Online and Reinforcement Learning (2025)
Home Assignment 5

Davide Marchi 777881

Contents

1 Deep Q-Learning

1.1 Neural architecture

1.1.1 Structure

The line

```
x = torch.cat((x_input, x), dim=1)
```

takes the original input `x_input` (the raw state features) and concatenates it with the hidden representation `x` along the feature dimension. This effectively merges the initial state features with the learned features from the hidden layers. Such a design is sometimes referred to as a *skip connection* or *residual-like* connection, because the network has direct access to the original input when producing the final Q-values.

1.1.2 Activation function

We use tanh instead of the standard logistic sigmoid function because:

- tanh is zero-centered (ranging from -1 to 1), which often helps with training stability.
- The logistic (sigmoid) function saturates more easily at 0 or 1, leading to slower gradients. In contrast, tanh keeps activations in a range that often accelerates convergence in practice.

1.2 Adding the Q-learning

The missing line to compute the target **y** (right after the comment "`# Compute targets`") is shown below. We also detach the tensor so that the gradient does not flow back through the next-state values:

```
max_elements = torch.max(target_Qs, dim=1)[0].detach()
y = rewards + gamma * max_elements
```

This implements the standard Q-learning target:

$$y_i = r_i + \gamma \max_{a'} Q(s'_i, a').$$

After adding this line, the training converged to policies achieving returns above 200 in many episodes, indicating that the agent successfully learned to land.

Learning curve

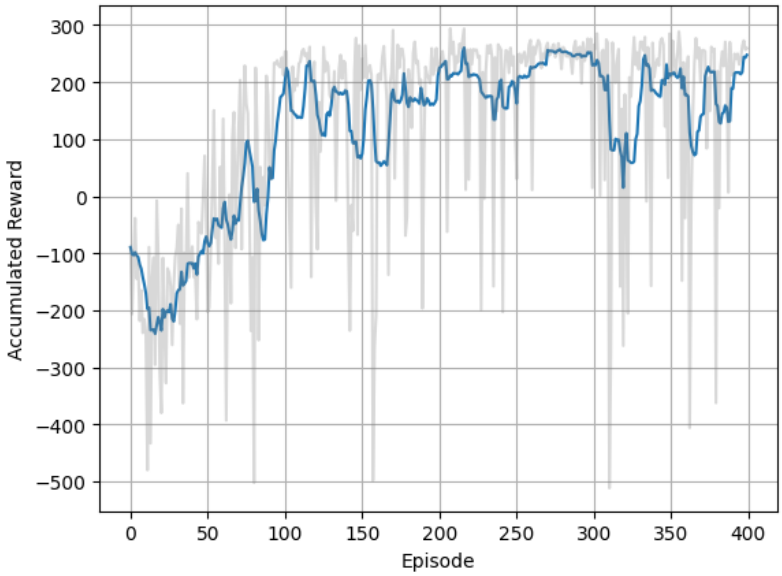

Figure 1: Accumulated reward per episode (in gray) and its smoothed average (blue).

1.3 Epsilon

The line

```
explore_p = explore_stop + (explore_start - explore_stop)*np.exp(-
decay_rate*ep)
```

implements an exponentially decaying exploration probability. At the beginning (when `ep` $= 0$), it starts near `explore_start`, and as the episode index `ep` grows, the probability `explore_p` exponentially approaches `explore_stop`. This ensures that early in training the agent explores more, and then it gradually exploits its learned policy more often.

1.4 Gather

The line

```
Q_tensor = torch.gather(output_tensor, 1, actions_tensor.unsqueeze(-
1)).squeeze()
```

selects the Q-value corresponding to the action actually taken in each state. Since `output_tensor` contains Q-values for all possible actions, we use `gather` along the action dimension to pick out the one Q-value that corresponds to the `actions_tensor`. In other words, it is a convenient way to index a batch of Q-value vectors by their chosen actions.

2 A tighter analysis of the Hedge algorithm

3 Empirical evaluation of algorithms for adversarial environments

4 Empirical comparison of UCB1 and EXP3 algorithms