

# Deep Q Networks

Saasha Nair

# Deep Q Networks (DQN)

TL; DR

Does not learn an explicit map/model of the environment

During training, the policy being learnt is different from the one collecting learning data from the environment

An off-policy, value-based, model-free RL algorithm. It learns to act in an environment with discrete action space by estimating Q-values.

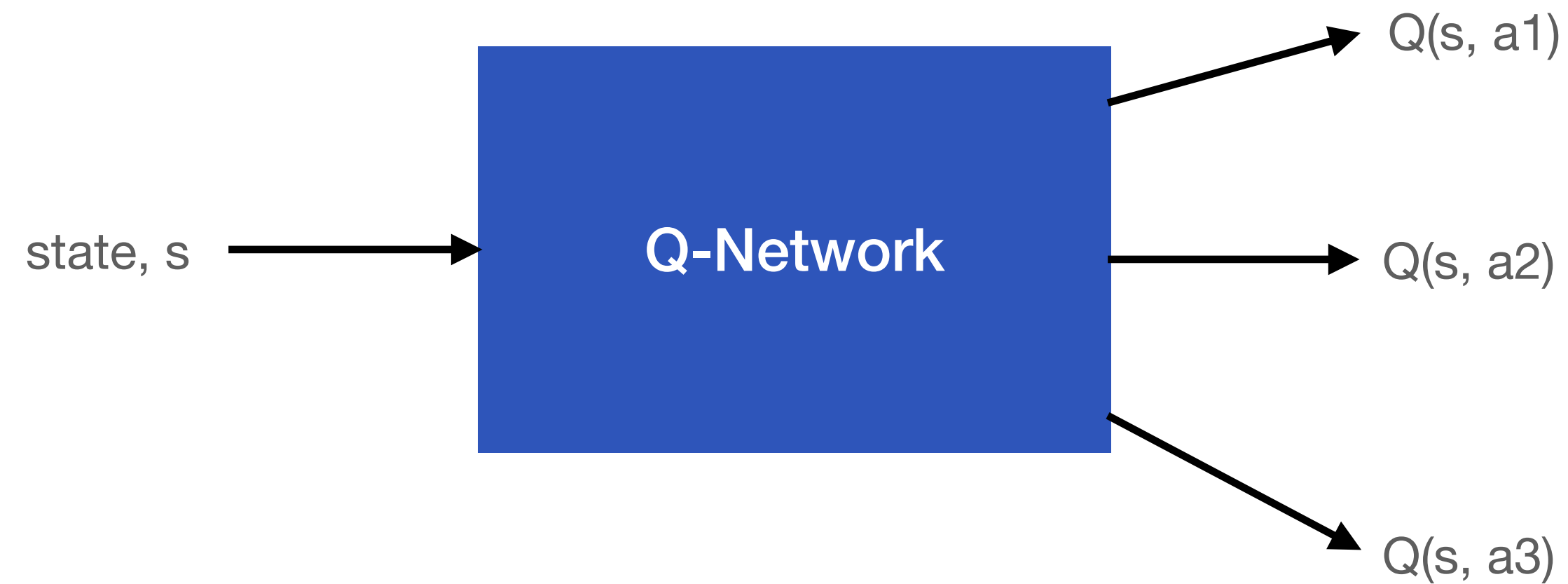
How good is a given state-action pair?

Policy not explicitly learnt, but derived from the learnt value function

# Idea behind DQN

$Q(s, a)$  = expected return from following a particular policy after performing action  $a$  in state  $s$

- Allow a neural network to learn Q-values



# Idea behind DQN

$$\text{Loss} = (\text{TargetQ} - \text{PredictQ})^2$$

Diagram illustrating the components of the DQN loss function:

- The expression  $\text{reward} + \text{discount} * \max_{a'} Q(s', a')$  represents the TargetQ value.
- The expression  $Q(s, a)$  represents the PredictQ value.

**But this naive implementation suffers from instability...**

# Non Stationary Targets

## Challenge #1

The problem:

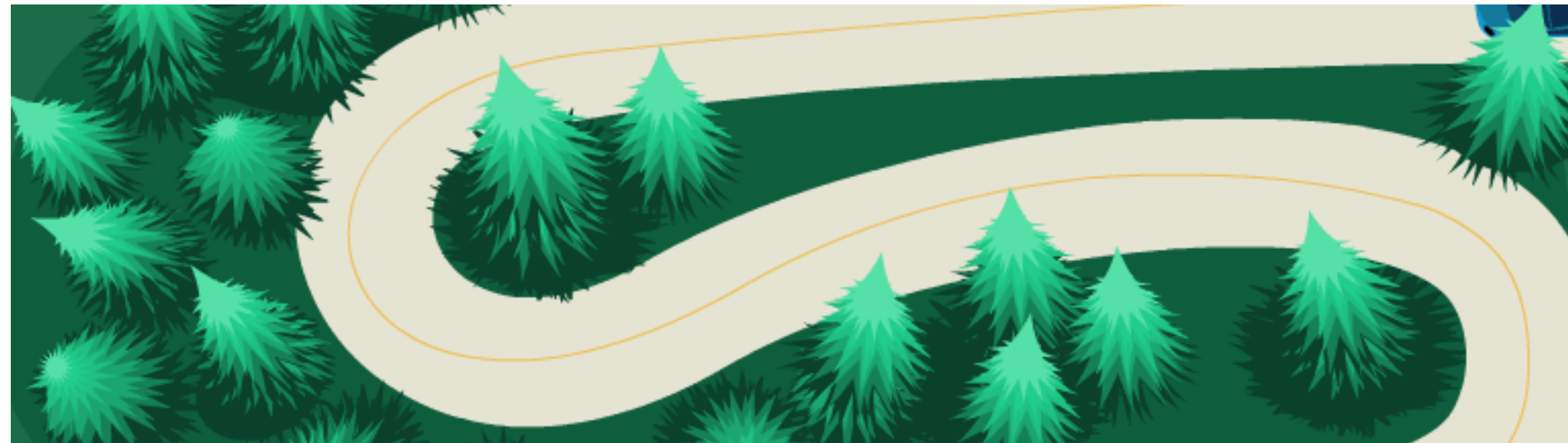


The solution: *Target Networks*

# Non IID Data

## Challenge #2

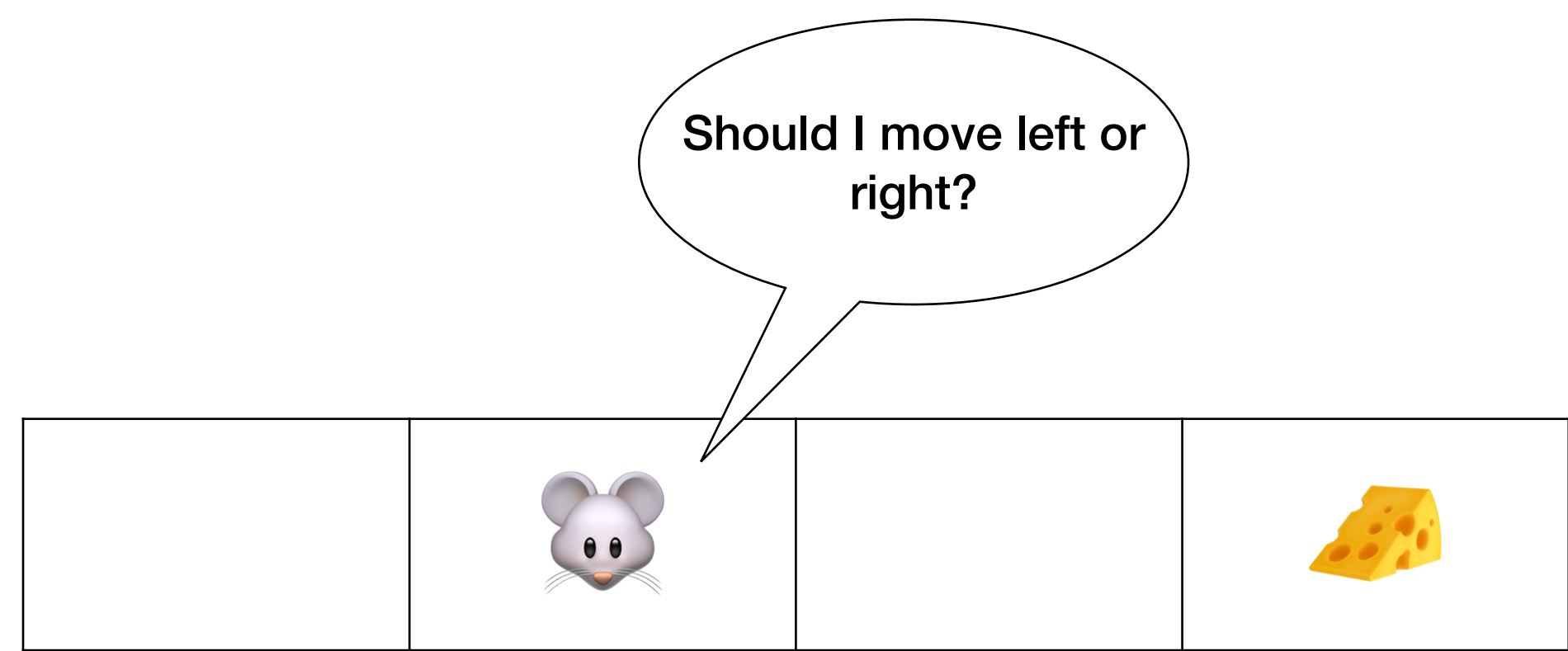
The problem:



The solution:

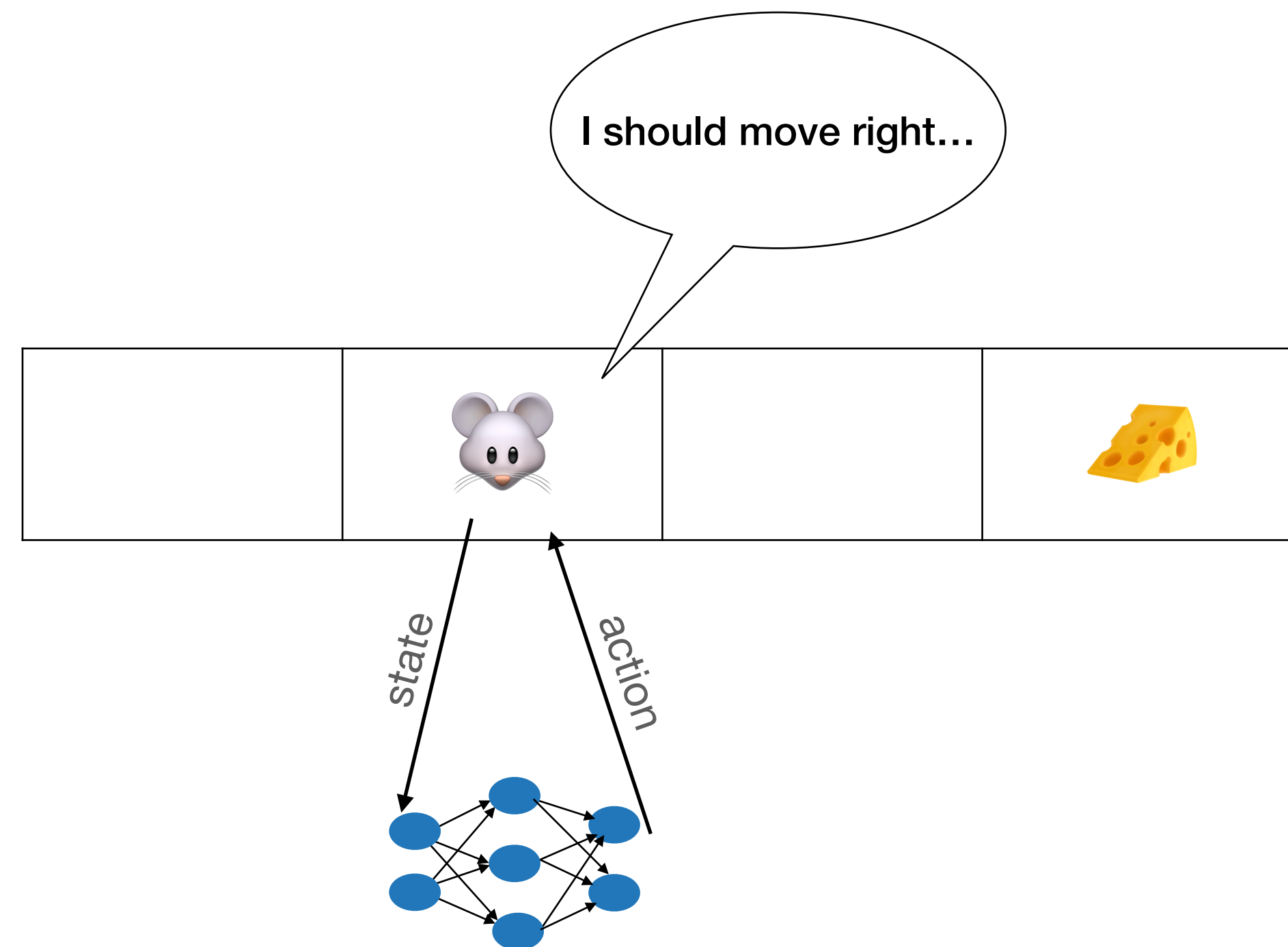
*Replay Buffer*

# DQN: Intuition

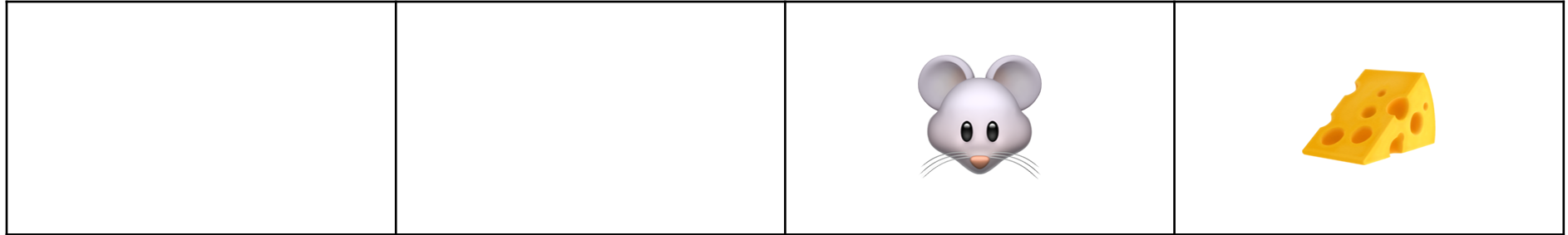




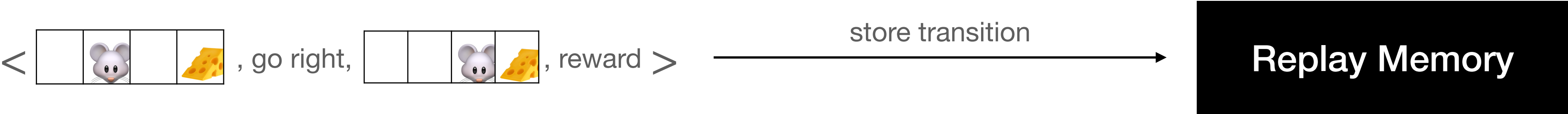
# DQN: Intuition



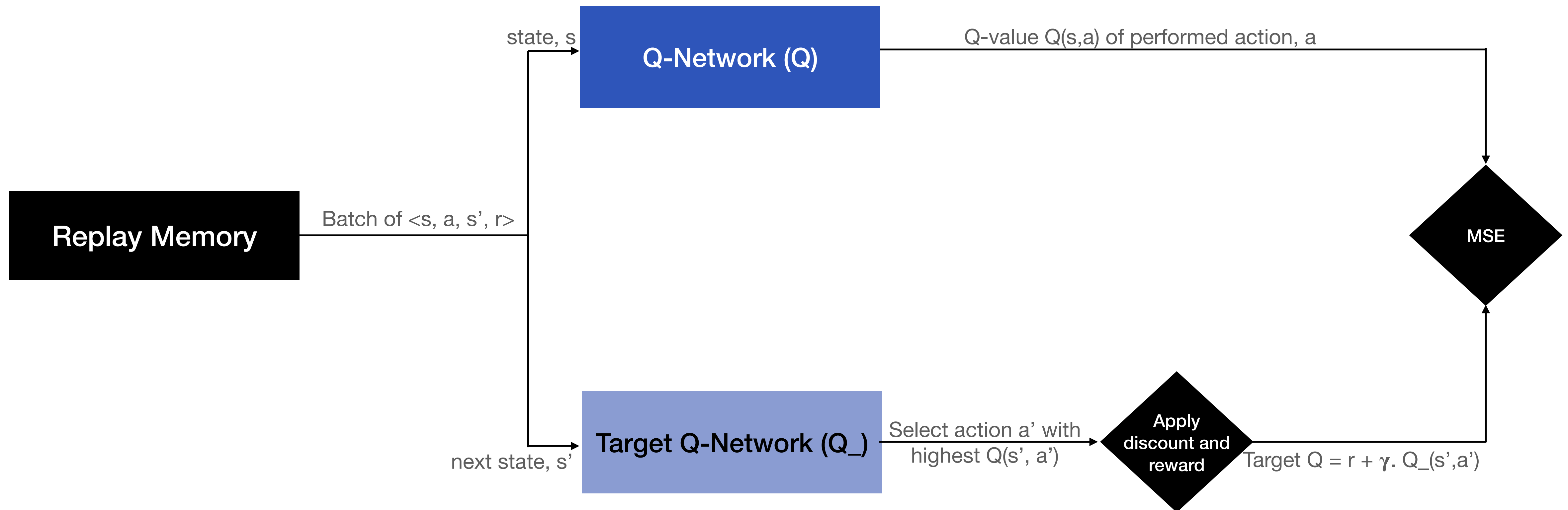
# DQN: Intuition



# DQN: Intuition

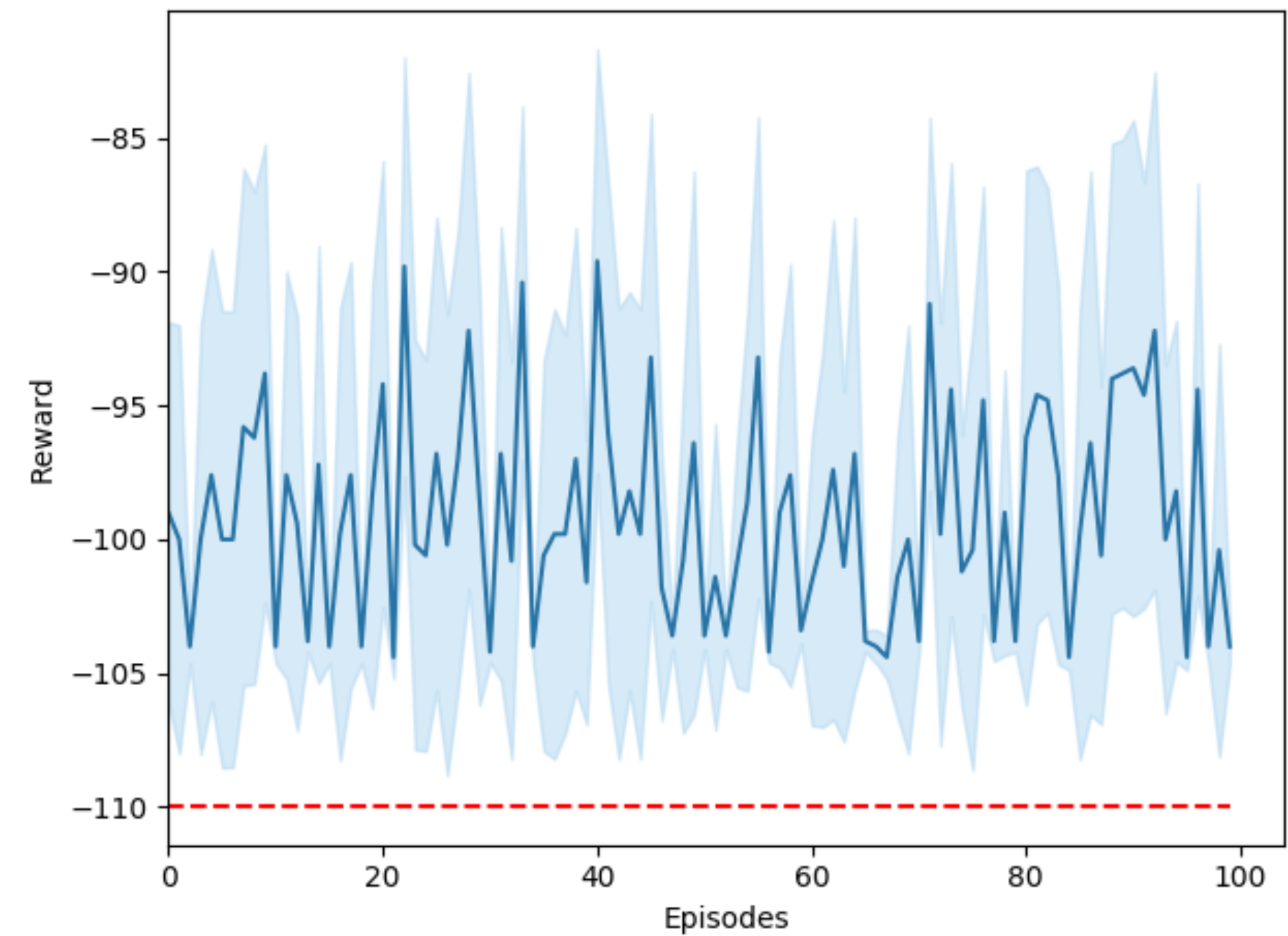
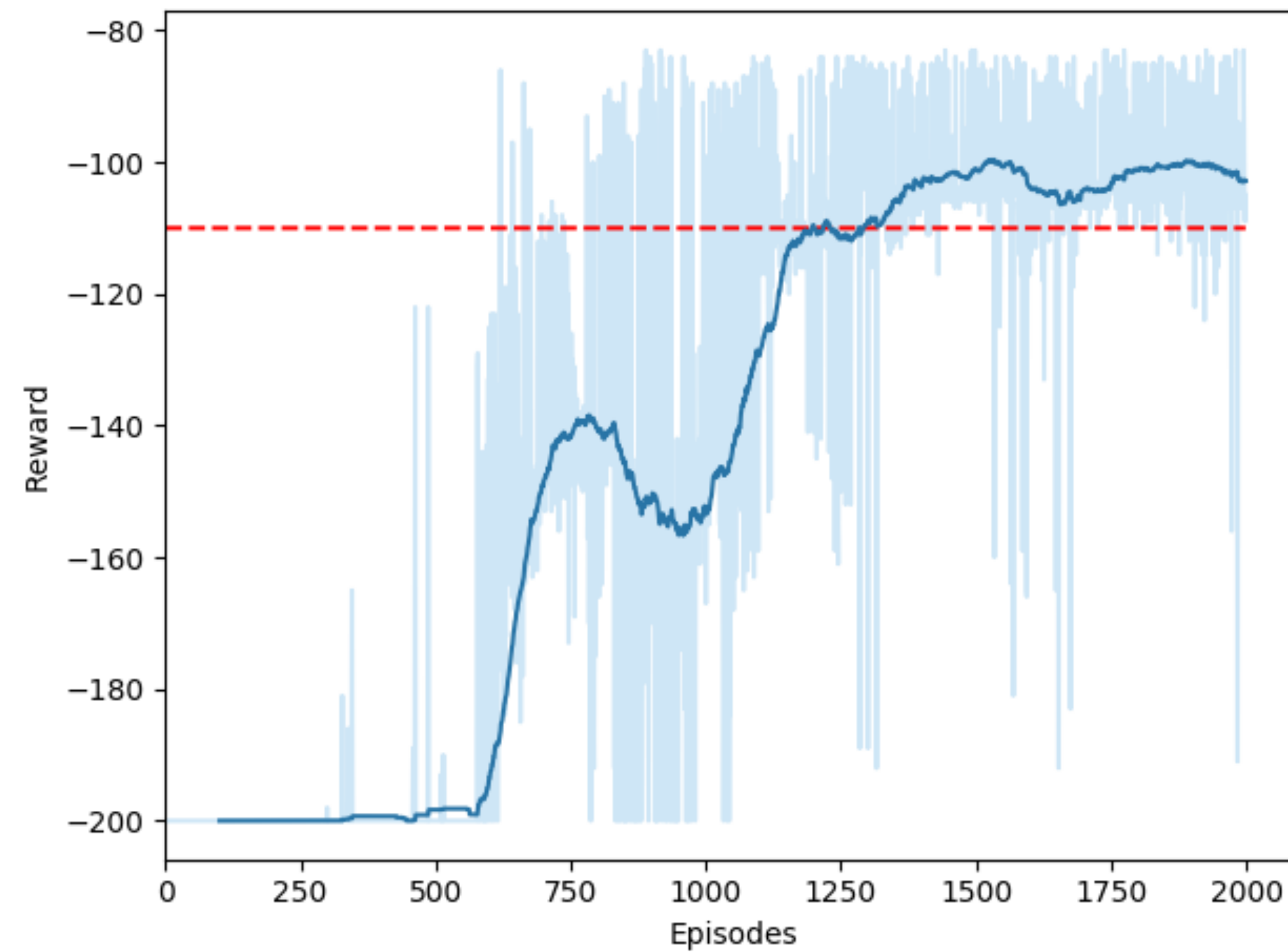
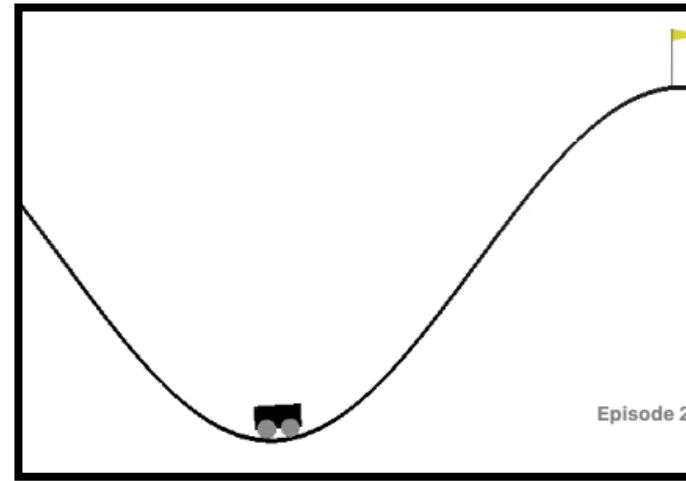


# DQN: Intuition



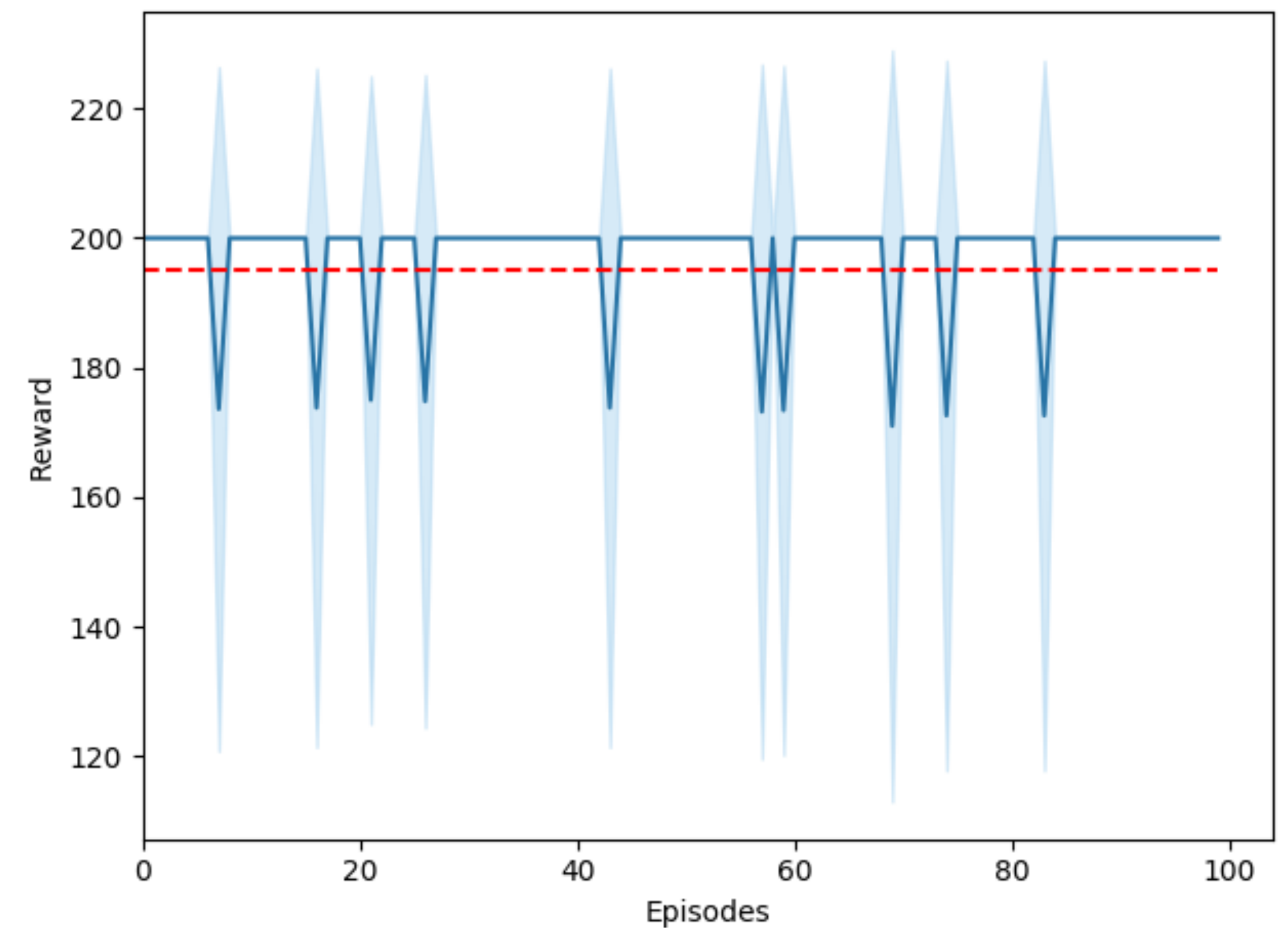
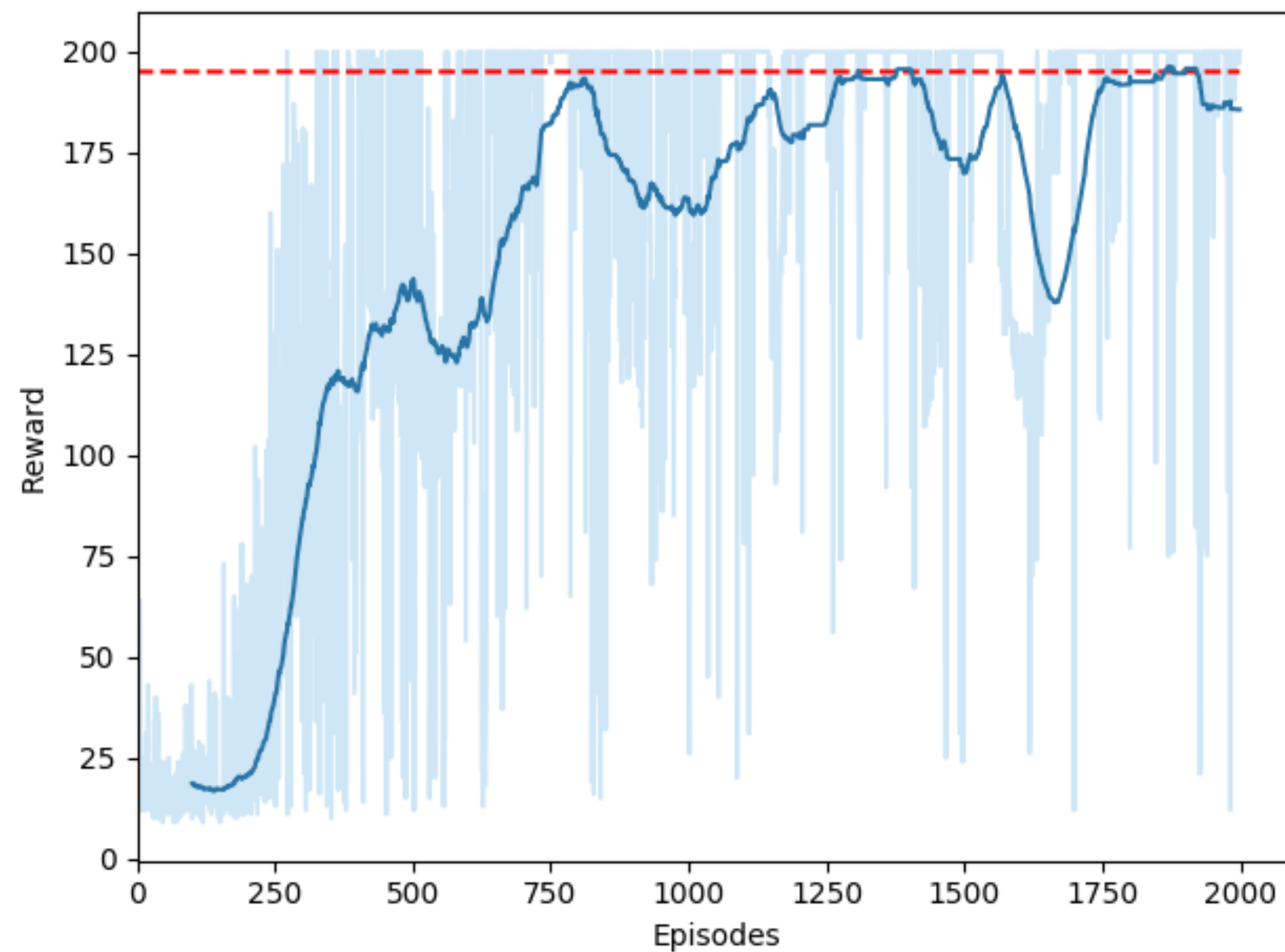
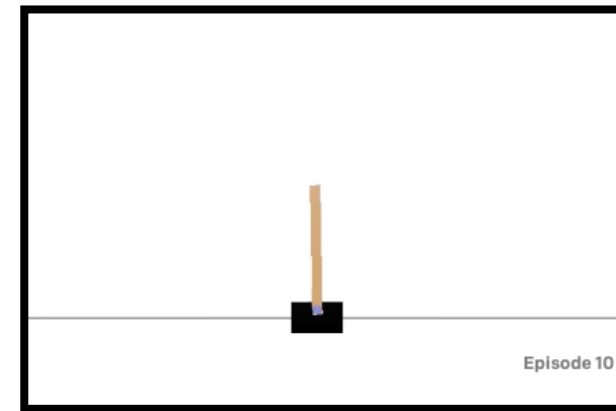
# Results

# MountainCar-v0



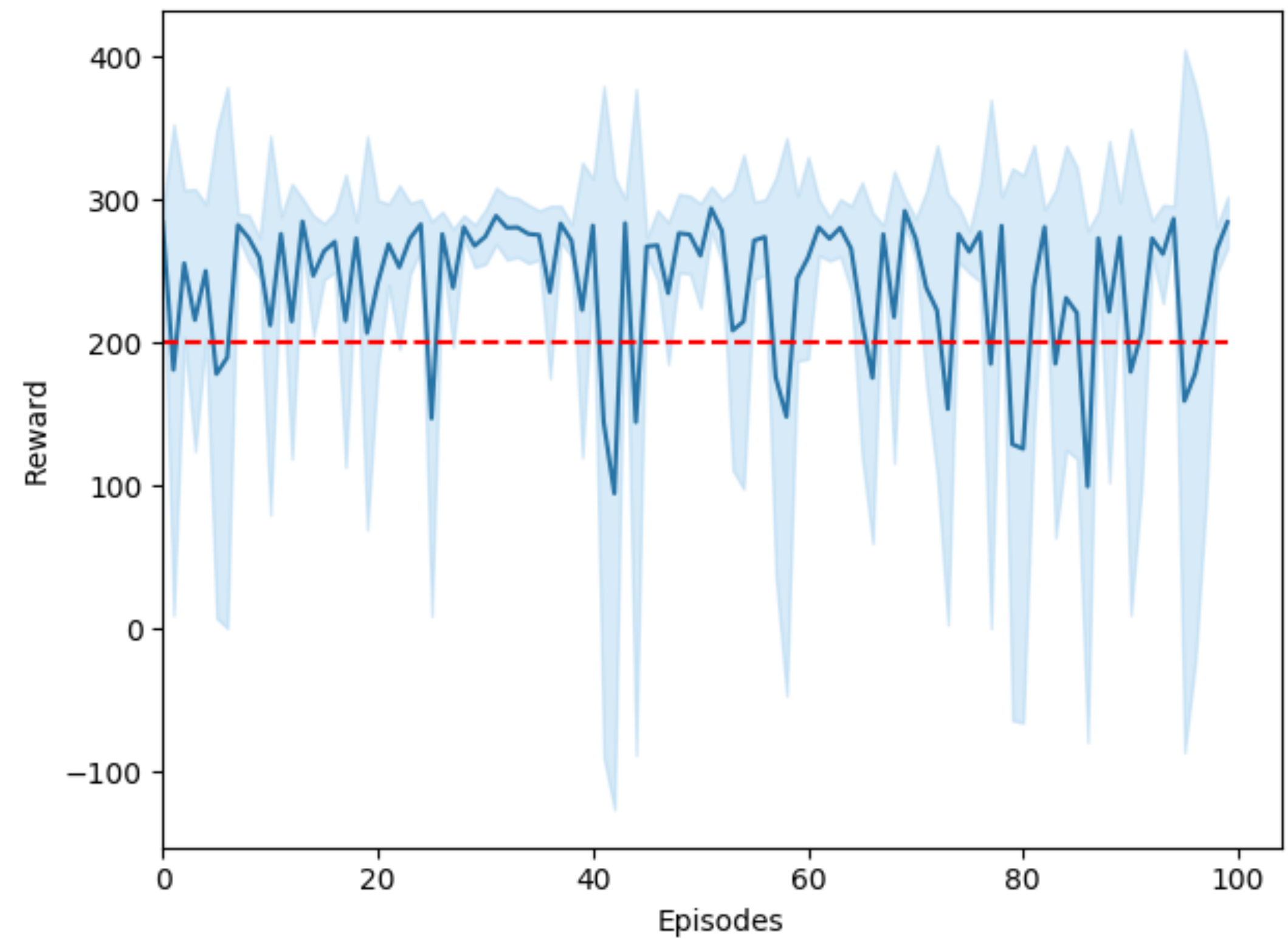
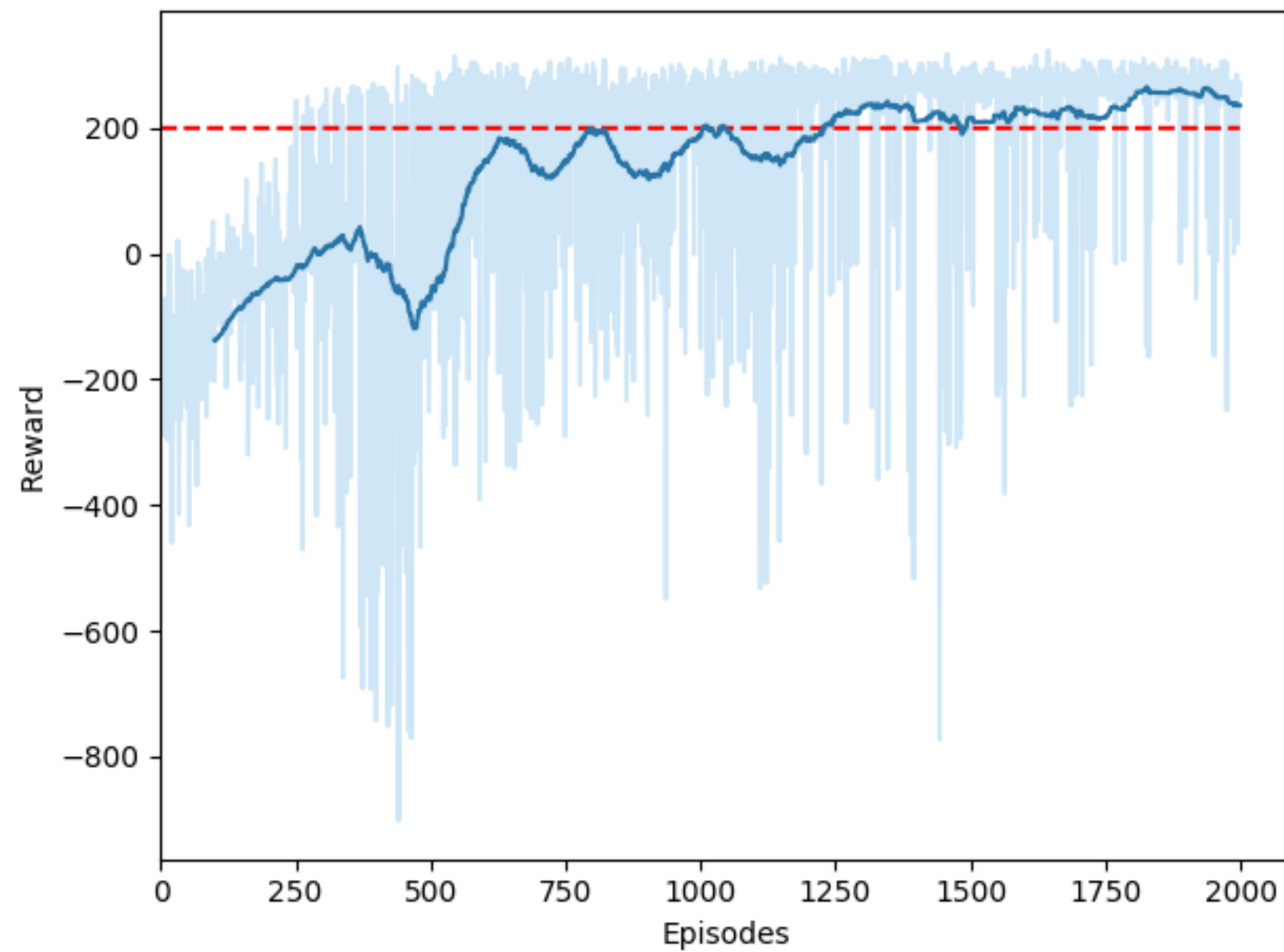
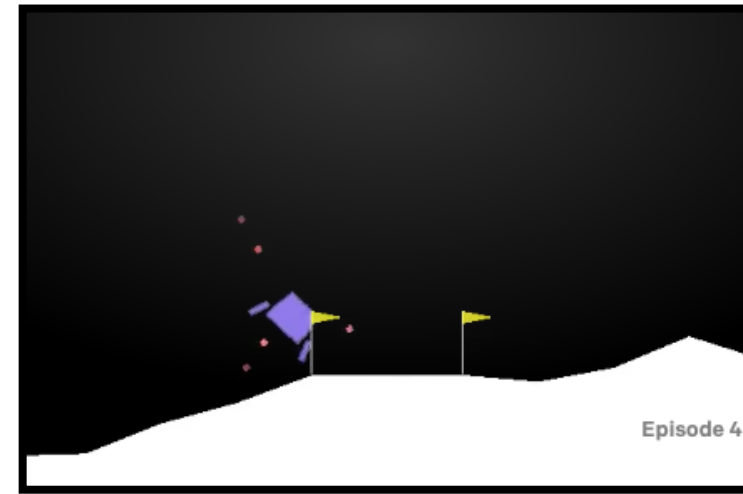
Plot 1: Training (left) vs Testing (right) in the MountainCar env

# CartPole-v0



Plot 2: Training (left) vs Testing (right) in the CartPole env

# LunarLander-v2



Plot 3: Training (left) vs Testing (right) in the LunarLander env



# DQN: Shortcoming

- discrete action space only
- overestimation bias

Target = reward + discount \*  $\max_{a'} Q(s', a')$

**Code**

# The Algorithm

## Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory  $D$  to capacity  $N$

Initialize action-value function  $Q$  with random weights  $\theta$

Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$

**For** episode = 1,  $M$  **do**

Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$

**For**  $t = 1, T$  **do**

With probability  $\varepsilon$  select a random action  $a_t$

otherwise select  $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$

Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$

Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the network parameters  $\theta$

Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

**End For**

### Replay Memory

- store experiences
- sample  $n$  experiences

### DQNNet

- network specifics

### DQNAgent

- instances of policy and target networks
- select action
- update target
- update epsilon
- learn



# Overestimation Bias

Target = reward + discount \*  $\max_{a'} Q(s', a')$

