RBE 595 — Reinforcement Learning
Deep Reinforcement Learning Assignment

Arjan Gupta

Problem 1

What are the two sources of error in Deep RL with function approximation?

Answer

The two sources of error in Deep RL with function approximation are as follows:

- **Bootstrapping error** — This is the error that arises due to the use of bootstrapping. Bootstrapping is the process of using the value of a successor state to update the value of a state. This is done in TD methods. The error is the difference between the target value and the current estimate value.
- **Approximation error** — This is the error defined as the difference between the true value function and the approximate value function. This error arises due to the use of function approximation itself.

Problem 2

In TD learning with a neural network what are we trying to minimize? What are we trying to maximize?

Answer

In general with any deep TD learning method, we aim to minimize the error between the target value and the current estimate Q-value. We try to maximize the value of the current state by choosing the action that maximizes the value of the next state.

We can use the example of gradient Q-learning to answer this question more specifically. In gradient Q-learning, we are trying to minimize the error given by the loss function as follows:

$$e(w) = \frac{1}{2}\left[Q_w(s,a) - \left(r + \gamma \max_{a'} Q_{w'}(s',a')\right)\right]^2$$

We are trying to maximize the value of the current state by choosing the action that maximizes the value of the next state. This is done by updating the weights of the neural network.

Problem 3

What is the main benefit of deep neural networks in function approximation compared to linear method? What is the linear function approximation useful for?

Answer

Deep neural networks (DNNs) have helped reinforcement learning become more powerful and practical. DNNs can be used to approximate the value function as a non-linear function of the state. This is more powerful than linear function approximation, which can only approximate the value function as a linear function of the state. A general advantage of DNNs is also that they learn their own features, which is not the case with linear function approximation.

Linear function approximation are still useful, however. Their advantage is that they can be used to verify theoretical results (in academia, for example). The mathematical analysis of linear function approximation is much easier than that of non-linear function approximation, which is why linear function approximation is still useful.

Problem 4

In DQN, what is the purpose of the target network and value network?

Answer

In DQN, two separate neural networks are used: the target network and the value network. The target network is used to estimate the target value, whereas the value network is used to estimate the Q-value. The purpose of having two separate networks is to serve as one of the two features of DQN that mitigates the problem of divergence (or weaker convergence) that occurs in gradient Q-learning. The other feature is experience replay.

We can provide more context on how DQN achieves this mitigation, as follows. The target network is updated less frequently than the value network for each mini-batch of (s, a, r, s') tuples from the experience replay buffer. This is done to make the target network more stable. The target network is updated as follows:

$$w_{i+1} = w_i + \alpha \left[Q_w(s, a) - r - \gamma \max_{a'} Q_{\overline{w}}(s', a') \right] \frac{\partial Q_w(s, a)}{\partial w}$$

Where the first Q in the square brackets is the value network, and the second Q is the target network. The second Q remains 'fixed' for a certain number of iterations. This is akin to the optimal value function being fixed while the value function for all the other states is being updated.

Problem 5

In the Deep Q-learning method, which are true:

(a) epsilon-greedy is required to ensure exploration.

(b) exploration is taken care of by the randomization provided by experience replay.

Answer

(a) is true. Epsilon-greedy is required to ensure exploration as seen in the 'Deep Q-learning with Experience Replay' paper's algorithm. This is because the action selection policy is greedy. This means that the agent will always choose the action that maximizes the Q-value. So, in order to ensure exploration, epsilon-greedy action selection is used.

(b) is false. Exploration is not taken care of by the randomization provided by experience replay. Experience replay is used to break the correlation between consecutive samples. This is done by storing the samples in a buffer and sampling from the buffer randomly. This is done to make the training process more stable. However, this does not take care of exploration because the agent is not experiencing or simulating any new orders of states. The agent is only experiencing the same states in a different order.

Problem 6

Value function-based RL methods are oriented towards finding deterministic policies whereas policy search methods are geared towards finding stochastic policies:

(a) True

(b) False

Answer

This is true, value-function based RL methods are oriented towards finding deterministic policies. The reason for this is that value-function based RL methods focus on estimating the value function in order to find the optimal policy. The optimal policy is the policy that maximizes the value function. This is by definition a deterministic policy. We can manually introduce some 'stochasticity' in the policy by using an ϵ-greedy policy, but this is not the same as a truly stochastic policy.

It is also true that policy search methods are geared towards finding stochastic policies. This is because policy search methods directly search for the optimal policy, so they focus on building a mapping which turns out to be a distribution over actions. This is by definition a stochastic policy.

Problem 7

What makes the optimization space smooth in policy gradient methods?

Answer

In policy gradient method, the optimization space is the policy parameter space. In general, stochastic policies are used in policy gradient methods. This means that the policy is a distribution over actions, like a Gaussian distribution. This makes the optimization space smooth.

Problem 8

How does actor-critic architecture improve the "vanilla" policy gradient?

Answer

The actor-critic architecture improves the 'vanilla' policy gradient in the following ways:

1. providing an improvement on the advantage function which regular baseline functions cannot provide. Specifically, REINFORCE-with-baseline tends to learn slowly and produce high variance. Actor-critic methods can improve on these issues by using temporal-difference (TD) learning as well as n-step bootstrapping.
2. using neural networks to approximate the value function and form a distribution for the policy. This provides gradients for both the value function and the policy, with which we can optimize both the value function and the policy.

The 'Critic' part of the actor-critic architecture is the estimate of the value function. The 'Actor' part of the actor-critic architecture is the policy. These are two separate neural networks.

Problem 9

What is the Advantage function, $A_\pi(s_t, a_t)$, in actor-critic architecture? Write down the equation for monte-carlo-based Advantage function and TD-based advantage function and comment on the pro and cons of each on.

Answer

In order to properly explain the advantage function, let us look at the formula for the 'vanilla' policy gradient, which is as follows:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) R(\tau^i) \tag{1}$$

In general we can improve the vanilla policy gradient by subtracting a baseline such that (1) becomes:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)(R(\tau^i) - b(s_t^i)) \tag{2}$$

This reduces the variance of the policy gradient. The (s_t^i) is called the baseline. We have different options for the baseline, such as the average reward of the trajectories or the expected return. We can also use the value function as the baseline. When we do this, it is called the *advantage function*. We can rewrite (2) as follows:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) A_\pi(s_t^i, a_t^i)$$

Where $A_\pi(s_t^i, a_t^i) = \sum_{k=t+1}^{T} r(s_k^i, a_k^i) - V_\pi(s_t^i)$ is the *advantage function*. It can take on different forms, such as the Monte Carlo-based advantage function and the TD-based advantage function. The Monte Carlo-based advantage function is as follows:

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$$

The TD-based advantage function is as follows:

$$A_\pi(s_t, a_t) = r_t(s_t, a_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$$

Problem 10

Can you find a resemblance between actor-critic architecture and generalized policy iteration in chapter 4?

Answer

Yes, there is a resemblance between actor-critic architecture and generalized policy iteration. In actor-critic architecture, the 'Critic' part of the architecture is the value function. The 'Actor' part of the architecture is the policy. These are two separate neural networks that can be trained separately to improve V and π respectively. In generalized policy iteration, there are similarly two separate processes: policy evaluation to estimate the value function and policy improvement to improve the policy. These two processes are iterated over until convergence, similar to how the actor-critic architecture is iterated over until convergence.