EXPLORING DEEP RL ALGORITHMS

CSE 4/546 REINFORCEMENT LEARNING

Team members:

Anuja Raghunath Katkar

Sagar Jitendra Thacker

University at Buffalo
School of Engineering and Applied Sciences

Project Description

- In this project, we propose to explore different deep reinforcement learning algorithms developed over the years on environments provided in OpenAI gym library.
- We have compared the performance of these algorithms in each of the environment to better understand how the algorithms affects the agent's behavior in those environments.

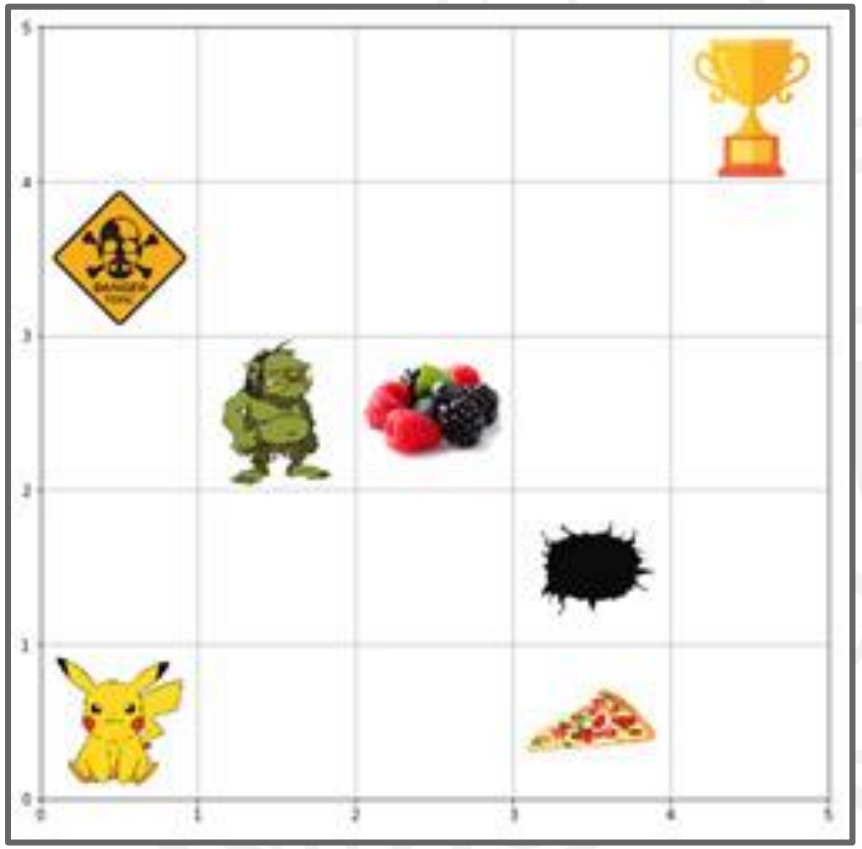
Algorithms used:	Environments used:
1. DQN	1. Grid World
2. Double DQN	2. Cartpole-v1
3. A2C	3. Acrobot-v1
4. Dueling DQN	4. BreakoutDeterministic-v1

Environment Description

1. Grid World Environment
   - State set: 25 states
   - Action set:

     A={Up, Down, Left, Right}
   - Reward set:

     R={-1,-3,5,10,-50,50}
   - Termination Condition:
     - Agent reached the trophy (winning state)
     - Agent fell in pit or met the monster (dead state)
     - Timesteps greater than 40
   - Solved Condition:
     - Reached the trophy with reward as 54

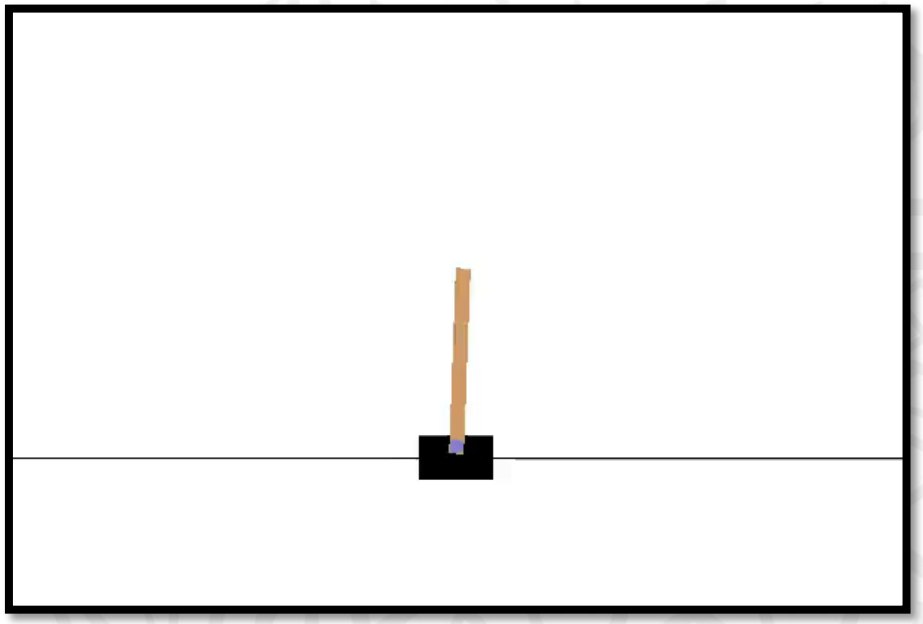


Environment Description

2. Cartpole-v1 Environment
   - Observation: {Cart Position ,Cart Velocity, Pole angle, Pole angular velocity}
   - Action set:

     A={Left, Right}
   - Reward set: for each step,

     R={+1}
   - Termination Condition:
     - Pole angle > ±12°
     - Cart position > ± 2.4 (center of the cart reaches the edge of the display)
     - Episode length > 500
   - Solved Condition:
     - Average return >=475.0 over 100 consecutive trials.

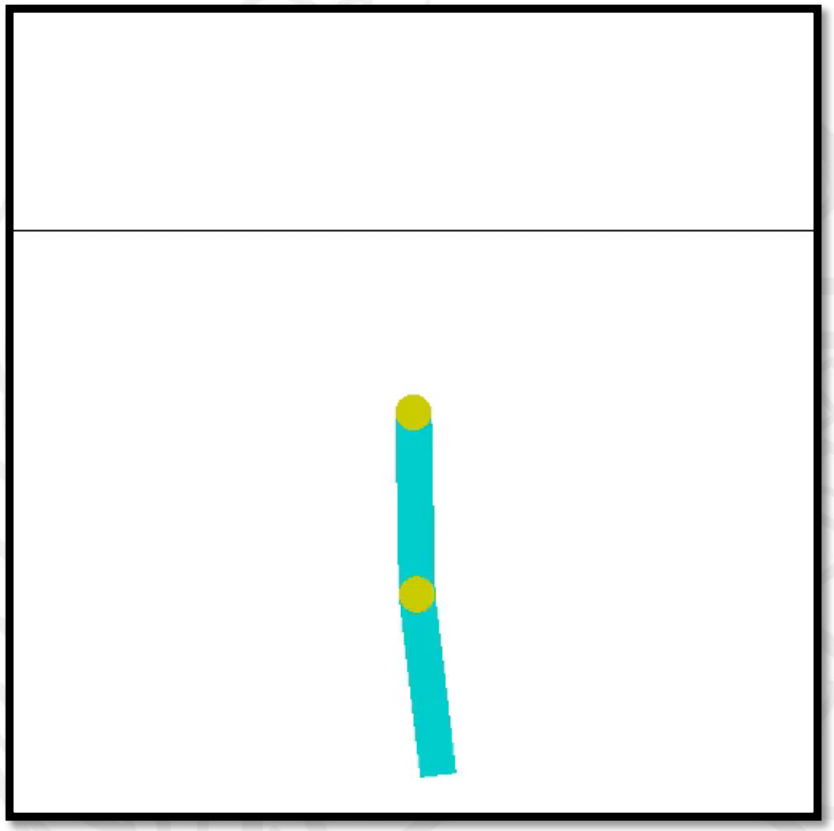


Environment Description

3. Acrobot-v1 Environment
   - Observation: {cost(theta1) sin(theta1) cos(theta2) sin(theta2) thetaDot1 thetaDot2}
   - Action set:

     A={+1,0,-1}
   - Reward set: for each action,

     R={-1}
   - Solved Condition:
     - If the average reward for 100 episodes is greater than -80 then it is considered the environment is solved and the agent has learnt.

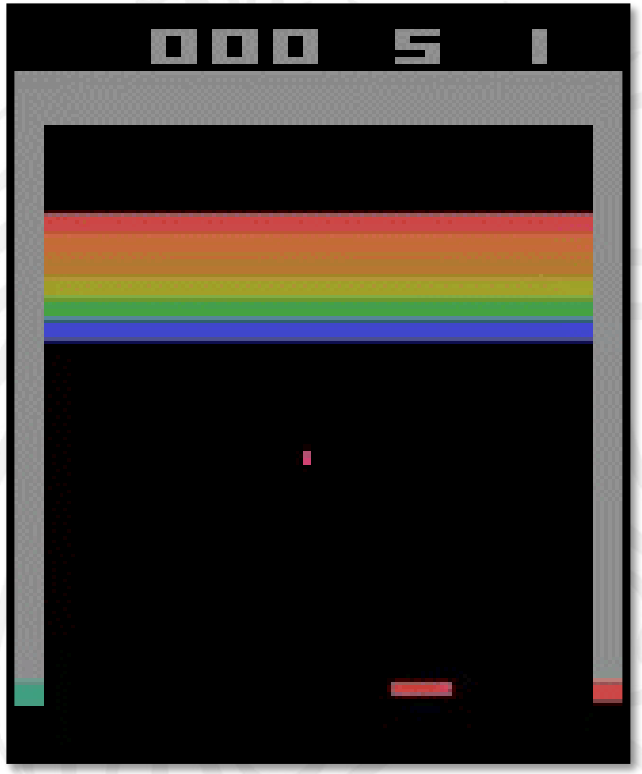


Environment Description

4. BreakoutDeterministic-v1 Environment
   - Observation: The observation is an RGB image of the screen, which is an array of shape (210, 160, 3)
   - Action set:

     A={Left, Right, Noop, Fire}
   - Reward set: for each action,

     R={+1}
   - Termination Condition: The episode terminates after losing 5 turns.



Algorithm Implemented

- Deep Q-Network:
  1. It is a value-based algorithm
  2. In DQN, we use experience replay to break the correlation of the data at the same time It is used to learn from past actions
  3. Here, we freeze the target Q-network to reduce variance in the Q-values which used to happen by slight changes in the policy.
  4. To converge in a faster way with optimal policy we clip the rewards with a certain range.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

End For

Algorithm Implemented

- Double Deep Q-Network
  1. It is a value-based algorithm.
  2. This algorithm is like DQN the key difference is we use same architecture to derive policy and target network without requiring extra parameters and networks.
  3. It uses the target network to reduce the overestimation.
  4. More stable results than DQN.

Algorithm 1: Double DQN (Hasset et al. 2015

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau < 1$, C
for *each iteration* **do**
  for *each step* **do**
    Observe state s_t and select $a_t \sim \pi(s_t)$
    Execute action a_t and observe next state s_{t+1} and reward r_t;
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$.
  end
  for *each update step* **do**
    Sample minibatch of transitions $(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q value:

$$Q^*(s_t, a_t) = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$$

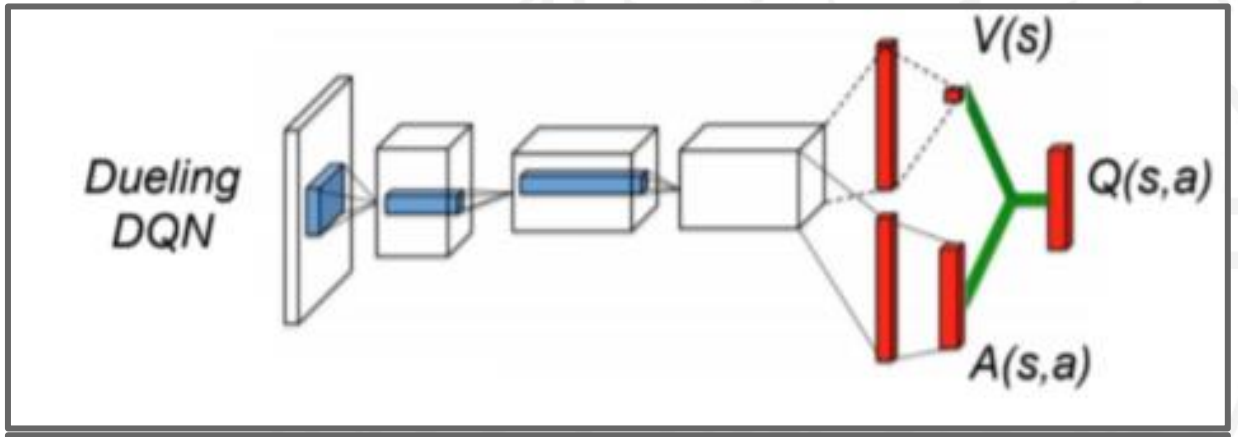
    Perform a gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$ with respect to the primary network parameters θ
    Every C steps update target network parameters:

$$\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$$

  end
end

Algorithm Implemented

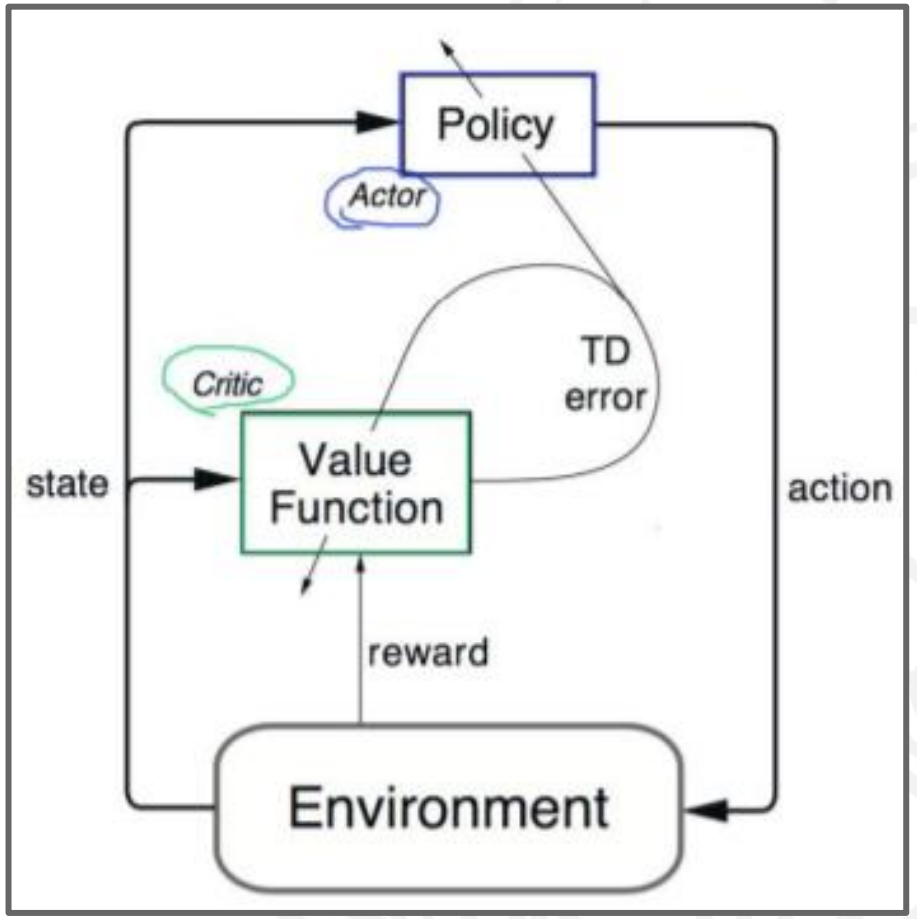
- Dueling Deep Q-Network
  1. It is a value-based algorithm.
  2. This architecture represents both the value function and advantage functions with a single network whose output aggregates both to produce a Q-value for a state-action pair.
  3. In this algorithm the agent can learn if a state is good or bad without having to learn q-value of each action in each state.



$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

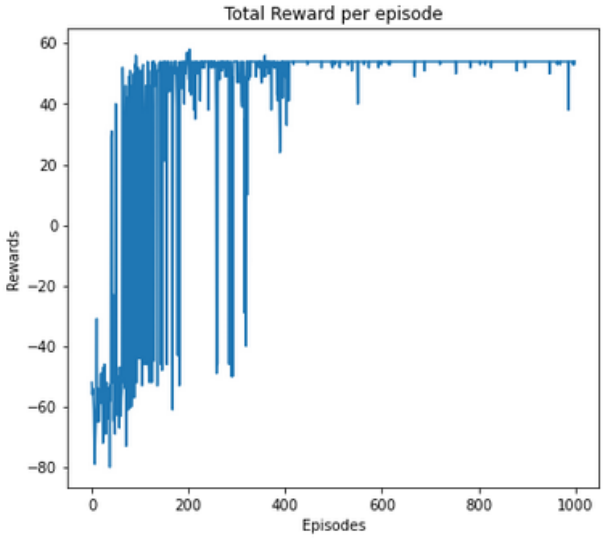
Algorithm Implemented

- Advantage Actor Critic
  1. It is an actor critic algorithm.
  2. It uses advantage function.
  3. It learns both policy as well as value function.
  4. This algorithm uses two different function approximators with two different parameters.

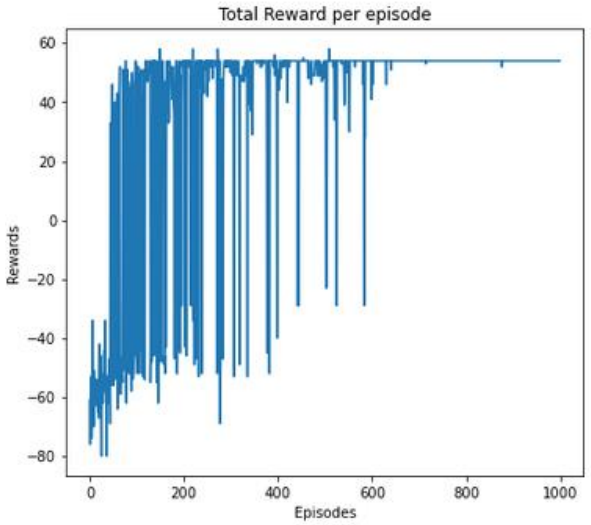


Results of Grid World environment

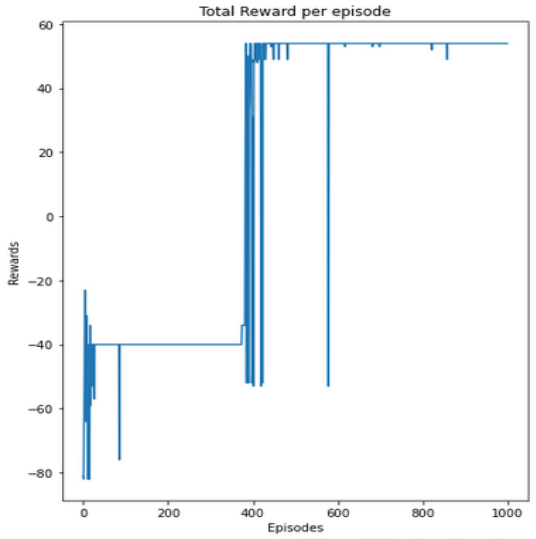
In the below graphs we compared the results of all the four algorithms on grid world environment. Grid World being the simplest environment all the algorithms are able to converge faster without need of hyper parameter tuning or excessive time to train. All the four algorithms were able to produce stable results and solve the environment.



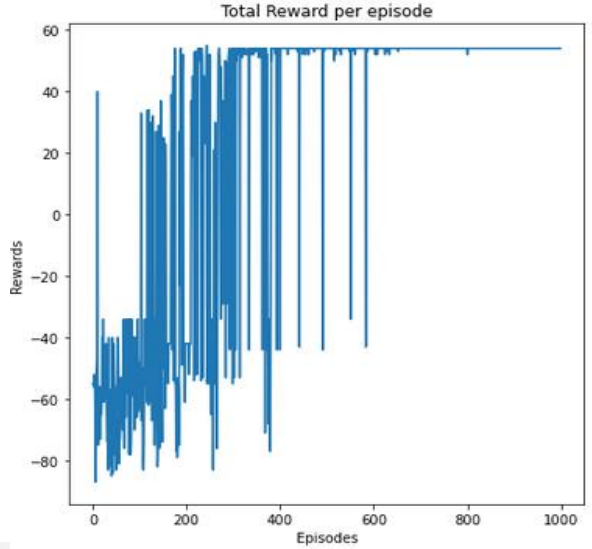
DQN



Double DQN



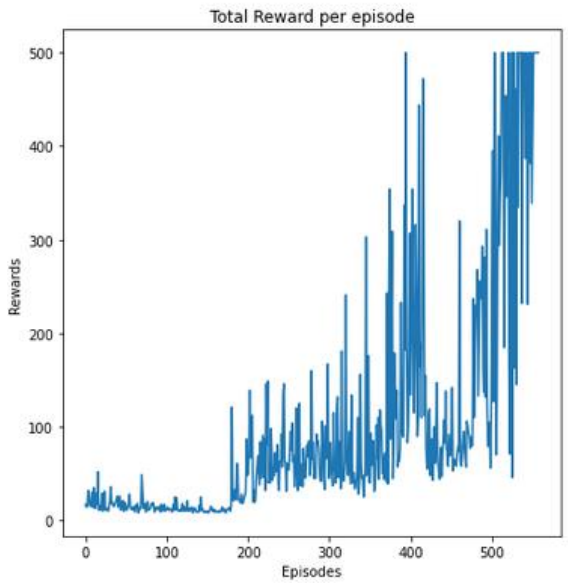
A2C



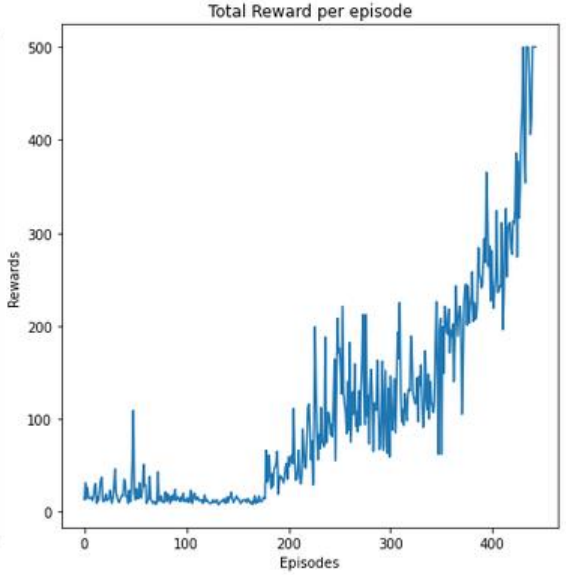
Dueling DQN

Results of Cartpole-v1 environment

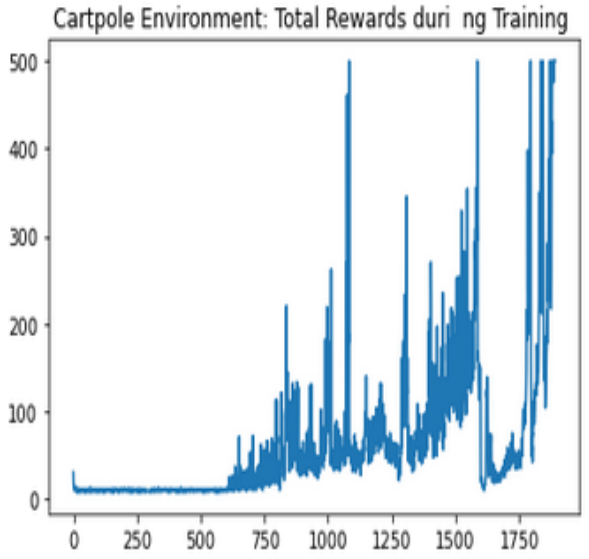
In the below graphs we can observe that agent was able to receive the maximum reward using each algorithms. Compared to DQN, Double DQN produced gradual and stable increase in rewards and solved the environment. But in A2C and Dueling DQN we can observe that the agent could reach the maximum reward multiple times but could not maintain the same. Nonetheless all the four algorithms are able to converge and find the optimal policy.



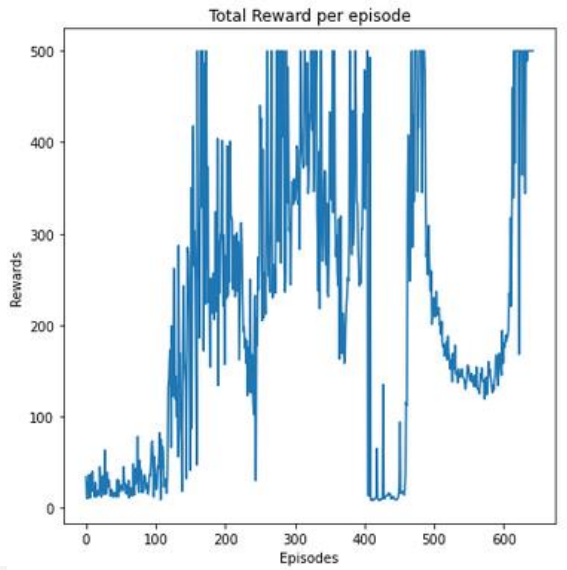
DQN



Double DQN



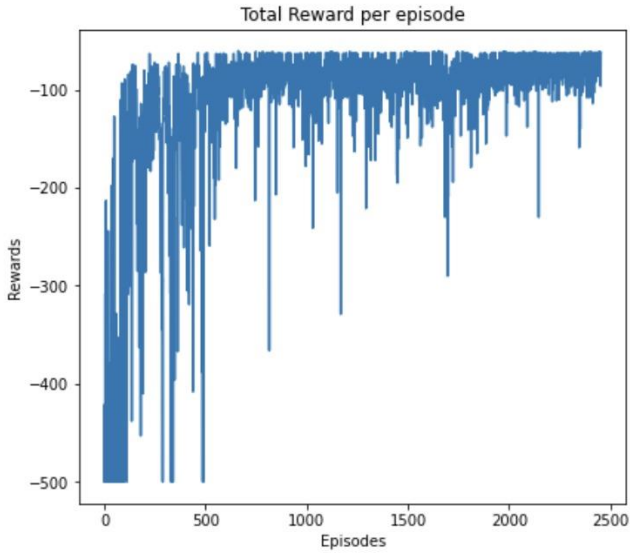
A2C



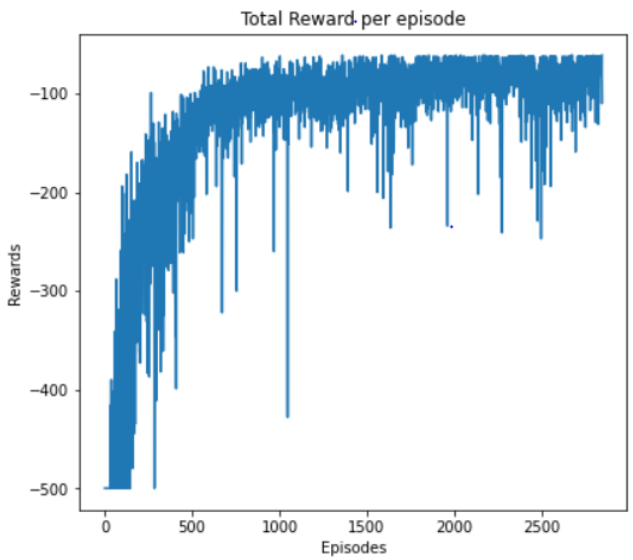
Dueling DQN

Results of Acrobot-v1 environment

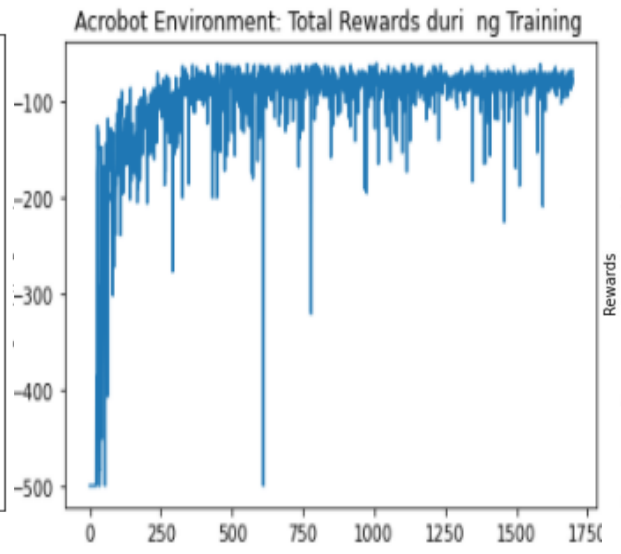
Out of all the algorithms we can observe that A2C has converged better than any other algorithm with fewer fluctuations. This could be due to the fact that DQN family algorithms which are value based and use maximization operation to find the best action which leads to the maximization bias issue. A2C learns the policy and the value function both, which might help to learn the optimal policy in a better way than only calculating the value function.



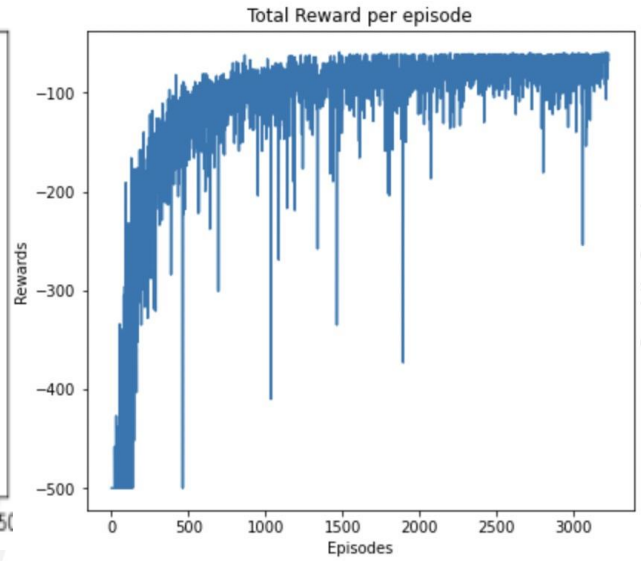
DQN



Double DQN



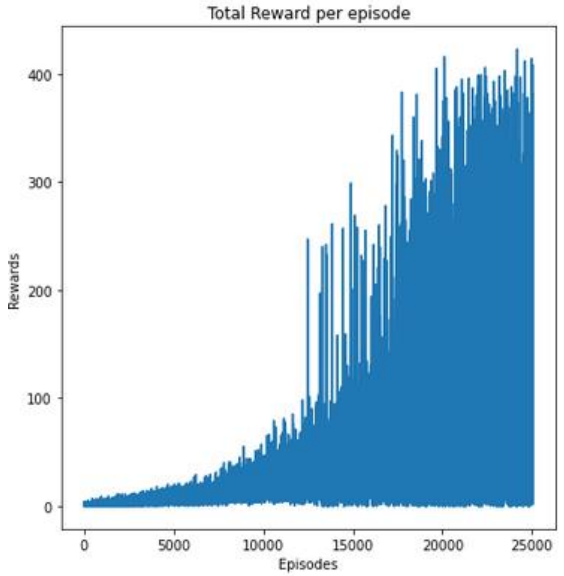
A2C



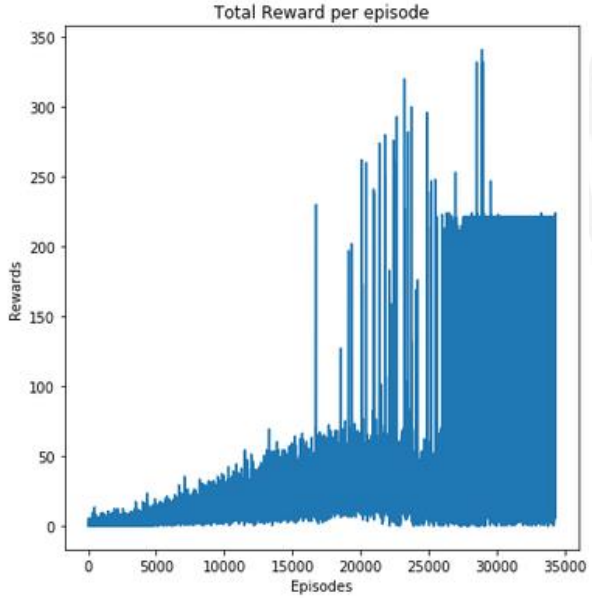
Dueling DQN

Results of Breakout environment

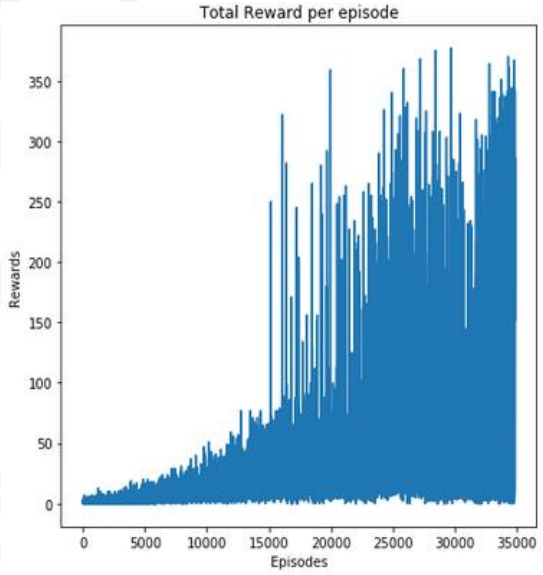
Compared to all the algorithms Double DQN does stabilize to a degree but often drop down. The graph for all gradually increases towards the reward we would like i.e., above 200. It is very difficult to try so many permutations but upon trial of tuning various parameters expect the batch size we tried to tune others and make the agent learn with maximum reward. Hence, hereby we conclude the agent would perform better and stabilize with trial higher batch size since these same algorithms are able to converge on other environments as well.



DQN



Double DQN



Dueling DQN

Technical Difficulties

- Primarily we faced issues with hardware limitations where we got "*Cuda Out of Memory*" error due to lack of GPU memory.
- Algorithms on complex environment like Breakout requires multiple trials to tune parameters which cost more time and resources
- A2C on breakout gave us "*cuda out of memory*" error even for small batch size.
- 24-hour session on Google Colab pro+ and limited access to CCR were factors contributed to time constraints for solving complex environment like Breakout.

Summary

- Deep reinforcement learning algorithms have enabled us to explore and solve complex problems
- Algorithm performance differs not only on different environment but even in the same environment
- As the environment gets more complex the tuning of the parameters play a crucial role on the performance of the agent.
- Normally we can see that Double DQN performance better than DQN on any of the three environments and produces much stable results.
- A2C and Dueling DQN are at par with the other two but take little more time and resources to train.
- On complex environment like breakout Dueling DQN starts to reach the optimal reward quicker than DQN and Double DQN.

Contribution Summary

Team Member	Contribution
Anuja Raghunath Katkar	50%
Sagar Jitendra Thacker	50%

Citations

- Lecture slides
- Reinforcement Learning 2nd Edition book

Thank you!

Any questions?