CS725 : Machine Learning Project Report
Deep Reinforcement Learning with External Memory

Varun Bhatt -140260004
Arka Sadhu -140070011
Parth Kothari -14D070019
Shray Sibal -14D070017

ABSTRACT:

In vanilla Deep Reinforcement Learning algorithms like Deep Q Network (DQN) , the agent learns the optimal policy through its interaction with the environment. The agent observes the reward for various actions and thereby learns which actions provide better rewards in a particular state by maintaining and updating action value function. In practical applications, there arise a particular set of problem statements where the agent is required to “remember” a particular observation, and consequently choose the next set of actions. Classical DQN architectures are not sufficient to handle such kinds of problems motivating the use of external memory. In this project, we explore the architecture of maintaining an explicit memory in addition to the DQN. This helps the agent learn the optimal route faster which is validated by our experiments.

EXPERIMENT’S DESIGN:

The Environment :

Our experiment revolves around a task which motivates the use of external memory that is, we will show faster convergence of a network using explicit memory by creating an environment which requires us to explicitly remember an observation. The environment is in the form of a I-shaped Maze. The agent starts from the top left corner. A key is placed in the top-right corner. There are two keys (Red, Yellow), randomly chosen in each episode, and one key corresponds to a particular colored door (In our case, Red corresponds to Blue, Yellow corresponds to Green). The agent gets the reward on reaching the doors **only if** the key has been collected prior to it. Hence, The agent is first required to learn to collect the key and then go towards the corresponding door (Blue or Green). The doors are placed in the lower left and lower right positions of the maze. Each cell in the maze is color coded. Normal cells are white. Key cells are either Red or Yellow while the door cells are Green or Blue as mentioned above. With reference to Figure 1 below, the blue dot denotes the current position of our agent. The key in figure 1 is Yellow in color and so the agent is expected to go to the Green cell after collecting the key. We created the game environment by altering the maze-implementation in OpenAI gym [1] to our requirements.

[1] "[1606.01540] OpenAI Gym - arXiv." Accessed November 24, 2017. https://arxiv.org/abs/1606.01540.

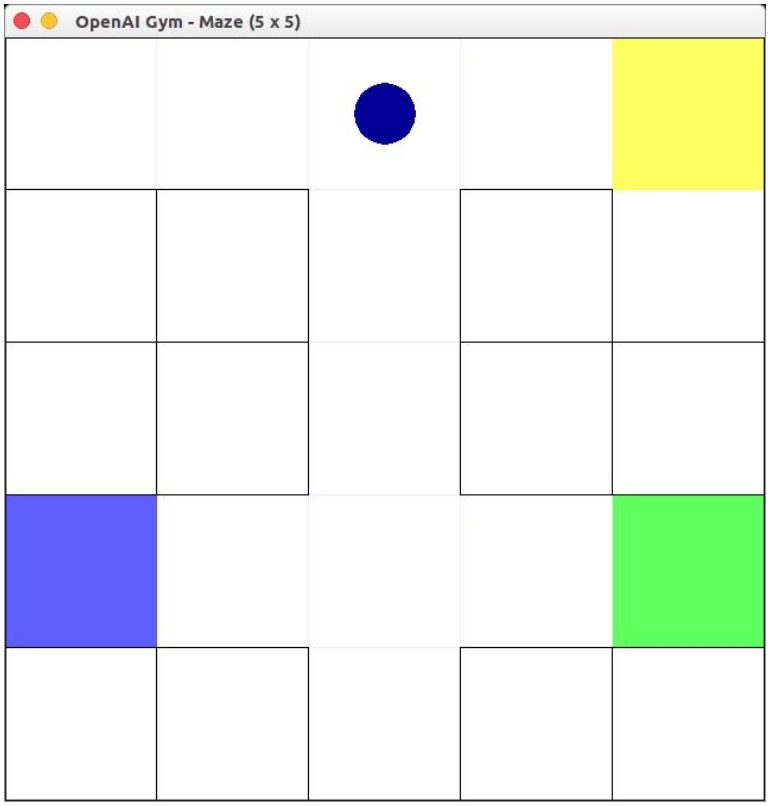

Fig 1. Our Environment.

The Observation Space :

After each action step, the environment returns the following 5 observations to the agent: color of cell of current state appended with color of the four surrounding cells in North-West-South-East format. If there is a wall in the surrounding, we return color as black. ([White, Black, White, White, White] with reference to Fig 1.). The colors are one-hot encoded before sending as an observation.

The Action Space :

At any time step, the agent has 4 actions to choose from : Move one step North, Move one step West, Move one step South or Move one step East. It is obvious that the step will be taken only if there is no immediate wall in that direction. If it tries to go in the direction which has an immediate wall, the agent does not move and stays in the same block.

The Reward :

For every step the agent moves, we give it a reward of -0.1/(size of maze^2) if the corresponding cell is WHITE. When it reaches the correct door it gets a reward of +2 and when it reaches the wrong door it still gets a reward of +1 but only if the agent had collected the key before it, as mentioned earlier. Else, a reward of -0.1/(size of maze^2) is given even if it ends up

at a door. The motivation is so that the agent also understands that to complete the game ,we need to go towards the door..

EXPERIMENT DETAILS :

We trained the agent using deep Q-learning using 4 different architectures for the Q-network:

- Deep Q-network (DQN[2]) - 2 dense layers of size 32 and output layer of size 4 for the 4 actions
- Deep recurrent Q-network (DRQN[3]) - A dense layer of size 32 followed by a LSTM layer and then a dense output layer over the last output of LSTM layer
- Memory Q-network (MQN) - A dense layer of size 32 for feature extraction and then memory layer similar to MQN[4]
- Recurrent memory Q-network (RMQN) - A dense layer of size 32 for feature extraction and then memory layer similar to RMQN[5]

In all the cases, observations from the past 50 frames were provided as the input to all the networks. In case of DQN, all observations were flattened before sending them to the network. For other architectures, we have retained them as a sequence.

Discount factor was kept as 0.99. Exploration rate was linearly annealed from 1 to 0.1 over 1000000 time steps. Replay memory of size 100000 was used. After at least half of the replay memory was filled, Q-network was trained by picking a random batch of size 32 every 4 time steps. RMSprop was used as the optimizer with learning rate 0.00025, momentum 0.95 and 0.01 was added to squared gradient in the denominator.

For 5x5 maze, the Q-networks were initialized randomly. For 7x7 case, the model obtained after training on 5x5 was used. The experiments for 5x5 maze were run for 15000 episodes while the experiments for 7x7 maze were run for 2000000 time steps.

RESULTS and DISCUSSIONS:

Figure 1 shows the smoothed score (reward at the end of the episode) for 5x5 maze; figure 2 shows the episode duration; figure 3 shows the average of the Q-values of chosen actions in an episode. It can be observed from the score and episode duration part that DQN was slowest to

---

[2] "Playing Atari with Deep Reinforcement Learning - University of ...." Accessed November 24, 2017. https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf.
[3] "[1507.06527] Deep Recurrent Q-Learning for Partially Observable MDPs." Accessed November 24, 2017. https://arxiv.org/abs/1507.06527.
[4] "Control of Memory, Active Perception, and Action in Minecraft - arXiv." Accessed November 24, 2017. https://arxiv.org/pdf/1605.09128.
[5] "Control of Memory, Active Perception, and Action in Minecraft - arXiv." Accessed November 24, 2017. https://arxiv.org/pdf/1605.09128.

converge while the rest converged fairly quickly. Also, towards the end, the score was oscillating between close to 2 and 1 but more than 50% of the times, the correct door was reached.

Fig 1: Smoothed score for 5x5 maze

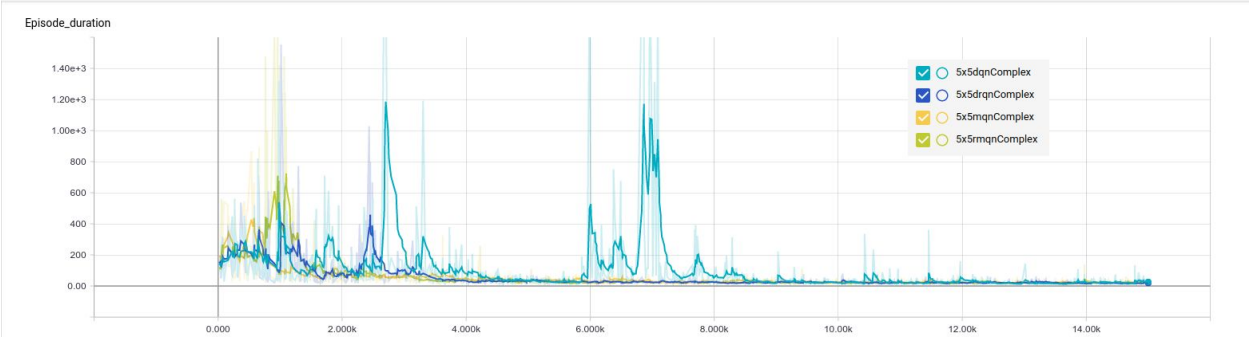

Fig 2: Episode duration for 5x5 maze

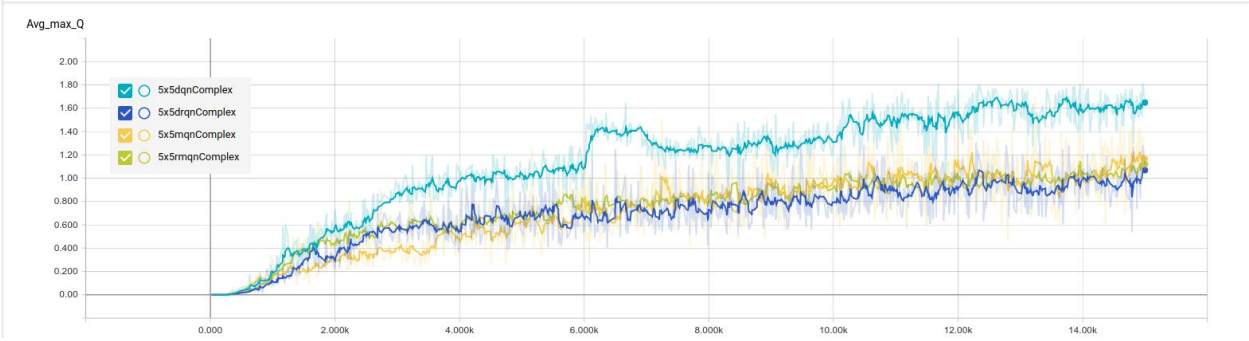

Fig 3: Average of Q-values of chosen actions for 5x5 maze

Figures 4, 5, 6 show the smoothed score, cumulative reward (sum of all rewards obtained till the current time step over all episodes) and the episode duration. Note that the x-axis is episode number for score, duration while it is the time step for the cumulative reward. Since the number of times steps were kept constant, the number of episodes vary between algorithms. It can be seen that the architectures with memory (MQN, RMQN) perform better than DQN.

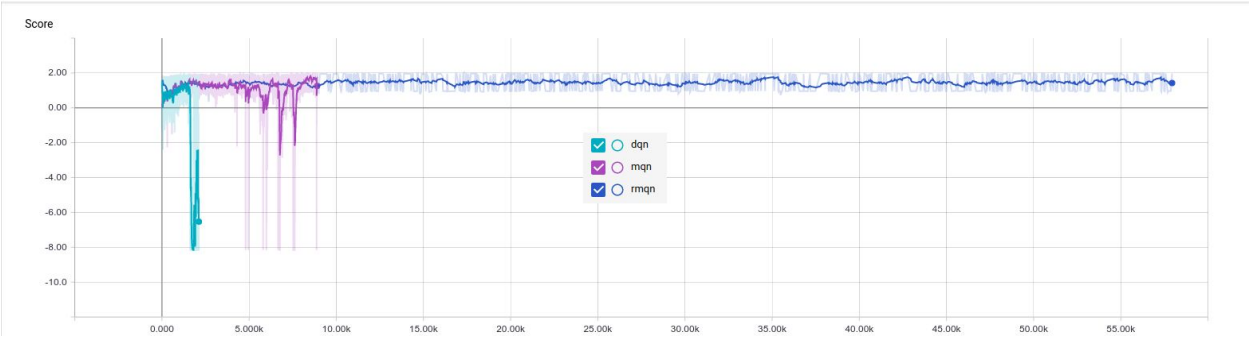

Figure 4: Smoothed score for 7x7 maze

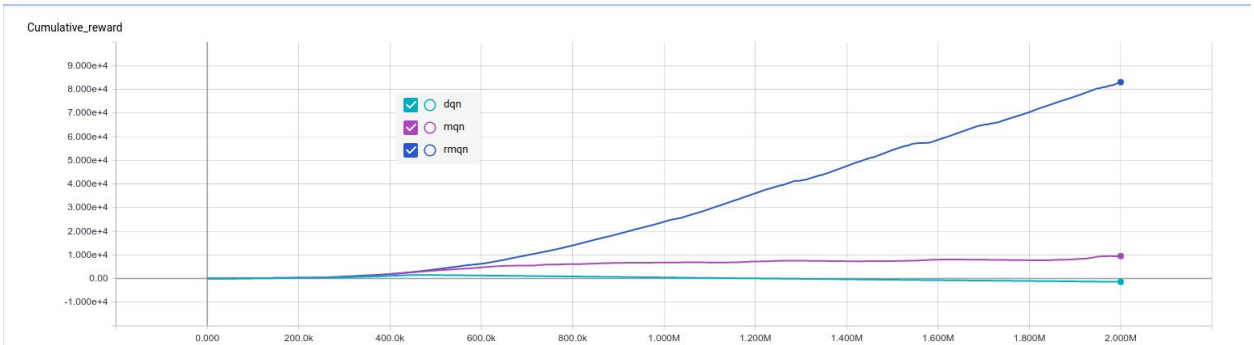

Figure 5: Cumulative reward for 7x7 maze

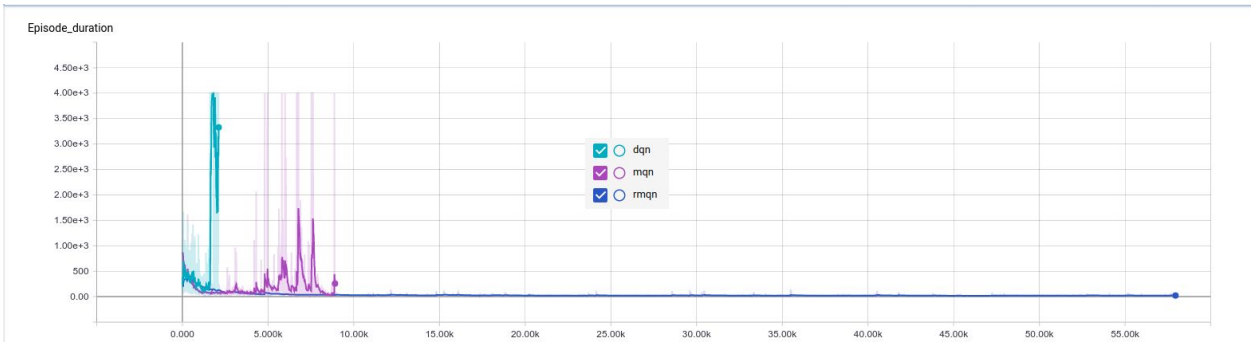

Figure 6: Episode duration for 7x7 maze

In both 5x5 and 7x7 case, the agents first learnt to go to one of the doors (randomly) after collecting the key and slowly started to improve by going to the correct door. Since the amount of training needed was very high for the second part, agent mostly ends up learning to go to one of the doors and doesn't always find the correct door.

During tuning of parameters, it was observed that the agent sometimes repeatedly visits the key block. It could be due to the requirement of getting the key to finish the maze which makes the agent initially give a high weightage to it.

The number of past frames given to the networks as input affected the convergence. Giving 4 frames was not enough; at 20, only MQN could learn to reach any door in 5x5 maze; at 50, all algorithms were able to get to one of the doors.

The experiments were first run by keeping the number of episodes constant. Since exploration rate was annealed at each time step, algorithms which converged quickly had a higher exploration rate than the ones which took longer. This made comparison with x-axis as episode number incorrect and hence, the later experiments were run for fixed number of time steps. This can be seen in figure 7 where in the 5x5, the exploration rates were different for different algorithms.

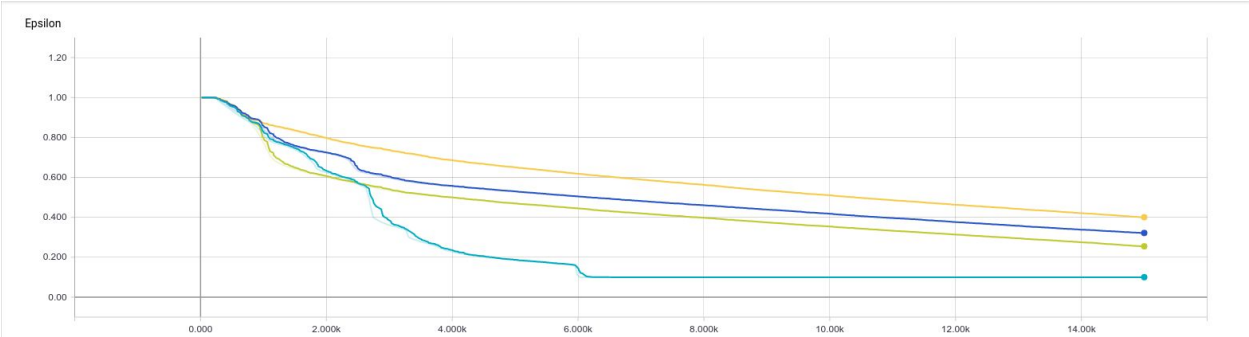

Figure 7: Exploration rate vs episode number for 5x5 maze

The reward given by the environment was seen to have a big impact on the convergence of algorithms even though, ideally, it should have found the optimal policy in all cases. Giving a reward of 0.5 for getting the key for the first time made the agent move continuously to key and never go to the door. +1 for correct door and -1 for incorrect door also made the agents get stuck in the maze.

If the exploration rate was annealed too quickly, it led to agent learning wrong policies like repeatedly going to the key block or not being able to solve the maze. Slower annealing improved learning at the cost of higher training time.

Other parameters were used directly from the original implementations of the architectures. Fine tuning them could possibly give an improvement.

APPLICATIONS :

Applications of such architectures is obviously on environments where tasks are performed in a sequential manner depending on the previous observations. Whenever an agent is required to perform tasks keeping in mind the previous observations, this is a better architecture than classical DQNs. Here, we kept the positions of the door and keys fixed, but we expect the agent to perform well even if the structure is different once it has learnt which key corresponds to which door. The above maze is can be thought of as a simplification of some

games where player needs to collect a key and go through the door (like Montezuma's Revenge in ATARI) and hence this architecture is expected to perform well in such games too.

CONCLUSION :

This project demonstrated the usefulness of explicit memory for reinforcement learning. Having explicit memory allowed the agent to model long term dependencies easily and selectively remember information from the environment which could be useful in future.

Algorithms rewarding exploration have been successful in solving problems which reinforcement learning algorithms have previously failed to solve. In future, architectures with memory can be combined with such algorithms to get better results.