Q-Learning vs. DQN

Comparing learning algorithms on Atari game Space Invaders

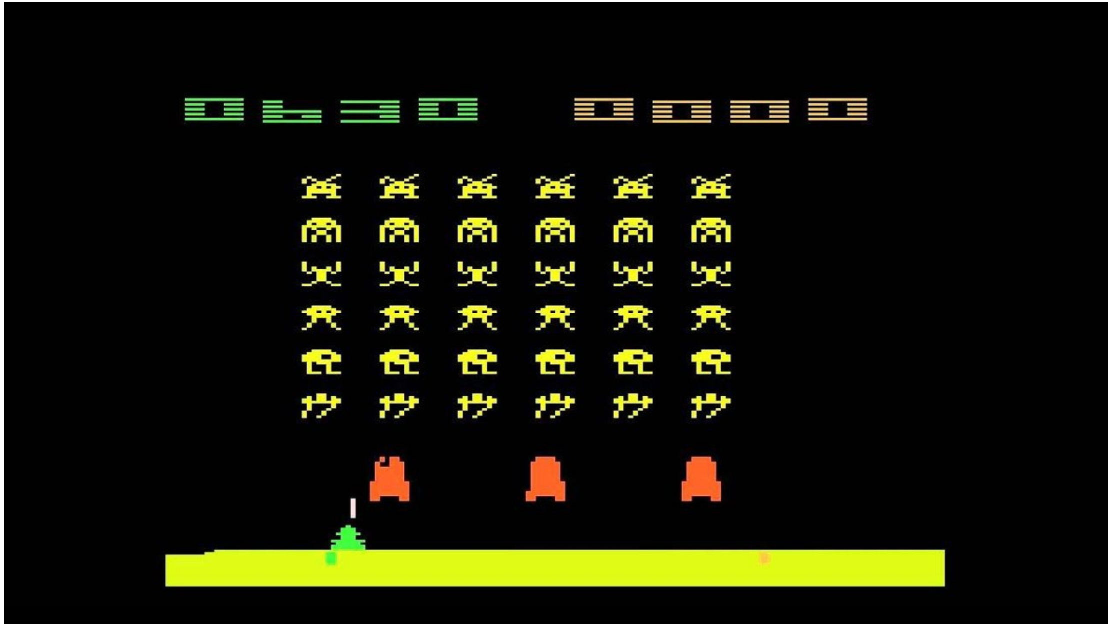
By Divyam Garg & John DiNofrio

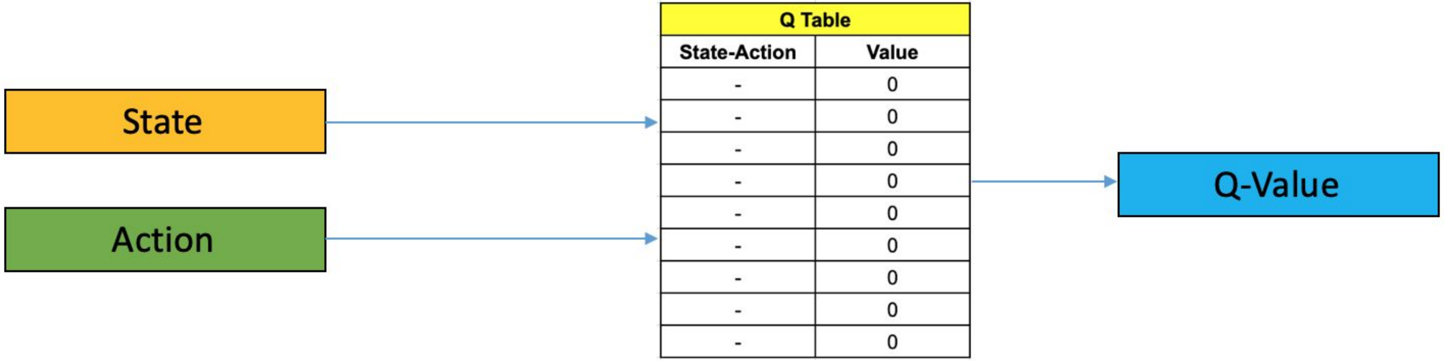
Reinforcement Learning

- Reinforcement learning is one of the three most basic machine learning techniques
- Unlike supervised and unsupervised learning
  - It does not require labeled data
  - It does not need to directly correct every action
- By setting positive and negative rewards, the program can teach itself how to play games or perform actions

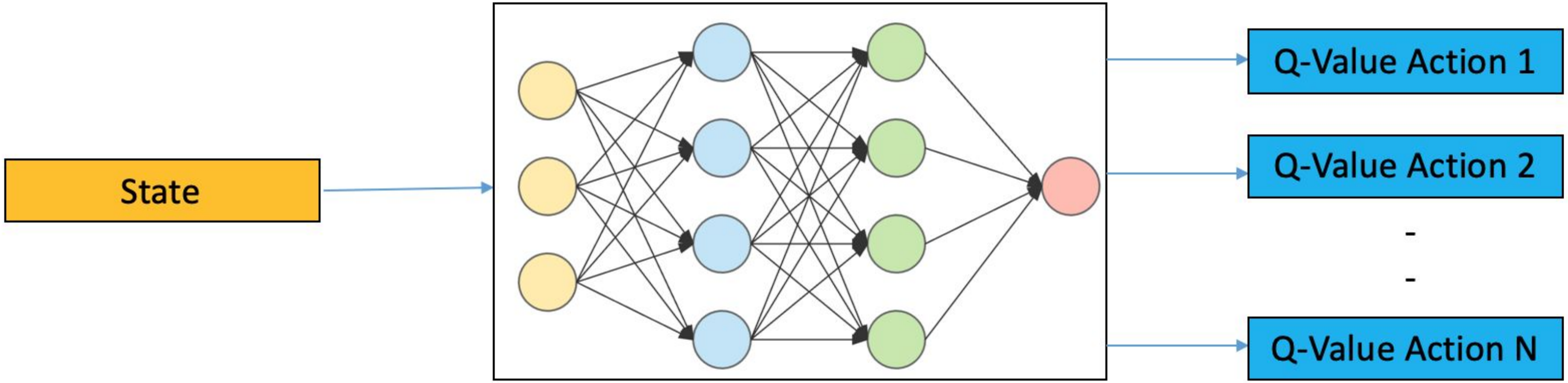
Space Invaders

- We used it to teach the computer how to play the Atari game **Space Invaders**
- We used both Q-Learning and DQN to demonstrate this and compare how the two methods worked





Q Learning



Deep Q Learning

Q-Learning

- Q-Learning is quite simple to implement
- The entire algorithm depends on an array called a **Q-Table**
- This Q-Table is n x m dimensions
    - Where n is the number of actions possible
    - Where m is the number of states possible
- For every current state and action that led it there, the value in the Q-Table will change
    - The value will go up if the reward was positive for that action
    - The value will go down if the reward was negative
    - The value could go up or down if there is no reward - this depends on the parameters you set

Q-table initialised at zero

	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

After few episodes

	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0

Eventually

	UP	DOWN	LEFT	RIGHT
0	0	0	0.45	0
1	0	1.01	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0

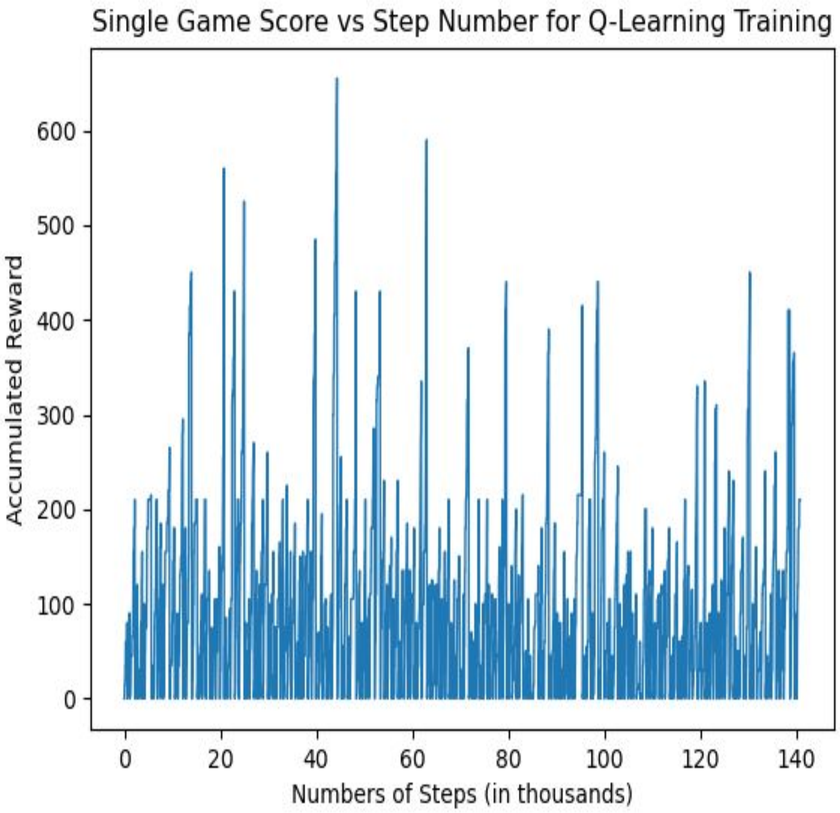
Q-Learning

- For both Q-Learning and DQN, there are certain parameters you can set
  - **Learning rate** - how much new data overwrites old data
    - 0 = no learning   1 = only new info is retained
  - **Discount rate** - short term or long term goals
    - 0 = short-sighted   1 = far-sighted
  - **Exploration rate** - how often the program tries random, new actions
    - 0 = never   1 = always new and random
  - **Number of episodes** - the amount of attempts it gets (how many games)
  - **Number of steps** - the number of actions it can take per episode

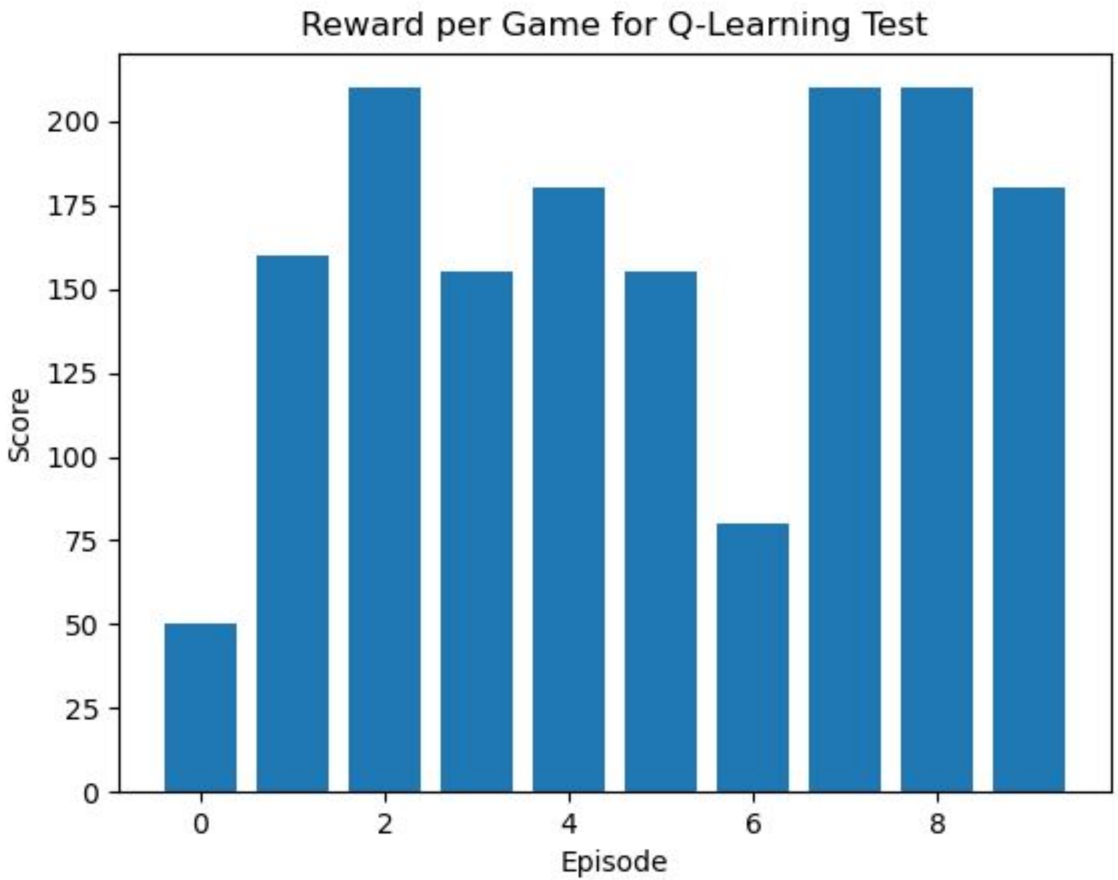
Q-Learning on Space Invaders

- **Learning rate** - 0.3
- **Discount rate** - 0.7
- **Exploration rate** - Started at 1 and exponentially went down
- **Number of episodes** - the amount of attempts it gets (how many games)
- **Number of steps** - Max 50000
- **Action space** - 6 (move left, move right, shoot, move left/shoot, move right/shoot, do nothing)
- **State space**- 160 (width of game)

Q-Learning Training



Q-Learning Test 200 Episodes

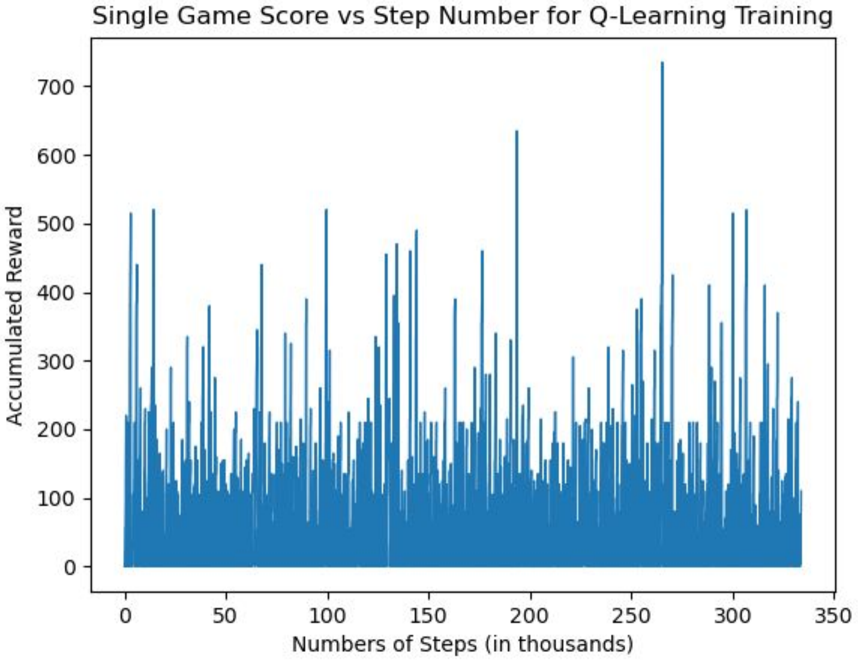


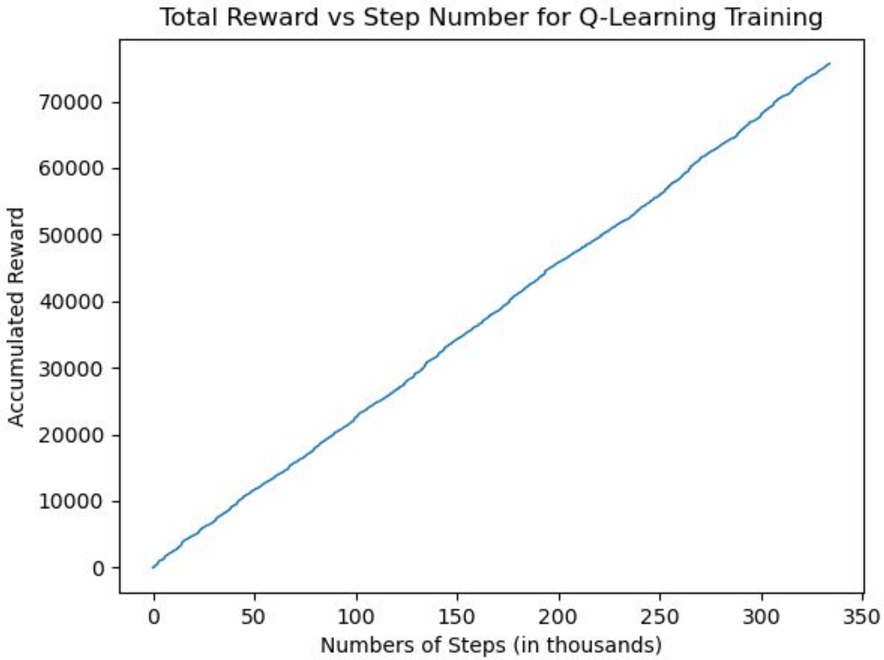
Q-Learning Test
200 Episodes

Score 50.0
Score 160.0
Score 210.0
Score 155.0
Score 180.0
Score 155.0
Score 80.0
Score 210.0
Score 210.0
Score 180.0

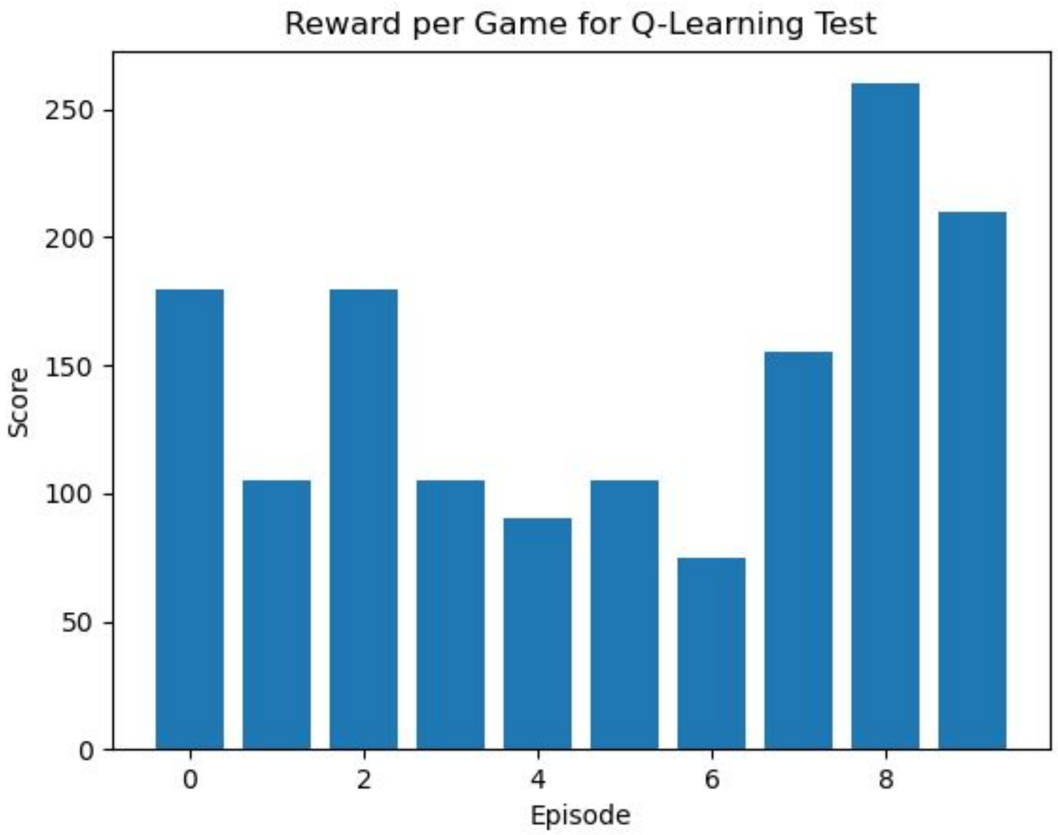
Average Score: 159.0

Q-Learning Training
200 Episodes





Q-Learning Test 200 Episodes



Q-Learning Test
500 Episodes

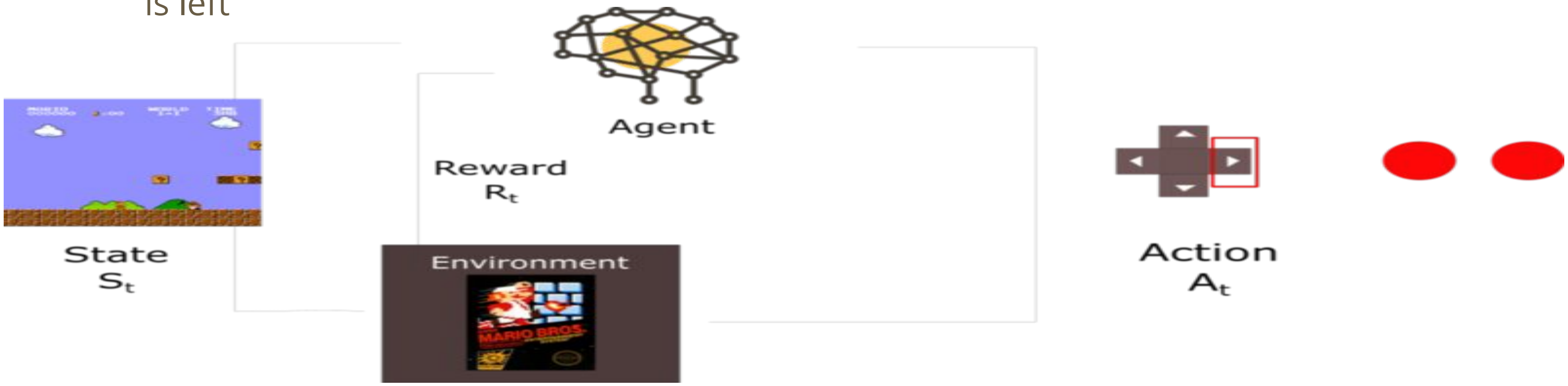
Score 180.0
Score 105.0
Score 180.0
Score 105.0
Score 90.0
Score 105.0
Score 75.0
Score 155.0
Score 260.0
Score 210.0
Average Score: 146.5

Q-Learning Results

- Unfortunately, the Q-Learning algorithm did not perform very well. It learned the basics and could get a few points in before it died, but it didn't *play* the game
- The reason for this is the state space
  - The state space for Q-Learning was the position of the spaceship
    - It had no idea where the bullets or ships were
- What it did learn
  - With the little information it had, it knew that it wouldn't get hurt while hiding under the shield
  - It also knew that if it kept shooting with half its body protected away it could get the most points
  - Q-Learning doesn't have the capacity to process that much information and fine two its training

Overcoming Shortcomings with Deep - Q Learning

- The problem with games is that the agent must take in images as perception; therefore, we need to use a convolutional neural network to add precision.
- The state space for DQN is a lot bigger, and it includes every pixel on the screen
  - This means that DQN knows where every bullet and enemy is as well as how much shield is left

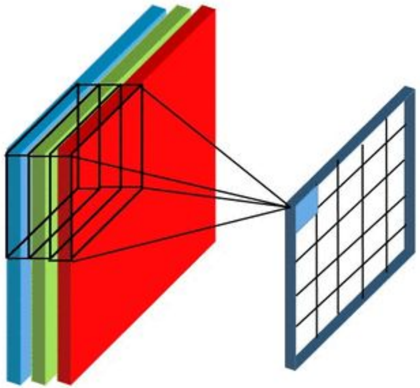


Implementation of Deep Q-learning Algorithm

- Initialize the environment of the Space-Invaders-Atari 2600 (8 actions possible)
- Preprocessing the frame:
  - Removal of unnecessary pixel (210,160) to (110,84)
  - Normalise for better distribution and resizing the image
- Getting stack frames
- Setting up training and hyper-parameters (state size : - 110,84,4)
  - Learning rate: - 0.0025 & decay rate = 0.0001
  - Total episodes = 150
  - Batch size = 64
  - Explore probability at start = 1.0
  - Discount = 0.9
  - Memory size = 1000000

Making it work

- Pre-Processing the frame
- Adding 4 frames/state as a single experience in a stack format
- Memory: - Storing previous experiences
- Replay Buffer: - Avoid reinforcing the same experience (overfitting) and making previous experiences more efficient.
- Creating the hidden layers
  - 3 Conv2d layers   activation = ‘elu’
  - flatten
  - 2 Dense layers   activation = ‘softmax’
  - Output layer size = action set (8)



Making it Work

- Exploration :- explore_start = 1.0 and explore_stop = 0.01

  explore_probability = explore_stop + (explore_start - explore_stop) * exp(-decay_rate * decay_step)

- Target networks and Loss Function

$$loss = \left(\mathrm{r} + \gamma \max_{a`} \hat{Q}(s, a`) - Q(s, a) \right)^2$$

Deep Q- learning training

$$\Delta w = \alpha[(R + \gamma \, max_a \, \hat{Q}(s', a, w)) - \hat{Q}(s, a, w)] \, \nabla_w \hat{Q}(s, a, w)$$

Change in weights

learning rate

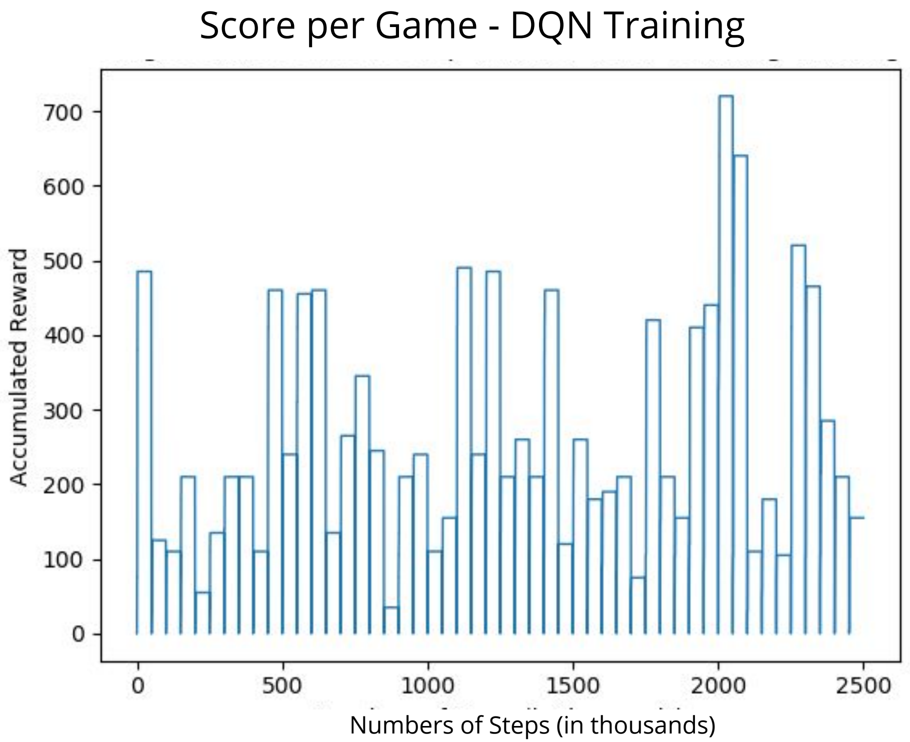
Maximum possible Qvalue for the next_state (= Q_target)

Current predicted Q-val

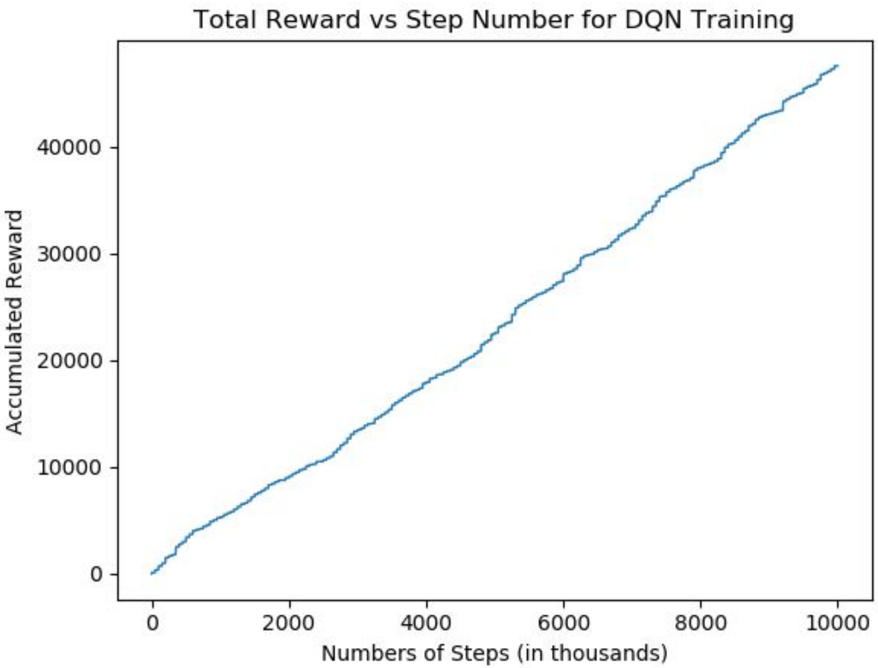
TD Error

Gradient of our current predicted Q-value

DQN Training - 50 Episodes

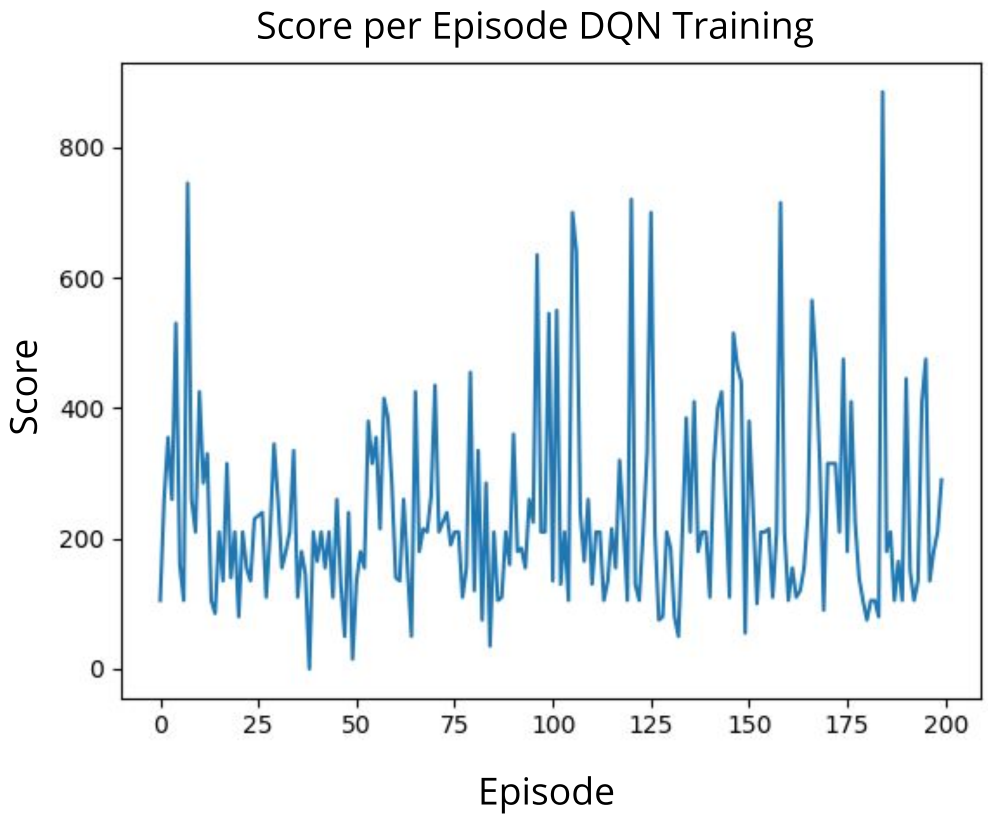


DQN Training (200 Episodes) vs Q-Learning (500 Episodes)



DQN Results - Training 200 Episodes/Test 10 Games

Score 480.0
Score 520.0
Score 260.0
Score 405.0
Score 190.0
Score 855.0
Score 630.0
Score 345.0
Score 305.0
Score 225.0
Average Score: 394.




0075
0000

DQN Results

- DQN performed a lot better than Q-Learning did, but it took a lot more time to train
  - Top Score - 855 (compared to Q-Learning's 260)
  - Avg Score - 394 (compared to Q-Learning's 146)
- With a very large state space, DQN could find the best move for every scenario
- This also meant that the training time was a lot longer
  - Training just 50 episodes took almost 3 hours
  - Training 50 episodes with Q-Learning only took about 10 minutes
- The DQN algorithm performed much better when we gave it more time to train
  - Q-Learning had a lot fewer steps (by a factor of 100) but a lot more episodes (300 more episodes), and it still did not do as well
- When it comes down to it, DQN actually played the game
  - It knew where to hide, when to shoot, and the pink think at the top is bonus points
  - Q-Learning had no clue about any of that except how to hide

Code

Use this link to access out GitHub

https://github.com/johndinofrio/ENPM690_Final_Project.git

References

https://www.freecodecamp.org/news/an-introduction-to-reinforcement-learning-4339519de419/

https://simoninithomas.github.io/Deep_reinforcement_learning_Course/

https://yilundu.github.io/2016/12/24/Deep-Q-Learning-on-Space-Invaders.html

https://www.youtube.com/watch?v=PnHCvfgC_ZA&list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT&index=4

https://www.youtube.com/watch?v=0g4j2k_Ggc4&list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT&index=5

https://www.youtube.com/watch?v=gCJyVX98KJ4