Summer Research Report
2018-2019

Matthew Aitchison u6857890@anu.edu.au
Nikhil Babu u5864691@anu.edu.au

Abstract

We conducted experiments in the TMaze environment using a DQN agent with LSTM units to evaluate the capability of LSTM. We trained two models with different neural architectures and also trained them using continual learning. Our experiments found LSTM units to be useful for learning dependencies up to 40-time steps but were difficult to train, and performance was dependent upon initial values. Continuous learning improved performance but was susceptible to catastrophic forgetting. Further experiments need to be run which look at the average performance over ‘n’ runs to control for random variance and also increase the upper limit of steps/episodes.

Introduction

Most Reinforcement Learning (RL) solutions focus on solving Markovian Decision Processes (MDPs), despite real-world problems mostly being non-Markovian and only partially observable. This is in large parts due to the difficulty in solving partially observable Markov decision processes (POMDP). Feature Reinforcement Learning (FRL) can help reduce non-Markovian problems to simple MDPs, allowing the RL algorithms to be applied to more general problems. However, a lot of further work is required in this field, especially to make that transformation in polynomial time. One approach in FRL is dependent upon the concept of memory and requires the agent to remember ‘useful’ information. We decided to test the performance of Long Short Term Memory (LSTM) units in remembering information over a period of time using a T-Maze environment. The idea was that these LSTM units could, in theory, extract useful features required for FRL. We also looked at how continuous learning would affect the performance of the agent. Our findings would indicate a direction for future research into the role of memory in FRL.

Background

While much progress in Reinforcement Learning (RL) has focused on solutions to Markovian Decision Processes (MDP), most real-world problems are non-Markovian and only partially observable. That is, optimal decision making depends not only on the current observation, but

also on the previous history of observations, actions, and rewards. This class of problems, partially observable Markov decision processes (POMDP), have proven to be very difficult to solve.

It has been shown that Feature Reinforcement Learning (FRL) can reduce these non-Markovian problems to simple MDPs by finding a mapping from the agent's history to an MDP state space (Hutter 2009). Reducing problems to the well understood, and highly tractable, MDP opens the possibility of applying RL algorithms to a much broader set of problems and potentially enabling more general algorithms. However, what is not well understood is the process by which these maps can be found, especially in a computationally tractable manner.

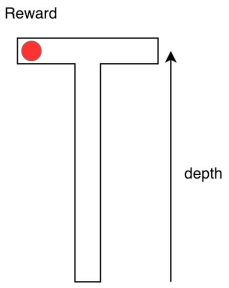
One approach to extracting useful features from history is memory. By learning which observations matter an agent can store, in its internal state, the salient parts of history, while ignoring the unimportant. Long-short term memory (Hochreiter & Schmidhuber, 1997) has been applied successfully over a broad range of tasks.

An alternative approach is to have the agent incorporate long term information in the weights of its neural network. Continuous learning allows agents to fine-tune their knowledge throughout their lifetime. However, by continuously updating the parameters, the agents are susceptible to catastrophic forgetting (Schwarz, 2018). That is, new knowledge is learned at the expense of old knowledge.

In order to evaluate the performance of agents, with respect to memory, an environment is needed that cannot be solved by simply looking at the current observation. The T-Maze environment provides just such a challenge (Bakker, 2018). An agent is given a signal at the start of a long corridor which it must remember, and then correctly pick a door based on that knowledge when it reaches the end. A successful agent must learn first to navigate the corridor, and secondly to select the appropriate door.

Method

The TMaze Environment



T-Maze is a non-Markovian environment where an agent must remember an initial observation at the origin of the maze and then apply that knowledge to pick the right door at the end of the maze. The environment has the following properties:

- Depth - The environment has a certain depth which is defined as the minimum number of steps it takes to reach the decision node or T junction of the maze starting from the initial position.
- Observations - Observation at the initial position is indicative of where the reward is at the end of the maze (left or right) and is randomised at the start of each episode. The decision node of the maze has a unique observation, and all the other corridor nodes have a fixed observation.
- Rewards - The agent receives a +4 reward for finding the goal, a -0.1 reward for bumping into the walls or picking the wrong door, and 0 reward (default) for each step it takes but this can be changed.
- Actions - The agent can move North, South, East or West, and if it bumps into a wall then it remains in the same position.

All models were trained using PyTorch v0.4.1 for a maximum of 200,000 episodes, using the Adam optimizer with a learning rate of 0.0003. Losses were accumulated during each episode then applied once the episode finished. Episodes were limited to a maximum of 250 steps each. Agents trained using the ε-greedy policy with ε=0.1 but were evaluated using a deterministic policy. Gamma was set to 0.98 as per (Bakker 2002).

Agents

Our agents are based on the Deep Q-Network (DQN) (Mnih, 2013) agent, but without any experience replay. The DQN algorithm minimizes the loss function L,

$$L = (r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a))^2$$

Where $Q_\theta(s, a)$ is the Q-value function for state s, action a parametrized by θ. Two agents (see figure 1) were developed:

- Original Agent - Based off of the network in the paper (Bakker, 2002). This network has 3 input units (the observation), 12 hidden units, 3 LSTM memory cells, and then 4 output cells, which are the Q values for the actions the agent can take.
- PassThru Agent - Has 3 input units, 3 hidden units and 3 memory cells, 12 hidden units, and then 4 output cells. This allows for the input to have a direct connection to the output without having to pass through the memory cells.

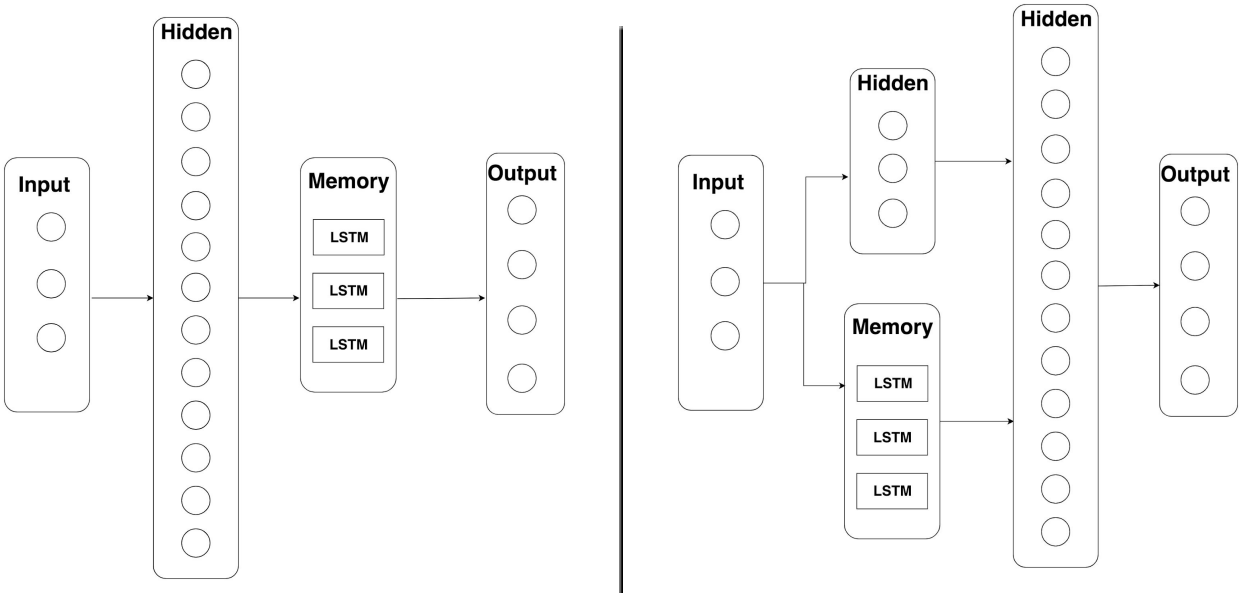


***Figure 1**. Left: The Original Model; Right: The PassThru Model*

Experiments

We ran 8 experiments for all combinations of the following variants (table 1).

Parameter	Variations
Agent	Original, PassThru
Step Cost	0, -0.01
Learning	Standard, Continual

Table 1 - Variations of agents used in experiments

The agent variations are as described above. Step cost is the reward given per step taken (to minimize redundant moves). Learning is either continual learning where the agent is not reset after each depth run and standard where it is.

Results

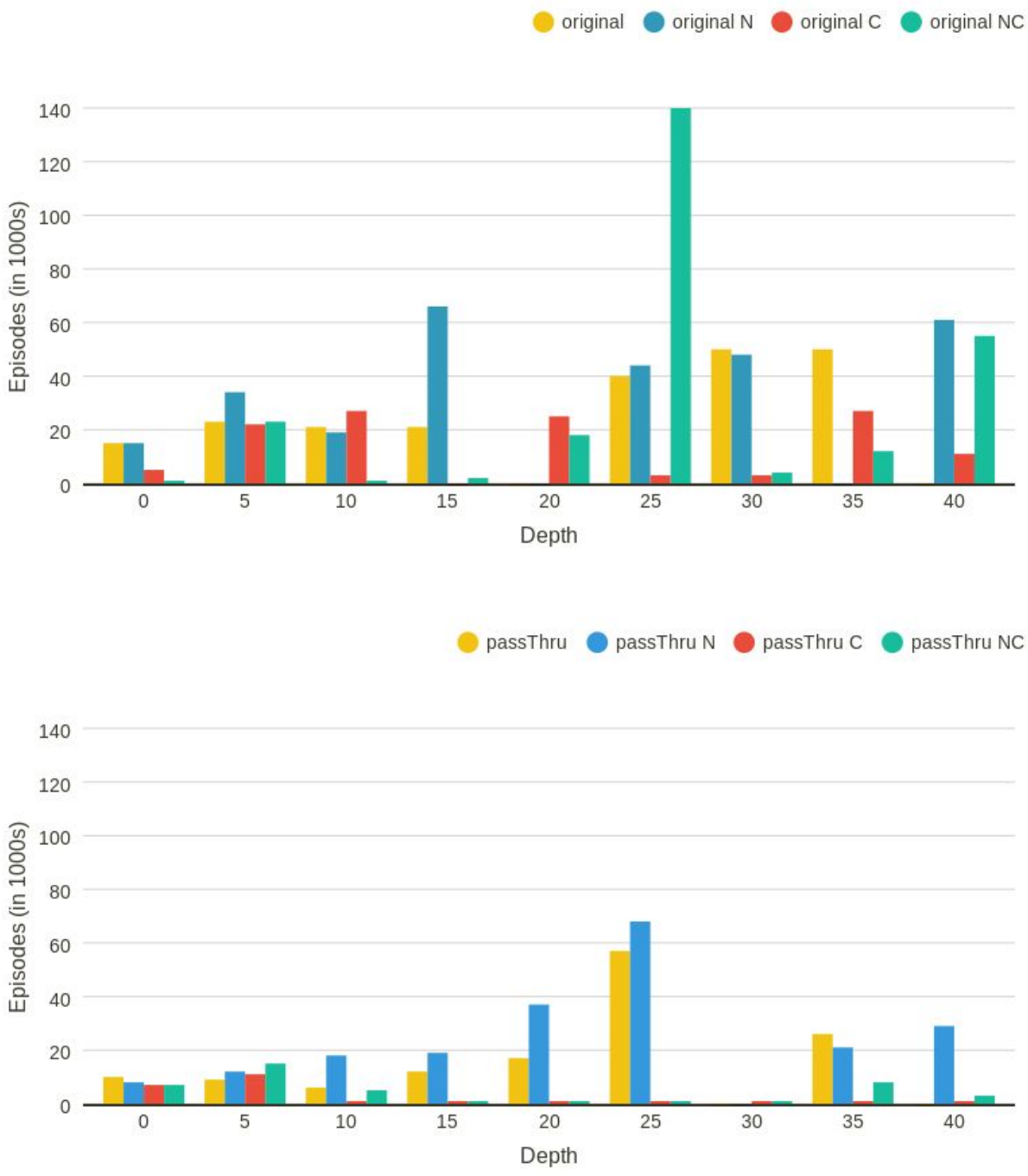


***Figure 2**. Top: The Original Model; Bottom: The PassThru Model.*

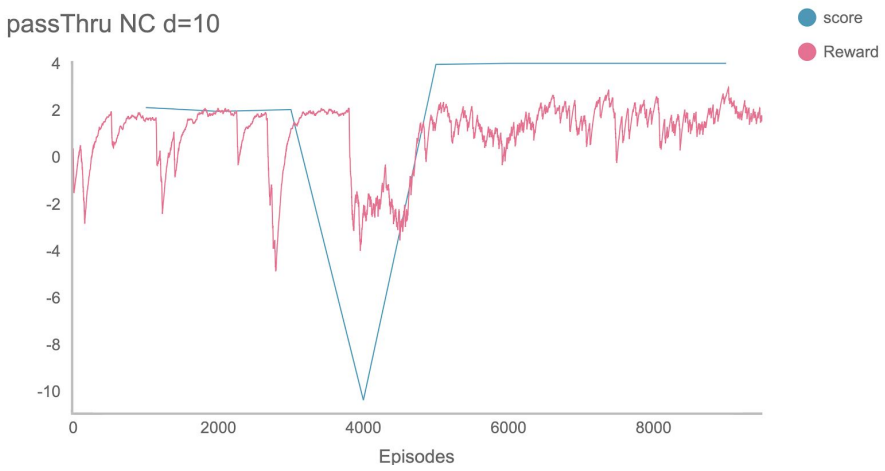


***Figure 3**. Training graph showing performance drop before settling on an optimal policy.*

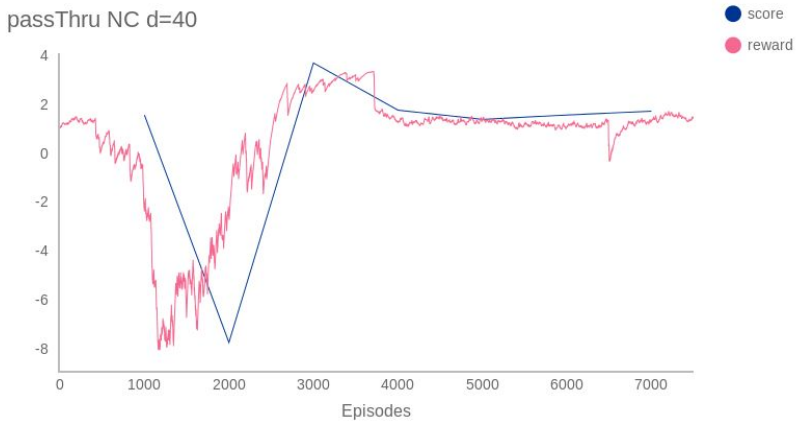


***Figure 4**. Training graph showing agent failing to converge.*

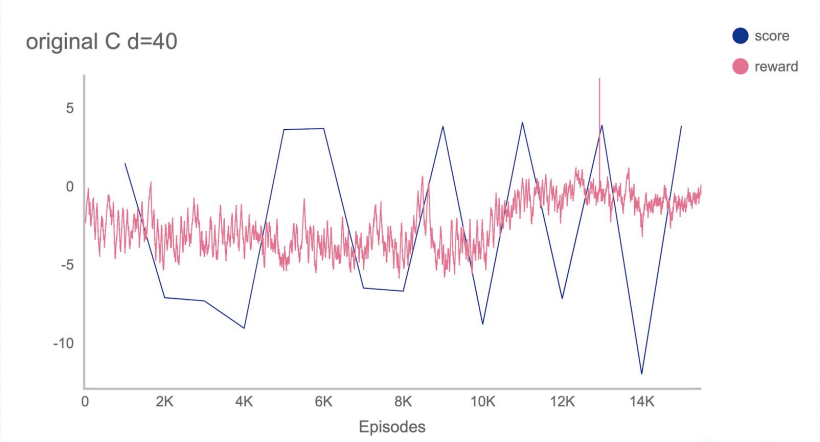


***Figure 5**. Agent oscillating between policies without converging.*

Training the LSTM based models proved to be very inconsistent. For example, the PassThru N agent took 68,000 steps to converge at a depth 25, but only 21,000 at the more difficult depth of 35. This was also seen in the continual learning case where for the original NC model convergence took 140,000 steps at depth 25, but only 4,000 at depth 30.

In general, continual learning worked well for this task, with the PassThru C model learning the additional depths in only a few thousand steps after the depth 5 task. The PassThru NC model performed slightly worse, needing only a little additional training at the longer depths.

We also noticed significant differences across runs, with, for example, the PassThru C model taking 116,000 steps to converge at depth 15 in a previous run, as compared to just 1,000 in the current batch of results. We believe this is due to divergence during training as seen in (figure 4) where the model would lose its previously apt parameters, and sometimes never find its way back to an optimal policy.

Due to these variations, in future work, we would like to run experiments 10 times and take averages, as well as increasing the upper steps limit from 200,000 to 1,000,000 or more. For higher depths, we noticed a tendency of the LSTM units to forget and the models with negative rewards (N) being more successful compared to the zero reward ones. We theorize this is because without negative rewards, going directly from initial position to goal is the same as just moving up and down a number of times, before reaching the goal. A scenario like that would mimic a TMaze with a longer corridor, i.e. depth and hence the LSTM agents would forget. Further experiments are needed to investigate this.

Conclusion

This project investigated the use of LSTM units as potential feature extractors for Feature Reinforcement Learning. Training LSTM units via DQN proved difficult due to divergence during training and dependence on initial conditions and training variations. Despite this models were developed that could solve T-Maze up to 40 steps, with the continuous learning based agent learning the 0 depth and 5 depth environments in just 18 thousand iterations, and then solving depths up to 40 with just 1 thousand iterations. This demonstrates the ability for the LSTM units to extract useful features from history that generalize across depth with very little additional training.

References

Bakker, B. (2002). Reinforcement learning with long short-term memory. *In Advances in neural information processing systems (pp. 1475-1482).*

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9(8), 1735-1780.*

Hutter, M. (2009). Feature reinforcement learning: Part I. unstructured MDPs. *Journal of Artificial General Intelligence*, 1(1), 3-24.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Schwarz, J., Luketina, J., Czarnecki, W. M., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., & Hadsell, R. (2018). Progress & Compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370.*