1. **What is Reinforcement Learning?**

It is the concept of generalized decision making. Given an environment and a set of actions, in which each action generates a reward. The Reinforcement Learning agent takes in the observation of the environment and takes a set of actions which maximizes the reward. This is the concept of RL on a high level.

2. **Keywords or Jargon of Reinforcement learning**

- Markov Decision Process- This is a mathematical concept or the objective which needs to be solved by the Reinforcement Learning algorithm. In this it is assumed that the whole environment is observable which is not true in practical scenarios. The sub concepts of MDP are as follows
    - States- the description of the environment in which the agent is present.
    - Model- the agent rules is defined by the model which basically states that, which action the agent needs to take given the state and will give the maximum reward.
    - Actions- set of steps which the agent can take given a state.
        - Deterministic actions can be there in which all the actions are distinct in nature
        - Stochastic actions can be there in which there is a degree of uncertainty which means a probability distribution over the range of actions is present.
    - Reward- the scalar value which the agent receives after taking an action.
- Policy- Transition function which maps the set of states to actions.
- POMDP(Partially observable MDP)- Same as MDP but more related to practical world as it considers that the model cannot observe the dynamics of the environment fully.

3. **Ways to solve MDP**

- Model-Based RL- In this the dynamics of the environment is used to determine the set of actions which need to be taken. This is not possible in a real environment. Normal planning algorithms are used for this task.
- Model-Free RL- These are not based on dynamics of the environment, rather a set of equations and notations are introduced which are optimized to solve the MDP.
    - Policy Optimization- Determining the ideal policy or the state action pairs. It requires a lot of history to converge. Eg- Policy gradients(on-policy)
    - Dynamic Programming- Evaluating the quality of actions in real time. This requires less history as compared to above methods. Eg- Value iteration(off-policy), Policy iteration(on-policy).

4. **Value Iteration**

This is a Model free based RL technique which is used to solve the MDP. This basically means that determining the best set of actions which gives the maximum reward.

5. **Policy Iteration**

This is also a Model Free based RL technique which is used to solve the MDP. Here the policy based parameters are used to determine the final policy and the state action pairs. The major drawback of this is a lot of samples are needed to converge on the policy.

6. **Solving both policy and value**

Both use the bellman equation to solve the MDP. The algorithms which are used to solve the bellman equation are as follows-

1. Temporal difference learning- This optimizes the error and tries to minimize it.
2. Monte Carlo Search tree- Uses bootstrapping and gradients to come up with the ideal values. They are useful in high dimension spaces.

7. **Why does Bellman Equation Converges**-

This is because it supports contraction mapping. Anything which supports this always converges to an optimal value or policy.

8. **Finite horizon vs infinite horizon**

Finite horizon means the end is deterministic while infinite horizon means the end it not deterministic. Therefore in that case a discount factor is introduced to make the non-deterministic as deterministic. This discount factor is between 0 to 1 which means that immediate reward is given more importance than the late rewards. This is the meaning of the discount factor introduced in the solving of the bellman equation.

Algorithms

1. Monte Carlo Based Vanilla Policy Gradient

This is an on-policy based algorithm which basically means that the updates are directly done on the policy rather than on the basis of any value functions. In this algorithm derivative of the policy gradient is done and gradient descent type algorithm is used to converge the gradients. The updates and the details of the algorithm has been stated in this link-https://www.freecodecamp.org/news/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f/

More Maths based details are stated in this link-https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html

Observations after implementing Monte Carlo Based Vanilla Policy Gradient-

1. Policy gradients are very uncertain and have high variance.
2. They take long time to learn
3. The Neural network is the one which determines the policy

Deep Q Learning

It is an off policy, model-free Reinforcement learning. Here the value function is directly estimated using a neural network. Firstly the state passes through the CNN and the linear output layer determines the Q values. The estimated Q value is moved towards the target optimal Q value which is determined better and better as and when the Q network gains experience. A memory is also maintained for a batch based update. Lastly, we manipulate with the frames so that differentiability between frames can be attained which basically means grabbing every 4th frame rather than every frame. If terminal then the reward is directly fed, else discounted reward using the determined Q value at that point of time is used.

The paper has been added to the repository for further details.

A few implementation details-

1. I have used Epsilon Greedy strategy so that equal amounts of exploration and exploitation can be done.

Observations-

1. The Algorithm starts to play well quite fast unlike vanilla policy gradient algorithms.
2. The hyperparameter setting and architecture determination is a difficult task.

A2C- Synchronous Actor to Critic

It is a hybrid method which combines both value based and policy based to create a Reinforcement Learning algorithm.

Critic- Measures how good a action is(Value based)
Actor- Behaves according to the environment(Policy)

It uses TD Learning

In a sense it uses a teacher student model. The updates are similar to when used individually just a few variable substitutions are there.

High level algorithm-

1. At each time-step t, we take the current state (St) from the environment and pass it as an input through our Actor and our Critic.
2. Our Policy takes the state, outputs an action (At), and receives a new state (St+1) and a reward (Rt+1).
3. the Critic computes the value of taking that action at that state
4. the Actor updates its policy parameters (weights) using this q value

The main intuition of A2C is that, normal policy based updates are having very high variance but work very well, so to deal with the variance we do a few baseline based substitution so that stability in the algorithm is attained without giving up on performance

Great explanation-
https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

One Additional Implementation change done-

1. Updating the Policy values in every step rather than waiting for the episode to complete. Traditionally in all the available algorithms of A2C, the values are being updated after the episode is complete which does does make use of the advantage which TD Learning gives rather is just monte carlo based estimate only.
2. Stacking of frames was used so that motion detection can be done.