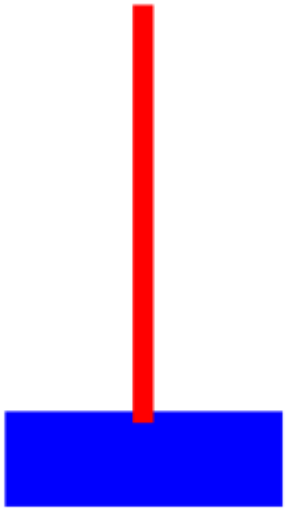
DQN for solving CartPole Env



CartPole Environment Overview

1. **Objective**:
    - Balance a pole on a moving cart for as long as possible by applying force to the left or right.
    - The environment provides a reward of +1 for every step the pole remains balanced.
2. **State Space (env.observation_space)**:
    - A 4-dimensional vector representing:
        1. Cart position (x).
        2. Cart velocity (x').
        3. Pole angle (θ).
        4. Pole angular velocity (θ').
3. **Action Space (env.action_space)**:
    - Discrete with 2 possible actions:
        - 0: Move cart to the left.
        - 1: Move cart to the right.

4. **Termination**:
    - Episode ends if:
        - Pole angle is more than ±12 degrees.
        - Cart moves beyond ±2.4 units from the center.
        - Episode length reaches 200 steps (for the default CartPole-v0).
5. **Reward**:
    - +1 reward per step until the episode ends.

Base Q Network:

This code implements a **Deep Q-Network (DQN)** for solving the "CartPole" environment from OpenAI Gym. The neural network has **one hidden layer** with **128 neurons**. This is sufficient for solving a simple environment like CartPole.

Please consider input of NN is state representation tensor from the ENV with (4,) shape and output of NN is one of the actions in CartPole ENV 0(left) or 1(right) so output has shape (2,).

Considering all these point the base Q_network is a Sequential container. It consists of:

1. A fully connected layer from obs_size to hidden_size.
2. A ReLU activation function.
3. Another fully connected layer from hidden_size to n_actions.

Target Network:

One of the key component in DQN is target Network but what is Target Network and its functionality?

The **Target Network** plays a crucial role in stabilizing the learning process of **Deep Q-Networks (DQN)**. It addresses the issue of **temporal difference (TD) error** and helps avoid the problem of **correlated updates** that might destabilize the training.

Let's dive into the concept of **Target Network** in DQN and how it functions:

1. The Problem with Regular Q-Learning

In standard Q-learning, the update rule for the Q-values is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

Where:

- Q(st,at) is the current Q-value for the state st and action at,
- rt+1 is the immediate reward for transitioning from st to st+1
- γ gamma is the discount factor,
- max Q(st+1,a') is the maximum Q-value of the next state over all possible next actions.

In this update rule, we are using the **current Q-values** for the **next state** to compute the target value of current state. This means the Q-values are being updated using our estimation for next state. This method which is known as bootstrapping creates a **correlation** between the target and predicted Q-values. This can lead to unstable learning because the network is trying to learn and target the same values simultaneously.

2. Introducing the Target Network in DQN

To stabilize this, DQN introduces a **Target Network**, which is a copy of the original Q-network but with **delayed updates**. This network is used to generate the **Expected Q-values** for the update rule, while the Base Q-network is used to predict the Q-values for the current state and action.

3. Synchronization of Target Network

In DQN, the **target network** is synchronized with the main Q-network after a certain number of steps. This is usually done at intervals (e.g., every 1000 steps), and the synchronization simply copies the weights from the Q-network to the target network.

Replay Buffer:

Another key component in DQN that addresses two major challenges in reinforcement learning: **correlated data** and **sample inefficiency**.

1. Role in DQN

In reinforcement learning, the data generated by interacting with the environment is sequential and highly correlated. Training a neural network directly on this correlated data can lead to unstable learning and divergence. The Replay Buffer mitigates this by decoupling the data generation and learning processes, enabling stable and efficient training.

2. How it Works

The Replay Buffer is a FIFO buffer that stores experiences collected during interactions with the environment. Each experience is typically stored as a tuple:
(state, action, reward, next_state, done), where:

- state: The current state observed from the environment.
- action: The action taken by the agent in the current state.

- reward: The reward received after taking the action.
- next_state: The state transitioned into after taking the action.
- done: A flag indicating if the episode has ended.

These experiences are accumulated in the buffer over time, and during training:

1. A random batch of experiences is sampled from the buffer.
2. The agent uses this batch to compute the Bellman target and update the Q-values.

By sampling randomly, the buffer breaks the correlation between consecutive experiences, improving the stability of the training process.