Machine Learning
Major Project

Reinforcement Learning

Team Members:

1	Anany Shrey Jain	2017A4PS0920G
2	Mihir Dharmadhikari	2016A8TS0616G
3	Rishikesh Vanarse	2017A7PS1913G

INTRODUCTION

RL is a general concept that can be simply described with an agent that takes actions in an environment in order to maximize its cumulative reward. The underlying idea is very lifelike, where similarly to the humans in real life, agents in RL algorithms are incentivized with punishments for bad actions and rewards for good ones.

ENVIRONMENT

In this project we will be training an agent on a modified cartpole environment. Simple cartpole environment defines a pendulum with a center of gravity above its pivot point. It's unstable, but can be controlled by moving the pivot point under the center of mass. The goal is to keep the cartpole balanced by applying appropriate forces to a pivot point.

In the modified cartpole problem various parameters related to action and sensors will be introduced to make the training more challenging. These parameters will be introduced task wise:

- ➢ Task 1: Friction and gravity
- ➢ Task 2: Friction and gravity with noisy controls
- ➢ Task 3: Friction and gravity with noisy controls and sensor observations

METHODOLOGY

Deep Q-Learning:

Q-learning is a reinforcement learning method that uses the knowledge of the expected reward (of every possible action) at each step to decide the next action. The expected total reward corresponding to each possible action is called the Q-value. In Deep Q Learning (DQL), a neural network is used to predict the Q-values at each step, using the observations at that step as an input.

The Q-value of an action takes into account the reward generated by that action, as well as the maximum reward possible through the new state corresponding to the action taken.

The Q-value for a given state-action pair is given by:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

Here ***r(s,a)*** is the reward gained on taking action ***a*** from state ***s.***
s' is the new state. ***max{Q(s', a)}*** is the maximum Q value at ***s'. Ɣ*** is the 'discount rate', which decides the importance given to future rewards.

A deep Q-learning approach uses ***s*** to predict ***Q(s, a)*** and ***s'*** to predict ***Q(s', a).***

The approach that has been used in the project is a ***Double DQL*** approach.

The basic steps of this approach are as follows:

1. Initially random weights are alloted for the model and based on those weights random actions are taken by the agent.

2. At every action the current state information and the actions leading to the current state along with the previous state is stored in deque.
3. Training of the model happens at every action and a small batch of randomly selected data from the deque is used for this purpose. There are two reasons why the model is not trained with just the current state and previous state information:
    a. Model becomes very unstable when fitting it with just one data point at every step.
    b. Neural nets are more efficient when data is fed in batches.
4. Two models are used; one for predicting Q-values of the state leading to the current state (Present-model), and the other for predicting Q-values of the current state (Future-model).
5. As the action corresponding to the max Q value is taken by the agent so it is used to update the Q value corresponding to the action taken in the previous state.
6. At every step, Q values corresponding to the states in the mini batch are updated as mentioned in step 5 after which the state information and updated Q values are fed into the current model.
7. After the training step agent takes an action and new information is pushed into the deque. For an agent to explore better options a decaying parameter (Epsilon) is used which decides if the agent makes a random move or it will move according to the Q values of the current state. As the epsilon values decay agent sticks to taking action according to the Q values.
8. After every few steps the future model is updated with the weights of the current model. The basic reason behind using two models is to increase the stability of the model.

Architecture & Hyperparameters:

For all three tasks a deep neural network with two hidden layers was used for predicting the Q-values for every state. An Adam's Optimizer and an MSE Loss was used for all models. Following are the architectures and the hyperparameters used for each task.

Training Logs:

TASK 1:

Architecture:

Layer 1 size:	64
Layer 2 size:	128
Dropout:	0.1 (at each layer)

Other Hyperparameters:

Learning rate:	0.001
Size of Memory:	20,000
Batch Size:	32
Discount rate:	0.997
Initial Epsilon:	1.0
Epsilon decay:	0.9975
Future model updated after:	5 steps
Episodes trained:	450

TASK 2:

Architecture:

Layer 1 size:	64
Layer 2 size:	128
Dropout:	0.2 (at each layer)

Other Hyperparameters:

Learning rate:	0.001
Size of Memory:	50,000
Batch Size:	64
Discount rate:	0.997
Initial Epsilon:	1
Epsilon decay:	0.9975
Future model updated after:	7 steps
Episodes trained:	400

TASK 3:

Architecture:

Layer 1 size:	64
Layer 2 size:	128
Dropout:	0.1 (at each layer)

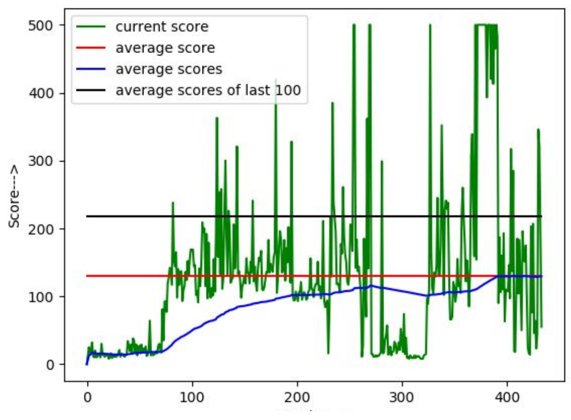
Other Hyperparameters:

Learning rate:	0.001
Size of Memory:	20,000
Batch Size:	32

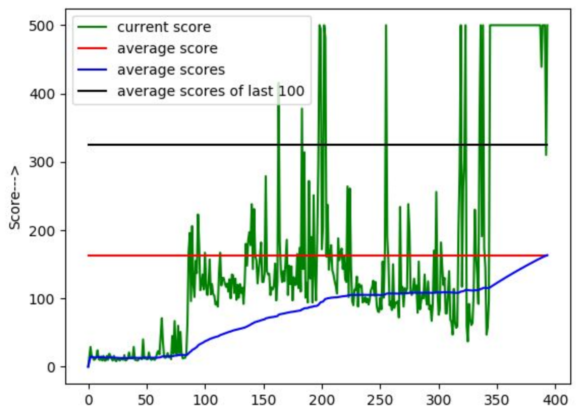
Discount rate:	0.997
Initial Epsilon:	1.0
Epsilon decay:	0.9975
Future model updated after:	5 steps
Episodes trained:	300

Training Graphs:

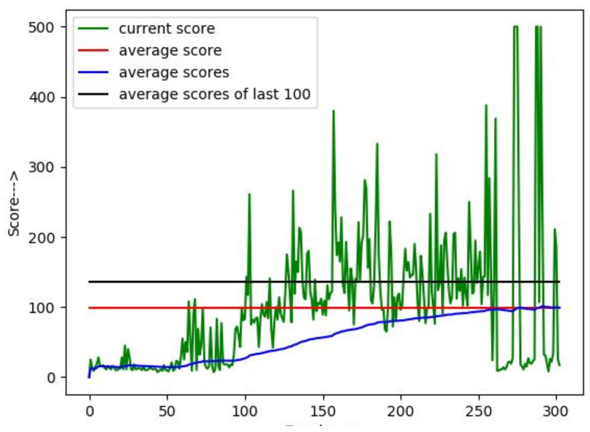
Task 1:



Task 2:



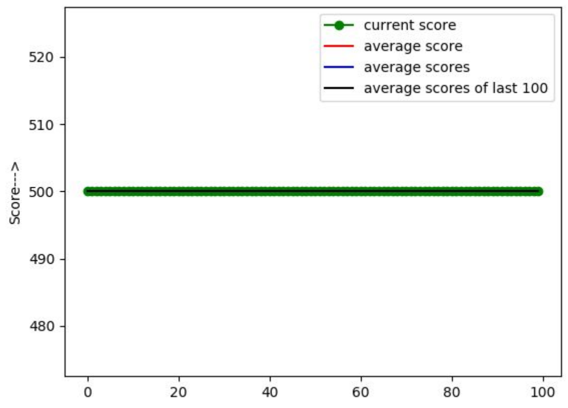
Task 3:



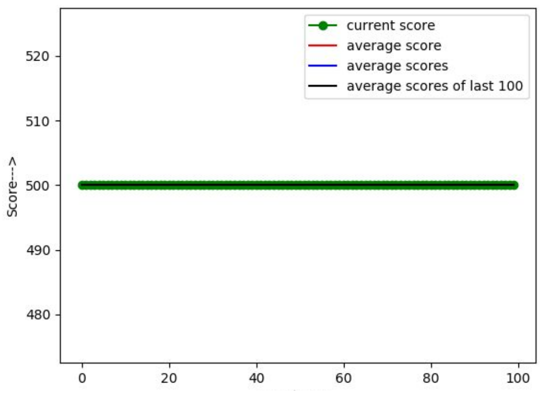
Results:

Full score of 500 was achieved on all 100 episodes of the three tasks.

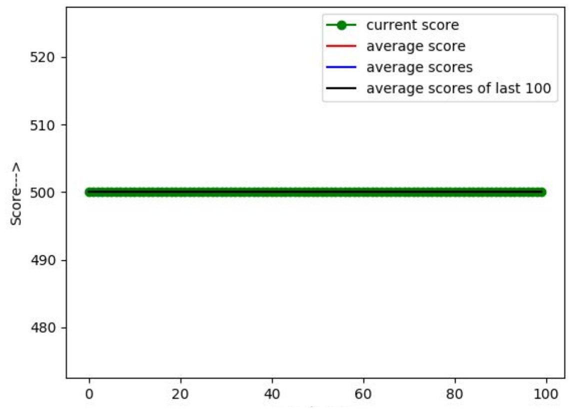
Task 1:



Task 2:



Task 3:



Alternate Approaches Implemented:

Method 1
Non Q Reinforcement Learning

In this method we train a neural network to predict the action to be taken based on the state of the game. The network is recursively trained using the data generated by the same network.

Initially a small set of data is generated using random games. In these, the actions are taken at random. The data point in the data set is the pair of current state and the action taken for that state. Among these data pairs the data for the games which have a score higher than a certain threshold will be selected for the training data. The score of the game is the number of frames for which the pole was upright. The features for the network are the values in the state and the labels are the actions taken for those states. The network outputs a probability value for each action possible. The action with highest probability is selected.

Now this trained model is used to generate the next dataset. The current state of the system is fed to the network and based on the predicted action the next state is found and fed again to the network. In this iteration, the threshold for selection of the data is increased. This is repeated for a certain number of iterations with the threshold increasing in every iteration.

Architecture:

For all three tasks a deep neural network with two hidden layers was used for predicting the action. An Adam's Optimizer and an MSE Loss was used for all models. Following are the architectures and the hyperparameters used for each task. Architecture has been kept same for all three tasks in Basic Neural Network approach

Architecture:	
Input Size:	4
Output Size:	2
Hidden layer 1:	64

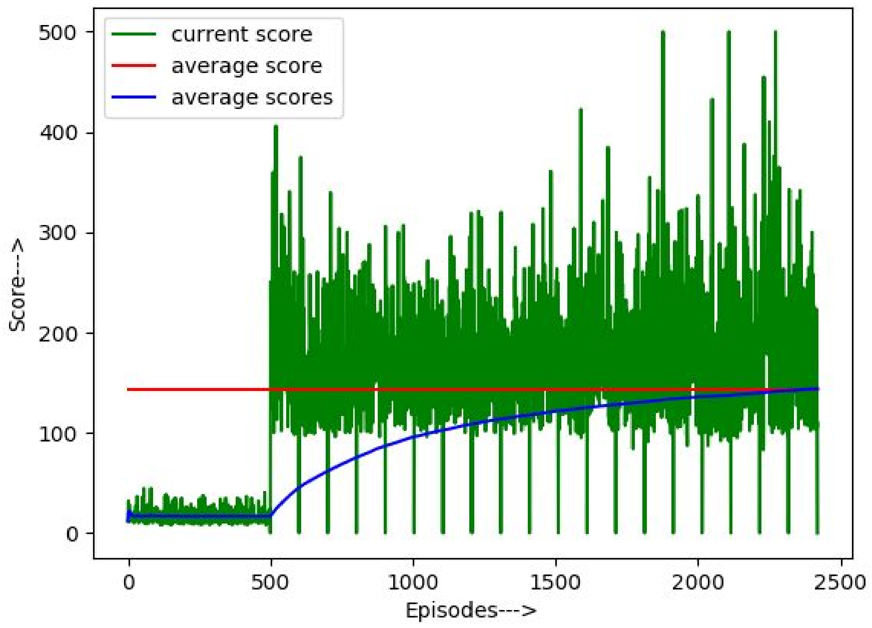
Hidden Layer 2:	128
Output size:	2
Dropout:	0.1
Optimizer:	Adam

Training Logs:

TASK 1:

Hyperparameters

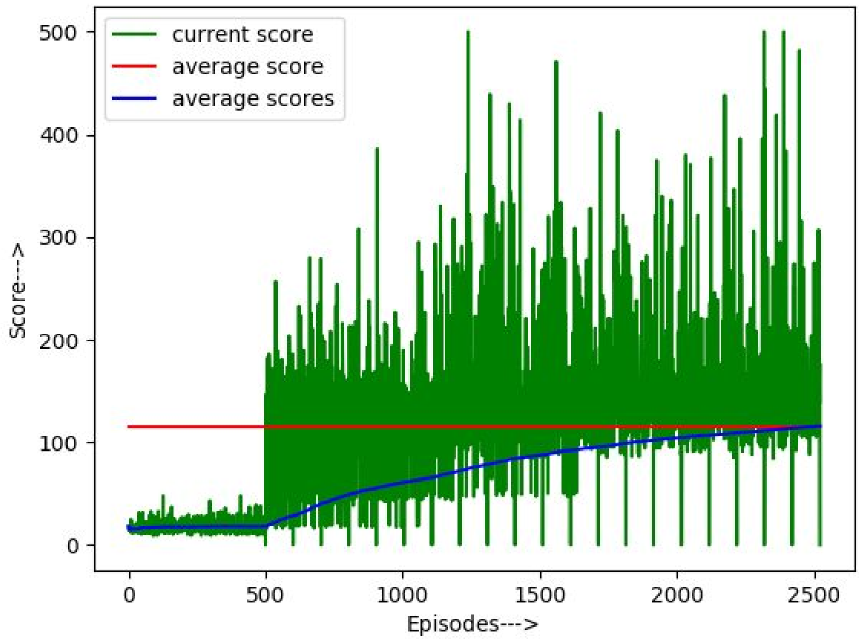
Learning Rate:	0.001
No. of games with random moves:	500
Min score required to classify as training game:	30
Increase in score requirement per trial:	20
No. of games played:	2500
Trials:	20
Epochs per trial:	5



TASK 2:

Hyperparameters

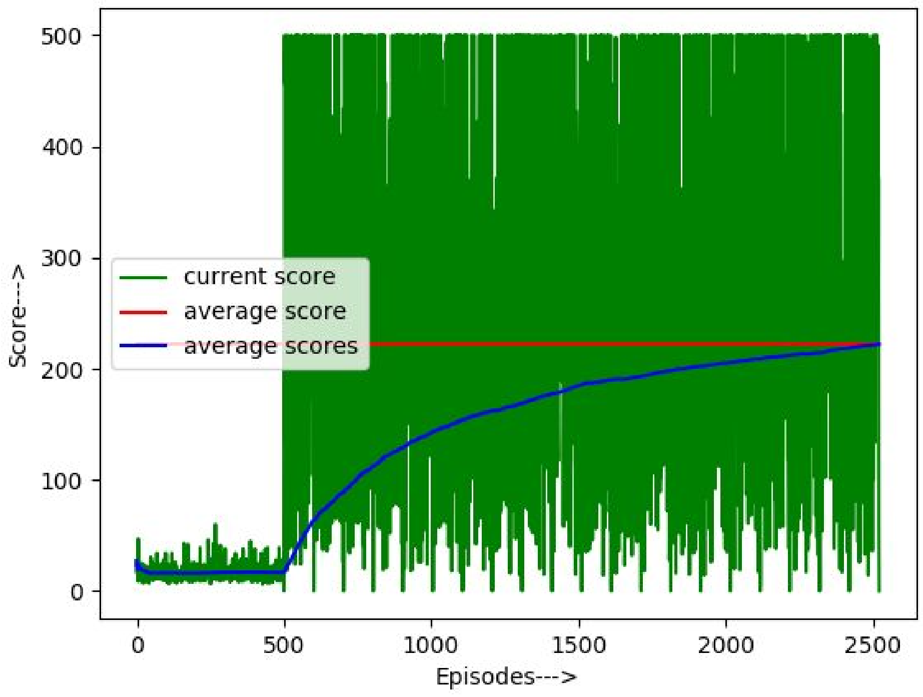
Learning Rate:	0.001
No. of games with random moves:	500
Min score required to classify as training game:	30
Increase in score requirement per trial:	20
No. of games played:	2500
Trials:	20
Epochs per trial:	5



TASK 3:

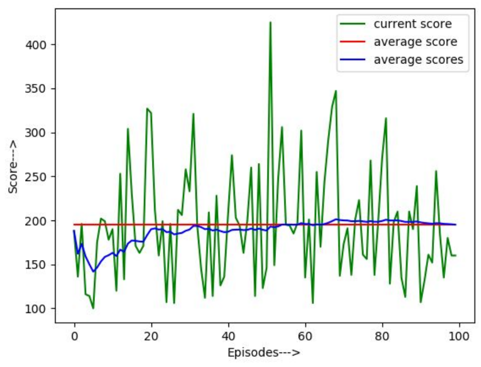
Hyperparameters

Learning Rate:	0.001
No. of games with random moves:	500
Min score required to classify as training game:	20
Increase in score requirement per trial:	20
No. of games played:	2500
Trials:	20
Epochs per trial:	10

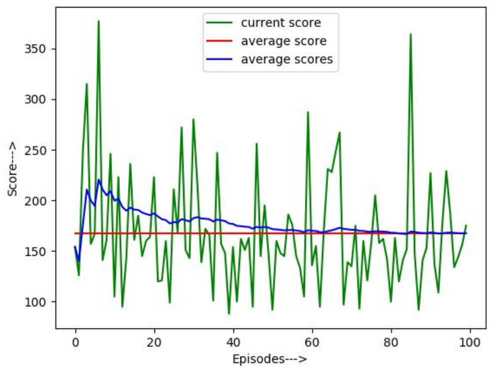


Results:

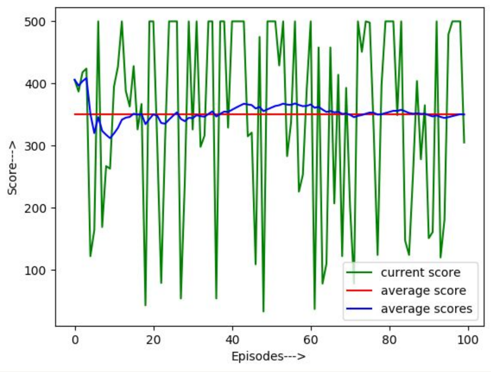
Task1:



Task2:



Task3:



Disadvantages:

The model only trains on the data of the last few instances. Also, the data is generated by the trained model. This makes the model train towards a certain approach. The model will learn to solve the problem in a certain way and perfect it but due to the lack of randomness it won't be able to find other ways of solving the problem.

Method 2
Q-Learning:

In this method, the observation space is approximated to a discrete size having sufficient granularity. Any observation is approximated to its closest value in this discrete set. A Q-table is made, which contains entries for every possible observation in the discrete observation space. These entries are the Q-values of each action for that particular observation.

In the beginning, all the Q-values are initialized randomly. At every step of training, the action with the highest Q-value is taken, to obtain the next state. The reward gained through this is used to update the Q-values corresponding to the previous state, using the following formula:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)}^{\text{temporal difference}}$$

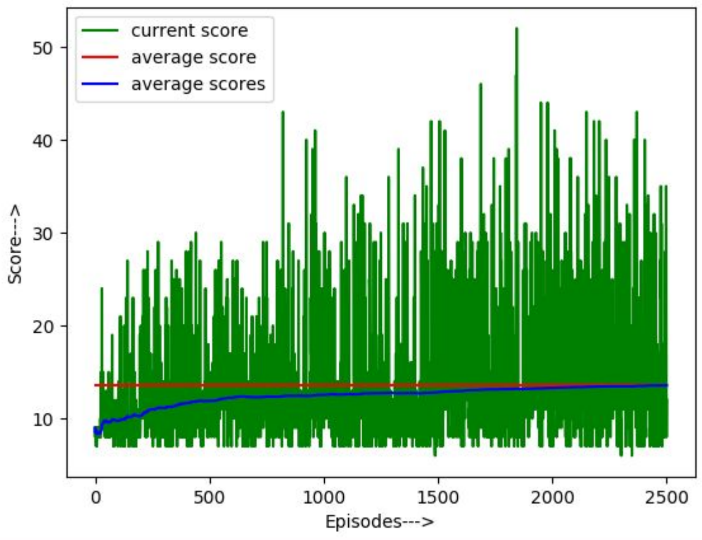
Here, ***Q(s, a)*** is the Q-value corresponding to action ***a*** taken at state ***s.***

Hyperparameters

Learning rate:	0.01
Discount rate:	0.997
Q Table Shape:	(20, 500, 20, 500, 2)
Episodes:	2500

Training Graph:

Since the observation space involves continuous values with an infinite range for the cart-pole problem, Q Learning is not a good choice. We implemented Q learning for task 1 but since the results were not appealing we skipped the other tasks. Below is the training graph for task 1.



Average scores are slowly increasing in the beginning but the rate of increase seems to be diminishing.

Disadvantages:

As the observation space is discrete, every observation has to be rounded to the closest approximation available in the Q-table. It is assumed that the best action for the approximation and for the actual observation will be the same; however, this may not always hold.

An attempt to solve this issue would mean a greater granularity. This leads to a very large table that takes up a lot of memory.

Furthermore, since the algorithm has to refer to the table before taking any action, the best action cannot be known for an unseen observation.

Running The Code:

There are 4 codes in the folder. The DQLearning, DLapproach and QLearning are the implementations of DQL, DL and Q learning approaches respectively. The *run_the_model.py* will play 100 games for the given task using a given model (neural network). For detailed information go through the Readme.txt file.

CONTRIBUTIONS

- **Anany Shrey Jain :**
    - Wrote the code for the deep Q Learning approach .
    - Trained model for deep Q Learning Task 1 and Task 3.
    - Wrote the code for Q Learning approach.
    - Contributed to completion of report.
- **Mihir Dharmadhikari :**
    - Wrote the code for the Non Q Reinforcement Learning approach.
    - Made improvements in the deep Q learning code.
    - Contributed to completion of report.
- **Rishikesh Vanarse :**
    - Trained model for deep Q Learning Task 2.
    - Contributed to completion of report.

REFERENCES

1. Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015)
2. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
3. https://hub.packtpub.com/build-cartpole-game-using-openai-gym/
4. https://en.wikipedia.org/wiki/Q-learning#Double_Q-learning
5. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/
6. https://pythonprogramming.net/openai-cartpole-neural-network-example-machine-learning-tutorial/