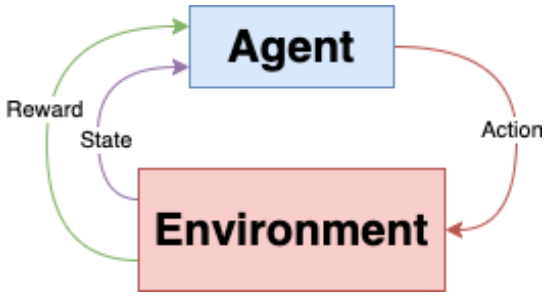
Introduction

The task of automation is not always easy or straightforward. That is apparent in constraint-systems that have agents acting in them to achieve goals. An example of that is games in general. They have many constraints and different objectives. On many occasions, the supervised method of learning is not enough. That is clear in competitive games, where the usage of a supervised model would never make the computer acts better than the data it was fed (i.e. the human who trained it).
One proven unsupervised learning solution is the use of reinforcement learning. One interesting and successful method in it is the deep reinforcement learning.

Reinforcement Learning

Reinforcement learning is an important type of machine learning where there will be an agent which will study the given environment or the environment around it and learn to perform an action based on the given state of the environment then it will evaluate the results to improve its actions in similar environment in the future. In recent years there are a lot of improvements in this fascinating area of research especially in games for example DeepMind AlphaGo could beat the champion of the Game of Go in 2016 which is a very complex game.

The main idea of Reinforcement Learning is learning while going, learning from experience or learning from interaction with the environment where the agent will get a reward for each action it makes, and the rewards could be negative (punishments) or positive and its objective is to learn to act in a way that will maximize its expected rewards over time.



Policy

The policy is the algorithm the agent uses to determine its best action to perform in the given state. They could be a Q-table or a neural networking taking an observation as inputs and outputting the action to take.

Q-Learning

Q-Learning is type of reinforcement learning where it uses Q-value (Quality Values) and Q-table to help the agent to decide which action it should take next. Q-Value takes two inputs and 'state' and 'action' and then it will return the expected future reward of that action at the given state and it will have hyperparameter gamma γ which will determine the weight for the future reward if close to 1 this means we give more weights for the future reward if it small then we only care for the current reward. Then it will use Q-table which the rows will represents all the possible states of the environment and the columns all the possible actions. The values for the Q-table will be initialized to zeros at the beginning and then it will be updated for each action based on the Q-Value, therefore after the training it should have the optimal Q-Value of the state-action pair in each cells of the table.

Exploration vs Exploitation

Exploitation where the agent will allows pick the best choice where it has the highest Q-value in the table so it might be sock between two states because they give it a positive rewards and the agent has not discover the other states, the problem with exploitation the environment the agent might not discover or reach some new states which might have higher rewards, and at the beginning because the agent doesn't know anything about the environment it starts by Exploration which it based on random action so if the agent rely only on exploitation it might get stuck in the first states of the environment without going to newer states on the other side if the agent only use exploration so the agent doesn't learn anything except it take action random. Of course, the agent should explore the environment as much as it can and doesn't take actions randomly therefore the option to balance between exploration and exploitation is using $\epsilon - greedy\ policy$ which the agent at each step will act randomly using exploration with probability ϵ and greedily using exploitation with probability $1 - \epsilon$ Therefore will initialize ϵ to 1 where it will have high probability to explore the environment and with each step ϵ will start decreasing and it will stop decreasing at given threshold. There with ϵ the agent will start acting randomly and explore the environment and with each step the agent good knowledge to start acting based what it knows about the environment.

Deep Q-Leaning

The problem with Q-Learning is that it doesn't scale well to large environment with many states and actions where it is difficult to store all the Q-value for each state-action. Therefore instead of using Q-learning it is better to use Deep Q-Learning where the agent will be a neural network instead of Q-table, therefore the neural network will state-action pair (s, a) and it will output approximation for the Q-value because of that it called Approximate Q-Learning. So the equation for the Q-value is $Q_{target}(s, a) = r + \gamma * max_{a'} Q_{\theta}(s', a')$

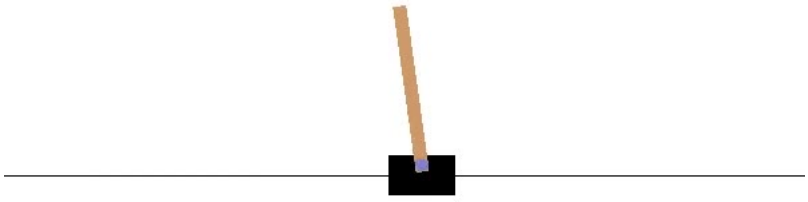
With this target Q-value we can run the training step using Gradient Descent algorithm, where we're going to minimize the squared error between the estimated Q-Value Q (s, a) and the target Q-Value.

Experience Replay

Instead of making our agent only train from latest experiences we can use experience replay to store all the experiences in a replay buffer while interacting with the environment, and then we will sample a random training batch from it at each training iteration. This will help the agent to learn more from previous experiences and reduce the correlations between the experience in a training batch.

Implementing Deep Q-Learning

CartPole (Our environment) "CartPole-v1"



Our environment is CartPole game which we will have a pole above a cart and the cart should balance the pole for as long as possible and the maximum period is 499 steps. In this environment we have 4 different states and 2 actions

States:

1- Cart position
2- Cart velocity
3- Pole angle
4- Pole angular velocity

Actions:

1- Move cart left
2- Move cart right

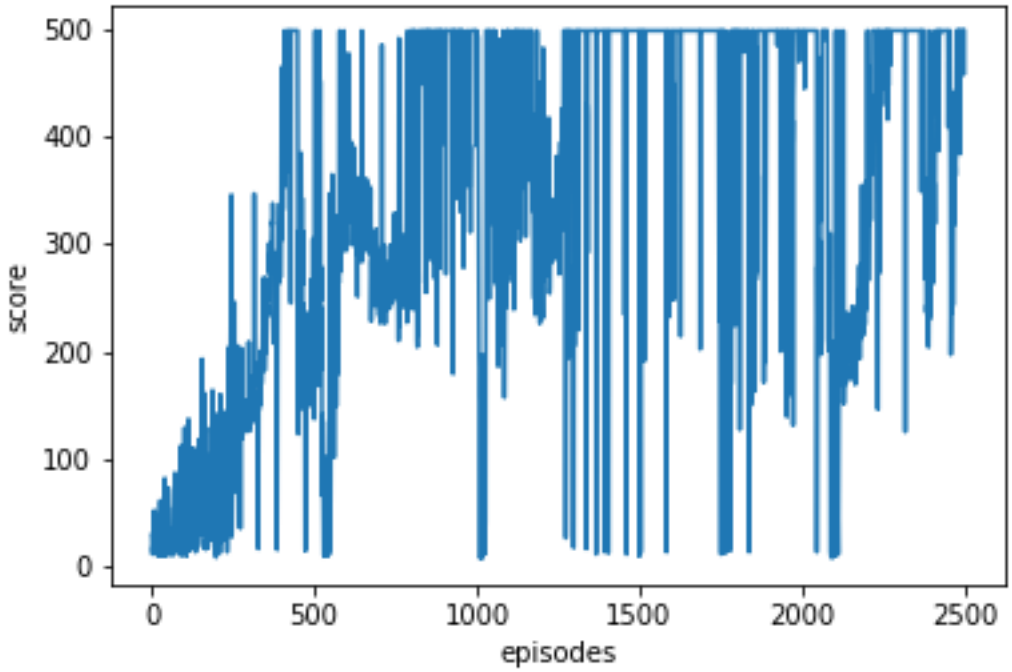
The Agent will lose (get negative reward) whenever the cart get out of the sides or the pole is 15 angular degree in either sides.

Neural Network (Agent)

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                320
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 16)                528
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 34
=================================================================
Total params: 7,122
Trainable params: 7,122
Non-trainable params: 0
```

This is a summary of our neural network.

Training



As shown in the graph the first 200 episodes the agents could not score or balance the pole for more than 200 steps and with each episodes the agent start learning how to balance the pole for longer time and more steps after 400 episodes the agent starts to balance the pole for 300 steps or more.