ML MAJOR PROJECT

A REPORT SUBMITTED IN

PARTIAL FULFILLMENT OF

BITS-F464

MACHINE LEARNING

TEAM MEMBERS

| # | Name | ID | Contribution |
| --- | --- | --- | --- |
| 1 | Ishan Sang | 2017A7PS0069G | * Implemented the DQN based model * Helped in debugging the errors * Fine tuned the model for better results |
| 2 | Purbayan Chatterjee | 2017A7PS0083G | * Contributed to the report * Helped in debugging the errors * Read various articles related to the project and contributed to the resources |
| 3 | Akshit Sharma | 2017A7PS0104G | * Tried various models using keras-rl and tensorflow * Provided comparisons amongst various policies and agents * Contributed to the report |

WORKFLOW DESCRIPTION

We implemented a DQN agent using Epsilon-Greedy policy and Experience Replay. On our way of exploring various approaches, we implemented various approaches whose results we have also added in the following report.

We used keras-rl APIs for checking which model should we choose to implement. In the following report, we have specified the flow of our work and how we reached our final submission.

APPROACHES

**Neural Networks**

We started by implementing the neural network based approach suggested by the blog post given to us in the resources document.

The input data for the neural network was generated by running the environment for 10000 episodes and capturing the episodes with a reward of 50 or more. These captured episodes were then fed into the neural network.

We used two different neural network models to solve the tasks -

Neural Network 1 : Uses a Deep Neural Network with 20 epochs

Neural Network 2 : Uses a Shallow Neural Network with 200 epochs

This approach did not work very well - we got average scores in the range of 100-200. So, we decided to move on to Q-Learning (a popular Reinforcement Learning Algorithm).

# Q Learning and Deep Q Networks

**Introduction**

Q Learning is an algorithm which finds the optimal policy which maximizes the reward by learning the optimal Q-values for each state-action pair. The Q-values are described by the Bellman Ford equation.

The algorithm uses a Q-Table to store and iteratively update the Q-values of all the State-Action pairs. Therefore to use the Q-Learning algorithm, we need to have an environment with a finite number of states.

However in our environment, we have an infinite number of possible states, as the environment is described by 4 real numbers. Therefore, we can not use Q-Learning directly.

Deep Q Networks (DQNs) solve this problem by using a neural network to estimate the Q-value function for all the states, hence there can be a continuous state-space.

**Policies**

We started with tensorflow and keras-rl to get an insight into how to go about training our agent. We read the implementations of DQN in keras-rl based on

We read about various policies and tried the following ones to get results using keras-rl first and then later implemented the Epsilon-Greedy Policy.

1. **Epsilon-Greedy policy** : When training an agent with Q-learning, you can either force the agent to take a random action with probability ϵ, or direct the agent to be greedy and take the action that corresponds to its policy with probability 1-ϵ (i.e. the action for a given state that has the highest Q-value). In our algorithm, we are not keeping the epsilon constant over time - it’s high initially, so that the model can experience different states; but with time it decreases and becomes stagnant to allow the model predictions to get verified.
2. **BoltzmannQ policy**: Here we are still acting greedy at times, but rather than blindly accepting any random action, when it comes time for the agent to explore the environment from a given states, the agent selects an action a (from a set of actions A) with some probability, for which it uses ranking and weighting all actions in the set of possible actions based on their Q-values. This system is often referred to as the softmax selection rule.

We also improved the learning and accuracy using **Experience Replay**. If the network only learns from consecutive samples, then the samples would be highly correlated, hence the learning would be inefficient. To break this correlation, we save previous observations in a replay memory and take random samples from it.

**State-Action-Reward-State-Agent (SARSA):**

SARSA agent very much resembles Q-learning. The key difference between SARSA and Q-learning is that SARSA is an on-policy algorithm. It implies that SARSA learns the Q-value based on the action performed by the current policy instead of the greedy policy. Basically, the Q-value is updated taking into account the action, A1 performed in the state, S1 in SARSA as opposed to Q-learning where the action with the highest Q-value in the next state, S1 is used to update the Q-table.

TRAINING LOGS

**TASK 1**

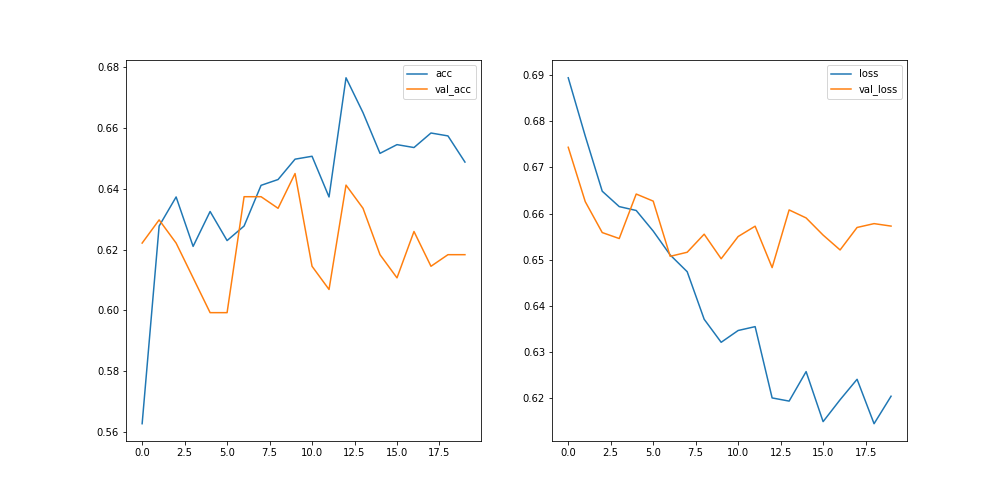
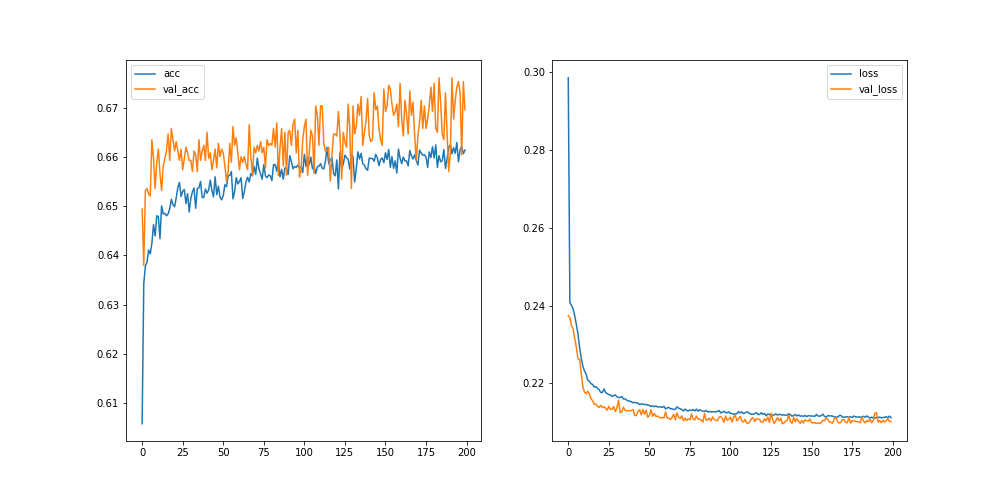


Fig 1 : Neural Network 1 plot with average reward of 103.35 (for 1000 episodes).

Fig 2 : Neural Network 2 plot with average reward of 141.75 (for 1000 episodes).

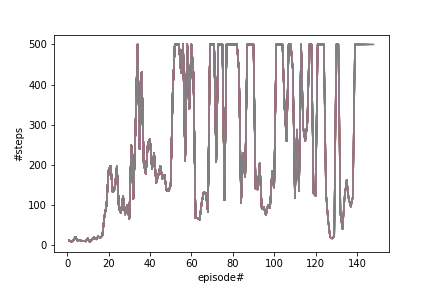
****

Fig 3 : Deep Q-Learning(with Epsilon greedy policy), average reward of 500 (for 1000 episodes).

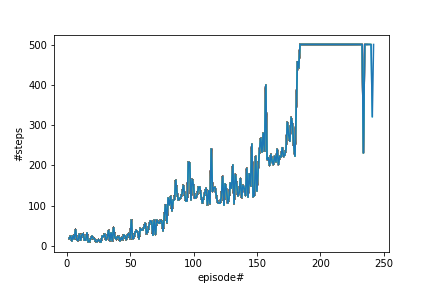
****

Fig 4 : Deep Q-Learning(with BoltzmannQ Policy), average reward of 500 (on 1000 episodes).

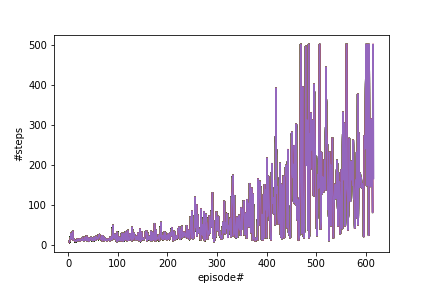


Fig 5 : SARSA Algorithm(with BoltzmannQ Policy), average reward : 413.58 (on 1000 episodes)

**TASK 2**

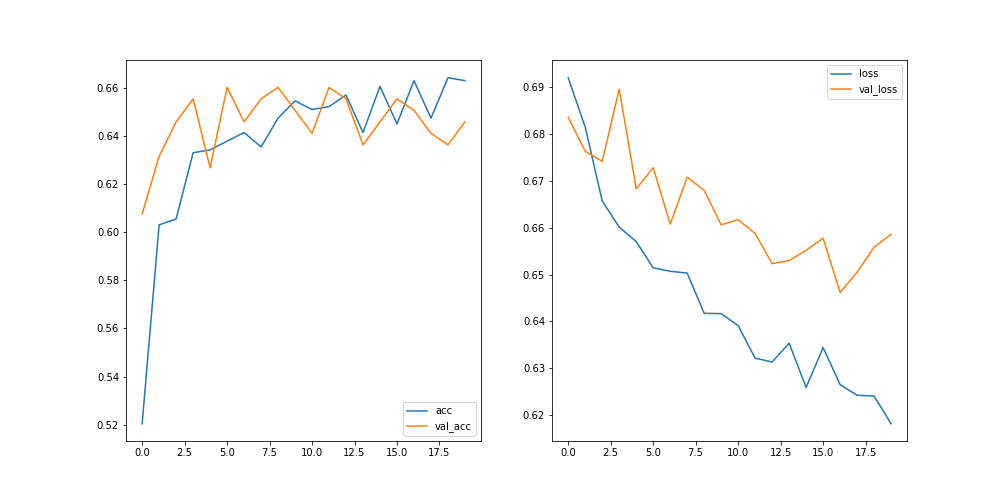


Fig 6 : Neural Network 1 plot with average reward of 159.262 (for 1000 episodes).

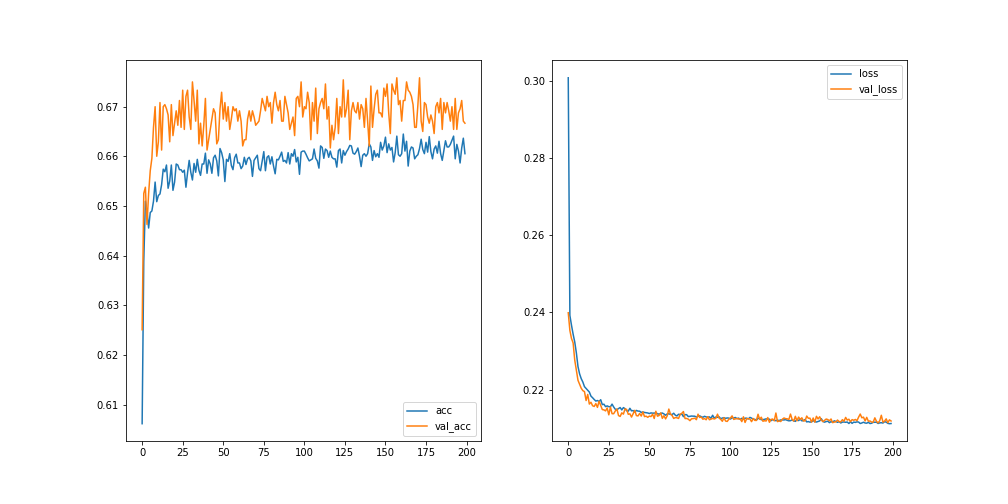


Fig 7 : Neural Network 2 plot with average reward of 204.075 (for 1000 episodes).

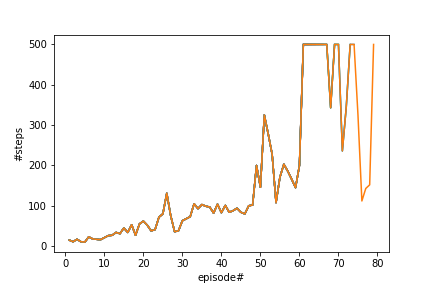
****

Fig 8 : Deep Q-Learning(with Epsilon greedy policy), average reward : 500 (on 1000 episodes).

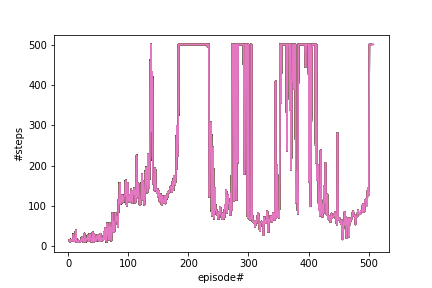
****

Fig 9 : Deep Q-Learning(with BoltzmannQ Policy), average reward : 500 (on 1000 episodes)

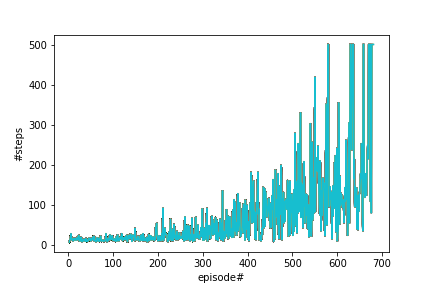
****

Fig 10 : SARSA Algorithm(with BoltzmannQ Policy), average reward : 494.8 (on 1000 episodes)

**TASK 3**

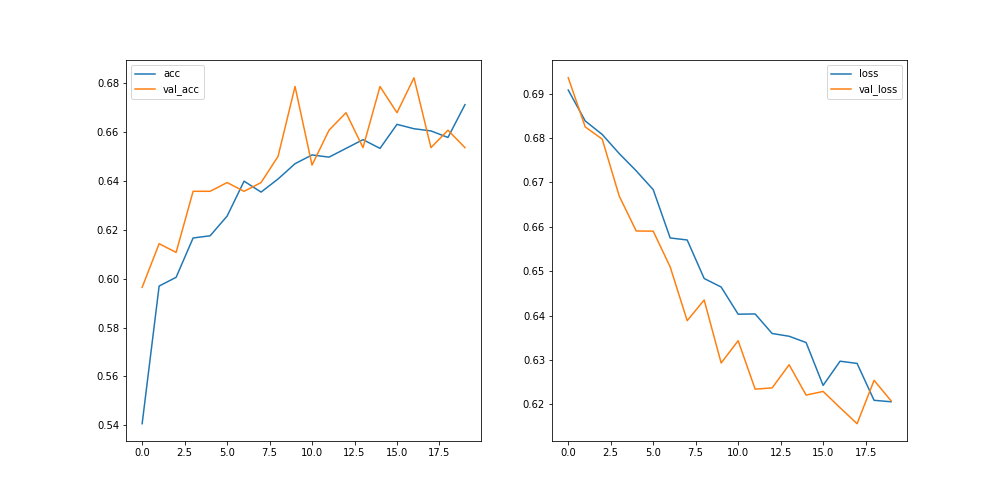


Fig 11 : Neural Network 1 plot with average reward of 135.987 (for 1000 episodes).

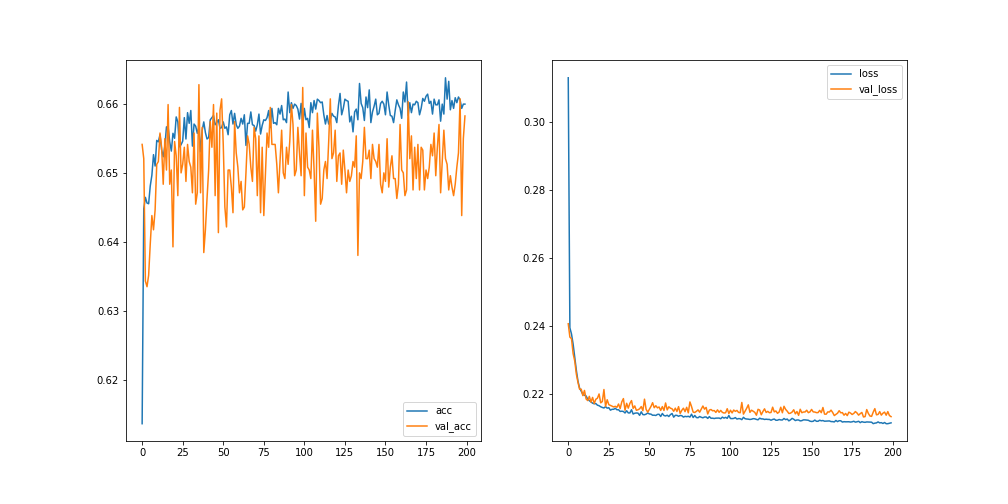


Fig 12 : Neural Network 2 plot with average reward of 262.514 (for 1000 episodes).

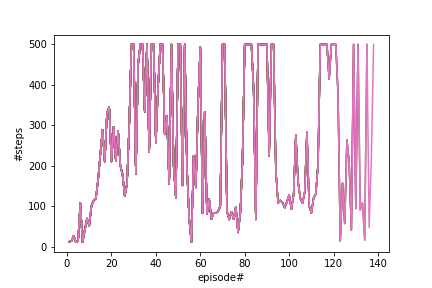
****

Fig 13 : Deep Q-Learning(with Epsilon greedy policy, average reward : 499.7 (on 1000 episodes)

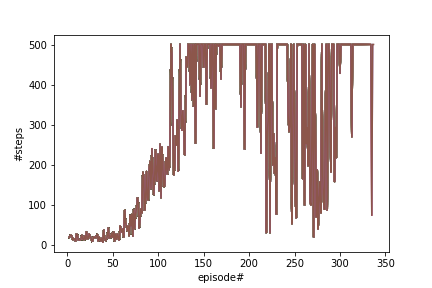
****

Fig 14 : Deep Q-Learning(with BoltzmannQ Policy), average reward : 500 (on 1000 episodes)

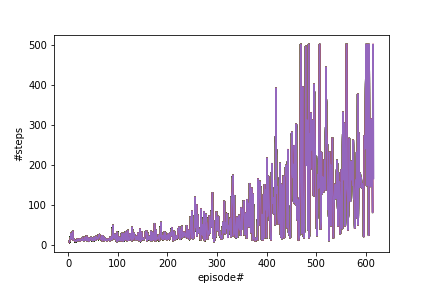


Fig 15 : SARSA algorithm(with BoltzmannQ Policy), average reward : 494.62 (on 1000 episodes)

GOOGLE DRIVE LINK

Contains all models, training logs that were generated while doing the project

<https://drive.google.com/drive/folders/1gSo-mBuWLIkVi56Hhh6MRJw52CRWo5Jv?usp=sharing>

RESOURCES

* [Python Programming Tutorials](https://pythonprogramming.net/openai-cartpole-neural-network-example-machine-learning-tutorial/)
* [keras-rl/keras-rl: Deep Reinforcement Learning for Keras.](https://github.com/keras-rl/keras-rl)
* [deep-learning/Q-learning-cart.ipynb at master · udacity/deep-learning](https://github.com/udacity/deep-learning/blob/master/reinforcement/Q-learning-cart.ipynb)
* [Practical Reinforcement Learning](https://www.coursera.org/learn/practical-rl/home/welcome)
* [Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG)](https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287)
* [dennybritz/reinforcement-learning: Implementation of Reinforcement Learning Algorithms. Python, OpenAI Gym, Tensorflow. Exercises and Solutions to accompany Sutton's Book and David Silver's course.](https://github.com/dennybritz/reinforcement-learning)
* [Cartpole - Introduction to Reinforcement Learning (DQN - Deep Q-Learning)](https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288)
* [Deep Q-Network with Pytorch - Unnat Singh](https://medium.com/@unnatsingh/deep-q-network-with-pytorch-d1ca6f40bfda)
* [RL — DQN Deep Q-network - Jonathan Hui](https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4)
* [Reinforcement Learning (DQN) Tutorial](https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html)
* [Build your first Reinforcement learning agent in Keras [Tutorial]](https://hub.packtpub.com/build-reinforcement-learning-agent-in-keras-tutorial/)
* [Deep Q-Learning with Keras and Gym · Keon's Blog](https://keon.github.io/deep-q-learning/)
* [CartPole with a Deep Q-Network](https://muetsch.io/cartpole-with-a-deep-q-network.html)
* [Introduction to Reinforcement Learning (Coding SARSA) — Part 4](https://medium.com/swlh/introduction-to-reinforcement-learning-coding-sarsa-part-4-2d64d6e37617)
* [Reinforcement Learning w/ Keras + OpenAI: DQNs](https://towardsdatascience.com/reinforcement-learning-w-keras-openai-dqns-1eed3a5338c)