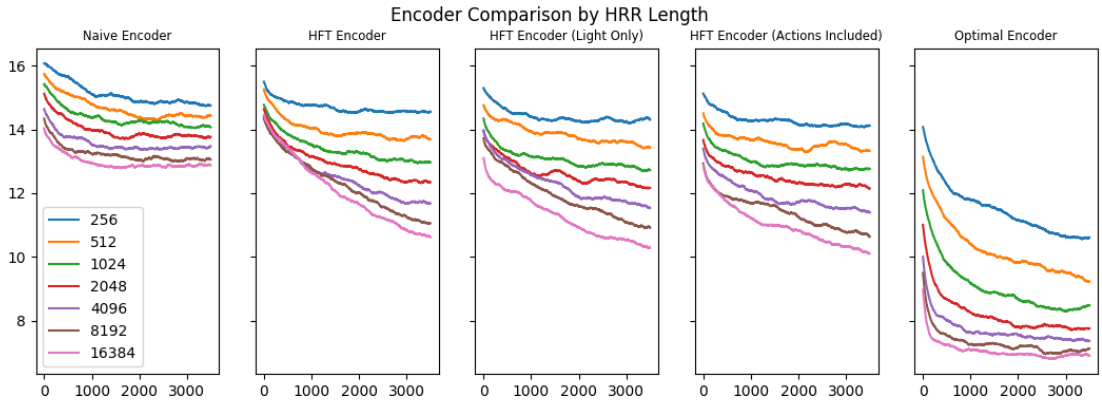

Figure 15: For each encoder, 64 models were trained with 7 different exponentially increasing HRR lengths, which resulted in roughly linear improvement in the average number of suboptimal steps per episode for the HFT encoders. The optimal encoder achieved an average of around 7 suboptimal steps per episode at best.

the total possible representations.

As we can see in Figure 17, the time required run to a single HFT model with HRRs of length 16384 was on the order of 10 hours and when running the hundreds of models required for statistical significance time quickly became a limiting factor in further exploration, despite yielding the most promising results.

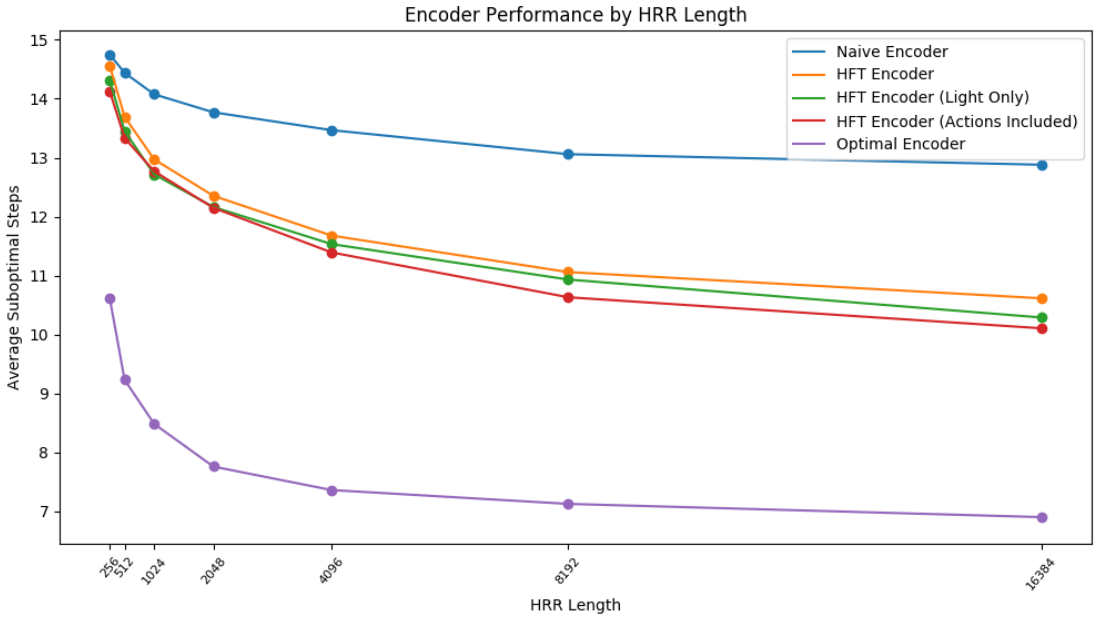

Figure 16: A snapshot of the average performance for each encoder for each HRR length during the last 100 episodes of training. This is a different view of some of the same data in Figure 15.

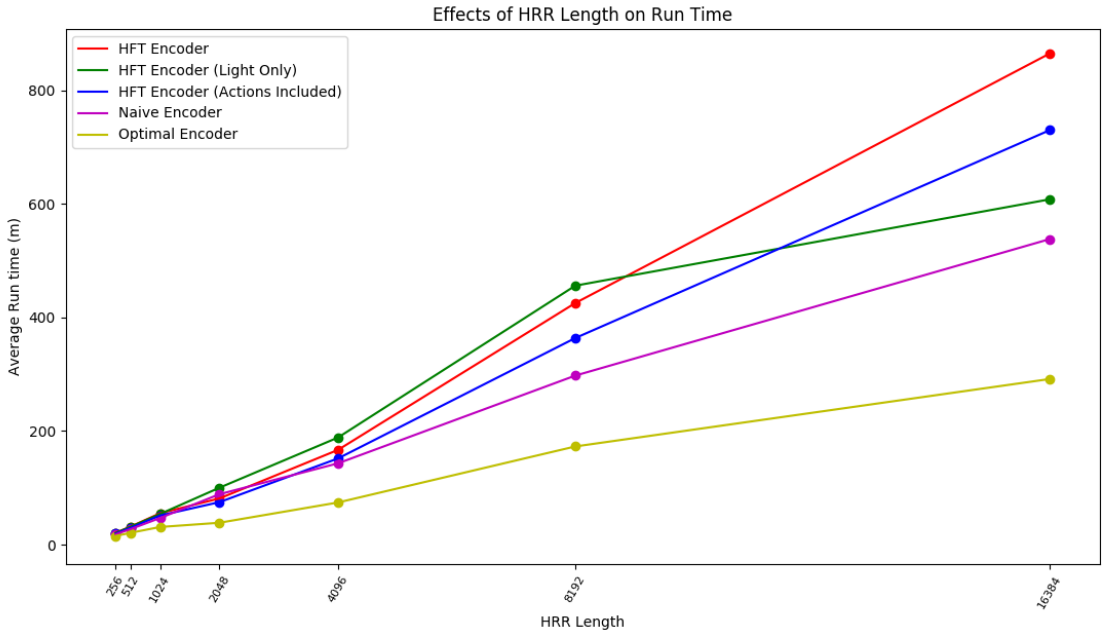

Figure 17: Because the FFT algorithm is $O(n \log n)$ we expect the time to increase super-linearly. While that is not clearly what we see here, the times shown are the wall-clock time and there are many other factors that influence the results. An unknown factor is also responsible for the behavior of the light-only HFT encoder as it increases from length 8192 to 16384, we suspect it is due to our implementation which saves previously calculated circular convolutions and that the light-only encoder experiences all possible traces relatively quickly, effectively becoming a constant-time lookup table.

CHAPTER V.

CONCLUSIONS

In order to contextualize a state, we can fold up the history of a sequence of steps into an episodic memory. This provides a sense of novelty to a state based on the steps that let to that state. A mechanism that does this is required to solve the flashing lights maze problem. While the HFT encoders we tested did not allow the agents to learn at the same level as the optimal encoder, they did make progress from the naive encoder which was unable to learn an effective policy at all. Further, when given sufficiently large HRRs, they would be capable of representing every possible path through the maze, and may be able to completely learn the optimal behavior. That said, the size of the HRRs would need to be prohibitively large even for a small task such as a flashing light maze of length 30.

Still, this process of being able to orthogonalize an episode based on the sequence of past steps in an on-line fashion may be useful. In this research we tried to use these encodings as standalone inputs to a neural network. However, because the representations are distributed in a way such that the network has no dedicated weights for processing other useful information such as agent position, the model has no direct way of tracking its location independently of the light value. We thought that perhaps the light-only HFT encoder would help resolve this, but because the representations resulting from the circular convolution of the trace and position are linearly independent from both, the model still has no sense of its location that is independent of the trace of the light. Said another way, it knows its location in space and time together, but not independently. This could be remedied by concatenating the position-action pair and light trace before being passed to the neural network. Testing encoders such as these are a promising prospect for future work, to use the HFTs as a tool to enrich already existing input instead of as standalone input.

We would also like to further explore the low values of zeta in Figure 13 at a very fine grain. It could be the case that a very small zeta value is actually more useful than the mid range values, as hinted at by the sharp dips in the graphs. We may simply need to search the space more thoroughly to see the actual potential of those values. A very small zeta value could apply just enough orthogonality to differentiate between tasks while keeping the representation of the position-action pair at the forefront.

The encoders explored show improvement in solving this task from a naive approach but do not completely solve to problem. Still, the test suite used in the evaluation of these encoders provides useful analysis of future encoders that seek to solve these kind of tasks.

BIBLIOGRAPHY

[1] H. ABDI. A neural network primer. *Journal of Biological Systems*, 02(03):247–281, 1994.

[2] P. Andersen. *Historical Perspective: Proposed Functions, Neural Characteristics, and Neurobiological Models of the Hippocampus*. Oxford University Press, 2007.

[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, Nov 2017.

[4] G. M. Dubois and J. L. Phillips. Working memory concept encoding using holographic reduced representations. In J. Licato, A. T. Hayes, and M. Glass, editors, *Proceedings of the 28th Modern Artificial Intelligence and Cognitive Science Conference 2017, Fort Wayne, IN, USA, April 28-29, 2017*, volume 1964 of *CEUR Workshop Proceedings*, pages 137–144. CEUR-WS.org, 2017.

[5] J. K. Kruschke. Alcove: An exemplar-based connectionist model of category learning. *Psychological Review*, 99(1):22–44, 1992.

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

[7] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[8] A. Williams and J. Phillips. Multilayer context reasoning in a neurobiologically inspired working memory model for cognitive robots. In C. Kalish, M. A. Rau, X. J. Zhu, and T. T. Rogers, editors, *Proceedings of the 40th Annual Meeting of the Cognitive Science Society, CogSci 2018, Madison, WI, USA, July 25-28, 2018*. cognitivesciencesociety.org, 2018.