# Recitation 8
# Reinforcement Learning

10-301/601: Introduction to Machine Learning

11/15/2024

## 1 Introduction to Reinforcement Learning

### 1.1 Markov Decision Process

A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma, s_0)$, where:

- $\mathcal{S}$ is the set of states
- $\mathcal{A}$ is the set of actions
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function
- $\gamma \in [0, 1)$ is the discount factor
- $s_0$ is the start state

When we play a game, we can model the game using a Markov decision process as follows:

1. We start in some state $s_0$.
2. Choose some action $a_0 \in \mathcal{A}$.
3. As a result of our choice, the state of the game transitions to some successor state $s_1$. For non-deterministic transition, we draw the next state according to our transition function. That is to say,
$$T(s_t, a_t, s_{t+1}) = P(S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t)$$
4. Then we pick another action $a_1$, and so on.

We can represent it as

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

Let $r_t = R(s_t, a_t, s_{t+1})$. Upon visiting a sequence of states $s_0, s_1, s_2, s_3 \dots$ with respective actions $a_0, a_1, a_2, \dots$ the **total discounted payoff** is given by

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$

Note that different sequences of states may have a different payoff value, and the reward is discounted if attained later.

## 1.2 Policy and Value Function

A **policy** is any function $\pi : \mathcal{S} \to \mathcal{A}$ mapping from the states to the actions. We say that we are executing some policy $\pi$ if, whenever we are in state $s$, we take action $a = \pi(s)$.

For the optimal policy function $\pi^*$ we can compute its **value function** at state $s$ as:

$$\begin{aligned} V^{\pi^*}(s) &= V^*(s) \\ &= \mathbb{E}\left[R(s_0, \pi^*(s_0), s_1) + \gamma R(s_1, \pi^*(s_1), s_2) + \gamma^2 R(s_2, \pi^*(s_2), s_3) \cdots \mid s_0 = s, \pi^*\right]. \end{aligned}$$

In other words, this is the best possible expected total payoff that can be attained using **any policy** $\pi$.

This **optimal value function** can be represented using Bellman's equations (named **Bellman optimality equation** for state value function), i.e.,

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s, a)(R(s, a, s') + \gamma V^*(s')).$$

If $R(s, a, s') = R(s, a)$ (deterministic transition), then we have the form we saw in class:

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^*(s') \right\}.$$

The interpretation of Bellman equation is also very intuitive: "how valuable is the current state under a policy?" Then it naturally follows that the optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ can be found as

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s, a)(R(s, a, s') + \gamma V^*(s')).$$

## 1.3 Q-Value

We can also define a $Q$ function $Q(s, a)$ that returns the expected discounted future value of taking action $a$ at state $s$.

**Question:** When would we want to use $Q(s, a)$ instead of $V(s)$?
**Answer:** When the reward function and/or transition probabilities are unknown

We can take a part of $V^*(s)$ as $Q^*(s, a)$ so that we have

$$\begin{aligned} Q^*(s, a) &= \sum_{s' \in \mathcal{S}} p(s'|s, a)(R(s, a, s') + \gamma V^*(s')) \\ V^*(s) &= \max_{a \in \mathcal{A}} \underbrace{\sum_{s' \in \mathcal{S}} p(s'|s, a)(R(s, a, s') + \gamma V^*(s'))}_{Q^*(s,a)} = \max_{a \in \mathcal{A}} Q^*(s, a). \end{aligned}$$

Then it follows that the optimal policy can be obtained as

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a).$$

# 2 Value Iteration

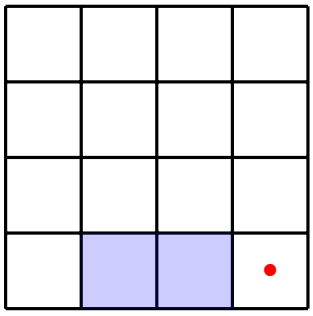


Figure 1: The cliff-walking environment

Here, we assume a $4 \times 4$ grid environment. The reward is 1 for reaching the cell with the red dot, $-1$ for reaching the shaded cells, and 0 for all other cells. The episode ends if the agent lands in either the shaded cells or the red-dot cell. The state space ($\mathcal{S}$) is all the cells. The action space ($A$) is up, down, left or right. If the agent tries to move out of the grid, it simply goes back to its previous state. The discount factor, $\gamma$, is 0.9.

**Bellman optimality equation** for state value function:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a)(R(s,a,s') + \gamma V^*(s'))$$

We can numerically approximate $V^*$ using synchronous value iteration and asynchronous value iteration. The recurrence relation is defined as follows for **synchronous** value iteration for all $s \in \mathcal{S}$:

$$V^{(t+1)}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a)(R(s,a,s') + \gamma V^{(t)}(s')).$$

Notice the time term $(t)$, hence the name synchronous. For **asynchronous** value iteration, we use for all $s \in \mathcal{S}$:

$$V(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a)(R(s,a,s') + \gamma V(s')).$$

1. Find the updated value of each cell after the first round and after the second round of synchronous value iteration. Assume deterministic transition function.

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

Initial

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |

After 1st round

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0.9 |
| 0 | 0 | 0.9 | 1 |
| 0 | 0 | 0 | 0 |

After 2nd round

2. Starting over, find the update value of each cell after one round of asynchronous value iteration. Visit each cell in two different orders: (1) bottom right to top left, and (2) top left to bottom right.

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

Initial

| | | | |
|---|---|---|---|
| $0.9^5$ | $0.9^4$ | $0.9^3$ | $0.9^2$ |
| $0.9^4$ | $0.9^3$ | $0.9^2$ | 0.9 |
| $0.9^3$ | $0.9^2$ | 0.9 | 1 |
| 0 | 0 | 0 | 0 |

BR to TL

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |

TL to BR

3. What is the policy learned after the one round of asynchronous value iteration when the cells were visited from bottom right to top left? If there is a tie, pick the action that comes first alphabetically (i.e., priority: $\downarrow > \leftarrow > \rightarrow > \uparrow$).

| | | | |
|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ |
| ↓ | ↓ | ↓ | ↓ |
| → | → | → | ↓ |
| ↓ | ⟲ | ⟲ | ⟲ |

4. Now suppose that the environment is sloped downward towards the cliff, with all the other settings unchanged. For every action taken, there is a 0.5 probability that the agent will move as intended and a 0.5 probability that the agent will slip and move 1 cell down instead. Report the values after two rounds of synchronous value iteration.

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

Initial

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | −.5 | −.5 | 1 |
| 0 | 0 | 0 | 0 |

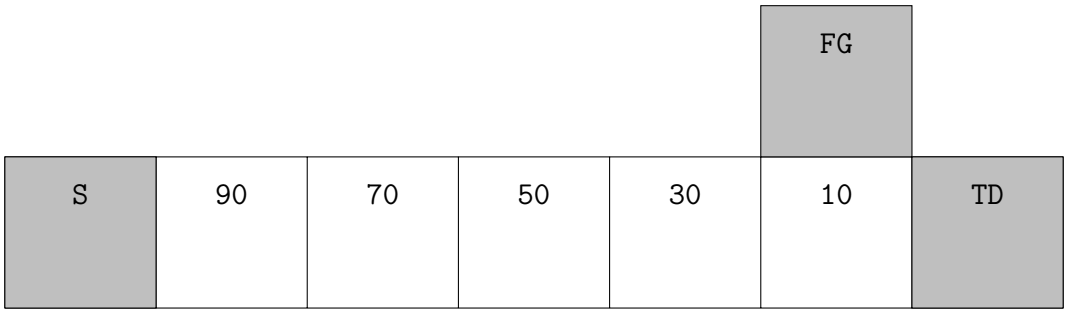
After 1st round

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | −.225 | −.225 | .9 |
| 0 | −.5 | −.05 | 1 |
| 0 | 0 | 0 | 0 |

After 2nd round

# 3 Q-Learning

Let's play some football (the ~~real~~ American kind). We will divide the field into eight states (labeled slightly differently from an official football field for clarity):




Here are some basic rules (slightly different than official football rules, but let's work with it for the sake of this example):

- There are two teams. For simplicity, let's call them Offense and Defense.
- The length of the field is 100 yards; we will set each 20-yard interval on this field as its own state, in addition to the end zones (`S`, `TD`, and `FG`).
- The drive (or, in reinforcement learning terms, episode) terminates if the ball is moved into the end zone.
- At each state, the Offense may either `run` the ball or `pass` it (always in the direction of `TD`).

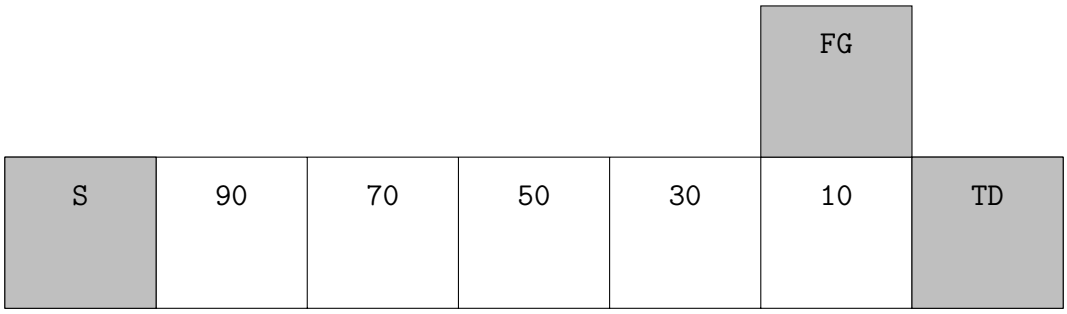
The head coach argues that it is not a good idea to assume a value for the transition probability from one state to the next. She also argues that since it is not the case that the Offense will always score seven points when going into the `TD` state (they can also score six points or eight) it is better to not assume a reward function. It is also possible to start the drive from anywhere in the field. For all these reasons, she encourages the offensive coordinators to try Q-learning.

1. The offensive coordinators, now convinced, decide to build a Q-value table that matches the state and action space. What is the size of this table?

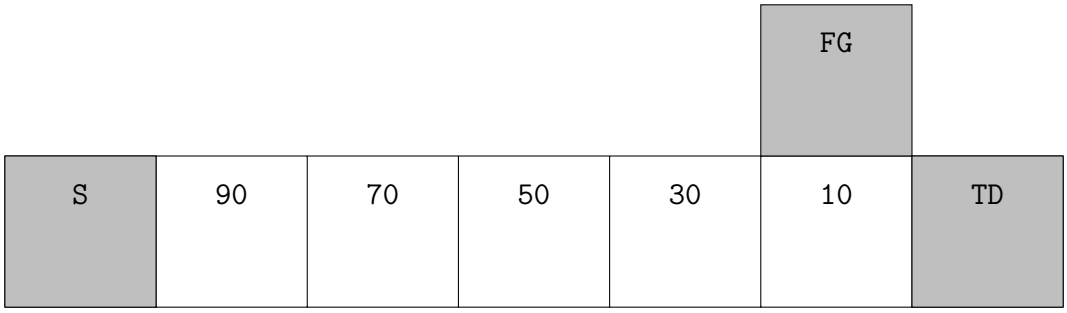
   8 states $\times$ 2 actions = 16 entries

2. The offensive coordinators initialize $Q(s, a) = 0 \ \forall s, a$. They decide to update it by sampling random drives from games previously played by the Offense this season (note that this approach uses experience replay and not a standard MDP since individual plays in a football game are not independent unlike moves in a grid world). They give you the empty Q-value table for ease:

   Q(·, `run`):




   Q(·, `pass`):




   Time to watch some football :) Assume the learning rate $\alpha = 0.1$ and the discount factor $\gamma = 0.5$. Update the Q-table above for each of the following episodes using the Temporal Difference error update:

   (a) **Episode 1, iteration 1**: Beginning at the `30` state, Offense `run`s the ball to `10`. Offense collects zero reward.
   $Q(\texttt{30}, \texttt{run}) = 0$

   (b) **Episode 1, iteration 2**: Offense `pass`es the ball from the `10` but stays at the `10`. Offense collects zero reward.
   $Q(\texttt{10}, \texttt{pass}) = 0$

   (c) **Episode 1, iteration 3**: From the `10`, Offense `pass`es the ball again and reaches the `TD` state. The drive terminates. Offense collects a reward of 7.
   $Q(\texttt{10}, \texttt{pass}) = 0.7$

   (d) After updating the $Q$-function for the first episode, the head coach decides to quiz the offensive coordinators. If the Offense is in state `10`, what is the best action?
   $\pi(\texttt{10}) = \texttt{pass}$

(e) **Episode 2, iteration 1**: Beginning at the `90` state, the Offense `runs` the ball, causing a safety (they enter the `S` state). The drive terminates. Offense collects -2 reward.
$Q(\texttt{90}, \texttt{run}) = -0.2$

(f) After updating the $Q$-function for the second episode, the head coach decides to quiz the offensive coordinators again. If the Offense is in state `90`, what is the best action to take?
$\pi(\texttt{90}) = \texttt{pass}$

(g) **Episode 3, iteration 1**: Beginning at `50`, the Offense `passes` the ball but stays at the `50`. Offense collects zero reward.
$Q(\texttt{50}, \texttt{pass}) = 0$

(h) **Episode 3, iteration 2**: Beginning at the `50`, the Offense `passes` the ball to the `30`. Offense collects zero reward.
$Q(\texttt{50}, \texttt{pass}) = 0$

(i) **Episode 3, iteration 3**: Beginning at the `30`, the Offense `runs` the ball to the `10`. Offense collects zero reward.
$Q(\texttt{30}, \texttt{run}) = 0.035$

(j) **Episode 3, iteration 4**: Beginning at the `10`, the Offense `runs` the ball and ends the drive after entering the `FG` state. Offense collects a reward of 3 points.
$Q(\texttt{10}, \texttt{run}) = 0.3$

(k) After updating the $Q$-function for the third episode, the head coach decides to quiz the offensive coordinators for the last time. If the Offense is in state `10`, what is the best action to take?
$\pi(\texttt{10}) = \texttt{pass}$

# 4 Deep Q-Learning

In Mountain Car, you control a car that starts at the bottom of a valley. Your goal is to reach the flag at the top right, as seen in Figure 2. However, your car is under-powered and can not climb up the hill by itself. Instead you must learn to leverage gravity and momentum to make your way to the flag. It would also be good to get to this flag as fast as possible.

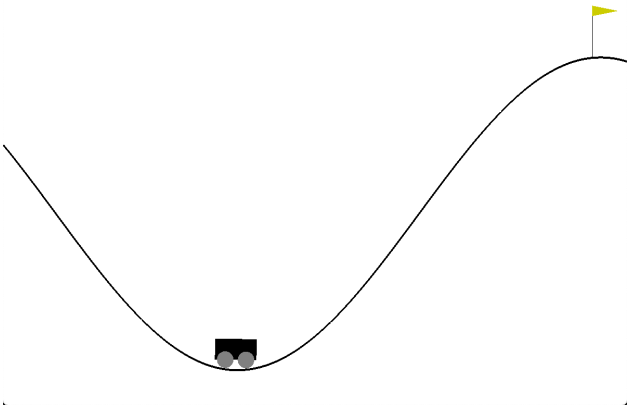


Figure 2: Mountain Car environment. The car starts at some point in the valley. The goal is to get to the top right flag.

The state of the environment is represented by two variables, `position` and `velocity`. `position` can be between $-1.2$ and $0.6$ inclusive and `velocity` can be between $-0.07$ and $0.07$ inclusive. These are just measurements along the horizontal axis.

The actions that you may take at any state are $\{0, 1, 2\}$ which respectively correspond to (0) pushing the car left, (1) doing nothing, and (2) pushing the car right. Note that we need Q-learning since we do not have the transition function.

1. Approximate Q-learning involves using a function that estimates $Q$ values given state and action pairs. Consider using the following linear approximation:

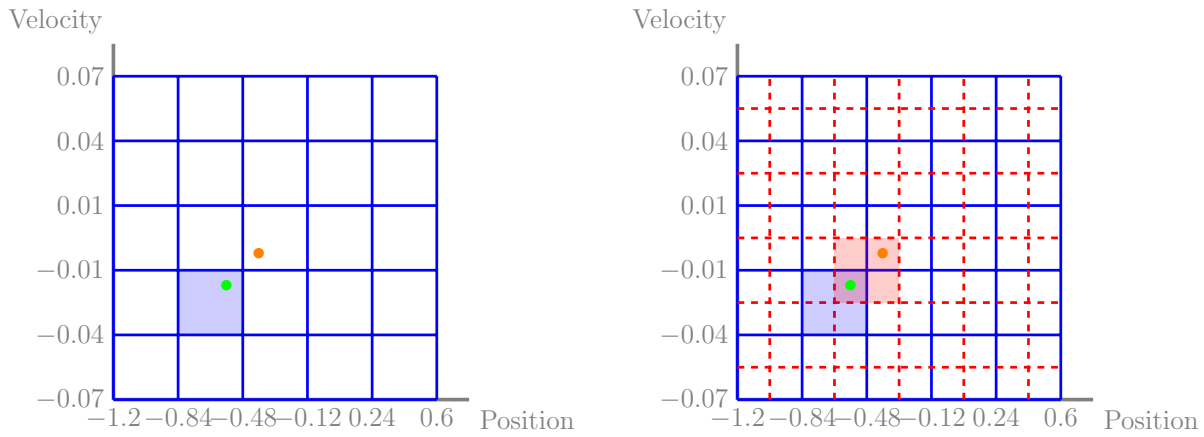
$$Q(\mathbf{s}, a; \mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2) = \mathbf{w}_a^T \mathbf{s} + b_a = \begin{cases} \mathbf{w}_0^T \mathbf{s} + b_0 & \text{if } a = 0 \\ \mathbf{w}_1^T \mathbf{s} + b_1 & \text{if } a = 1 \\ \mathbf{w}_2^T \mathbf{s} + b_2 & \text{if } a = 2 \end{cases}$$

   Let $\mathbf{s} = [0.5, 0.5]^T$, $\mathbf{w}_0 = [-1, -10]^T$, $\mathbf{w}_1 = [0, 1]^T$, $\mathbf{w}_2 = [2, 10]^T$, $b_0 = b_1 = b_2 = 0.5$. What are the $Q$ values for all actions performed at state $\mathbf{s}$?

   $q(s, 0) = -5$, $q(s, 1) = 1$, $q(s, 2) = 6.5$

2. We try a naïve state representation for Q-learning. What are the disadvantages with the following state representation when using Q-learning with linear approximation?

$$\mathbf{s} = \begin{bmatrix} \texttt{position} \\ \texttt{velocity} \end{bmatrix}$$




(a) A discretization of the state space of Mountain Car

(b) A tiling of the state space of Mountain Car

Figure 3: State representations for the states of Mountain Car

The linear approximation can only learn linear relationships between the position and velocity of a state and its value. In the Mountain Car environment, this would mean values of states would decrease gradually from right to left (values are highest near the flag and decrease as the car moves away from it). However, this is not representative of the true values since we would want to penalize the states at the bottom of the valley as opposed to the leftmost states. This relationship would not be learned well with this "raw" state representation.

3. For the Mountain Car environment, we know that `position` and `velocity` are both bounded. What we can do is draw a grid over the possible `position-velocity` combinations as seen in Figure 3a. We then enumerate the grid from bottom left to top right, row by row. Then we map all states that fall into a grid square with the corresponding one-hot encoding of the grid number. For efficiency reasons we will just use the index that is non-zero. For example the green point would be mapped to $\{6\}$. This is called a *discretization* of the state space. **Is there any downside to this approach?** If the discretization is not fine enough i.e. grids are large, nearby states with different true values would be incorrectly considered the same. For example, in the grid world environment with the cliff near the goal, the edge of the cliff and the goal may be grouped into the same state and inaccurately assigned the same reward. If the discretization is too fine, the dimensionality of the state representation becomes too large, slowing down training.

4. In addition to 2D discretization, we can draw two grids over the state space, each offset slightly from each other as in Figure 3b. The grids are indexed in row major order left to right and then bottom to top, then blue grid to red grid. For example, the top right blue grid is 24 while the bottom left red grid is 25. The new state is a "two-hot" encoding marking which blue and red grid the point falls within. **What are the two indices,**

**one for each grid that map the green point?**

$\{6, 39\}$

Note the implementation for tiling in the homework doesn't give these exact indices. Do not need to worry about the tile implementation, but it's doing something along these lines.[1]

5. Suppose we decrease the size of the grid squares in Figure 3a to match the precision of the tiling in Figure 3b. Compare the dimensionality of states under this new grid scheme versus the tiling scheme in Figure 3b. That is, what are the sizes of the vectors of each state representation? What is the advantage of tiling over this single grid discretization?

   Single grid: 100
   Tiling: $25 + 36 = 61$
   Tiling reduces the dimensionality of states for the same level of precision. More tiles lends more precision for the same number of dimensions.

# 5 Experience Replay Buffer

Consider the deep Q-learning algorithm where on each iteration, you take 1 step, receive some information $(s, a, s', r)$, and then apply the Q-learning update on that most recent step. The issue with this is that we are updating our weights using highly correlated samples (i.e., we are taking a step, updating, taking another step, updating, and so on). This is not desired because when performing this type of gradient descent, we would like i.i.d. samples instead.

The solution is to use a replay buffer that stores past experiences. Then in each iteration, instead of applying the Q-learning update based on the step we just took, we can randomly pick a previous experience and update on that one instead. This makes our sampling more independent and identically distributed than just updating based on the most recent step.

- **Idea:** maintain a "replay buffer" $\mathcal{D} = \{e_1, e_2, \ldots, e_N\}$ of the $N$ most recent experiences $e_t = (s_t, a_t, r_t, s_{t+1})$

  This keeps the agent from "forgetting" recent experiences

- In each iteration, we:

  1. Sample some experience $e_i$ (or a mini-batch of experiences $E = \{e_1, \ldots, e_T\}$) uniformly at random from $\mathcal{D}$ and apply the Q-learning update
  2. Add a new experience to $\mathcal{D}$

[1] What's really happening is that it allocates 2048 indices and assigns them first-come-first serve to the tilings the environment encounters

- Can also sample experiences from $\mathcal{D}$ according to some distribution that prioritizes experiences with high error

**In Homework 8, we will ask you to implement a replay buffer, add a new experience to it in each iteration, and then randomly sample a batch of past experiences to update on.**