MINI PROJECTREPORT

## Balancing Inverted Pendulum

*Submitted in partial fulfilment of the requirements for the degree of*

**Bachelor of Engineering**

in

**Computer Science and Business System**

*Submitted By*

**Hritvik Naru (101918012)**

**Samaksh Gulati (101918020)**

**Aadish Mehta (101918024)**

**Abhishek Pandey (101918063)**

Under the supervision of:

**Sushma Jain**

Designation of Supervisors



**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**PATIALA – 147004**

**December 2021**

Table of Contents

[1. Introduction](#_2s8eyo1) 1

[2. Background](#_17dp8vu) 2

[3. Objectives](#_3rdcrjn) 3

[4. Methodology](#_26in1rg) 4-5

[5. Observations and Findings](#_lnxbz9) 6-7

[6. Limitations](#_35nkun2) 8-9

[7. Conclusions and Future Work](#_1ksv4uv) 10

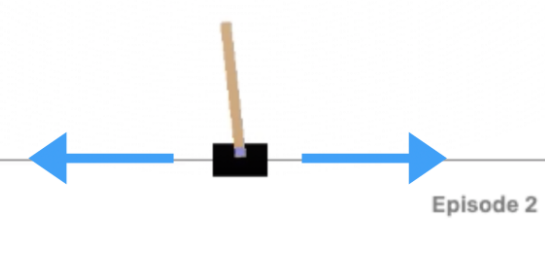
[8. Bibliography/References](#_44sinio) 11

9. Plagiarism Report 12

# 

# Introduction

CartPole, also known as [inverted pendulum](https://en.wikipedia.org/wiki/Inverted_pendulum), is a game in which you try to balance the pole as long as possible. CartPole is the structure where a pole is attached to the cart and the cart is free to slide across the frictionless surface. The goal of this task is to move the cart left and right so that the pole can stand (within a certain angle) as long as possible.



We will look at reinforcement learning, a field in artificial intelligence where the AI explores the environment all by itself by playing the game many times until it learns the right way to play the game.



As you can see here, at the beginning of the training, the agent has no idea of where to move a cart. After a while, the agent moves toward a direction but, of course, it is impossible to bring the pole to the other side with such speed. For the last one, the agent knows the correct way to balance the pole which is to move left and right repeatedly.

# Background

The cart pole problem has been thoroughly studied by the research community as a fundamental problem, yielding fruitful results in both closed-form control strategies and learning-based strategies. Fixed Point Controllers and Stabilization of the Cart-pole System and the Rotating Pendulum discussed control theory-based stabilisation of cascaded nonlinear systems and applied the theorem to the cart pole problem. Algorithms for Reinforcement Learning Comparison Applied to the Cart-pole Problem compared the performance of several popular RL algorithms with numerical simulations. In 2016, Open AI Gym was released, which provides standardised environments for reinforcement learning tasks. Many attempts have been made in recent years to solve the Cart Pole problem with as few episodes as possible.

These submissions, each with a distinct approach and performance, are available on the OpenAI Gym leader board. We discovered that implementing a closed-form solution would be the quickest solution, followed by genetic algorithms and deep reinforcement learning methods, both of which are excellent approaches to determining a sequence of decisions, after surveying this leader board. We observe that closed-form solutions are much less scalable as the problem becomes exponentially more complex as more states are introduced into the environment. The tutorial balancing a Cart Pole System with Reinforcement Learning walks readers through a series of different Q-Learning methods for solving the Cart Pole problem, which we used as a starting point to understand and work on the problem.

Since OpenAI Gym provided us with a simple and standardised testing environment, variations of Q Learning methods have received a lot of attention. There are numerous publications relating to our problem that we discovered, and here are a few that influenced our implementation. We learned from Deep Reinforcement Learning with Double Q-learning that Q learning can introduce bias, which prompted us to work on double Q learning as well. We saw how using a neural network in conjunction with reinforcement learning allows us to solve the problem with fewer episodes and greater adaptability. We also learned the impact of selecting a different number of discrete states on the traditional Q learning method and how neural networks eliminate that process by taking in either continuous kinematic state values or screenshots of the environment from previous research.

# Objectives

* Stabilize the Inverted Pendulum (CartPole).
* Keep it from falling or moving out of the range.
* Finding the best model to get maximum cumulative reward.

# Methodology

### Markov Decision Processes (MDPs):

Markov decision processes give us a way to formalize sequential decision making. This formalization is the basis for structuring problems that are solved with reinforcement learning.

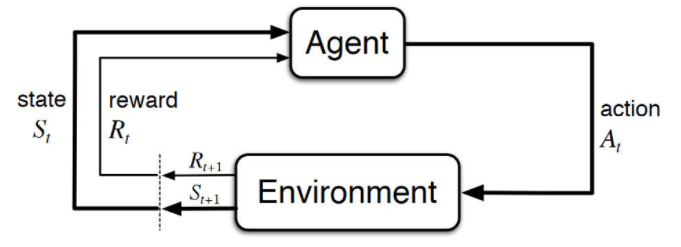
In an MDP, we have a decision maker, called an *agent*, that interacts with the *environment* it's placed in. These interactions occur sequentially over time. At each time step, the agent will get some representation of the environment's *state*. Given this representation, the agent selects an *action* to take. The environment is then transitioned into a new state, and the agent is given a *reward* as a consequence of the previous action.

Components of an MDP:

* Agent
* Environment
* State
* Action
* Reward

This process of selecting an action from a given state, transitioning to a new state, and receiving a reward happens sequentially over and over again, which creates something called a *trajectory* that shows the sequence of states, actions, and rewards.

Throughout this process, it is the agent's goal to maximize the total amount of rewards that it receives from taking actions in given states. This means that the agent wants to maximize not just the immediate reward, but the *cumulative* rewards it receives over time.



For our project, we are implementing Q Learning, Double Q Learning, and Deep Q Network to solve the inverse pendulum problem.

**Q Learning**

 Q, or the quality index, is the direct measurement of the future expected reward which we want to maximize in this problem. In the training process of the Q learning algorithm, we use the value of epsilon to control whether we want to explore or exploit our environment. Exploration performs a random action from the action space, and Exploitation performs the optimal action from our Q Table. This process allows the algorithm to learn and update the Q Table iteratively to keep the pole from falling.

Our problem follows a Markov chain where the cart performs an action, which triggers a new state and corresponding reward from the environment, which are then used to update the optimal action to take to keep the environment stable. The update rule for the Q table is also known as the Bellman equation for the action value function, where we are updating each state and action’s Q value by its corresponding old Q value plus the learning rate, alpha, times the sum of reward, discount factor gamma times the estimate of optimal future value, and negative old Q value. This update is performed iteratively until the pole falls from the cart, the cart moves too far away from the center, or the maximum number of steps is reached.

*Bellman Eq.*



**Deep Q learning**

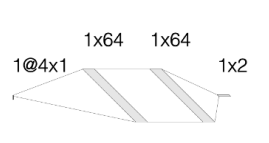
The main idea behind Q-learning is that we have a function  Q:State×Action→R, which can tell the agent what actions will result in what rewards. If we know the value of Q, it is possible to construct a policy that maximizes rewards:



However, in the real world, we don't have access to full information, that's why we need to come up with ways of approximating Q. One traditional method is creating a lookup table where the values of Q are updated after each of the agent's actions. However, this approach is slow and does not scale to large action and state spaces. Since neural networks are universal function approximators.

With the advent of deep learning, researchers have proposed to adopt deep learning techniques to calculate the Q function. This is based on the intuition that given a properly designed network architecture and sufficient unbiased training data, a deep neural network can be trained to approximate an arbitrary given function. Deep Q Learning is more suitable for problems with continuous state or control space, since it requires little effort to modify a deep learning model to take continuous input variables and generate continuous outputs.

Under this setup, our goal is to build a Deep Q Network that mimics the Q Table. That being said, the network should take an input vector with length 4 and generate a corresponding output vector with length 2. Since the input is already a compact feature representation of the system’s kinematic state, we don’t need to design complex feature extraction structures any more. Following the convection of neural architecture design for low-dimension input and output, we adopt a 3-layer fully connected network (FCN) as our Deep Q Network. For activation function, we adopt ReLU for the first 2 layers as well as for the Output layer.



## 

## Deep Q Learning with Experienced Replay

The approximation of Q using one sample at a time is not very effective. The graph above is a nice illustration of that. The network managed to achieve a much better performance compared to a random agent.

We implemented experience replay to improve network stability and make sure previous experiences are not discarded but used in training.

Experience replay stores the agent's experiences in memory. Batches of experiences are randomly sampled from memory and are used to train the neural network. Such learning consists of two phases--gaining experience and updating the model. The size of the replay controls the number of experiences that are used for the network update. Memory is an array that stores the agent's state, reward, and action, as well as whether the action finished the game and the next state.

## Double Deep Q Learning

Deep Q Learning tends to overestimate the reward, which leads to unstable training and lower quality policy. Let's consider the equation for the Q value:



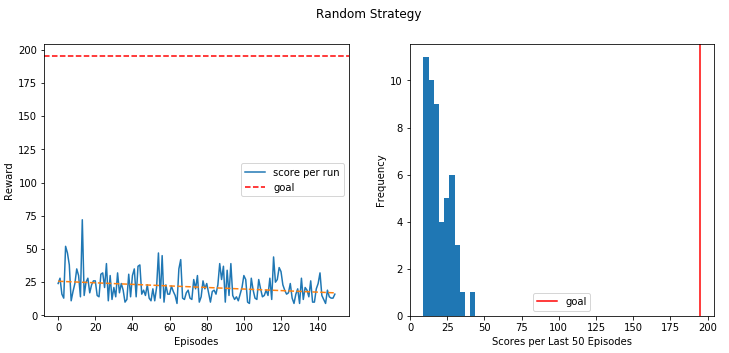
The last part of the equation takes the estimate of the maximum value. This procedure results in systematic overestimation, which introduces a maximization bias. Since Q-learning involves learning estimates from estimates, such overestimation is especially worrying.

To avoid such a situation, we will define a new target network. The Q values will be taken from this new network, which is meant to reflect the state of the main DQN. However, it doesn't have identical weights because it's only updated after a certain number of episodes.

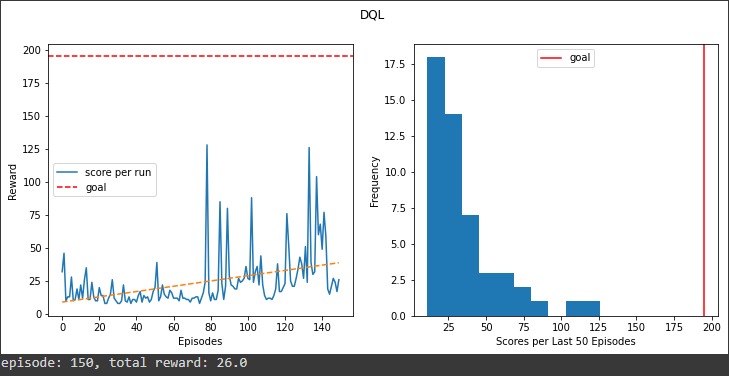
The addition of the target network might slow down the training since the target network is not continuously updated. However, it should have a more robust performance over time.

# Observations and Findings

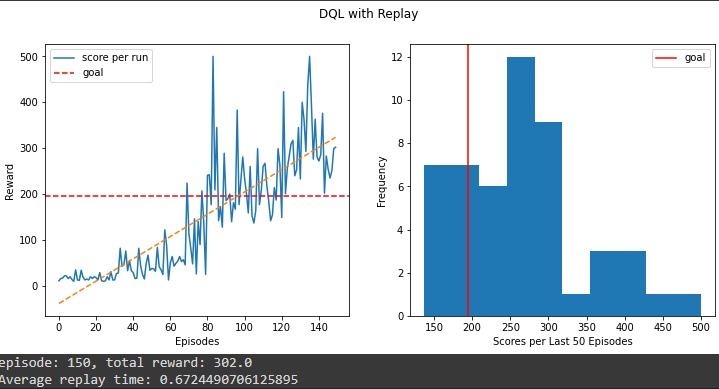
Before implementing any deep learning approaches, we wrote a simple strategy where the action is sampled randomly from the action space. This approach will serve as a baseline for other strategies and will make it easier to understand how to work with the agent using the Open AI Gym environment.



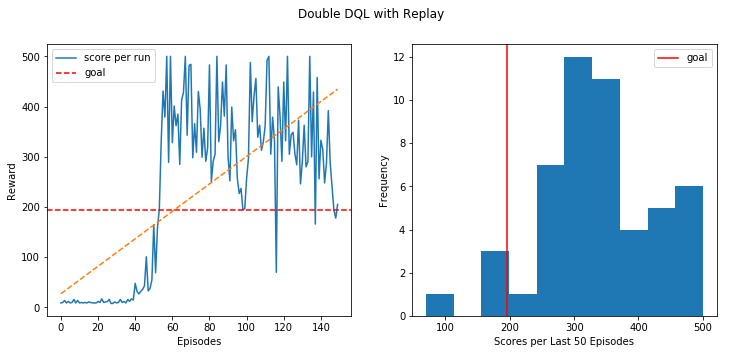
The random strategy is depicted in the plot above. It is, as expected, impossible to solve the environment using this method. The agent isn't learning from their mistakes. Despite being fortunate on occasion (receiving a reward of nearly 75), their average performance is as low as 10 steps.



The graph above shows that the agent's performance has significantly improved. It got to 175 steps, which is impossible for a random agent, as we've seen before. The trend line is also positive, and the performance improves over time. At the same time, the agent failed to cross the finish line after 150 epochs, and its average performance is still around 15 steps, indicating that there is definitely room for improvement.



As expected, the neural network with replay appears to be much more robust and intelligent than its counterpart that only remembers the most recent action. After about 60 episodes, the agent was able to cross the winning threshold and maintain it. I also received the highest possible reward—500.

****

The previous version of Double DQL with replay outperformed it and consistently performed above 300 steps. Because of the separation of action selection and evaluation, performance appears to be a little more stable. Finally, let's look at the most recent change to the DQL agent.

# Limitations

The standard Q-learning algorithm (using a table) applies only to discrete action and state spaces. [Discretization](https://en.wikipedia.org/wiki/Discretization) of these values leads to inefficient learning, largely due to the [curse of dimensionality](https://en.wikipedia.org/wiki/Curse_of_dimensionality). However, there are adaptations of Q-learning that attempt to solve this problem such as Neural Network Q-Learning. The disadvantage of deep q-networks approach is that the smaller action set might not be sufficient for achieving the desired behavior.

# Conclusions

We investigated the inverse pendulum problem and used four different RL approaches to solve it. The FCN (Fully Connected Network) based DQN with experience replay and the target network in the Open AI Cart pole environment outperforms traditional RL that generates a better control strategy in fewer episodes.

# Bibliography/References

**<https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html>**

[**https://gym.openai.com/envs/CartPole-v0/**](https://gym.openai.com/envs/CartPole-v0/)

[**https://medium.com/@leosimmons/double-dqn-implementation-to-solve-openai-gyms-cartpole-v-0-df554cd0614d**](https://medium.com/@leosimmons/double-dqn-implementation-to-solve-openai-gyms-cartpole-v-0-df554cd0614d)

[**https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/**](https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/)

[**https://www.baeldung.com/cs/epsilon-greedy-q-learning**](https://www.baeldung.com/cs/epsilon-greedy-q-learning)

[**https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/**](https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/)