Q Learning Methods

Devesh Nath

February 10, 2025

1 Q-Iteration

Q-Iteration is a method used in reinforcement learning to find the optimal policy for a given Markov Decision Process (MDP). The algorithm iteratively updates the Q-values for each state-action pair until convergence. The update rule for Q-Iteration is given by:

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

where:

- $Q_k(s,a)$ is the Q-value for state s and action a at iteration k.
- $P(s'|s,a)$ is the transition probability from state s to state s' given action a.
- $R(s,a,s')$ is the reward received after transitioning from state s to state s' with action a.
- γ is the discount factor.

2 Q-Learning

Q-Learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It does this by learning the Q-values for each state-action pair through exploration and exploitation.
The update rule for Q-Learning is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

where:

- $Q(s,a)$ is the Q-value for state s and action a.
- α is the learning rate.
- $R(s,a,s')$ is the reward received after transitioning from state s to state s' with action a.
- γ is the discount factor.
- $\max_{a'} Q(s',a')$ is the maximum Q-value for the next state s' over all possible actions a'.

Q-Learning is model-free, meaning it does not require knowledge of the transition probabilities or the reward function. It can be used in environments where the model is unknown or too complex to be explicitly defined.

3 Deep Q-Learning

Deep Q-Learning (DQN) is an extension of Q-Learning that uses deep neural networks to approximate the Q-values for each state-action pair. This allows the algorithm to handle high-dimensional state spaces that are infeasible for traditional Q-Learning.
The key components of Deep Q-Learning are:

- **Experience Replay**: A technique where the agent stores its experiences (state, action, reward, next state) in a replay buffer and samples mini-batches from this buffer to train the neural network. This helps to break the correlation between consecutive experiences and improves the stability of the training process.
- **Target Network**: A separate neural network that is used to generate the target Q-values for training. The weights of the target network are periodically updated to match the weights of the main network, which helps to stabilize the training process.

The update rule for Deep Q-Learning is given by:

$$Q(s,a;\theta) \leftarrow Q(s,a;\theta) + \alpha \left[R(s,a,s') + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta) \right]$$

where:

- $Q(s,a;\theta)$ is the Q-value for state s and action a approximated by the neural network with weights θ.
- α is the learning rate.
- $R(s,a,s')$ is the reward received after transitioning from state s to state s' with action a.
- γ is the discount factor.
- $\max_{a'} Q(s',a';\theta^-)$ is the maximum Q-value for the next state s' over all possible actions a' approximated by the target network with weights θ^-.

Deep Q-Learning has been successfully applied to a variety of complex tasks, including playing Atari games at a superhuman level.

4 Double Deep Q-Learning

Double Deep Q-Learning (Double DQN) is an improvement over Deep Q-Learning that addresses the overestimation bias present in the Q-learning update. This bias occurs because the same values are used to both select and evaluate an action, leading to overly optimistic value estimates.
Double DQN mitigates this by decoupling the action selection from the action evaluation. It uses the main network to select the action and the target network to evaluate the action.
The update rule for Double Deep Q-Learning is given by:

$$Q(s,a;\theta) \leftarrow Q(s,a;\theta) + \alpha \left[R(s,a,s') + \gamma Q(s', \arg\max_{a'} Q(s',a';\theta);\theta^-) - Q(s,a;\theta) \right]$$

where:

- $Q(s,a;\theta)$ is the Q-value for state s and action a approximated by the neural network with weights θ.
- α is the learning rate.
- $R(s,a,s')$ is the reward received after transitioning from state s to state s' with action a.
- γ is the discount factor.
- $\arg\max_{a'} Q(s',a';\theta)$ is the action that maximizes the Q-value for the next state s' according to the main network.

- $Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-)$ is the Q-value for the next state s' and the selected action, evaluated by the target network with weights θ^-.

By using separate networks for action selection and evaluation, Double DQN provides more accurate value estimates and improves the stability and performance of the learning process.

4.1 Steps for Implementing Double Deep Q-Learning

The steps for implementing Double Deep Q-Learning are as follows:

1. **Initialize the main network and target network**: Create two neural networks with the same architecture. The main network is used for action selection, and the target network is used for action evaluation.
2. **Initialize the replay buffer**: Create a replay buffer to store the agent's experiences, including the state, action, reward, and next state.
3. **Set hyperparameters**: Define the learning rate (α), discount factor (γ), batch size, and the frequency of updating the target network.
4. **For each episode**:
    - (a) Initialize the environment and get the initial state.
    - (b) **For each step in the episode**:
        - i. Select an action using an ϵ-greedy policy based on the Q-values from the main network.
        - ii. Execute the action in the environment and observe the reward and next state.
        - iii. Store the experience (state, action, reward, next state) in the replay buffer.
        - iv. Sample a mini-batch of experiences from the replay buffer.
        - v. For each experience in the mini-batch:
            - A. Compute the target Q-value using the target network:

              $$y = R(s, a, s') + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-)$$

            - B. Update the main network by minimizing the loss between the predicted Q-value and the target Q-value.
        - vi. Periodically update the target network by copying the weights from the main network.