Deep Q-Networks (DQN)

October 1, 2023

1 Introduction

Deep Q-Networks (DQN) are a combination of Q-Learning and deep neural networks, introduced by [Mnih(2013), Mnih(2015)]. DQN extends traditional Q-Learning by using a neural network as a function approximator for the Q-function, enabling the handling of high-dimensional input spaces, typically images.

2 Q-Learning

The Q-Learning update rule is defined as follows:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) \right) \tag{1}$$

Where:

- $Q(s_t, a_t)$ is the Q-value of taking action a_t in state s_t.
- α is the learning rate.
- r_t is the immediate reward of taking action a_t in state s_t.
- γ is the discount factor.
- $\max_a Q(s_{t+1}, a)$ is the maximum Q-value over all possible actions in the next state s_{t+1}.

3 Deep Q-Networks (DQN)

DQN uses a deep neural network to approximate the Q-function. The primary innovations introduced by DQN are Experience Replay and a Target Network.

3.1 Experience Replay

DQN stores the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ in a replay buffer. Random mini-batches of experiences are sampled from this buffer to update the Q-network, breaking the correlation between consecutive experiences and stabilizing training.

3.2 Target Network

DQN uses two separate networks with identical architectures but different parameters: the Q-network with parameters θ, and the target Q-network with parameters θ^-. The Q-network is used to select actions and is updated at every iteration, while the target Q-network is used to compute the target values in the Q-learning update and is updated less frequently.

The update for the Q-network parameters is given by:

$$\Delta\theta = \alpha(r + \gamma \max_a Q(s', a; \theta^-) - Q(s, a; \theta))\nabla_\theta Q(s, a; \theta) \tag{2}$$

The target Q-network parameters θ^- are updated less frequently by copying the Q-network parameters θ:

$$\theta^- \leftarrow \theta \tag{3}$$

4 Algorithm

The DQN algorithm can be outlined as follows:

1. **Initialize:** Initialize the Q-network and the target Q-network with random weights.
2. **Experience Replay Buffer:** Initialize the experience replay buffer.
3. **Exploration Policy:** Choose an exploration policy (e.g., epsilon-greedy).
4. **For** each episode:
   (a) **Initialize State:** Observe the initial state s.
   (b) **For** each time step:
      i. **Select Action:** Select an action a using the exploration policy based on the Q-network.
      ii. **Execute Action:** Execute the selected action a and observe the reward r and the next state s'.
      iii. **Store Experience:** Store (s, a, r, s') in the experience replay buffer.
      iv. **Sample Mini-Batch:** Sample a random mini-batch of experiences from the buffer.
      v. **Compute Targets:** Compute the target values using the target Q-network.
      vi. **Update Q-Network:** Update the Q-network parameters using gradient descent.
      vii. **Update Target Network:** Every few iterations, update the target Q-network parameters.
      viii. **Next State:** Set $s \leftarrow s'$.
   (c) **End For**
5. **End For**

References

[Mnih(2013)] Volodymyr Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Mnih(2015)] Volodymyr Mnih. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.