Module: INM707: Deep Learning 3: Optimization
Professors: Dr. Eduardo Alonso/ Dr. Michael Garcia Ortiz

Reinforcement Learning

Author:
Jasveen Kaur - 190020638
Elisabeta Monica Furdui - 190045971

1.Basic reinforcement learning

1 Domain and task

Finding the shortest path is a crucial task for autonomous devices such as drones, vacuum cleaning devices and stockroom robots. With faster delivery times required, autonomous devices are the core of the supply chain industry.The task for a path finding robot is to reach its goal state from any other state in an unknown environment without external help. For this basic task we will use Reinforcement learning's Q-learning algorithm for navigation of a robot in an unknown environment such as the stockroom of a supply chain company. We are presenting a toy environment of a stockroom utilizing (3x3) cells where a robot can move freely anywhere bounded by the 4 walls. The gray cells are walls of packages, accessible by the robots however they are as well negatively rewarded. The white cells are where robots can freely move with minimal negative reward. We set up a minimal negative reward for the white cell so the robots are encouraged to reach the goal state as fast as possible. There are 5 actions robots can make: move up, move down, move left, move right, do nothing. There are 2 ramps in the environment, represented by the arrows, which provide shortcuts for the robots and they are also with minimal negative reward just like the white squares.

2 State transition function and reward function

The starting state in our toy example can be anywhere on the 3x3 grid, except terminal states.

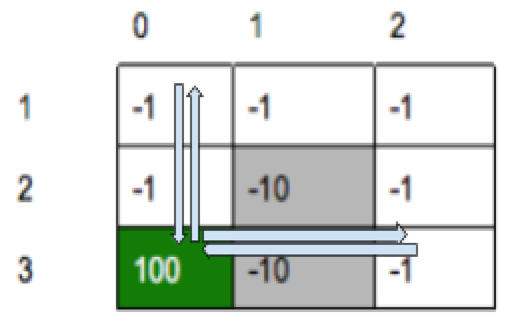


Figure 1. Toy stockroom environment with rewards at each state

Terminal state is where reward is 100 (the goal state). The robots are able to move freely to any adjacent cell, however no diagonal moves. The state transition function is defined by: $s_{t+1} = \delta(s_t, a_t)$ with s=state, a=action, t =time step

0	1	2
3	4	5
6	7	8

Figure 2. Toy stockroom environment - states

These are the valid transition states for every state:

0-> { 0,1,3,6}
1-> { 0, 1, 2,4}
2-> { 1, 2, 5 }
3-> {0, 3, 4, 6}
4-> {1, 3, 4,5,7}
5-> {2,4,5,8}
6-> {0,3,6,7,8}
7->{4,6,7,8}
8-> {5,6,7,8}

The reward function is defined by $r_{t+1} = r(s_t, a_t)$ and the values are in the R-matrix Figure 3.

3 Policy

The policy is defined by **π ($s_t = a_t$).** It is an epsilon-greedy policy which allows the robots to explore and exploit the stockroom environment. We first select an ε value between 0 and 1, and then we generate a random number between 0 and 1. If the random number is smaller than ε, the robot will explore the environment by randomly choosing one of the available actions. If the random number is higher than ε , the robot will exploit the environment by selecting the best action from the available actions that leads to the highest reward.
We also experimented with a decaying epsilon value in (section 1.8). This ensures that the policy changes from exploring, in the beginning, when there is no knowledge of the environment (Q table is zero), to exploiting slowly over time. For this the epsilon value will be multiplied by 0.99999 if ε ≥ 0.5 or by 0.9999 if ε < 0.5 to decrease its value slowly at every time step.

4 Graphical representation and R-matrix

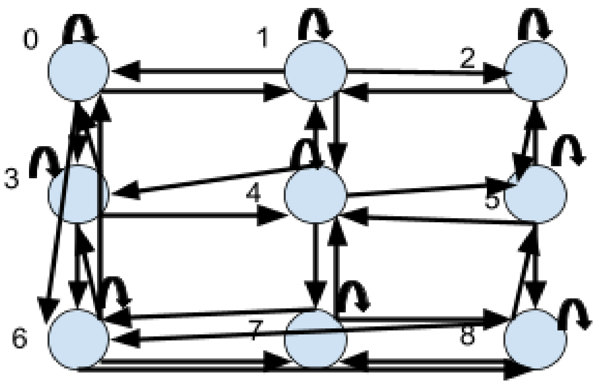
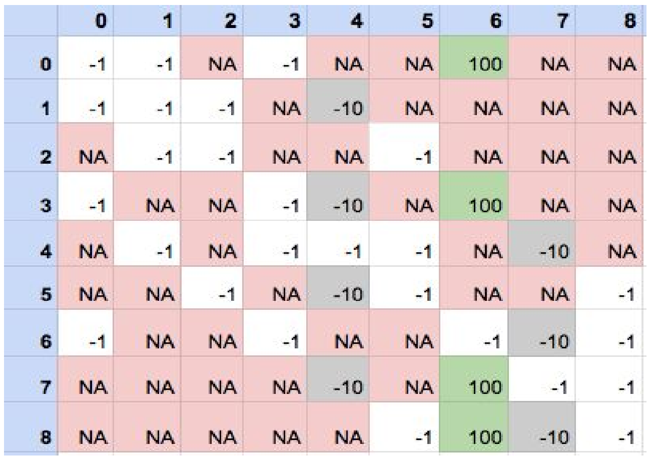


Figure 3. Graphical representation of valid states



	0	1	2	3	4	5	6	7	8
0	-1	-1	NA	-1	NA	NA	100	NA	NA
1	-1	-1	-1	NA	-10	NA	NA	NA	NA
2	NA	-1	-1	NA	NA	-1	NA	NA	NA
3	-1	NA	NA	-1	-10	NA	100	NA	NA
4	NA	-1	NA	-1	-1	-1	NA	-10	NA
5	NA	NA	-1	NA	-10	-1	NA	NA	-1
6	-1	NA	NA	-1	NA	NA	-1	-10	-1
7	NA	NA	NA	NA	-10	NA	100	-1	-1
8	NA	NA	NA	NA	NA	-1	100	-10	-1

Figure 4. Toy stockroom R-matrix

5 Parameter values for Q-learning

There are two parameters in the Bellman equation of Q-learning algorithm, alpha which is the **learning rate** and gamma which is the **discount factor**.
The learning rate signifies the degree new information overrides old information. Higher alpha means the agent will learn faster.
The discount factor signifies how much value we give for future rewards. If gamma is closer to 0 the robots will only consider immediate rewards and when gamma is closer to 1 the robots will place equal importance on future rewards and immediate rewards in order to obtain long term high rewards.
Initial parameter values:

- alpha = 0.9
- gamma=0.9
- epsilon=0.9

6 Original Q-matrix and updates during learning

Episode 1 walkthrough:

500 episodes, alpha = 0.9
gamma = 0.9, epsilon = 0.1

Starting state is '0'

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0

Figure 5. Original Q-matrix

From state 0, we choose from the best actions(all are zero because the Q matrix is empty) to go to state 6 using the ramp.

Bellman equation:
Qnew(s, a) = Qold(s, a) + α[(R(s, a) + γ.argmax{Q(next s, all a)}- Qold(s, a)]

Qnew(0, 6) = Qold(0, 6) + α[(R(0, 6) + γ.argmax{Q(6, 0),Q(6,3),Q(6,6),Q(6,7),Q(6,8)}-Qold(0, 6)]
Qnew(0, 6) = 0 + 0.9[(100)+0.9.argmax{0,0,0,0,0}-0]
Qnew(0,6) = 90.0

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	**90**	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0

Figure 6. Updated Q-matrix

	0	1	2	3	4	5	6	7	8
0	87	70	0	89	0	0	100	0	0
1	89	79	70	0	68	0	0	0	0
2	0	79	70	0	0	79	0	0	0
3	89	0	0	88	69	0	100	0	0
4	0	71	0	89	0	71	0	0	0
5	0	0	62	0	62	79	0	0	89
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	63	0	100	89	80
8	0	0	0	0	0	78	100	80	89

Figure 7. Final Q-matrix generated with 500 episodes, alpha = 0.9 gamma = 0.9, epsilon = 0.1 . It correctly identifies the shortest paths from each state to each state.

7 Performance vs Episodes

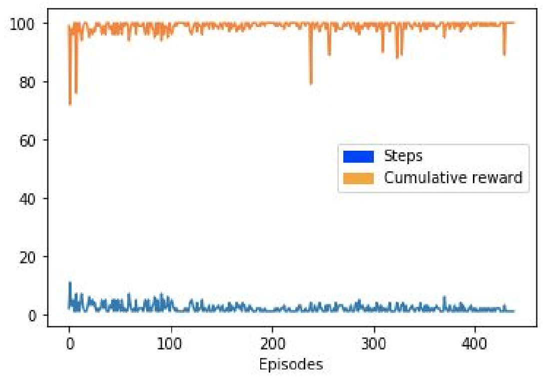


Figure 8. generated with 500 episodes, alpha = 0.9 gamma = 0.9, epsilon = 0.1

In Figure 8 we can see that the number of steps to complete an episode decreases around 100 episodes. Similarly the cumulative reward is higher on average beyond 100 episodes, to around 98-100.The Q-matrix leads to optimal paths.

8 Experiments with different parameters and policies

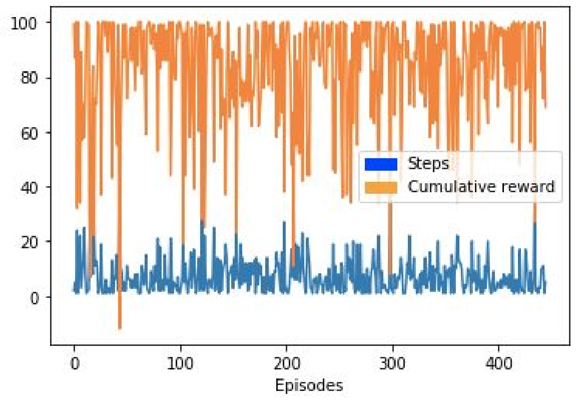


Figure 9 . Trial 1 γ = 0.90; α = 0.90; ε = 0.9 with 500 episodes

In figure 9 the trial converges to a correct q-matrix but we can notice large variation in the cumulative reward and number of steps taken. This is due to the randomness of the action taken because of the epsilon (explore). We used the decaying epsilon explained in section 1.3.

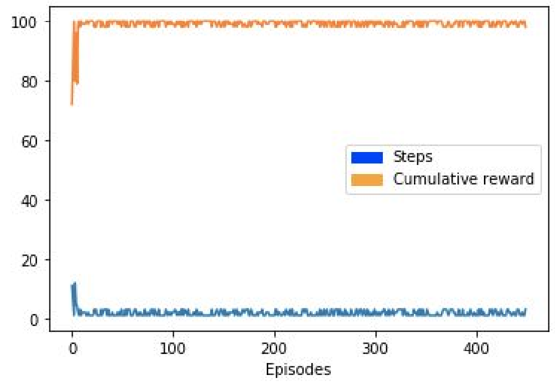


Figure 10. Trial 2 γ = 0.9 α = 0.9 ε =0 with 500 episodes.

In Figure 10 after just a few episodes there isn't much variation in the steps taken and cumulative reward. The q-matrix indicates that state 4 should go to 5, instead of 3 in order to reach the goal which is not the best path. We get this result due to being in the exploit mode always because of epsilon=0 and this result shows that in order to get a good q-matrix we need some amount of randomness in the actions (explore). 5000 episodes still lead to faulty q-matrix results.

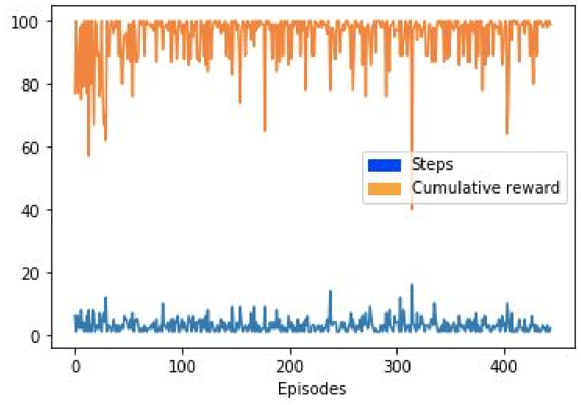


Figure 11. Trial 3 γ = 0.90; α = 0.90; ε = 0.5. with 500 episodes

This trial leads to a correct q-matrix. There is a balance between exploration and exploitation because of epsilon =0.5 though the number of steps are quite high per episode and the rewards are on average not maximised with a lot of them around 90.

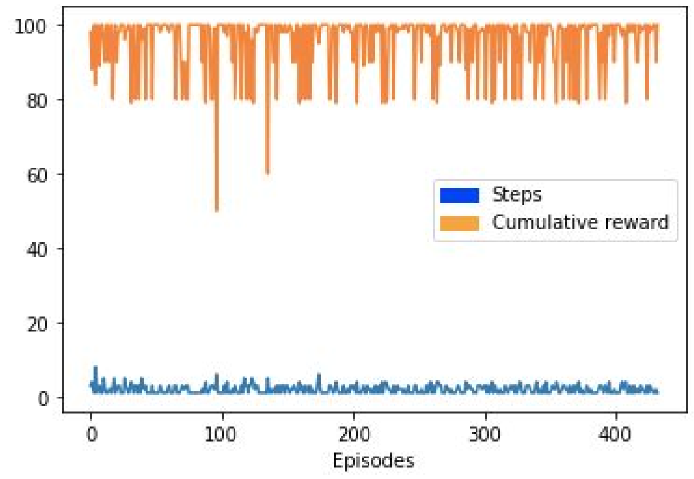


Figure 12. Trial 4 γ = 0.90; α = 0.20 ε =0.1

In Figure 12 if we reduce the learning rate we get average cumulative rewards between 80 and 100 with 500 episodes, the number steps to goal are minimal. The q-matrix is faulty with state 1 going to 4 instead of 0. State 4 going to 7, instead of 3. However after 5000 episodes the q-matrix is correct and our graph improved with rewards averaging between 90 and 100.

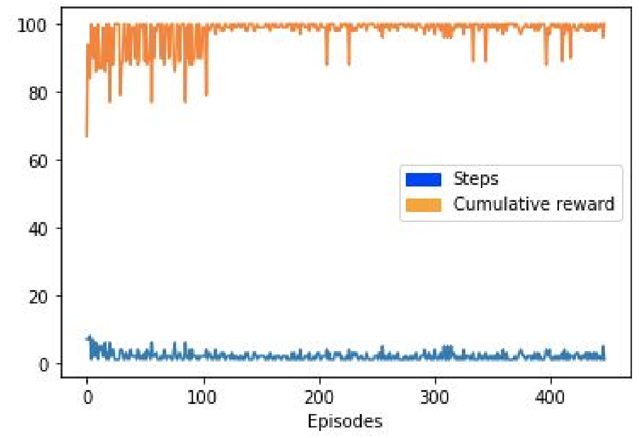


Figure 13. Trial 5 γ = 0.20; α = 0.90; ε =0.1.

In Figure 13 gamma is low and this means future rewards are considered less. The cumulative reward after around 100 episodes improves a lot averaging around 98-100. The q-matrix is correct in this particular case, however, there were 500 episode trials conducted that still lead to an non optimal q-matrix, therefore not consistent results with low gamma.

9 Quantitative and qualitative analysis

The analysis of the trials was conducted in section 1.8. Regarding epsilon value, exploitation needs some amount of exploration in order to obtain optimal results for the q-matrix. As shown in Trial 2 an epsilon that's too low (exploitation) can lead to non-optimal paths especially for larger state-spaces as not all states are explored even after a large number of episodes. One downside of epsilon-greedy strategy is that it depends on the value we select for epsilon. The algorithm Epsilon-BMC (Gimelfarb, Sanner and Lee, 2020) can be used to adapt epsilon using experiences from the environment in constant time.

Regarding the parameters of the Bellman equation, with a reduced learning rate, the learning improves very slowly and we need more episodes to obtain optimal results. If the discount factor is low immediate rewards are given more weight which also can lead to non-optimal results.

In terms of limitations, Q-learning algorithm only works when the environment has finite states and actions. If we combine Q-learning with function approximation, such as neural networks, we can use the algorithm to solve problems where state space is continuous. This is done in the following section of this paper.

2. Advanced Task

10. Advanced RL (motivations and expectations)

This section focuses on implementing Deep Reinforcement learning techniques to solve an OpenAI gym Atari 2600 environment (OpenAI, n.d.). OpenAI Gym is an open source reinforcement learning environment. It provides a range of environments with a common baseline to be used on various reinforcement learning algorithms.

Deep Q-Network (DQN) (Mnih et al., 2013) and Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2015), were implemented on Space Invaders Atari game, in which the agent tries to maximize the reward by shooting and destroying the aliens that appear from the top of the screen and preventing them to reach the ground level i.e. the level of the agent. If the aliens reach the bottom, then the agent dies and the game ends.

Environment: The Space Invaders environment consists of 6 actions: no-op, fire, right, left, right-fire, left-fire. Our observation space is an RGB image of screen i.e. consists of frames of size {210,160,3}, which are preprocessed and stacked before passing through the convolutional layer.

The environment is partially observable since the agent observes one frame (stack of 4 frames) at one time-step and is stochastic in nature as each state-action pair is random.

10.1 Methods

10.1.1 Deep Q Network (DQN)

DQN is a combination of Q-learning and neural network. It is an off-policy model-free method, where the agent has no prior knowledge of the environment. It uses raw pixels as input to neural network (Policy Network) to approximate the action-value function:

$$Q^{*}(s,a) = max_{\pi}\ E[R_t | s_t = s,\ a_t = a,\ \pi]$$

$$Q = r + \gamma\ max\ Q\ (s', a')$$

Discount factor: $\gamma = 0.99$ Policy: $\pi = P(a|s)$

However, it is still not guaranteed to converge, and the network weights may diverge or oscillate. This drawback of the DQN makes it unstable and ineffective, therefore, we use Experience replay and Fixed Q target network.

Experience Replay/ Replay Buffer: It stores the experiences of the agent at each time-step in the form of a tuple, and then randomly selects the given experiences. Randomization is important to remove correlations in the consecutive samples of observation space and smoothly reflect changes in data sampling, making the training data diverse. It makes the learning process more efficient.

In practice, the replay buffer has a finite memory to store the experiences, therefore, after the maximum memory location is reached the buffer removes the first tuple and adds the most recent one. For our experiment, we fixed the buffer size to 500000 and initialized the buffer at 50000.

Epsilon greedy strategy (as explained in section 3)**:** The value of epsilon varies between 1.0 and 0.1 and the agent explores for 250000 frames before exploiting the environment.

Frame skipping: Instead of considering all the input frames, the agent executes the action after every kth

frame. This permits the agent to play k more games in the given time span. (Mnih et al., 2015). For this environment, we experiment with k = 3 and k = 4.

Preprocessing: The input to the neural network is in the form of images. These images are the frames of the Atari game i.e. Space Invaders. These frames are preprocessed by first converting them to grayscale images and then scaling them down to 80x80 and then 42x42. Hence the input to the neural network becomes 42x42x1 (HWC). 4 consecutive frames were stacked before passing through the convolutional layer.

Policy network and Target network: Target network prevents the action-values from oscillating and the algorithm from diverging. Hence making the training and learning process more stable. The weights of the target network are updated after every 5000 episodes.

Both the network architectures consist of 3 convolutional layers and 2 linear layers with 512 units in the hidden layer, followed by ReLu nonlinearity.

10.1.2 Asynchronous Advantage Actor Critic (A3C)

Description: It is an action-based method that learns a policy and a value through function approximator. Action-based methods are more stable than value-based methods and require fewer samples than policy-based networks. It is used for both continuous and discrete action spaces.

Asynchronous: It refers to the multiple agents that run in parallel, asynchronously to explore and share information. Since all the agents share the same network, therefore they also share common weights.

Advantage: It gives an estimate on how well a particular action performs as compared to the others by comparing the Q value of an action to the value of being in a particular state. It can be calculated by:

Advantage: $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$

Actor-Critic: Since it is a policy gradient method, it has two loss functions to approximate the policy (Actor) and action-values (Critic). The value estimated by the Critic is used to update the policy by the Actor.

Actor: $(R - V(s; \theta_v))\, log\, \pi(a|s; \theta)$

Critic: $(R - V(s; \theta_v))$

Preprocessing: The input frames were preprocessed by scaling down and resizing to 80 x 80 and then to 42 x 42. They were then converted to grayscale and normalized before passing through the convolutional layers.

A3C Architecture: We use convolutional layers for feature extraction directly from the input frame. Our model consists of 4 convolutional layers (stride 2 and padding 1). Last layer and the weights were changed according to the addition of the following layers:

A3C FF: A linear layer was added at the end of the network. All the layers in the network were followed by ReLU nonlinearity.

A3C LSTM: LSTM layer was added at the end in place of the hidden layer. All the layers in the network were followed by Elu (Exponential Linear Units) nonlinearity, which does not have the dying ReLu problem. The LSTM Cell performs the task of storing memory and can suggest about the previous state the agent was in.

10.2 Motivation and expectations: Based on the disadvantages of previously implemented basic algorithm for our advanced environment:
Limitations of Q-learning: It can be implemented on discrete state space involving discrete actions.

Our environment is continuous, hence applying Q learning on discretization of the state spaces may lead to inefficient learning.

Advantages of DQN over Q learning: Uses a neural network as a function approximator, which generalizes previous experiences to unseen states. It also uses experience replay and target network (section 2.1).

Limitations of DQN: 1. Experience replay consumes more memory and computational power.
2. Since it is an off-policy method, it updates the Q values using older data.

Advantages of A3C over DQN: 1. The agents work in separate environments and can periodically update global parameters. Since the agents run in parallel, they can share information and explore in different ways.
2. It runs on CPU, hence, eliminating the need for GPU power.
3. It does not use experience replay and instead uses parallel memory, which stabilizes the training process, indicating that a replay buffer is not always necessary.
A3C was also chosen based on its comparison with other methods in the original paper(Mnih et al., 2015).

11. Quantitative Analysis

DQN:Ablation Study: Comparing the performance of different parameters and then displaying parameters for the best results.

1. Batch Size

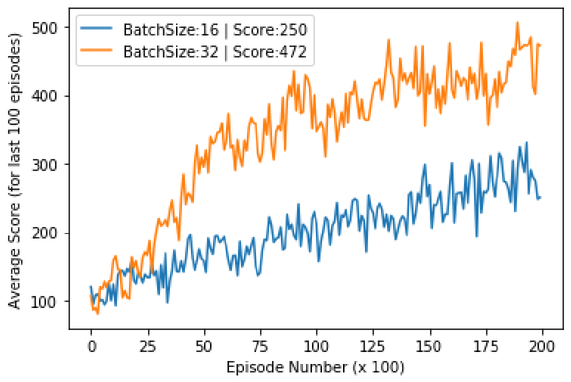


Figure 14: Agent’s performance with different batch sizes. The learning is noisy and rewards are fluctuating.

2.Optimizers:

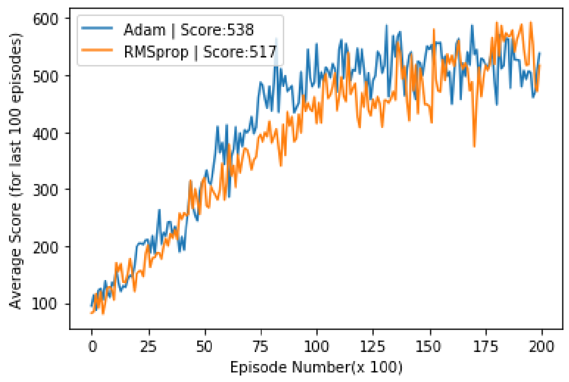


Figure 15: Plotting the graphs of different optimizers with same learning rate.

3. Skip Frame

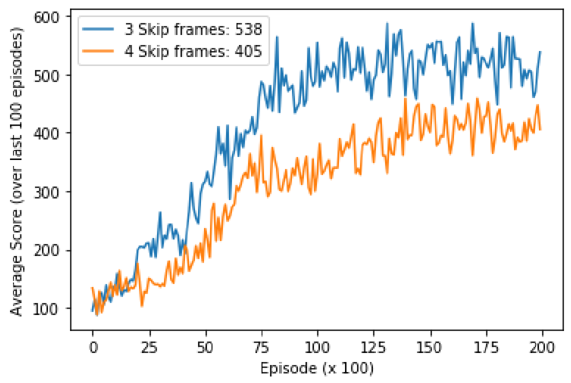


Figure 16: Comparing the avg. score received when the agent skips 3 and 4 frames

4. Loss Functions:

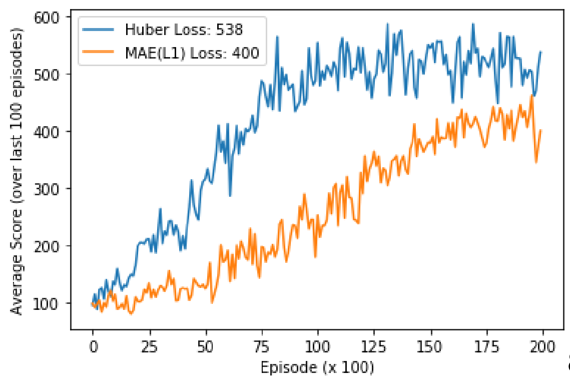


Figure 17: Comparison of the results of two loss functions: Huber and Mean Absolute Error.

Best Runs

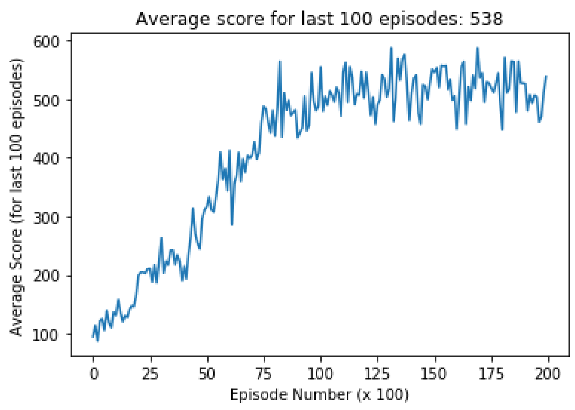


Figure 18: Best Run: It does not converge but reward is high

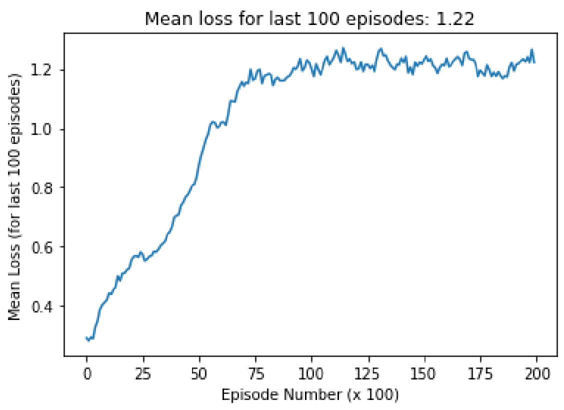
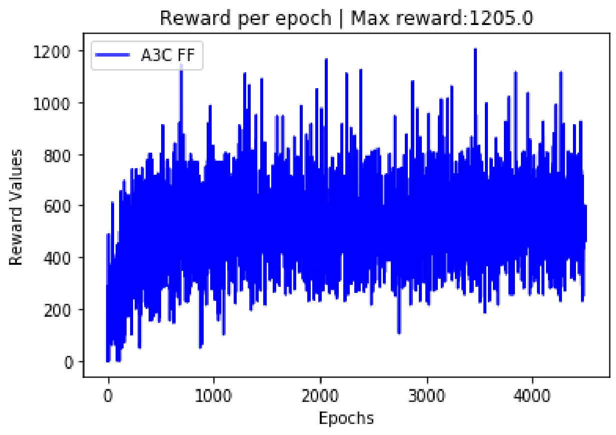


Figure 19: Best Run: Mean loss - Not a good indicator

Maximum episodes	Batch size	Skip frame	Optimizer	Learning Rate	Alpha	Gamma	Loss Function
20000	32	3	Adam	0.0001	0.95	0.99	Huber Loss

11.2 A3C



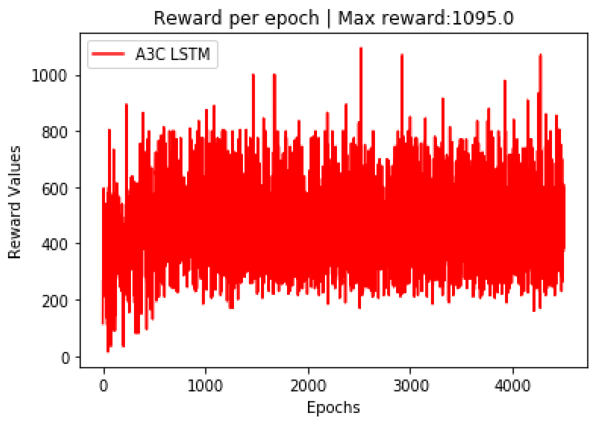


Figure 20. A3C FF rewards graph for max. 10000 episodes
Learning is noisy and the model does not converge

Figure 21. A3C FF rewards graph for max. 10000 episodes.
Model does not converge, however the reward is not zero anymore

	Maximum episode length	No. of Steps	Optimizer	Epsilon	Learning Rate	Gamma	Epochs	tau
LSTM and FF	10000	20	Adam	1e-8	1e-3	0.99	4500	1.

12. Qualitative Analysis

12.1 DQN

Batch size: 16 vs 32: The batches contain randomly sampled experiences from the replay buffer. As Fig.14 suggests, batch size of 32 gave higher average reward(472) as compared to batch size of 16 i.e. 250.Batch size of 16 gave lower rewards since a small batch size makes the performance noisy and gradient estimates less accurate. It may lead to slower convergence as well (in case of more trials) (Haleva, 2020), hence, decreasing the average reward value.

A batch size larger than 32 was not used due to memory constraints.

Optimizer and Learning rate: The performances of both the optimizers i.e. Adam and RMSProp were similar.

As Fig.15 indicates, RMSprop seems to perform better in the middle of the training process. However, at the end of 20000 episodes, Adam gave a mean average score of 538, which is greater than that of RMSprop i.e. 517. Despite comparable results of both the optimizers and use of RMSprop in Deepmind's paper (Mnih et al., 2013), we chose Adam for the final model.Since a lower learning rate helps the model to converge faster, the learning rate of 0.0001 gave better results as compared to 0.00025.

Frame skip: 4 vs 3: Although 4 frame skipping speed up the training time, Space Invaders perform better with 3 frame skipping (Fig16). This is due to the fact that skipping 4 frames made the lasers invisible due to their blinking interval(Mnih et al., 2013).

Loss Function: Huber loss (smooth L1 loss) vs MAE loss(L1_loss) (Fig.17)

The maximum reward was found to be with Huber loss (538). Reward by using MAE loss was 400. Since Huber is squared at errors below a threshold (usually around 1), therefore the loss is differentiable at zero. Whereas MSE is not differentiable at 0.

MSE loss was also implemented, however, the loss value was too large (48.61) in this case. Huber and MAE loss do not penalize the outliers heavily, whereas MSE does. Since MSE tried to keep the variance at a minimum, therefore it did not perform well. Although loss function helps increase the reward, the graphs indicate that the loss per episode graph is not a good indicator of the trained model. In reinforcement learning, the loss is not always expected to decrease.

12.2 A3C (Fig. 20 and Fig. 21)

Although both the graphs are noisy, i.e. the model does not converge, the results show that A3C LSTM performs better than A3C FF network. According to our expectation, A3C LSTM should have outperformed A3C FF, since LSTM plays an active role in memory sharing. Observing more epochs might solve this issue.

12.3 DQN and A3C

According to Fig. 20 and 21, both A3C FF and A3C LSTM outperformed DQN. Since A3C uses parallel training it is able to collect and share more diverse data (since the experiences of all the agents are different), increasing the efficiency of the training process (Juliani, 2017).

According to our expectations, A3C should have been easier to train. But since it does not use GPU and operates on CPU using multiple processors, it was difficult to train. It took 5 days to achieve the given reward value, using 8 processors. Whereas, DQN was trained in 2 days using GPU. It is also believed to have faster convergence rate since multiple processes run in parallel but the desired results were hard to achieve in both the algorithms.

Conclusion and Future Work

The above experiment indicates that the A3C algorithm outperforms DQN on the Space Invaders environment. DQN method can be further improved by: prioritizing the experience replay; using double, dueling or rainbow DQN. Normalizing, aggregating and re-scaling the outputs is another method. Clipping the rewards between 1 and -1 may improve the performance. A3C can be improved by running more agents in parallel, i.e. using better CPU for training and increasing the number of episodes. Other policy gradient methods like Proximal Policy Optimization (PPO), TRPO can also be used.

3. References

1. Gimelfarb, M., Sanner, S. and Lee, C.-G. (2020). ε-BMC: A Bayesian Ensemble Approach to Epsilon-Greedy Exploration in Model-Free Reinforcement Learning. [online] Available at: http://auai.org/uai2019/proceedings/papers/162.pdf [Accessed 21 Aug. 2020].

2. Techopedia.com. (n.d.). *Reinforcement Learning Vs. Deep Reinforcement Learning: What's the Difference?* [online] Available at: https://www.techopedia.com/reinforcement-learning-vs-deep-reinforcement-learning-whats-the-difference/2/34039 [Accessed 3 May 2020].

2. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, [online] 518(7540), pp.529–533. Available at: https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf.

3. Kostrikov, I. (2020). *ikostrikov/pytorch-a3c*. [online] GitHub. Available at: https://github.com/ikostrikov/pytorch-a3c [Accessed 3 May 2020].

4. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning*. [online] Available at: https://arxiv.org/pdf/1312.5602.pdf.

5. OpenAI (n.d.). *Gym: A toolkit for developing and comparing reinforcement learning algorithms*. [online] gym.openai.com. Available at: https://gym.openai.com/envs/SpaceInvaders-v0/ [Accessed 21 May 2020].

6. Juliani, A. (2017). *Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C)*. [online] Medium. Available at: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2 [Accessed 20 Jun. 2020].

7. Haleva, R. (2020). *How to Break GPU Memory Boundaries Even with Large Batch Sizes*. [online] Medium. Available at: https://towardsdatascience.com/how-to-break-gpu-memory-boundaries-even-with-large-batch-sizes-7a9c27a400ce [Accessed 17 Aug. 2020].