Double Q-learning and Deep Q-learning with a Working Memory Concept

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

May 2020

Thesis Committee:

Dr. Suk J. Seo

Dr. Joshua L. Phillips

ACKNOWLEDGEMENTS

I'd like to thank my instructor, Dr. Suk Seo, for providing a great introduction and learning experience in the field of research. I'd also like to thank my advisor, Dr. Joshua Phillips, who was of great assistance in both teaching and guiding me through reinforcement learning and his working memory toolkit concepts.

ABSTRACT

Working memory is a part of our brain's memory system that temporarily retains a small amount of information which we use to accomplish tasks. Using a framework called the Working Memory Toolkit, we can easily model working memory based on cognitive neuroscience models using reinforcement learning (RL). The toolkit implements RL traditionally on a single-layer neural network, and the conceptual information encoded as holographic reduced representation (HRR) vectors can get quite large. In more complicated problems, learning can also be unstable and tricky to converge due to "randomness" in the environment, requiring additional effort to solve. By replacing the single-layer neural network with a multi-layer neural network using deep Q-learning, we show that the size of the HRR vectors can be reduced while retaining a reliable learning ability. The gathered evidence also suggests that double Q-learning reduces the noise in the learned Q-functions from exploratory actions. By incorporating combinations of these algorithms into the WMtk, the memory usage required for the tasks can be reduced while still learning and further stabilizing the desired Q-function; thus, leading to potential advancements in working memory modeling.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I.

INTRODUCTION

When you are taking notes during a lecture, or quickly writing a phone number down, you are actually taking advantage of your ”working memory,” an aspect of your brain’s short-term memory system. Working memory temporarily retains a small amount of information that is used to solve or complete a task. Old information that is now no longer useful can then be quickly swapped out for new information. While it would probably be convenient to remember everything that we experience, most of those experiences are repetitive or do not have any significance in our everyday lives, making it ”wasteful” to store. This short-term memory system allows our brains to be much more efficient by differentiating what’s important for our lives currently, versus what will be important in the future.

In the realm of biologically-inspired working memory models, Phillips and Noelle created a software framework called the Working Memory Toolkit that incorporates working memory models based on cognitive neuroscience [4]. The toolkit makes implementing working memory models significantly easier by greatly automating much of the machine learning process. By providing the user with a simple interface, the toolkit handles the artificial neural networks and machine learning, as well as other complicated aspects, in the backend away from the user. This allows developers who aren’t so familiar with the machine learning concepts to incorporate working memory models into their AI agents.

This project explores the implementation and integration of two new temporal-difference learning algorithms into the WMtk. In some stochastic environments where learning may be noisy or have difficulty converging, we investigate the effects and results of the double Q-learning algorithm. Furthermore, we also examine the incorporation of the deep Q-learning algorithm to improve performance by shrinking the memory-space required for the conceptual information, resulting in faster convolution calculations.

CHAPTER II.

BACKGROUND

One of the core concepts of the Working Memory Toolkit (WMtk) is its ability to utilize holographic reduced representation (HRR) vectors to automatically convert conceptual information from symbolic encodings to distributed encodings. An HRR vector is a fixed-width vector of random values pulled from a Gaussian distribution [5]. Each piece of conceptual information, such as ”big,” ”red,” ”dog,” etc., can be represented by a unique HRR vector. Through the use of circular-convolutions, these various concepts can be combined (e.g. ”blue*car”), resulting in a new unique HRR vector of the same width that is representing the total concept. With a reasonable vector width, these convolved vectors will be orthogonal to the original vectors [5]. The original vectors can also be retrieved through circular-correlation [5]. Since the HRR vector is a form of distributed encoding, it can be plugged straight into a neural network [1, 5]. By combining conceptual information, the agent can learn to perform multiple tasks with a single neural network. Additionally, since the HRR vectors are fixed-width, new information can be added without affecting the structure of the neural network. This learning has been termed n-task [3]. The agent can then be given working memory ”slots” that can be used to hold on to these HRR vectors, giving it a form of memory [4, 1].

The WMtk implements a reinforcement learning approach known as temporal-difference learning (TD-learning) on a single-layer neural network. This neural network, also called the critic network [1], takes as input the an HRR vector which results in the expected reward. By convolving the current state with an action, the neural network can learn and predict the expected reward based on the given state-action pair [1]. HRR vectors currently stored in working memory slots can also be convolved with the state-action, changing the potential value of the actions to take.

Moving focus into TD-learning, Q-learning is by far one of the most popular and one of

the more accurate TD-learning algorithms available. Traditional TD-learning attempts to learn the value function $V(S)$ to calculate an expected reward value for each state [6]. Using this function, an agent can look at the values of reachable states and determine which state to transition to. While this approach is simple, it's a bit unrealistic in the sense that the agent is essentially looking into the future; as if it has already made the transition into the state its observing. Q-learning is one such algorithm that resolves this issue. Rather than looking directly at the other states for their expected reward values, Q-learning learns and utilizes the *Q-function* $Q(S,a)$ which associates the value of an action at a particular state [8]. This grants the agent the ability to determine where to go based solely on where it is now, and what actions it can take/perform.

While we typically use Q-learning with the WMtk, a recent modification to the algorithm was developed known as double Q-learning [2]. In his paper, Hasselt explains the problems that standard Q-learning is susceptible to, in particular that it can overestimate in certain stochastic environments [2]. As a result, this overestimation can hurt the overall learning performance in some problems. As a Q-learning agent observes a new state, it updates its knowledge of that previous state with what it *knows* now, or what it has just learned about the new state. Essentially, the same function that the agent is predicting on is also used for updating its knowledge set. Double Q-learning attempts to resolve this issue by implementing two Q-functions that are used to update each other, rather than themselves [2]. An example of why this would benefit would be when playing a game of Roulette. If an agent wins on 13 black, standard Q-learning would learn that 13 black is the solution to the problem. Double Q-learning's second Q-function would heavily weigh against this idea, thus making the learning of such a problem much more stable. In working memory problems where the environment is more stochastic, double Q-learning could potentially assist in stabilizing the learning process.

Another potential improvement to the model is to reduce the size of the HRR vectors by

incorporating deep Q-learning. One of the issues with the current WMtk is the fact that the size of the HRR vectors grow very fast, and can get quite large even in small state-spaces [1, 9]. Since the WMtk uses a single-layer neural network [1], its overall capability in terms of what it can distinguish is fairly limited. The idea of swapping this single-layer neural network out for a multi-layer neural network could enable the ability to better differentiate the input vectors. However, simply adding additional layers to the network introduces the ”moving target” problem. Since deep neural networks require more effort to train, and since the neural network is constantly updating itself, the target value that it tries to learn is always changing, making it difficult for the network to converge. This issue is solved in deep Q-learning by creating two neural networks, freezing the parameters on one of them and using that frozen neural network to calculate the target value [7].

CHAPTER III.

METHODS

Stack and Framework

A relatively small tech-stack was used to build the models and perform the experimentation. All of the models were implemented within a Jupyter Notebook. The HRR engine to calculate and handle all of the HRR encoding of conceptual information was kindly provided by Dr. Phillips, the creator of the WMtk. To create the neural networks used for reinforcement learning, the Keras framework with Tensorflow-GPU backend was used. The models were trained on a single desktop GPU.

As this project is a standalone project, separate from the actual WMtk, the first step was to build a small reinforcement learning framework. A custom wrapper was created for the Keras neural network to more easily support the features required for double/deep Q-learning. Using the HRR engine, the wrapper automatically encodes into HRR vectors the conceptual information given to the neural network. The length of these HRR vectors is set to match the size of the neural network's input layer. Q-learning and its variations are then implemented as a single class that utilizes this neural network wrapper. The traditional Q-learning implementation uses a single-layer neural network that outputs a single value, indicating how valuable the given action is. The double Q-learning implementation extends this neural network to two output values, one for each Q-function. Finally, the deep Q-learning implementation adds hidden layers into the neural network, where the number of layers as well as the size of each layer is specified as a parameter to the Q-learning class constructor.

1D-Maze Working Memory Task

With a simple framework in place, the next step is to create a new task that cannot only be used for simple reinforcement learning, but can also scale up to a working memory problem. For this, a simple 1D maze task was created as illustrated in Figure 1. This

task on its own provides quite a bit of the technical aspects and potential difficulties that a reinforcement learning problem would run into. The task involves spawning an agent into the maze at a random position. By moving either left or right, the agent must ”learn” the optimal path to the goal position, regardless of where the agent is in the maze. The maze is also periodic, in that moving to the right at the end of the maze would result in the agent landing at the beginning of the maze, and vice versa.

Building up toward incorporating working memory, the 1D maze task first needs to be extended to an N-task problem to support multiple goals. Rather than having the agent learn a single goal position, the agent can learn to solve for multiple goals with the same neural network. By convolving the current state with an HRR vector representing the desired goal, the agent can learn a completely different function for each goal. This is one of the core concepts of the working memory tasks as the agent will need to learn to control its working memory in the next maze variation. This task was used to confirm the proper implementation of the Q-learning algorithms as well as the state-goal convolutions.

Finally, to transform the maze into a working memory task, the maze needs to be converted to a partially observable environment. This is done by adding two goals in the maze, with a signal presented only at the very beginning of each episode to indicate which goal to seek out. By adding a single working memory slot to the agent, it must learn to **store** the goal’s position into memory at the beginning of the episode, and to **protect** its

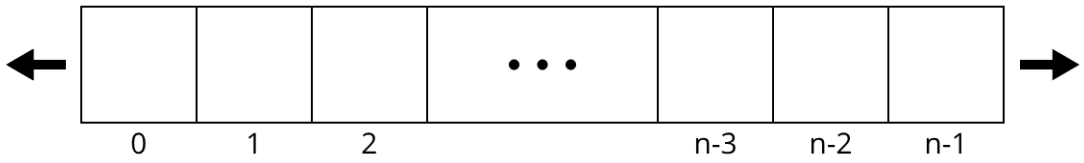

Figure 1: The 1D maze with `n` states. The agent must move left or right to find the goal state. If the agent moves left at state 0 or right at state `n-1`, the agent will end up on the other side of the maze.

memory slot afterwards so that it doesn't "forget" the signal in the middle of the episode. The memory that is stored is a new HRR vector representing an "internal" representation of the goal, different from the goal's actual HRR vector. In other words, by having a separate HRR vector for the same goal that an agent can memorize, the agent can distinguish what it is remembering vs. what is actually in the environment. Using the same HRR vector for both a signal in the environment and for memorization would result in the agent "hallucinating."

Experimentation

The first aim of the project was to experiment with and gather results from double Q-learning. Since double Q-learning was created for stochastic environments [2], the current maze problem is not necessarily a good candidate. To resolve this issue, a new random reward schedule was created to reward the agent at random points in its journey to the goal. The idea of this is to simulate the stochastic environment that double Q-learning seeks to solve in hopes of obtaining a noticeable improvement in learning. This experiment consisted of the simple 1D maze task with 20 states and a single goal placed at state 0, HRR size of $n = 256$, learning rate of $\alpha = 0.1$, discount factor of $\gamma = 0.95$, and an ε-greedy policy with random-action eligibility factor of $\varepsilon = 0.2$. The parameters were identical for both Q-learning and double Q-learning.

The second aim was to show that deep Q-learning could retain the learning ability with smaller HRR vectors. This process was done in two phases. The first phase was to demonstrate that deep Q-learning could indeed learn the function reliably with HRRs smaller than that of where standard Q-learning successfully learned. This was done with the standard maze problem using a maze size of 20 with a single goal located at 10, HRR size of $n = 64$, learning rate of $\alpha = 0.05$, discount factor of $\gamma = 0.95$, an ε-greedy policy with random-action eligibility factor of $\varepsilon = 0.2$, and a target-copy frequency of $C = 50$. With supporting evidence that deep Q-learning could learn the function with a smaller HRR vector, the second phase moved up to the working memory task. This task had the same

maze size of 20 with goals at positions 0 and 10, an HRR size of $n = 1024$, learning rate of $\alpha = 1.0$, discount factor of $\gamma = 0.95$, and an ε-greedy policy with random-action eligibility factor of $\varepsilon = 0.25$. These parameters were again retained for both Q-learning and deep Q-learning.

CHAPTER IV.

RESULTS

The first task that was investigated was the double Q-learning algorithm. While we had good expectations for the random reward schedule maze task, the results that came out of it showed no improvement. Both standard Q-learning and double Q-learning showed significant interference with their learned Q-functions. Due to time constraints, further investigation regarding this issue was abandoned. However, during experimentation when using double Q-learning with the typical reward schedule and identical parameters next to the standard Q-learning implementation, there appeared to be a consistent reduction in noise from exploratory actions. With more time allotted, further investigation into this issue would have continued. At this point, any advantage/disadvantages found by implementing double Q-learning within the working memory tasks has been inconclusive thus far.

Deep Q-learning on the other hand immediately yielded promising results. After first examining the results of deep Q-learning with a smaller HRR vector size on the simple 1D maze task, it was clear right away that deep Q-learning managed to learn the function. Through additional runs, it was found that deep Q-learning was also learning it reliably, 100% of the time in testing, while standard Q-learning failed in every case. The results from this observation are portrayed in Figure 2.

When applied to the working memory task, it was again immediately obvious that the algorithm managed to learn the function (or come very close) with an HRR vector of half the size as used in the standard working memory task. However, the difficulty of achieving good convergence increased slightly, becoming almost tricky. Through experimentation, the final parameters used managed to obtain some fairly good results. The results of the working memory maze task with deep Q-learning are shown in Figure 3.

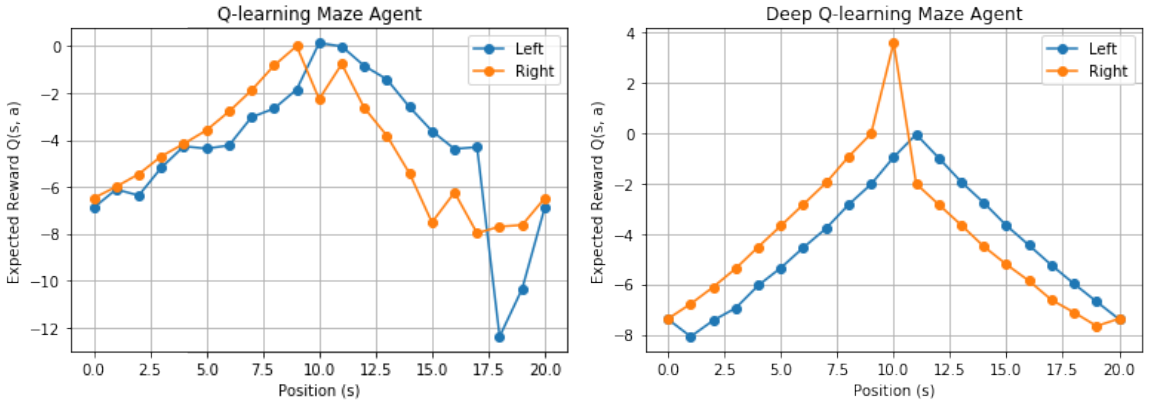

Figure 2: Q-learning vs. deep Q-learning with small HRR vectors using the simple 1D-maze with a single goal at state 10 after 10,000 episodes. Learning parameters for both: $n = 64$, $\gamma = 0.95$, $\alpha = 0.05$, $\varepsilon = 0.2$. This shows that deep Q-learning is capable of reliably learning the Q-function with an HRR smaller than standard Q-learning.

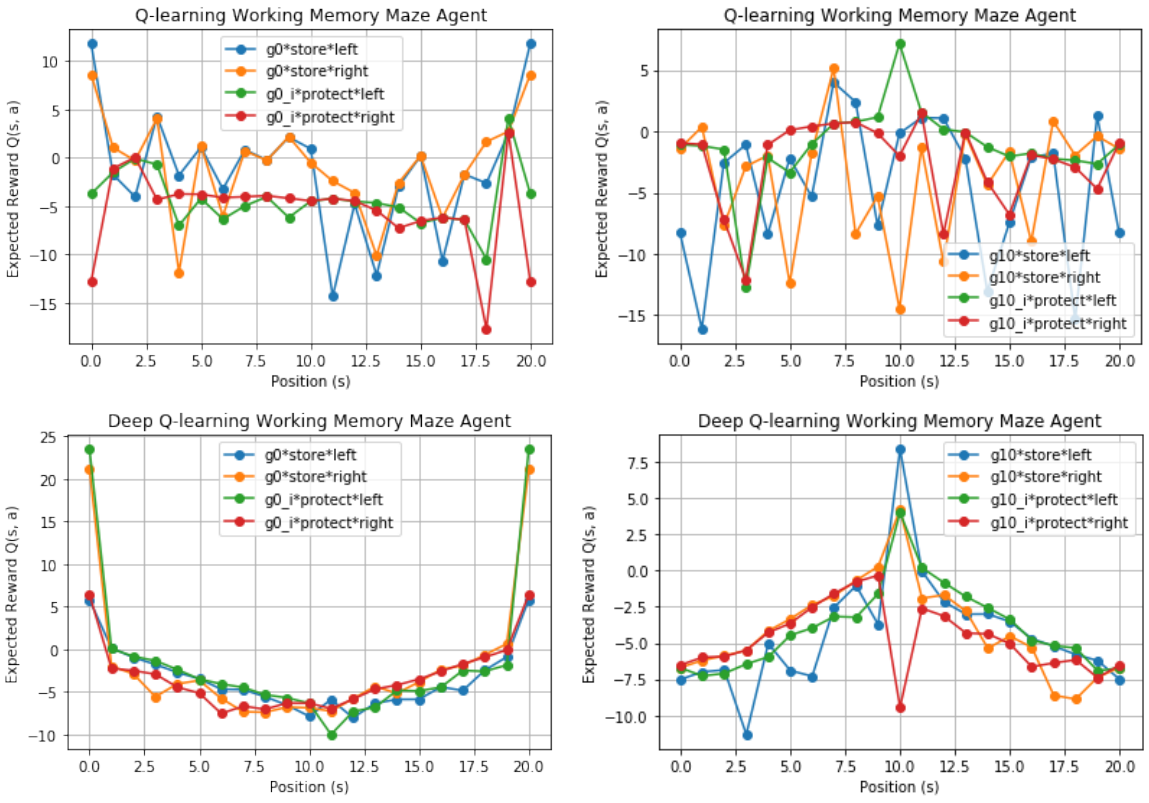

Figure 3: Q-learning vs. deep Q-learning with small HRR vectors using the working memory 1D maze with a single goal at state 10 after 15,000 episodes. Learning parameters for both: $n = 1024$, $\gamma = 0.95$, $\alpha = 1.0$, $\varepsilon = 0.25$. This is further support of the fact that deep Q-learning is capable of learning the Q-function reliably with smaller HRR vectors while the standard Q-learning algorithm results in garbage.

CHAPTER V.

DISCUSSION

In the end, we were unable to determine any improvement from the implementation of double Q-learning. While there were signs of improvement during the experimentation phase where noise from exploratory actions was reduced, there just wasn't enough supporting evidence to come to any immediate conclusions. With more time allotted, it would be very beneficial to view the sub-optimal moves over time. This would provide real evidence of the suspected noise reduction. It would also be worthwhile to explore more working memory tasks, especially tasks that have a natural randomness to their reward schedule. Even if only the noise can be shown to be reduced from exploratory moves, the time it takes for an agent to optimally learn a function could drastically be reduced.

The results that we were able to gather from deep Q-learning with the time restrictions that were in place were very promising. Deep Q-learning was able to reliably learn the Q-function with significantly smaller HRRs, down to half the size required for standard Q-learning in testing. While this was promising, the difficulty in achieving good convergence did increase a bit. If more time were available, it would be worth investigating how the number of and size of the hidden layers affects learning rate of the function; furthermore, it would also be worth taking look at how the target copy frequency of deep Q-learning affects the learning reliability. With a better understanding of the deep Q-learning integration into these working memory tasks, the memory-space required for the conceptual information could be reduced alongside potentially increasing the speed in the application since the convolution calculations would be much smaller.

BIBLIOGRAPHY

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

[2] Hasselt, H. V. Double q-learning. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2613–2621. Curran Associates, Inc., 2010.

[3] Jovanovich, M. and Phillips, J. L. 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.

[4] Phillips, J. and Noelle, D. Working memory for robots: Inspirations from computational neuroscience. 01 2006.

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

[6] Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.

[7] van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[8] Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[9] Williams, A. S. and Phillips, J. L. 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.