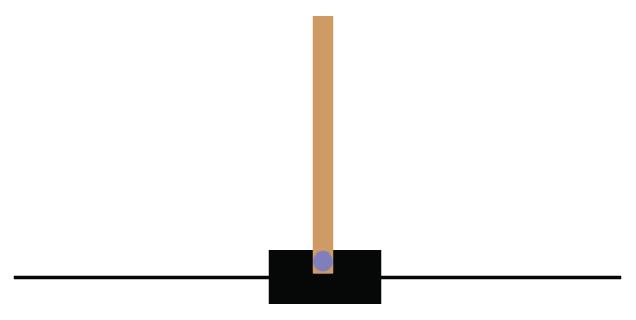
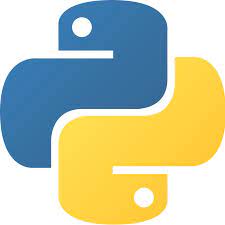
Balancing Cart Pole in Open AI Gym using Deep Reinforcement Learning

The Ultimate Guide for Implementing a Cart Pole Game using Python, Deep Q Network (DQN), Keras and Open AI Gym.







**Blog By: Chaithanya Vamshi Sai**

**Student Id: 21152797**

# **Introduction**

Reinforcement Learning is a type of machine learning that allows us to create AI agents that learn from their mistakes and improves their performance in the environment by interacting to maximize their cumulative reward.

AI agents will learn it by trial and error and agents are incentivized with punishments for wrong actions and rewards for good ones.

In this blog, I will demonstrate and show how we can harness the power of how Deep Reinforcement learning (Deep Q-learning) can be implemented and applied to play a Cart Pole game using Keras, DQN Algorithm and Open AI Gym.

# **Steps to Implement Cart Pole Game using Keras, Deep Q Network (DQN) Algorithm and Open AI Gym**

1. Problem Statement
2. Importing Libraries
3. Setting up the Environment with Open AI Gym
4. Implementing DQN Algorithm by applying LinearAnnealed - EpsGreedyQPolicy
5. Building DQN Agent with Keras-RL
6. Testing the DQN Agent for 20 Consecutive Episodes
7. Saving the Best DQN Model Weights

# **Problem Statement**

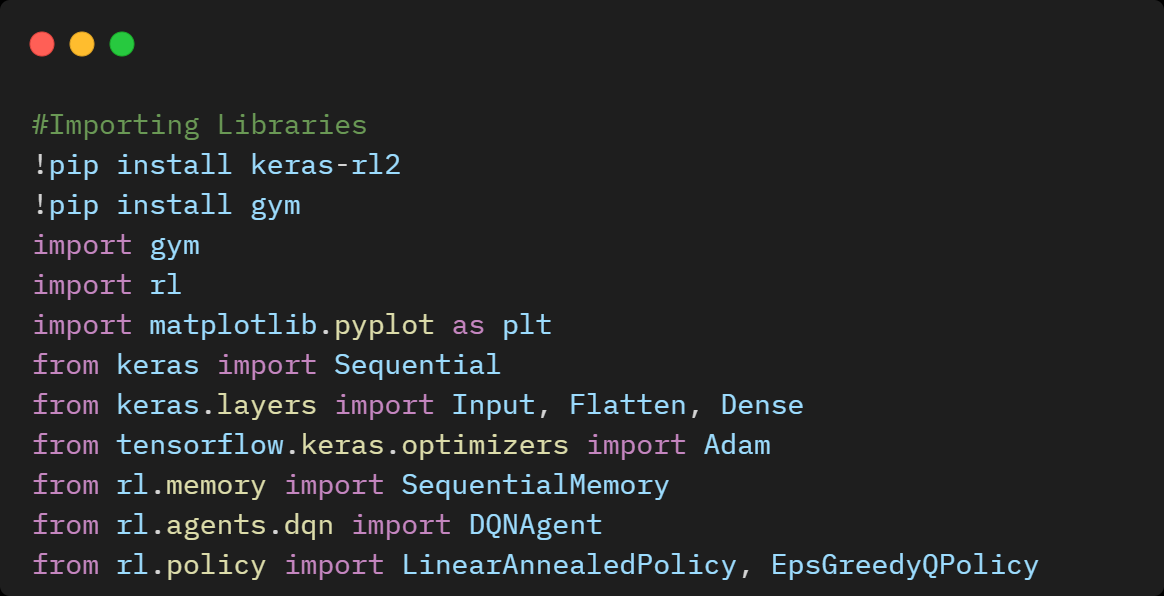
The main objective of this task is to apply Deep Reinforcement learning to replace the human element in the CartPole-V0 environment in Open AI Gym environment using the Deep Q Network (DQN) algorithm.

**Conditions for Cart Pole Game**

* The goal is to balance the pole by moving the cart from side to side to keep the pole balanced upright.
* If the pole angle is more than 12 degrees or the cart moves by more than 2.4 units from the centre, then the game will end and if the pole remains standing for 200 steps, then the game is successful.
* Apply Linear annealed policy with the EpsGreedyQPolicy as the inner policy.
* Achieve a DQN model that trains in the least possible number of episodes.
* Balance pole on the cart for 200 steps (Maximum Reward) for 20 consecutive episodes while testing.

# **Importing Libraries**

* **Keras - rl2**: Integrates with the Open AI Gym to evaluate and play around with DQN Algorithm
* **Matplotlib**: For displaying images and plotting model results.
* **Gym**: Open AI Gym for setting up the Cart Pole Environment to develop and test Reinforcement learning algorithms.
* **Keras**: High-level API to build and train deep learning models in TensorFlow.



# **3. Setting up the Cart Pole Environment with Open AI Gym**

Cart Pole is one of the simplest environments in the Open AI gym which is a collection of environments to develop and test Reinforcement learning algorithms.

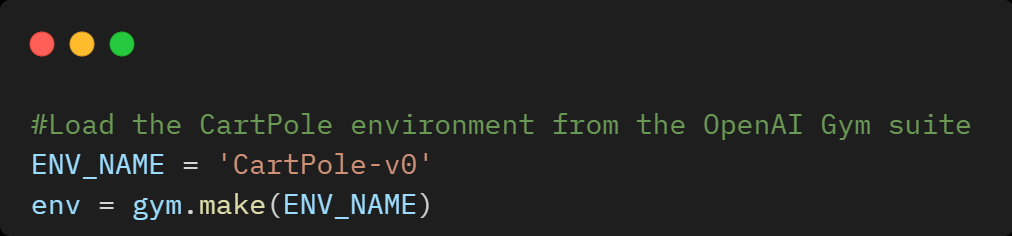
The goal of the Cart Pole is to balance a pole connected with one joint on top of a moving cart. An agent can move the cart by performing a series of 0 or 1 actions, pushing it left or right.

1. Observation is an array of 4 floats that contains

* Angular Position and Velocity of the Cart
* Angular Position and Velocity of the Pole

1. Reward is a scalar float value
2. Action is a scalar integer with only two possible values

* 0 — "move left"
* 1 — "move right"



# **4. Implementing DQN Algorithm by applying Linear Annealed - EpsGreedyQPolicy**

A policy defines the way an agent acts in an environment. Typically, the goal of reinforcement learning is to train the underlying model until the policy produces the desired outcome.

In this task, we will set our policy as Linear Annealed with the EpsGreedyQPolicy as the inner policy, memory as Sequential Memory because we want to store the result of actions we performed and the rewards we get for each action.

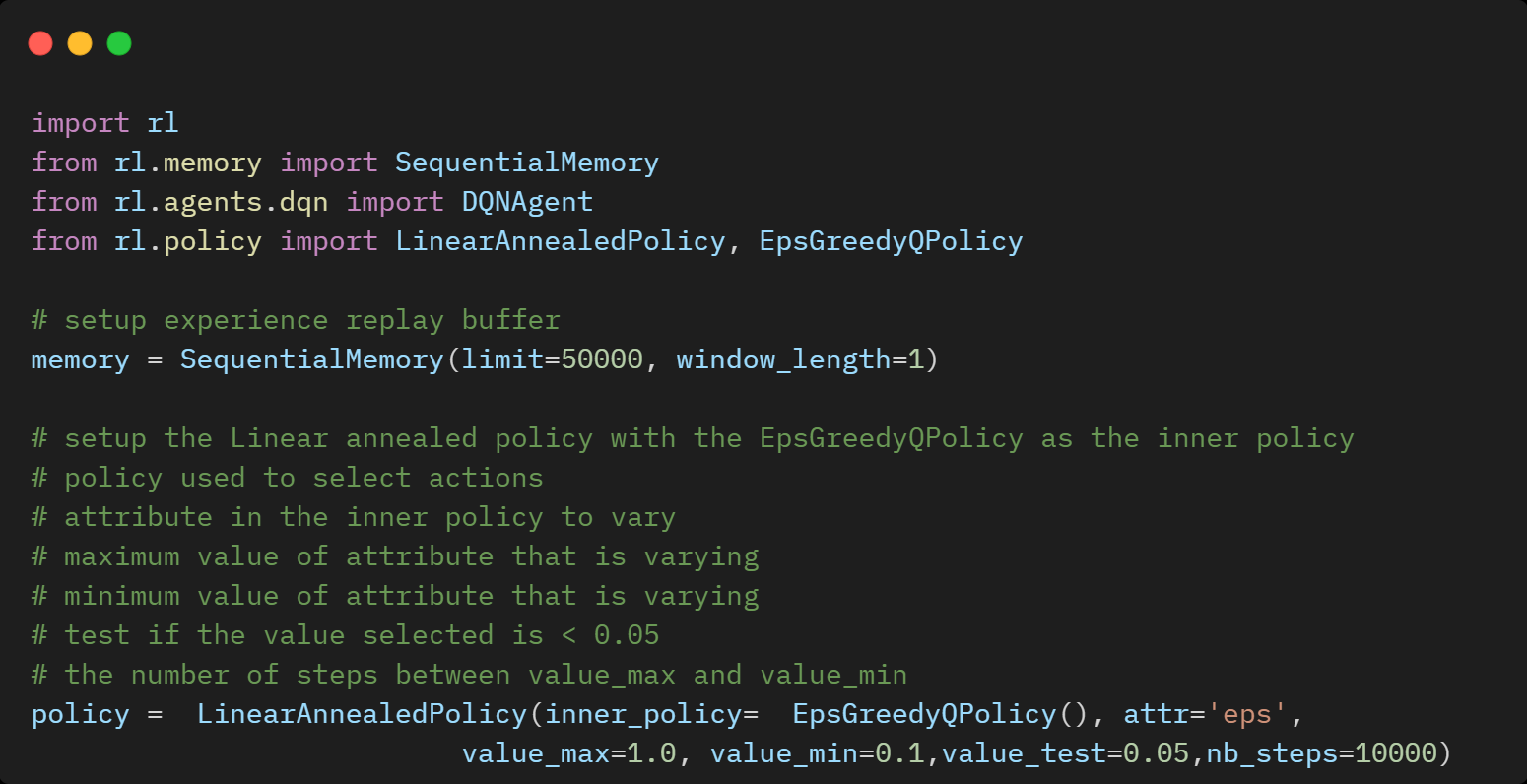
Epsilon-Greedy Policy

Epsilon-Greedy means choosing the best (greedy) option now, but sometimes choosing a random option that is unlikely (epsilon).

The idea is that we specify an exploration rate - epsilon, which is initially set to 1.

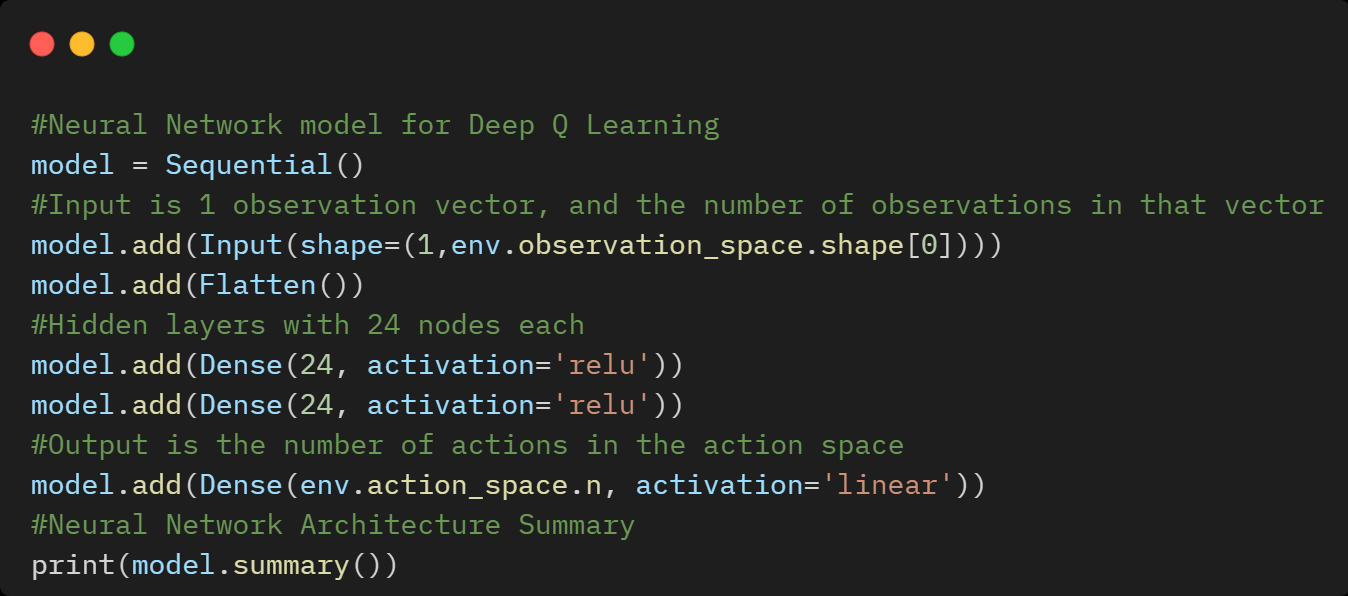
In the beginning, this rate should be the highest value because we know nothing about the importance of the Q table.

In simple terms, we need to have a big epsilon value at the beginning of Q function training and then gradually reduce it as the agent has more confidence in the Q values.

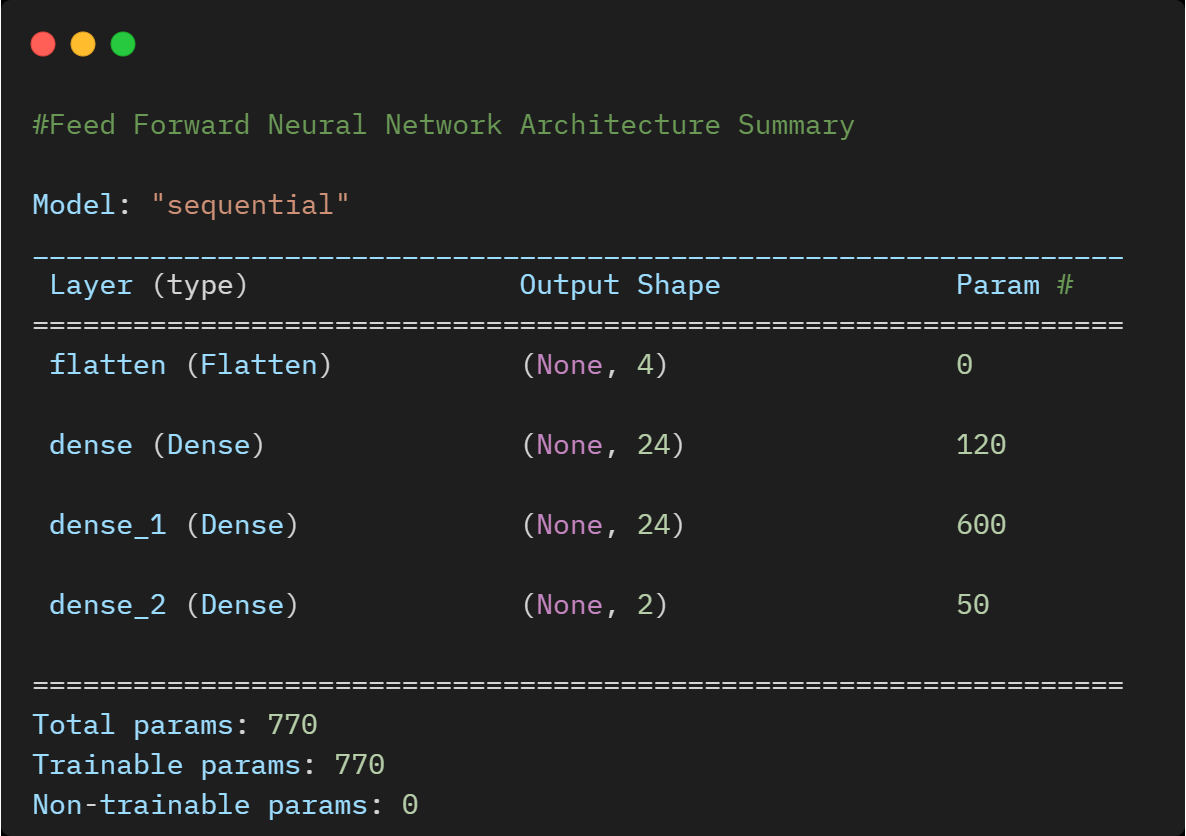


# **5. Building DQN Agent with Keras-RL**

## Feed-Forward Neural Network Model for Deep Q Learning (DQN)



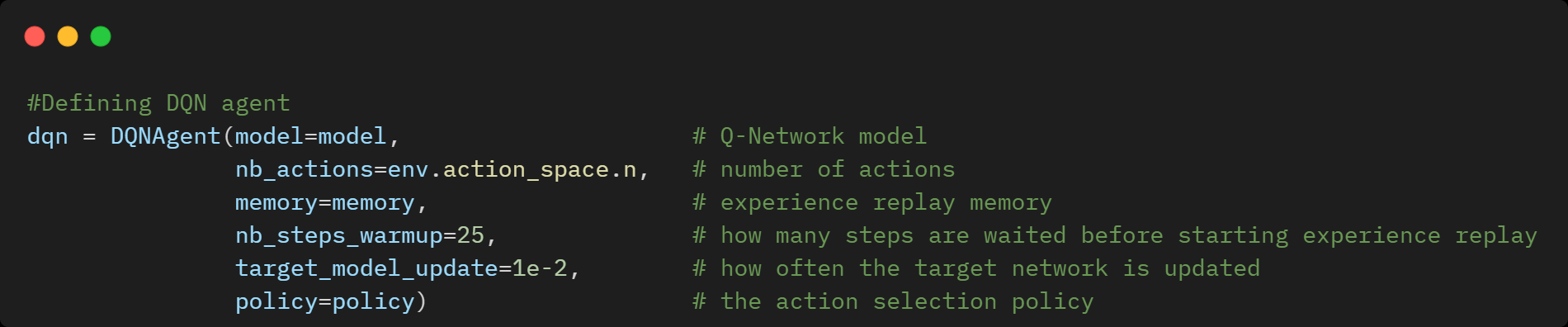
## Feed-Forward Neural Network Architecture Summary



## Defining DQN Agent for DQN Model

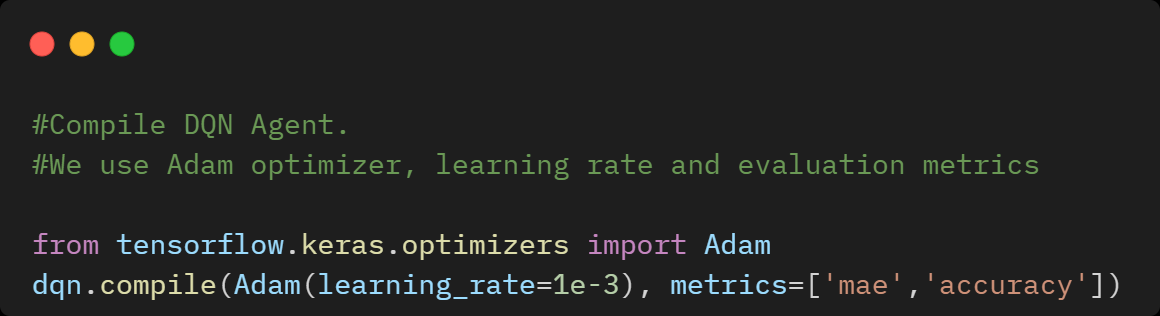
The DQN agent can be used in any environment which has a discrete action space.

The heart of a DQN Agent is a Q - Network, a neural network model that can learn to predict Q-Values (expected returns) for all actions, given an observation from the environment.



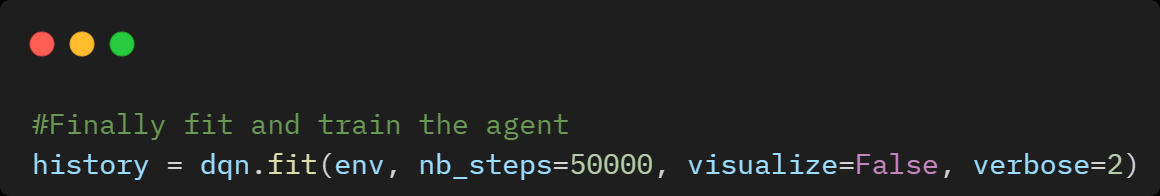
## Compiling the DQN Model

Once the layers are added to the model, we need to compile the DQN agent by using Adam Optimizer, Learning rate and evaluation metrics like MAE and Accuracy.



## DQN Model Training

To start training the DQM model, we will use the “model.fit” method to train the data and parameters with nb\_steps=50000.



## Summary of Training episode steps and the total episodes of the DQN Model

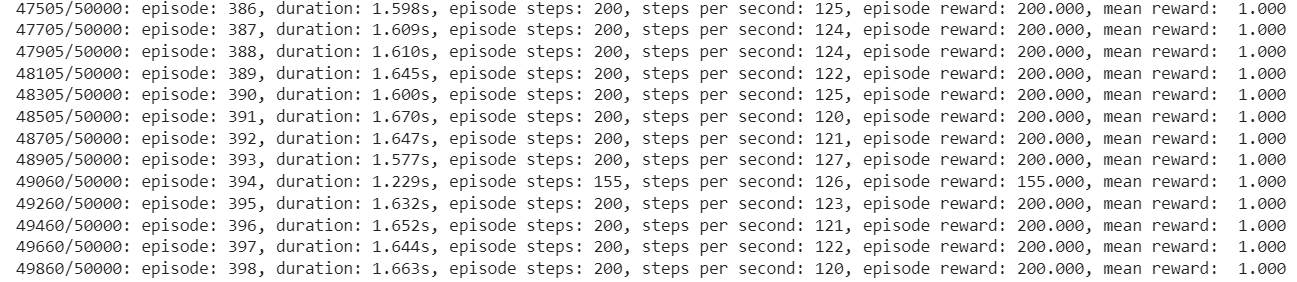
One iteration of the Cartpole-v0 environment consists of 200-time steps.

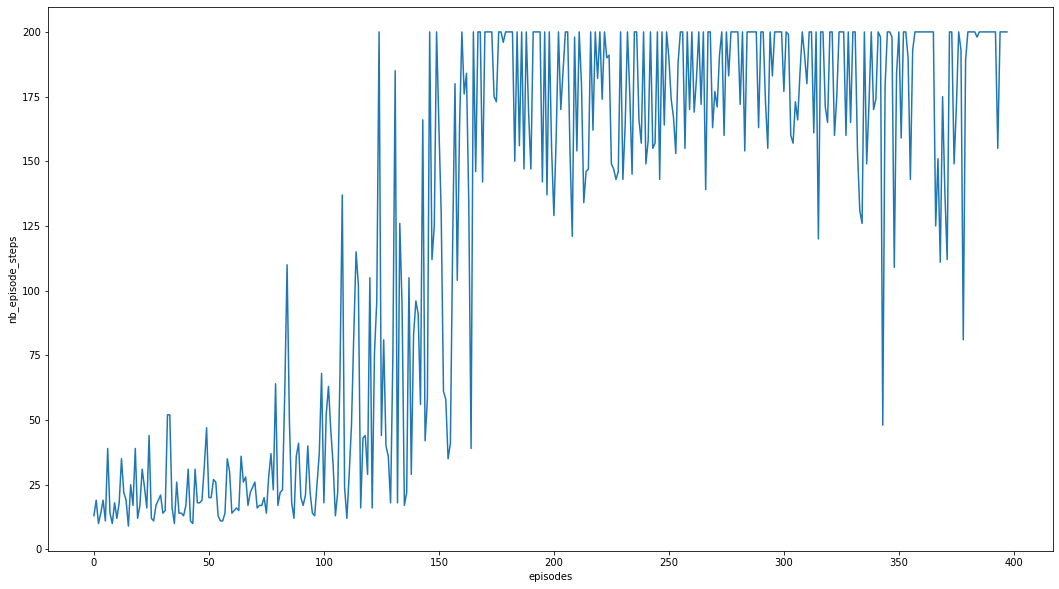
The environment gives a reward of +1 for each step the pole stays up, so the maximum return for one episode is 200.

During its training, this instance of the DQN agent was able to achieve a **maximum reward** of **200** at **125 episodes**, meaning it reached 200 steps without dropping the pole with the least number of possible episodes while training.

From the graph, we can observe how in the first episodes, the rewards stay low as the agent is still exploring the state-space and learning the values for each state-action pair. However, as we complete more episodes, the agent’s performance keeps improving and more episodes are completed.

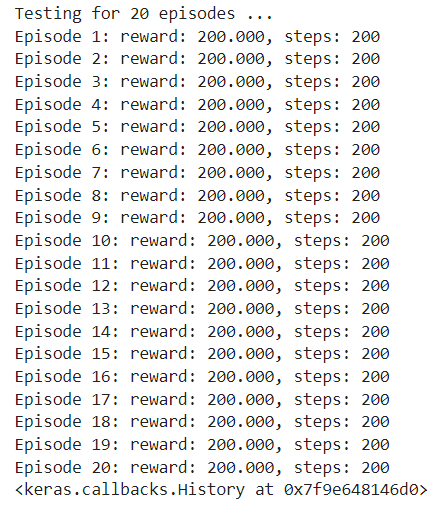
Python Implementation of Balancing Cart Pole using Keras, DQN and Open AI Gym is available on [**GitHub**](https://github.com/ChaithanyaVamshi/CartPole-DQN-Reinforcementlearning/blob/main/CartPole_DQN.ipynb)





# **6. Testing the DQN Agent Model for 20 Consecutive Episodes**

Finally, after testing 20 consecutive episode steps, DQN Model was able to achieve a maximum reward of 200 at each step without dropping the pole.



# **7. Saving the Best DQN Model Weights**

# 

# **Conclusion**

In this blog, we discussed how to implement balancing Cart Pole Game using Deep Q Network (DQN), Keras and Open AI Gym.

While this DQN agent seems to be performing well already, its performance can be further improved by applying advanced Deep Q learning algorithms like Double DQN Networks, Dueling DQN and Prioritized Experience replay which can further improve the learning process and give us better scores using an even lesser number of episodes.

# **References**

[1] <https://gym.openai.com/docs/>

[2] <https://gym.openai.com/envs/CartPole-v0/>

[3] <https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial>