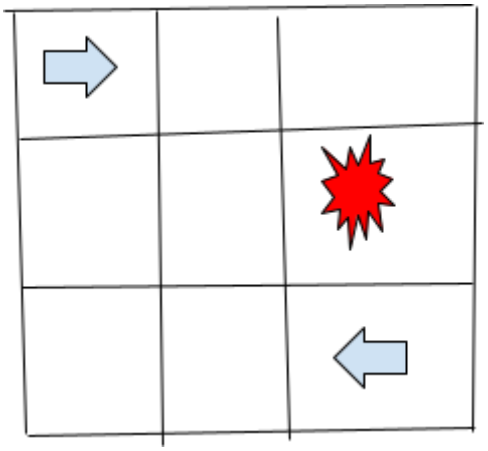
Hello,

Today I am going to explain about q learning and PPO, which took me quite a time to understand. So, without further adieu, let us begin,

In Q learning, we have q tables, which help us in making progress. Typically we use q learning in easy environments. So, for this scenario let us take the example of a game, which is an easy game and there are very few moves alright. So, in this game there is a start point and an end point and there are several other points. Take a grid as shown below, where the top left is a starting point and the bottom right is the end point. And there is a bomb in one the cell and that will give us a huge negative reward and we want to step on this bomb. So, the goal of this problem is to reach the end goal without stepping in a bomb and be at the end state as quickly as possible.



So, in Q learning we will have a table for each state action pair that is possible and there will be some q value assigned to it. Also, not to forget that there will q values for all the actions that are possible from that state and we want to improve those q values and that is what we will do over here.

For each episode(game play):

Get a state by doing a reset in case using a gym environment, which gives us an observation, reward and whether done or not.

For each time step:

Get the q values for the state that we got.

max(q-vals)

Take an action for which the q val is max.

State and reward(prev selected action)

q-vals(for that state)

Update the q values of the earlier state based on this reward, if you get some reward.

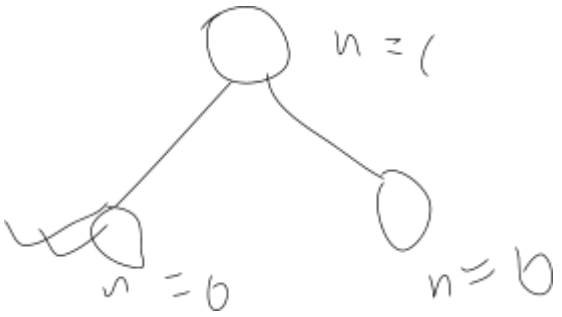
Initially what the algorithm will do is it will take some random steps cause there would be some random values to it and after the first game play there will be a change in the last value. And again if it plays a game there will be some updates to avoid certain action or to encourage some action to take place so in a way it is actually trying to prune the tree.

It is not fully using Monte carlo algorithm but it is using some parts of it. In Monte carlo what happens is, it traverses the whole tree and not just looks at the next state at each state.

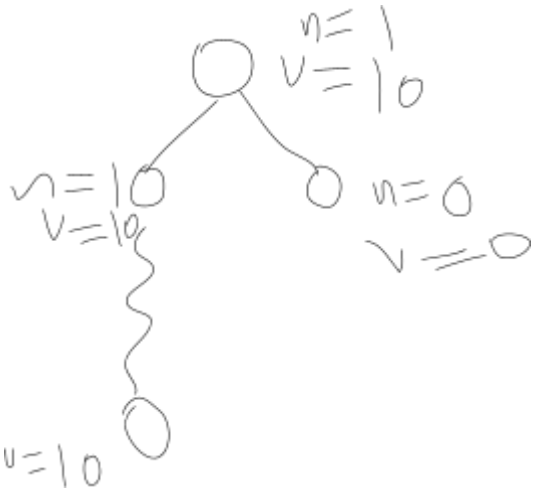
Monte Carlo algorithm:

In Monte carlo algorithm there is something called UCB score, which tells us whether to explore or to exploit. So, it gives an equation, where we will be using the number of times the node was traversed and also the value which will essentially the sum of all the values down that tree
Before any traversal actually there would be a tree given where in we will have some nodes already attached to the root node.

It starts with the root node and then traverses it if it is already not traversed and takes the best child acc to UCB score, so both of these would be best because both of them will have the same UCB score and then it takes the first one



in order and asks if that is a leaf node and then if



so it does the roll out and updates the value in the backward fashion. As we can see that in the above figure and the next iteration then begins in which, it will be asked if the root node has child and yes it does and then it calculates the UCB score of both these child and picks up the one which has the max value and asks if that is a leaf node and if so it will roll out else it will traverse further.

Now, let us move on this new algorithm called DQN, the reason why we introduced this algorithm is because q learning is not really good for the environments which are complex and the reason is because, the q table that we are maintaining becomes exponentially large and we will run into these memory issues so it does not do quite well in that case you know.

Now let us go through the algorithm and rather take a complex environment, let us take an example of a game in this case, and let us take the example of this star trek game in which we have like a player playing a game and there are horrendous amounts of action from any particular state. Or you could also take an example of this atari game where there are just two actions to take on any given state but the thing is that we need to know the velocity and everything, basically the information about the ball whether it is going up or down or so we will have images as input in this case.

Now, you might think of a lot of things in your mind like how will you define a new state or something and for each action who will give us a new state, just like I had a lot of questions in my mind when I was trying to wrap my head around this problem. So, now

Let us go through the algorithm,
We will have a bunch of images as an input for this game, let us say atari in this case.

Now, we will give these bunch of images as input to the neural network and neural network which predict the q values for these bunch of images. Q values will be for each given action that could take place at a certain state. So, initially we will get some values to this neural network cause of course it is not trained.

So,

For each episode:
    game.reset()
    For each time step:
        Images as input to the neural network()
        Neural network will give us some q values for these inputs
        Max(q -values)
        Give us a new state New_state = this new state
        MemoryBufferStore(State, Action, Reward, New_state)

        Now, sample a batch from this memory buffer and you will get a bunch of these pairs that you stored in your memory buffer.
        (S, A,Rt+1, St+1)
        Now, do the forward pass for these S from the policy network and you will get the q values of the different possible actions that we could take from this state S and now do the second pass from the target network for the new state for the action that you have taken and you will get different q values from this target network and you need to select the best of the q values from these actions that are there.

Now, then what you need to do is calculate the loss for this and update the policy network and you update the target network after some fixed number of steps, alright!

What at this point is a little unclear about the next action that we gonna take at the beginning of the time step, not really sure which action we are talking about over here.

Next I wanted to discuss the PPO algorithm, and this algorithm is the recent one that we got to know in 2017.

So, what is different in this algorithm is we do have stored experiences here like the older one. Rather what we do over here is we reject the exp once we learn something from them rather than storing it in the memory buffer.
It could be used both for deterministic and non deterministic algorithms. And both for continuous and discrete ones. Now, what we have over here is rather complex I guess.

So, we have two neural networks over here.
In here as well we run it for several episodes and what we do have over here is we have a state and and in each time step what we do is we have something called actor and critic model which will basically take the images of the states that we gave. In the actor model what we do is we take a state and it gives us some probabilities so basically q values for all the possible actions that one can take from a given state. And we store those probabilities and we store that state and we also store the reward. We also have another neural network which is responsible for telling about how good a given state is. So it will just give us back a single value unlike the actor model that we defined earlier. Now we store this value as well so we keep on doing that for certain time steps basically for batch time steps

Once we are done storing all these values, what we do next is we get the advantages and returns for these values, and these values will be useful while defining the PPO loss,

In get advantage function what we do is, we iterate the loop of stored batches in a reverse fashion

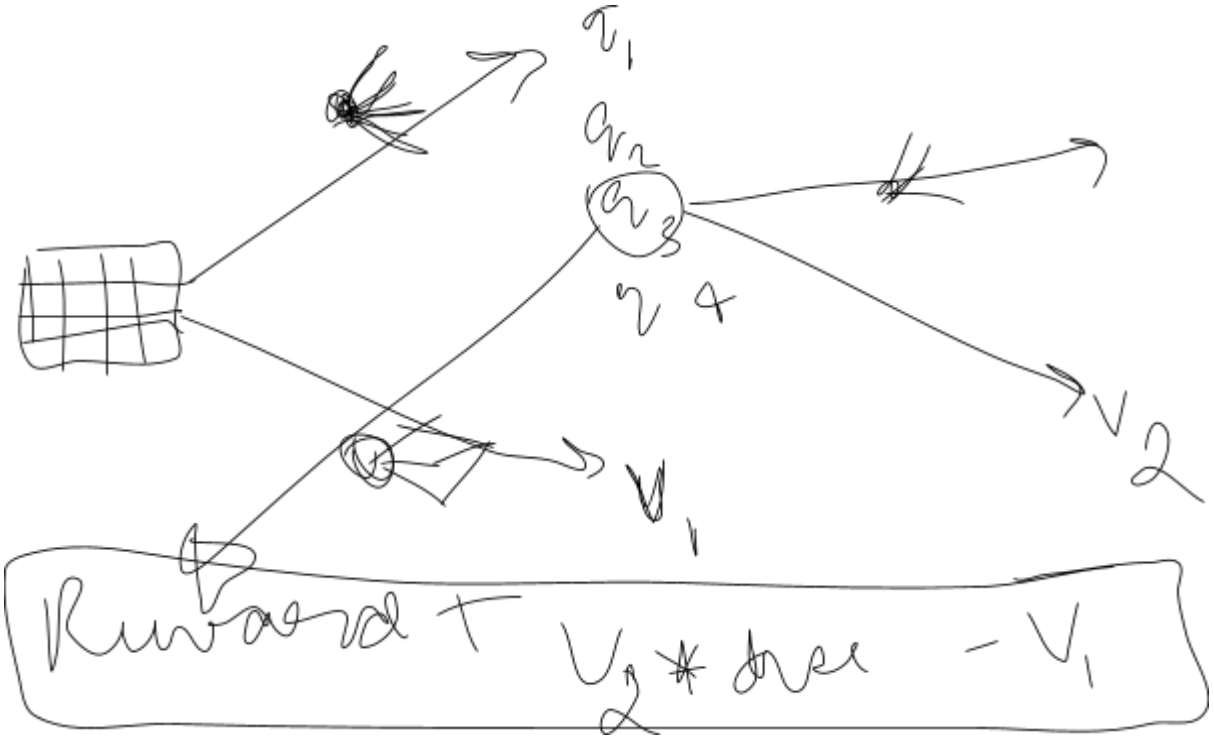
$$delta = rewards + gamma * values[i+1] * mask - values[i]$$

$$gae = delta + gamma * lambda * masks * gae$$

$$returns.insert(0, gae + values[i])$$

Intuitively what we are doing over here is we are taking the rewards that we got by taking some action and we are adding gamma times values that we got from the critic model by

taking that action and subtracting the value that we already got from the actor model.



After that what we try to do is we try to add this gain during this gain over the future iteration and we take care of the discount factor.

We then insert this gae over time and also the values and we also store and that would just be the returns - values that we added in the last step.

Next what we do is we call the actor model to fit these values that we provided that is we will provide is we are providing action probabilities and we are giving results, states, advantages, values etc and we also call critic model to fit these values which mainly aims at giving us the value for a state to describe how good a state is.

What will happen during fitting the actor model for these values:

ppo_loss():
Old policy probs are just the probs that we are giving and new policy probs would just mainly be the probs once we fit those, so of course we will get something out of it, but what I was thinking is wouldn't those values just be the same as the old ones?

We calculate the ratio between these probs and
P1 = ratio * advantages
P2 = clip the ration between max and min values.
Actor loss
Critic loss
Total loss
Still am not completely getting how this ypred is happening