Training Robot Policies using External Memory Based Networks Via Imitation Learning

by

Nambi Srivatsav

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Recent advancements in external memory based neural networks have shown promise in solving tasks that require precise storage and retrieval of past information. Researchers have applied these models to a wide range of tasks that have algorithmic properties but have not applied these models to real-world robotic tasks. In this thesis, we present memory-augmented neural networks that synthesize robot navigation policies which a) encode long-term temporal dependencies b) make decisions in partially observed environments and c) quantify the uncertainty inherent in the task. We extract information about the temporal structure of a task via imitation learning from human demonstration and evaluate the performance of the models on control policies for a robot navigation task. Experiments are performed in partially observed environments in both simulation and the real world.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

Machine learning has transformed many branches of science and technology with innovative applications. Advancements in neural network modeling were able to solve complex machine learning problems in multiple domains like computer vision, speech recognition, video understanding, etc. Recurrent Neural Networks, such as Long Short-Term Memory (LSTM) networks, have brought breakthroughs in solving various complex sequence modeling tasks which traditional machine learning methods were unable to solve. However, even such RNNs come with a couple of inherent flaws that limit their applicability to relevant tasks and challenges. For example, RNNs model the data as a black box where no interpretations can be derived from the memory learned by the network. Furthermore, the memory of the network is coupled with computational units making it challenging to adjust memory without adjusting the computation. Recent work has proposed decoupling memory and computation, which results in an approach called Neural Turing Machines (NTMs). In this work, we investigate how NTMs could be used to solve specific real-world robotics tasks that exhibit long-term temporal dependencies. Any neural network model, in general, makes it challenging to introspect its internal behavior, as well as the inherent stochasticity. As the controller in the NTM is a neural network, we also present a method to estimate the uncertainty involved in the model which helps in evaluating the safety of executed robot actions.

Consider the task of driving down the interstate and navigating to the desired exit. One may observe a sign with information indicating that the desired exit is on the right side at a distance of one mile. A human solving this task must act on that

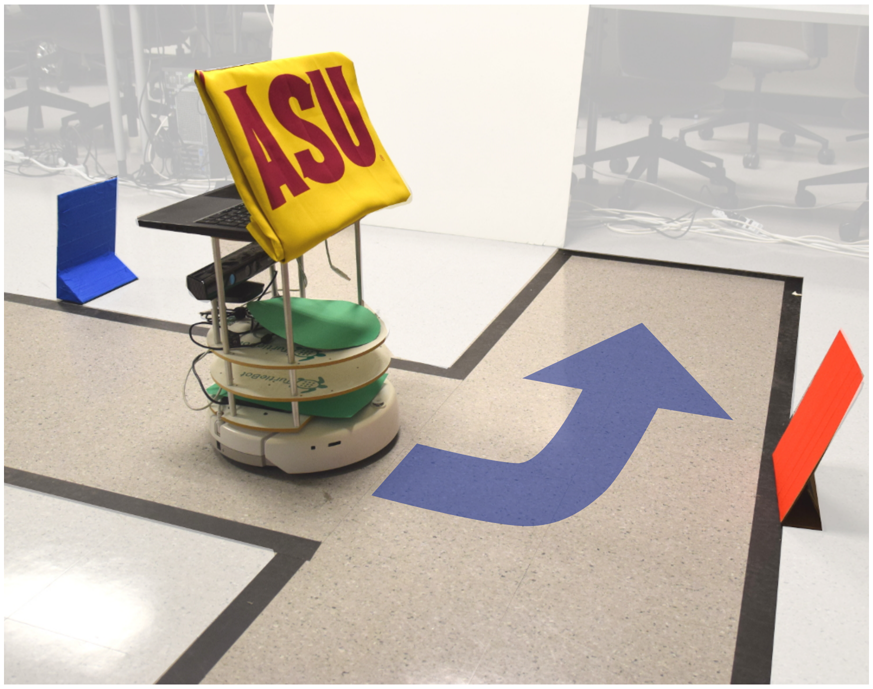


Figure 1.1: A Robot Moving to a 3-way Junction. Before Reaching the Intersection, the Robot Is Provided Information about Which Direction to Turn. This Is Realized via a Colored Sign (Blue or Green Rectangle). At the Intersection, the Robot Should Make a Decision to Turn Based on the Previously Observed Signal.

information over an extended time frame, even after the initial observation has long passed. To solve this task, which millions of people easily do every day, all necessary information must be (a) identified, (b) stored and (c) adequately incorporated into the decision-making process.

For robots to behave intelligently in similar complex, partially-observed environments, they must also be able to handle long-term dependencies effectively. Figure 1.1 depicts a similar experimental setup. While driving towards an intersection, a robot is presented with the optimal turn direction via a visual sign, e.g., a colored panel. Depending on the color (blue or green), the robot should turn either left or right. As soon as the panel has passed, the agent no longer has access to information about the turn direction and, therefore, cannot behave optimally if we model the robot behavior using a reactive policy. In general, memory plays a critical role in theoretical models of computation and is often the deciding factor which determines the expressiveness of a given model or framework [12].

To overcome the above challenge, many existing approaches for partially observed control problems are based on Long Short-Term Memory (LSTM) structures [14]. In the last few years, new models with separate, explicit memory structures, such as Neural Turing Machines (NTM) and its extensions [5, 8, 6] have shown promise in supervised tasks with long-term dependencies or algorithmic properties, such as copying or sorting a sequence. They can store information discovered to be relevant for a task and use it to make optimal decisions at a much later time. Memory transforms the partially observed task above into a fully-observed task when action selection is conditioned on its history by incorporating the memory state in addition to the current environment state. This behavior can formally be modeled as an n-th order Markov model where n is learned from data.

While researchers have explored the use of memory-based models in simulated control tasks they have so far not been successful in real-world robotics [7, 6, 10]. We apply these techniques to real-world robotics tasks through imitation learning and present results in training controllers for partially-observed robotic tasks by imitation in both simulations and on a physical robot.

The key contributions of this work can be summarized as:

1. The incorporation of Neural Turing Machines in real-world robotic tasks within partially observable environments, and

2. The utilization of stochastic forward passes to quantify the uncertainty inherent to the robot's actions at run-time.

Chapter 2

BACKGROUND

2.1 Neural Networks

Artificial neural networks are devised to replicate the manner in which the neurons in our brain work. The neural network is organized in layers of neurons or nodes with connections in between them. The first layer of neurons gets the input and the final layer of neurons gives the output. In between both of the layers, there are multiple hidden layers with interconnecting neurons. In an artificial neural network, the interconnecting synapse identical to the brain are the weights.

Similar to synapse in neuroscience, each weight refers to the strength of the connection between nodes. An activation function is embedded in each node that determines the output of that node for the given input or set of inputs. Along with input, weights are also passed into the activation function that determines if that respective node gets activated or not. If the nodes in the current layer get activated, the output from that layer is passed to the next layer. The process repeats until the final layer. The weighted sum of the last hidden layer produces the output required. Training of an artificial neural network is modifying the weights in between nodes in such a way that the error is decreased in getting our desired output.

2.2 LSTMs

Long short-term memory (LSTM) s are the type of recurrent neural networks that emerged as an effective neural network solution for sequential learning problems. Earlier, the traditional neural network was unable to be tailored for the longtime dependencies but we can do so now with LSTM. The main idea behind LSTM architecture is that there is a memory cell that stores the state information over time. There are gating units that regulate the data flow into and out from the memory cell. From, its original invention, so many versions of LSTMs have been proposed to solve numerous machine learning problems involving handwriting recognition and generation, language modeling and translation, analysis of audio and video data, etc.

2.3 Neural Turing Machines

In this section, we describe the fundamental concept of the external memory augmented network called Neural Turing Machine. A Neural Turing Machine comprises two main components called a controller (which is a neural network) and a memory bank (which is a matrix). Figure 2.1 is the high-level diagram of NTM. NTM receives the input and produces output using the controller. The controller interacts with the memory bank by writing the encoded data to it and reads the content from it and decodes the data when requested. The entire architecture with each component being differentiable is straightforward to train with gradient descent.

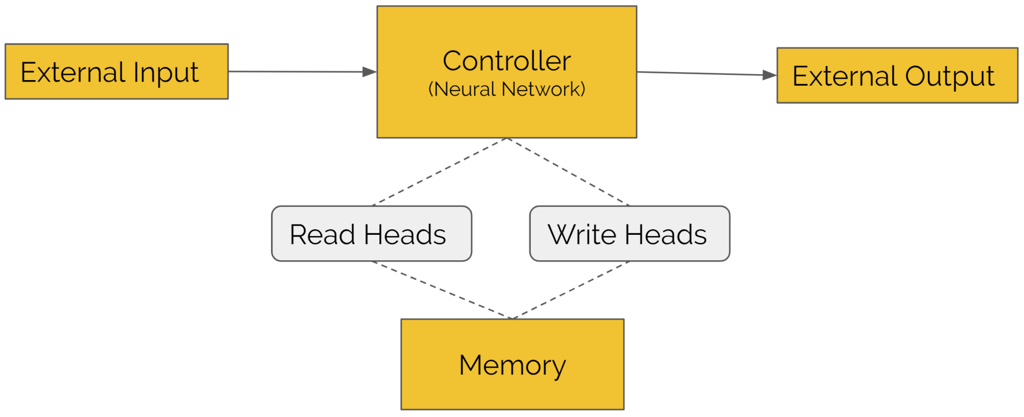


Figure 2.1: High-level diagram of NTM.

2.3.1 Controller

The controller in the NTM is a neural network and the architecture proposed in [5] typically uses two types of feedforward and LSTM. We have explored a various version of neural network controllers comparing the performance of them on the sequence memorization tasks. LSTM and GRU have their internal memory and might complement the larger memory present in the NTM. To have transparency we used feedforward controller for all of our simulation and hardware experiments and they are learning faster as per our comparative study between the controllers on the benchmark tasks. The NTM architecture involves multiple free parameters like memory, number of read and write heads, type of controller, range between the shifts, etc. It provides a great flexibility to modify the individual components of the architecture for respective tasks.

2.3.2 Memory

It mimics the RAM of the computer. It is another essential component of Neural Turing Machine and is a matrix where the elements can be written and read from. The size of the columns represents the length of the encoded input vector that is stored and the size of rows represents the number of memory slots present in the matrix. The controller acts the interface to memory while performing read and write operations. The memory is not fixed and is not attached to the computational units. This gives the flexibility to increase or decrease the memory of the NTM based on the task required to solve.

Chapter 3

METHODOLOGY

Imitation learning, also known as learning from demonstration, is a well-established method of enabling robots to perform new tasks autonomously [2, 1]. In this thesis, we address the use of memory-augmented neural networks for imitation learning of robotic control tasks with longtime dependencies.

3.1 Neural Turing Machine for Imitation Learning

In imitation learning, an agent or robot mimics the human demonstrations for given task. The agent is trained to perform a task by collecting human demonstrations and fitting a policy that matches the observed human data. Accordingly, human behavior is assumed to be optimal. The imitation learning setup formulates the learning process in a supervised manner. The procedure starts by collecting human demonstrated actions and the corresponding sensory data. In this work we consider stochastic policies modeled as $a_t \sim \pi(a_t|\ x_1, \ldots, x_t)$ where π gives a distribution over actions, a, given the past states, x_1 through x_t. Then, given a sequence of inputs $x_1, x_2, \ldots, x_n$ and outputs $a_1, a_2, \ldots, a_n$. we wish to minimize the negative log likelihood of $\pi(a_t|\ x_1, \ldots, x_t)$. Formally, we define the loss to be the cross entropy loss, $\mathcal{L} = -\frac{1}{T}\sum_t^T y_t \log \pi(a_t \mid x_t)$, where y_t is the true action at time t. The history is implicitly accounted for by the use of a memory augmented model. The memory augmented model learns to encode and store the environment perception data in its external memory matrix. Encode, decode, read and write operations are not manually written in the architecture but are learned by the NTM through back propagation. Figure 3.1 Schematic visualization of the memory-augmented model used in this work.

First, the environment perception data is collected and fed as an input to the NTM controller. Based on the task and robot used, the environment perception data could vary. If it is an image recognition task the input could be an image and robot arm movement task it could be joint angles. In our case, we have used a turtle-bot which has a camera mounted on it. So, we have processed the images using OpenCV, constructed a customized state vector and sent it to the NTM controller. The NTM controller receives the input, stores the received information in the memory and fetches the action required for a given input. To get the confidence of action proposed by NTM, we do multiple stochastic forward passes to the NTM with the same input and generate a probability distribution with mean and variance.

3.1.1 Reading

The observation space vector is fed to the neural network controller of the NTM. The controller emits the w_t is the weighting vector at time t normalized over the number of rows of the memory matrix.

Let us consider the size of the memory matrix is NxM where N is the number rows and M is the number of columns. Rows of the matrix represent memory locations and columns of the matrix represent the size of the encoded input vector. Let M_t is content of the matrix at time t.

$$0 \leq w_t(i) \leq 1, \qquad \sum_{i=1}^{R} w_t(i) = 1 \tag{3.1}$$

The following combination of read vector r_t generated and memory matrix $M_t(i)$ gives the content from the memory matrix:

$$r_t \leftarrow \sum_{i}^{R} w_t(i)\mathcal{M}_t(i) \tag{3.2}$$

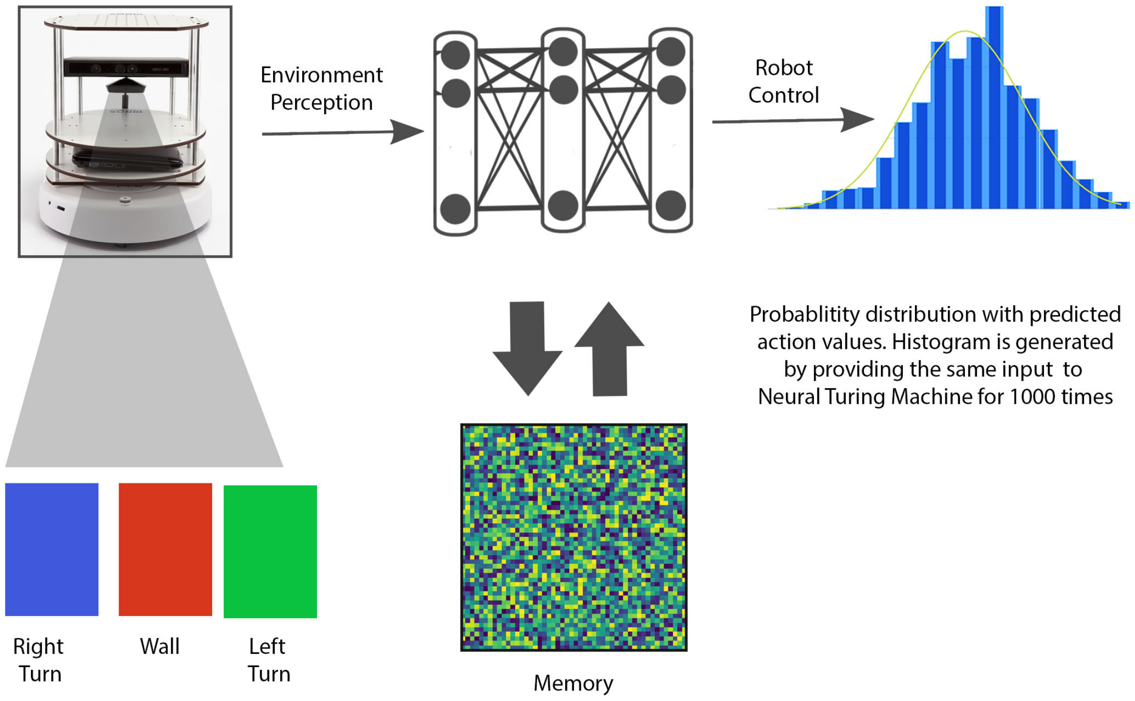

Environment Perception
Robot Control
Probablitity distribution with predicted action values. Histogram is generated by providing the same input to Neural Turing Machine for 1000 times
Right Turn
Wall
Left Turn
Memory

Figure 3.1: Schematic Visualization of the Memory Augmented Model Used in This Work. On the Left, Environmental Perception Information Is Fed as an Input to the Neural Network Controller. Then Controller Writes and Reads the Data from Memory Generating the Respective Control Action Distribution.

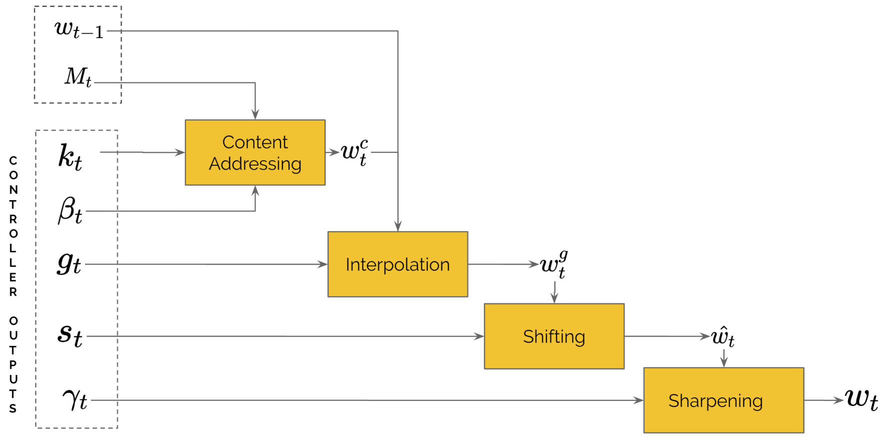
3.1.2 Writing

Writing to memory matrix is done two parts erasing from the memory matrix and adding to the memory matrix. The write head generates three components from the controller outputs: weight vector (w_t) with N elements, erase vector (e_t) with M elements and add vector (a_t) with M elements. The values of erase and vector lie in the range of (0, 1). Erasing from the matrix is done as follows:

$$\mathcal{M}_t^{erased}(i) \leftarrow \mathcal{M}_{t-1}(i)[\mathbf{1} - w_t(i)e_t] \tag{3.3}$$

Elements of the memory matrix are unchanged only if both erase vector and weight vector are zero. Memory matrix elements are set to zero only if weight vector and erase vector are set to 1. After erasing, add operation is performed as follows with generated add vector and intermediary memory matrix.

$$\mathcal{M}_t(i) \leftarrow \mathcal{M}_t^{erased}(i) + w_t(i)a_t \tag{4}$$



3.1.3 Addressing Memory

In both erase and add operations we used weight vector generated from respective heads. These weight vectors are generated by using two addressing mechanisms combined: content-based addressing and location-based addressing.

Focusing By Content:

Each head whether it is read or write head first generates a key vector k_t of length M. A similarity measure is computed with key vector and every row in the memory matrix $M_t(i)$. It generates a normalized weighting and is amplified with the scalar β which determines the key strength.

$$w_t^c(i) = \frac{exp\Big(\beta_t K(k_t, M_t(i))\Big)}{\sum_j exp\Big(\beta_t K(k_t, M_t(j))\Big)} \tag{3.4}$$

Cosine similarity is used as the similarity measure:

$$K(u,v) = \frac{u \cdot v}{\|u\| \cdot \|v\|} \tag{3.5}$$

Interpolation:

The next steps followed in modifying the weight vector is focusing by location. The main idea to have this to provide a facility in accessing multiple locations iteratively and be able to jump locations randomly in between them. To perform this operation, we begin with the scalar generated from the controller g_t and to blend between weight vector generated from content addressing and the previous weight vector. It is called interpolation.

$$w_t^g \leftarrow g_t w_t^c + (1 - g_t) w_{t-1} \tag{7}$$

Shifting:

Once, interpolation is performed the generated the weight vector is then multiplied with shift vector which allows us to move the location upper or lower. Shift weighting is a normalized distribution over the allowed number of shifts

$$\tilde{w}_t(i) \leftarrow \sum_{j=0}^{R-1} w_t^g(j) s_t(i-j) \tag{8}$$

Sharpening:

The final step is to sharpen the weight vector. The weight vector generated after shifting operation is sharpened by using another scalar γ from the controller as follows. It prevents the blurring of the weightings.

$$w_t(i) \leftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}} \tag{9}$$

Combining content-based addressing and location-based addressing, we generate a final weight vector to access the locations and store data in the memory matrix.

3.2 Uncertainty Estimation

A critical feature of our approach involves estimating the uncertainty, or conversely the confidence of the actions generated by the memory-augmented neural network. Uncertainty metrics generally used to measure the quality and accuracy of the deep learning model outputs [13]. The controller is a critical component in our model. As it is a neural network, it makes it difficult to get an insight into the internal behavior. The stochastic nature of the controller also raises the question reliability of the actions from the model. To overcome this, we have implemented a technique [4] to characterize the uncertainty affecting the network prediction. When generating actions by robot policies, the model output could be filtered with necessary confidence information before executing robot controllers. The actions can be prioritized based on the threshold of the uncertainty estimation. In doing so, we could derive better performance from the model and also evaluate the actions generated the model in uncertain environments.

Given the input of the observation space x^* to our model, it outputs the predictive mean $E(y*)$ which represents the action to be taken by the robot and also the predictive variance $var(y*)$ which gives the insight on the confidence of that action predicted by the model. To incorporate this into the NTM, we modified the layers in NTM controller layers to remove a random selection number of inputs, outputs, and recurrent connections and added noise to input data passed to NTM. The random selection is maintained in all steps of training of the experimental data and is varied for each stochastic forward pass. We do around T stochastic forward passes across the network and we estimate the predictive mean and predictive variance from our samples by the following equations:

$$
\begin{aligned}
\mathbb{E}(\bar{y}) &\approx \frac{1}{T}\sum_{t=1}^{T} y_t(x) \\
Var\big(\bar{y}\big) &\approx \tau^{-1} I_D + \frac{1}{T}\sum_{t=1}^{T} y_t(x)^T y_t(x) \\
&\quad - \mathbb{E}(\bar{y})^T \mathbb{E}(\bar{y})
\end{aligned}
$$

To add stochasticity, let us consider L hidden layers in the neural network controller where index l where $l \in \{1, ..., L\}$ Let z^l be the inputs and y^l be the outputs of layer l. W^l be the weights and b^l be the biases of layer l. So, in our experiments, we use two types of weight approximations for the forward passes.

1. Bernoulli approximating the distribution implemented as dropout:

$$\epsilon_1 \sim Bernoulli(p)$$
$$\bar{W}^l = diag(\epsilon_1)W^l$$
$$y^l = \bar{W}^l z^l + b^l$$

The operation has the effect of modifying the weighting tensors in NTM controller network by setting a random number of weights to zero. The randomness is being controlled by dropout rate(e.g if p is 0.01, it implies 1 percent of the weights are set to zero)

2. Multiplicative Gaussian approximating distribution, implemented as multiplicative Gaussian noise:

$$\epsilon_2 \sim \mathcal{N}(1,\, p_i/(1-p_i))$$
$$\bar{W}^l = diag(\epsilon_2)W^l$$
$$y^l = \bar{W}^l z^l + b^l$$

We have experimented both of the above methods on NTM controller by also passing random gaussian noise in the inputs to the neural network. This technique addresses both quantifying uncertainties and combining many different neural network architectures efficiently in the model.

Performing the stochastic forward passes multiple times with the same input leads to a set of multiple outputs, each output being different from the previous one. We can generate a histogram which approximates the distribution underlying the predictions of the neural network. Visualization of such stochastic forward passes can be seen in Figure 3.2. The left of Figure 3.2 shows the current scene in the simulator, presenting the car position on the map and the right graphs show the histogram of the stochastic forward passes. We would be able to observe that the certainty of an action predicted

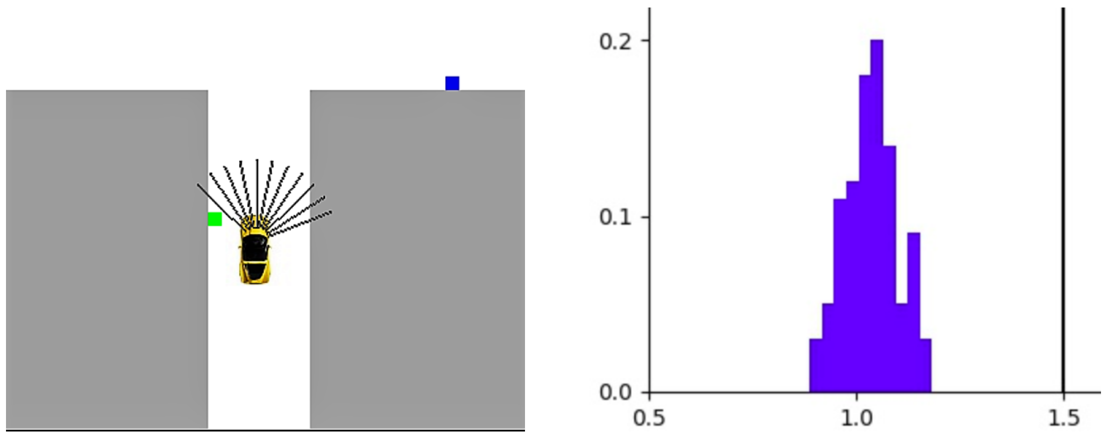


Figure 3.2: The position of the Car on the Map (Left) and the Respective Action Probability Distribution(Right).

by the network is high if the variance is low and certainty of an action predicted by the network is low when the variance is high.

Chapter 4

EXPERIMENTAL RESULTS

To evaluate the performance of NTM, we compared it to LSTM on the standard sequential task of memorization [9]. After initial experiments, we then shifted to training robot policies by imitation for partially observed tasks. We have also trained robot policies in simulation using the simulator we have developed in pygame. We designed two environments where the vehicle must navigate based on the signals it received many time steps in the past. We first adapted the classic T-Maze problem as a robotic driving task [3]. Then we made the problem more difficult by generalizing the T-Maze structure to a grid map. In both of these environments, the agent received data from a simulated radar sensor as input. Renderings of the environments are in Figure 4.2.

For all tasks we use the RMSProp optimizer with initial default parameters $\alpha = 0.99$ and $\epsilon = 1 \cdot 10^{-8}$ with a learning rate of 0.0001 for benchmark task and the imitation tests. All models were trained with a hidden size of 32 on the benchmark and imitation tasks. We have implemented all variants in the PyTorch library [11].

4.1 Supervised Benchmark Task - Memorization

To evaluate the NTM on sequential learning tasks we have taken a standard benchmark task of memorization. The task was intentionally designed by [5] to be a particularly challenging to learn by recurrent neural networks. The task here is that the neural network is given a series of random binary strings and it needs to learn to output the same binary strings in respective order. We have varied the sequence length from 5 to 20 and the string width was 8 bits.

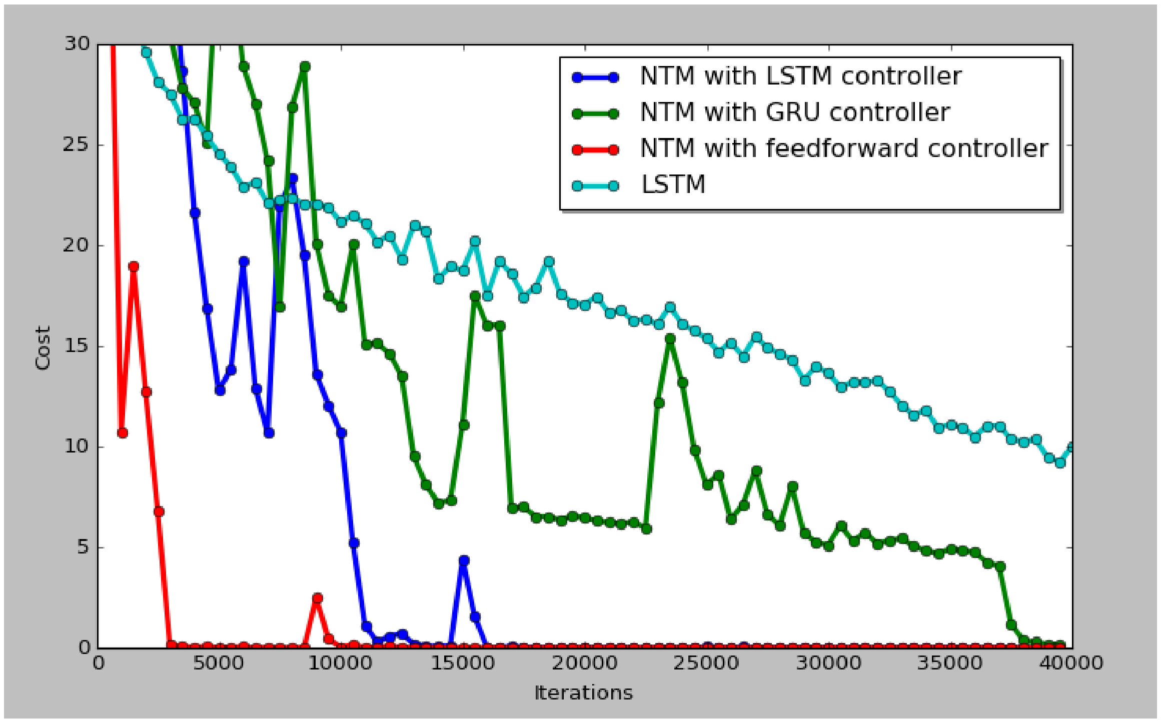


Figure 4.1: Mean of the Learning Performance of Various NTM Controllers and LSTM on Memorization with Five Different Random Generator Seeds. NTM with Feedforward piecewise Linear Controller Outperforms Other Models.

4.2 Imitation Learning in a Partially Observed Driving Task - Simulation

We now transition to applying our model to control tasks in partially observed environments using the simulation. We have designed two tasks, grid task; grid memorization task as follows.

The Grid Task

This task generalizes the T-Maze task to a more complex setup where the agent must navigate a series of left and right turns, dependent on earlier signals. We design this task so that the environment has a total of two to eight turns for the car to reach its destination. We have evaluated variants where the starting position, the

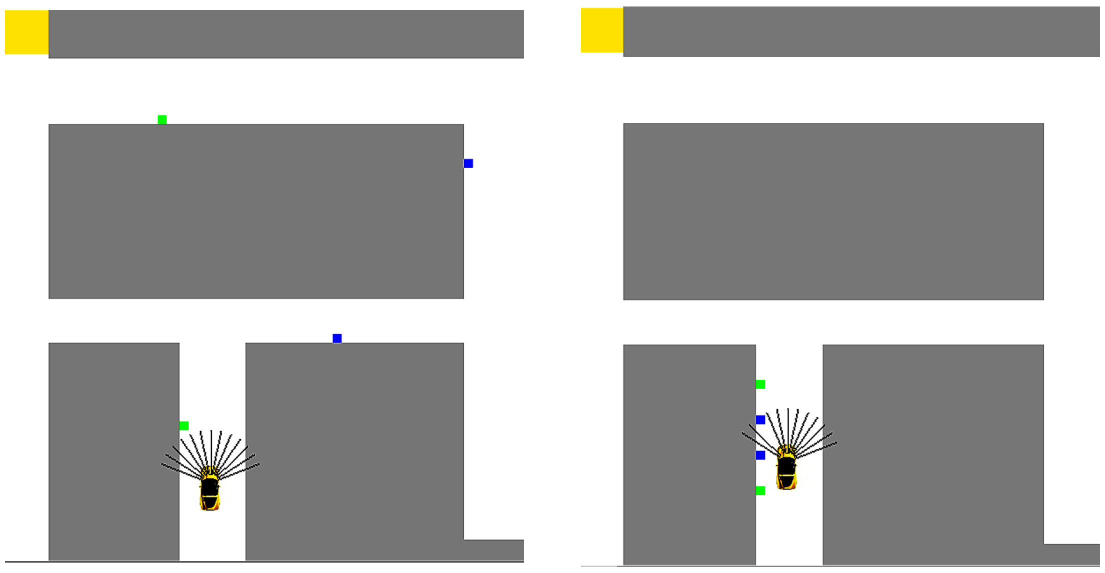


Figure 4.2: The Two Tasks That We Have Tested. Grid Task (Left) with Multiple Turns in a Single Task and Grid Memorization Task, a More Complicated Driving Task (Right) Where It Needs to Remember All the Turns on the Map Before Hand. Green Represents Right Turn and Blue Represent Left Turn and Yellow Represents the Destination.

signal position, or both the starting position and the signal position are randomized to increase the difficulty. The entire sequence length here varied from 100 to 250 steps. 4.3 has the mean performance between NTM and LSTM. NTM was learning ten times faster than LSTM.

The Grid Memorization Task

This task is similar to grid task with an even more difficult setup where the agent must navigate a series of left and right turns, again dependent on earlier signals. However, in this task, we give all the turn signals to reach the destination at the starting of the task. Here as well, we have evaluated the variants where the starting position, the signal position, or both the starting position and the signal position are

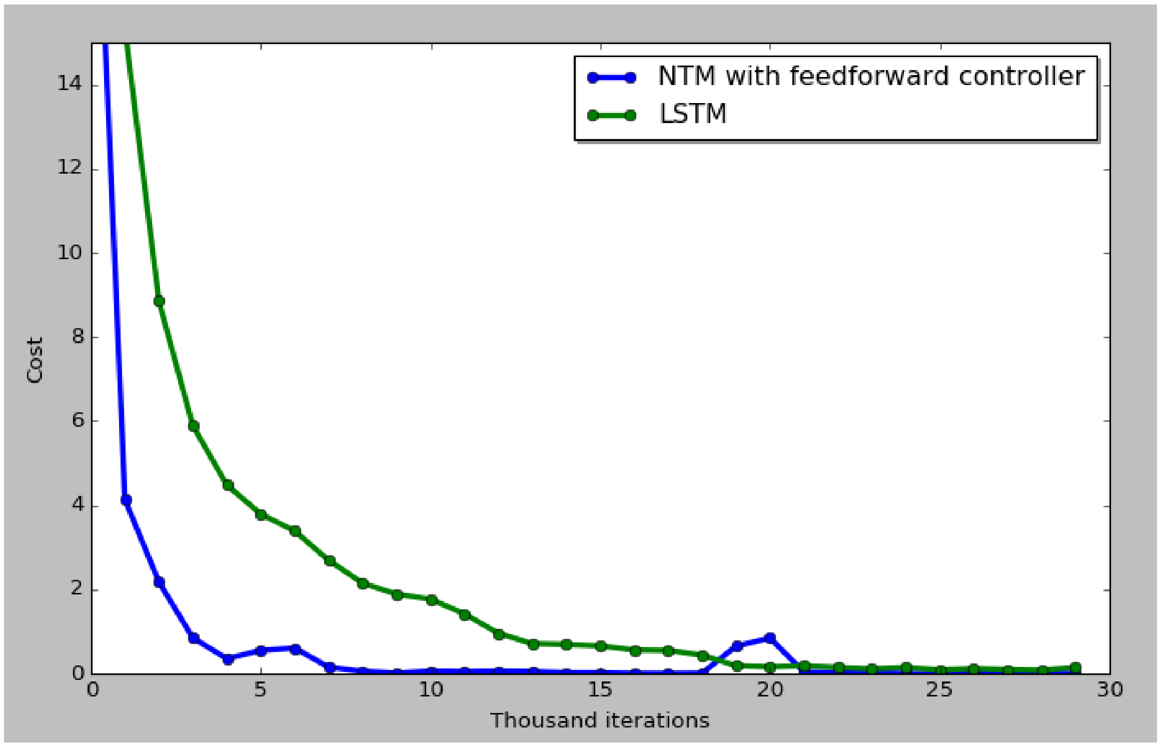


Figure 4.3: Comparison of NTM Feedforward Controller and LSTM on the Grid Task Having a Left or Right Turn Signal Randomly in Between the Sequence.

randomized to increase the difficulty. The entire sequence length here varied from 100 to 250 steps. Figure 4.4 has the mean performance between NTM and LSTM. NTM was able to learn in 40,000 iterations but LSTM was unable to learn much.

4.3 Robot Experiments

4.3.1 T-Maze task

To investigate the applicability of our model to real-world scenarios, we also tested it on the robot navigation task introduced in Figure 1.1. Figure 4.5 shows another depiction of the experiment. A turtle-bot robot is moving along a 3-way junction. The red label marks the intersection. Along the way to the intersection, the turtle-bot sees either a blue or a green sign. Optical recognition of the sign is performed with

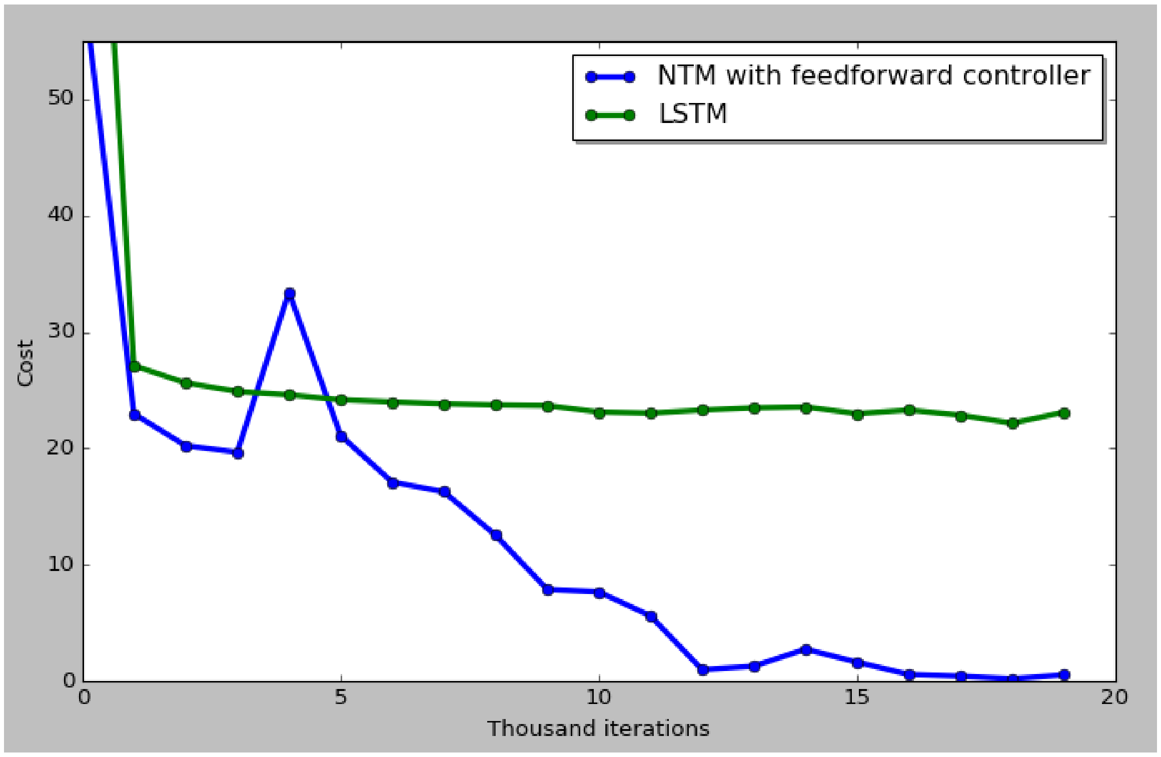


Figure 4.4: Comparison of Various NTM Feedforward Controller and LSTM on the Grid Memorization Where All the Turn Signal Are given at the Beginning of the Task.

computer vision techniques. Dependent on whether a green or blue label has been observed, the robot needs to turn left or right respectively. Note, however, that the labels indicating the turn direction are not visible at the time point when a decision has to be made. Hence, the robot has to learn that this particular information needs to be stored. In addition, it needs to learn to associate this previous information with the action to be performed when reaching the intersection. A complicating factor is that the position of the green and the blue sign is variable during test time. It is, therefore, not sufficient to associate a specific time step to the actionable information, but rather a specific visual stimulus.

The robot control policy learned via NTM takes as input the current distance to

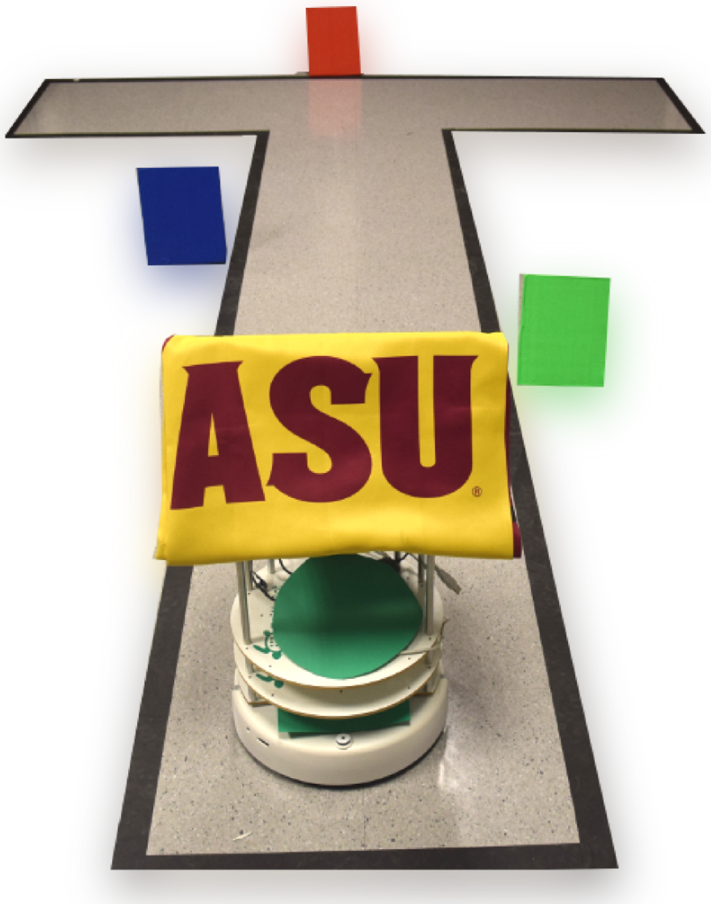


Figure 4.5: Experimental Setup. A Robot Moves a Long an Intersection and Performs a Turn Based on Previously Observed Visual Markers (Signs).

the red label, as well as a two-dimensional binary vector indicating the existence of either the blue or green sign. The output of the neural network is a discrete control action indicating whether to move forward, turn right, or turn left. The network is activated $\approx$ 100 times within each run of the robot. We ran 22 trials in all of which the turtle-bot made the correct turning decision when reaching the correct marker. Note that we did not specify that the robot has to decide when reaching the red marker. Instead, this information was automatically extracted from the human demonstrations. Figure 4.6 shows the movement trajectories of the robot (birds-eye view) during the 22 trials. The trajectories were generated using odometry data, and noise of about 2 cm can be expected in the depicted results. All but a single execution stayed within the envelope defined by the 3-way junction.

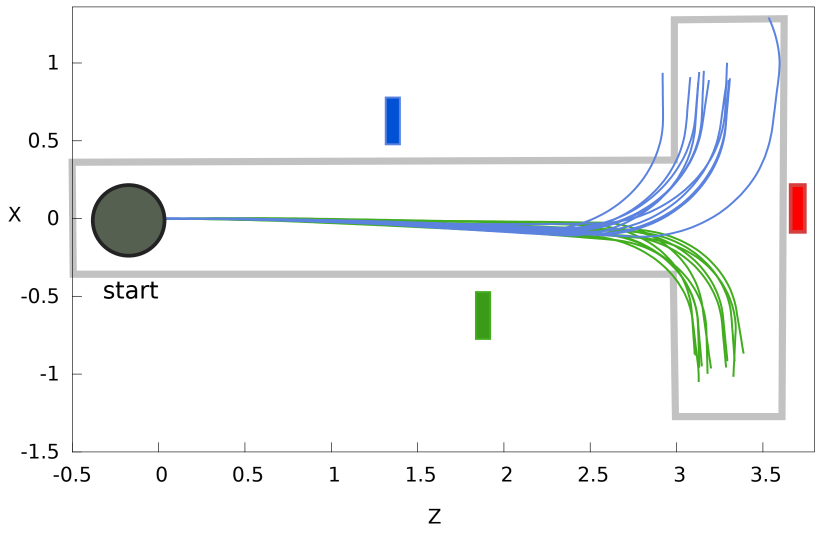


Figure 4.6: Bird's Eye View of the Experimental Setup. A Robot Moves along an Intersection and Performs a Turn Based on Previously Observed Visual Markers (Signs).

In the above, the first experiment, the robot was always initialized from the same starting position in each trial. Accordingly, the learned strategy may exploit the static nature of the distance to the intersection. To test the robustness of the approach to variable robot starting locations we performed an additional experiment in which the robot position was initialized within an area of ±1m of the original start position (2m variations). We performed an additional 22 tests in which, again, the robot was able to perform the successful turn in all of the trials.

Chapter 5

CONCLUSION

Acting with partial observations is a long-standing problem in robotics and control, and with the rapid development of application such as autonomous vehicles and assistive robotics, it is becoming increasingly essential to handle partially-observability over long timescales. In this thesis, we have presented an effective method to use external memory based network that handles long-term information by encoding it to a memory matrix. We have shown that the proposed method handles temporal dependencies better than a simple recurrent network or LSTM network these results open up a future sample-efficient robot learning with memory-based policies. For example, this structure could be integrated into a reinforcement learning algorithm or fine-tuned with after pre-training by imitation. Evaluating the confidence of the actions generated by the neural network before executing on the hardware of the robot could be a great safety check. It not only provides the security measure to prevent the robot from unexpected behavior but also helps us analyze the uncertainty present in the learned network. Using stochastic forward passes is one of the best ways to achieve uncertainty estimation and multiple architecture training.

REFERENCES

[1] Argall, B. D., S. Chernova, M. Veloso and B. Browning, “A survey of robot learning from demonstration”, Robotics and autonomous systems **57**, 5, 469–483 (2009).

[2] Atkeson, C. G. and S. Schaal, “Robot learning from demonstration”, (????).

[3] Bakker, B., “Reinforcement learning with long short-term memory”, in “Advances in neural information processing systems”, pp. 1475–1482 (2002).

[4] Gal, Y. and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”, in “international conference on machine learning”, pp. 1050–1059 (2016).

[5] Graves, A., G. Wayne and I. Danihelka, “Neural turing machines”, arXiv preprint arXiv:1410.5401 (2014).

[6] Graves, A., G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou *et al.*, “Hybrid computing using a neural network with dynamic external memory”, Nature **538**, 7626, 471 (2016).

[7] Heess, N., J. J. Hunt, T. P. Lillicrap and D. Silver, “Memory-based control with recurrent neural networks”, arXiv preprint arXiv:1512.04455 (2015).

[8] Kaiser, Ł. and I. Sutskever, “Neural gpus learn algorithms”, arXiv preprint arXiv:1511.08228 (2015).

[9] Martens, J. and I. Sutskever, “Learning recurrent neural networks with hessian-free optimization”, in “Proceedings of the 28th International Conference on Machine Learning (ICML-11)”, pp. 1033–1040 (Citeseer, 2011).

[10] Oh, J., V. Chockalingam, S. Singh and H. Lee, “Control of memory, active perception, and action in minecraft”, arXiv preprint arXiv:1605.09128 (2016).

[11] Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer, “Automatic differentiation in pytorch”, (2017).

[12] Sipser, M., *Introduction to the Theory of Computation* (International Thomson Publishing, 1996), 1st edn.

[13] Strickland, M., G. Fainekos and H. B. Amor, “Deep predictive models for collision risk assessment in autonomous driving”, in “2018 IEEE International Conference on Robotics and Automation (ICRA)”, pp. 1–8 (IEEE, 2018).

[14] Wierstra, D., A. Förster, J. Peters and J. Schmidhuber, “Recurrent policy gradients”, Logic Journal of the IGPL **18**, 5, 620–634 (2010).