SMU: Lecture 4

Monday, March 7, 2022

(Heavily inspired by the Stanford RL Course of Prof. Emma Brunskill, but all potential errors are mine.)

Plan for Today

- Recap of important concepts from lectures 1, 2 and 3.
- Value function approximation.
- Control with value function approximation.

Part 1: Where are we? (Recap from the previous lectures)

Markov Decision Process

- **Markov decision process = Markov reward process + Actions**
- **An MDP is given by:**
  - A set of states S.
  - A set of actions A.
  - A transition model $P[X_{t+1} = s' | X_t = s, A_t = a] \underbrace{= P(s' | s, a)}_{\text{notation}}$
  - A reward $R(s, a) = \mathbb{E}[R_t | X_t = s, A_t = a]$, i.e. the expected reward that the agent receives when performing action a in state s.
  - Discount factor γ.

Policy

- Policy determines which action to take in each state s.
- It can be either deterministic or random — that is also why policy will not simply be a function from states to actions.
- **We define policy:** $\pi(a \mid s) = P(A_t = a \mid X_t = s)$.
- **Example** (policy for our ant 🐜):
  - $A = \{\text{left}, \text{right}\}$
  - $\pi(\text{left} \mid 1) = 0$, $\pi(\text{right} \mid 1) = 1$, $\pi(\text{left} \mid 2) = 0.5$, $\pi(\text{right} \mid 1) = 0.5, \ldots$

(Bellman equation for MDP)

State Value Function of MDP

General case:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a, s) \cdot \left[R(s, a) + \gamma \cdot \sum_{s' \in S} P(s' \mid s, a) \cdot V^{\pi}(s') \right]$$

Version for deterministic policy:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \cdot \sum_{s' \in S} P(s' \mid s, \pi(s)) \cdot V^{\pi}(s')$$

MDP Control Problem

How to find $\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$???

State-Action Value Q

- **Definition:**

$$Q^\pi(s,a) = R(s,a) + \gamma \cdot \sum_{s' \in S} P(s' \mid s,a) \cdot V^\pi(s').$$

- **Intuition:**
  - The value of the return that we obtain if we first take the action a in the state s and then follow the policy π (including when we visit s again).
  - *Think of it as perturbing the policy π — we deviate from following the policy π only in the first step in s.*

Policy Improvement Step

- **Given:** An MDP and a **policy π_i that we want to improve** (if possible).
- **DO:**
    - For all $s \in S$, compute $Q^{\pi_i}(s, a)$ as defined on the previous slide, i.e. $Q^{\pi_i}(s, a) = R(s, a) + \gamma \cdot \sum_{s' \in S} P(s' | s, a) \cdot V^{\pi_i}(s')$.
    - **Compute new policy for all $s \in S$:**

$$\pi_{i+1}(s) = \arg\max_{a \in S} Q^{\pi_i}(s, a)$$

Here, we use the fact that our policy is deterministic for simpler notation (treating policy as a function). Using our previous notation we could write:

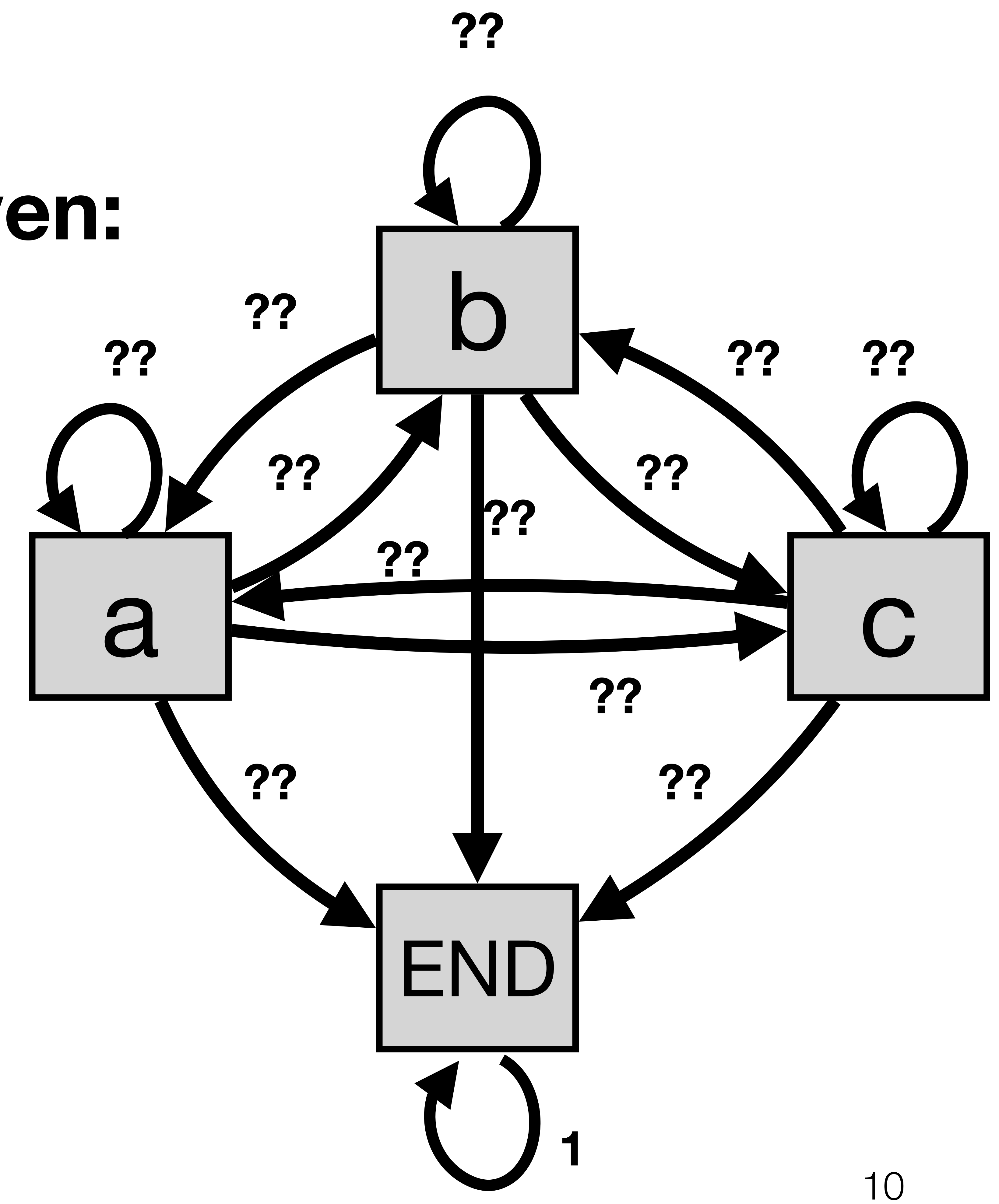
$$\pi(a | s) = \begin{cases} 1 & \textit{if } a = \arg\max_{a \in A} Q^{\pi_i}(s, a) \\ 0 & \textit{otherwise} \end{cases}$$

Example

Agent: 🐸

Rewards??

States are given:



Actions are given:

$$A = \{l, r\}$$

🐸

Policy is given, e.g.:

$$\pi(l\,|\,a) = 0.2,\ \pi(r\,|\,a) = 0.8,$$
$$\pi(l\,|\,b) = 0.3,\ \pi(r\,|\,b) = 0.7,$$

...

Problem: Model-Free Policy Evaluation

- Given a policy and an MDP with unknown parameters (or generally an environment with which we can interact), **estimate the value function.**

Model-Free Control

- Given a policy and an MDP with unknown parameters (or generally an environment with which we can interact), **find the optimal policy π.**

Three Methods Last Time

- Monte Carlo Control, SARSA and Q-Learning.
- All three using the concept of ε-greedy policy.

MC On Policy Improvement

```
**Initialize:** $G(s,a)=0,\ N(s,a)=0,\ Q(s,a)=0$ **for all** $s \in S, a \in A$.
**Initialize:** $\varepsilon = 1,\ k = 1$
**For** $i = 1,\ldots,N$:
    Sample episode $e_i := s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \ldots, s_{i,T_i}$ **given $\pi_k$.**
    **For** each time step $1 \le t \le T_i$:
        (**If** $t$ is the first occurrence of state $s$ in the episode $e_i$  -  Use this if you want first-visit MC)
         $s_t$ is the state visited at time $t$ in the episode $e_i$
         $a_t$ is the action taken at time $t$ in the episode $e_i$
          $g_{i,t} := r_{i,t} + \gamma \cdot r_{i,t+1} + \gamma^2 \cdot r_{i,t+2} + \ldots + \gamma^{T_i - t} \cdot r_{i,T_i}$
          $N(s) := N(s) + 1$ /* *Increment total visits counter* */
          $G(s_t, a_t) := G(s_t, a_t) + g_{i,1}$ /* *Increment total return counter* */
          $Q(s_t, a_t) := G(s_t, a_t)/N(s_t, a_t)$ /* *Update current estimate* *
    **EndFor**
    $k = k + 1,\ \varepsilon = 1/k$
    $\pi_k = \varepsilon -$**greedy policy w.r.t.** $Q$
```

SARSA

- SARSA is an on-policy algorithm.

1. **Initialize:** set π to be some ε-greedy policy, set $t = 0$
2. **Sample** a using the distribution given by π_0 in the state s_0 *(for sampling, we will use the notation $a \sim \pi(s)$)*. **Take** the action a and **observe** r_0, s_1**.**
3. **While** s_t is not a terminal state:
    1. **Take** action $a \sim \pi(s_t)$ and observe r_{t+1}, s_{t+a}.
    2. $Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$
    3. $\pi :=$ ε-greedy(Q)
    4. Set $t := t + 1$. Update ε, α /* *see next slides* */

Q-Learning

- Q-Learning is an off-policy algorithm.

1. **Initialize:** set π to be some ε-greedy policy, set $t = 0$
2. **Sample** a using the distribution given by π_0 in the state s_0 *(for sampling, we will use the notation* $a \sim \pi(s)$*)*. **Take** the action a and **observe** r_0, s_1.
3. **While** s_t is not a terminal state:
    1. **Take** action $a \sim \pi(s_t)$ and observe r_{t+1}, s_{t+1}.
    2. $Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t) \right)$
    3. $\pi := \varepsilon$-greedy(Q)
    4. Set $t := t + 1$. Update ε, α */* see next slides */*

ε-Greedy Policy

We assume ties are decided consistently

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A|} & \text{when } a = \arg\max_{a \in A} Q(s,a) \\ \frac{\varepsilon}{|A|} & \text{when } a \neq \arg\max_{a \in A} Q(s,a) \end{cases}$$

Part 2: RL with Function Approximation (Problem Statement)

Limitations of What We Saw So Far

- In the previous lectures, we assumed discrete MDPs with number of states that was not too large (i.e. the set S was not too large).

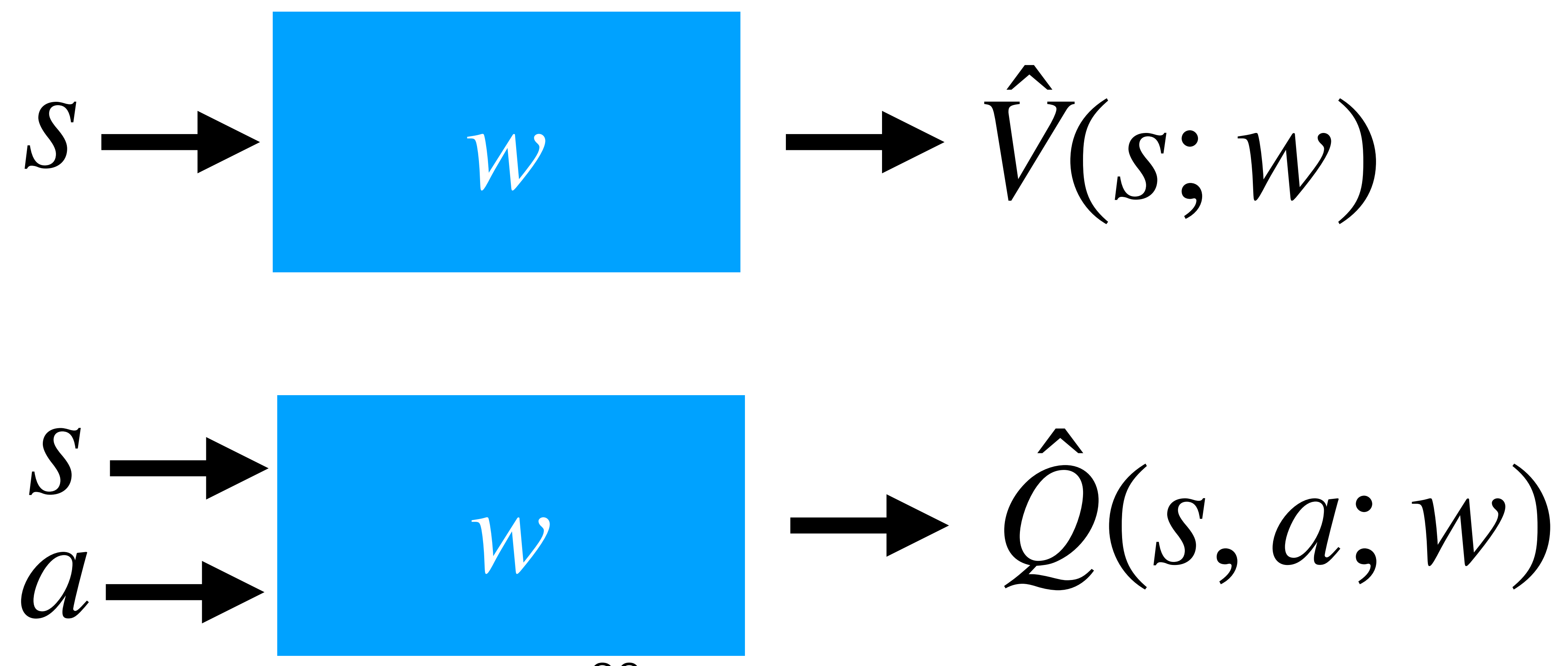
- Now imagine that we want to learn to play Atari games (which is what DeepMind did!) and we want to do it form the pixel inputs. How many states would we need if we wanted to use what we learned in the previous lectures? … Then we would need at least $128^{160 \cdot 192}$ states (128 colors with resolution 160 x 192 pixels).

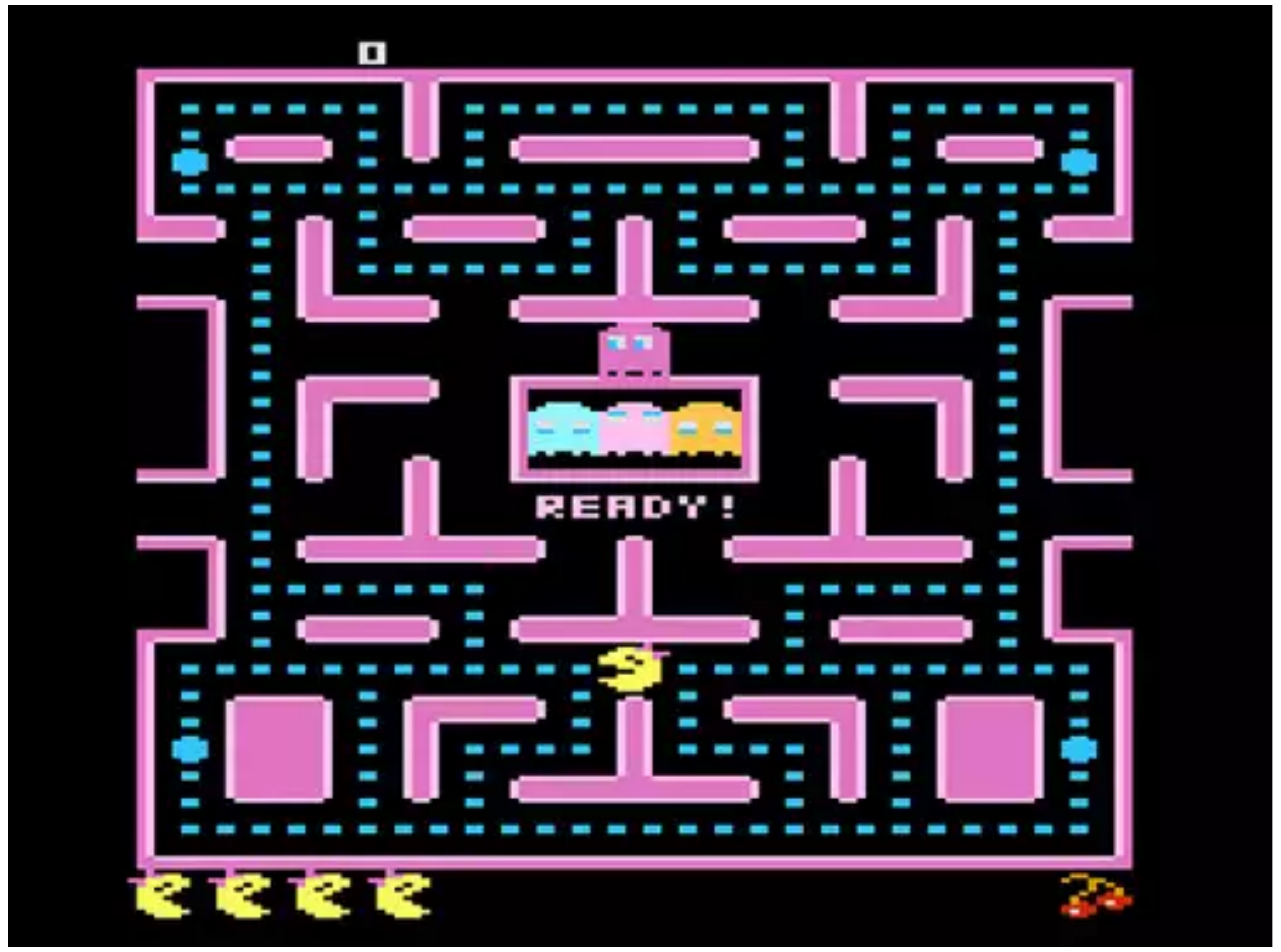
- **What we need is *function approximation.***

Basic Idea

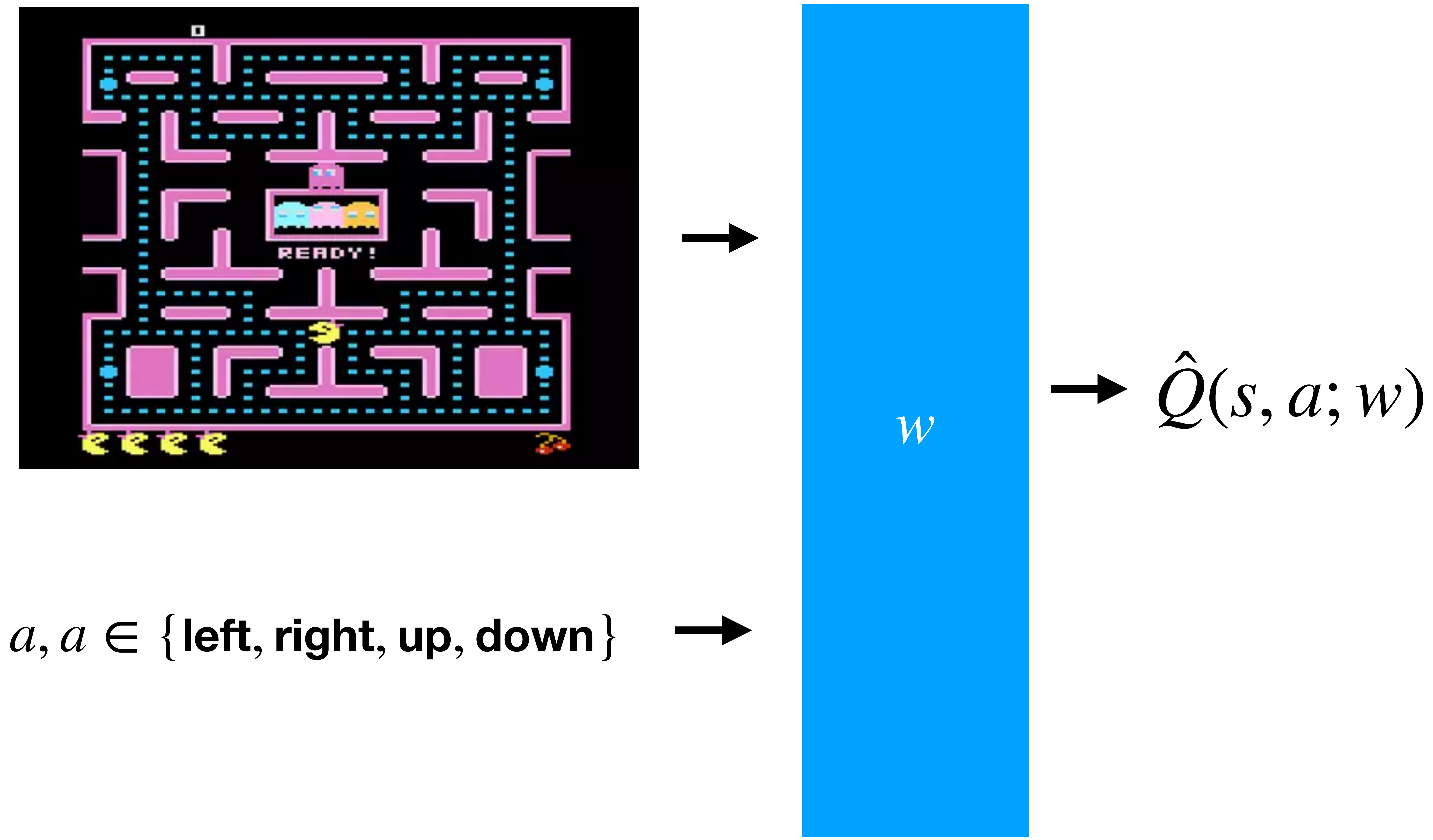
- Do not represent the state value function V or the state-action value function Q explicitly.

- Represent the state value function $V(s)$ or the state-action value function $Q(s, a)$ approximately using a function from some parametrized family, e.g. as a neural network, linear function, decision tree…





$$\begin{tikzcd} {} \arrow[r] & \boxed{w} \arrow[r] & \hat{V}(s; w) \end{tikzcd}$$


READY!
w
$\hat{Q}(s, a; w)$
$a, a \in \{$left, right, up, down$\}$

Part 3: Some Background

Gradient Descent (1/3)

- A method for finding a (local) optimum of a function.
- In our setting, we want to find $\mathbf{w} \in \mathbb{R}^d$ that is a local minimum of a function $J(\mathbf{w})$.
- We do that using *gradient descent.*

Gradient Descent (2/3)

Gradient:

$$\nabla J(\mathbf{w}) = \left(\frac{\partial J}{\partial w_1}(\mathbf{w}), \frac{\partial J}{\partial w_2}(\mathbf{w}), \ldots, \frac{\partial J}{\partial w_d}(\mathbf{w}) \right)$$

Example:

$$J(\mathbf{w}) = w_1 \cdot w_2 + w_1, \, \mathbf{w} \in \mathbb{R}^2.$$

Then

$$\nabla J(\mathbf{w}) = (w_2 + 1, w_1).$$

Gradient Descent (3/3)

Gradient descent update rule:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \cdot \nabla J(\mathbf{w}_n)$$

(gradient descent algorithm iterates this rule).

A Useful Property of Mean Squared Loss

Let $Y_1, Y_2, \ldots, Y_n$ be an independent sample from some distribution with expected value $\mu = \mathbb{E}[Y_i], \forall i$.

What is the value y (~prediction) that minimizes the mean squared error $\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - y\right)^2$?

It is the sample average $y = \frac{1}{n}\sum_{i=1}^{n} Y_i$, which, for $n \to \infty$, converges to the mean μ.

Consequence: Learning a predictor under mean squared loss leads to learning a predictor for conditional expectation (*we will explain later what it means for RL*).

Stochastic Gradient Descent

- We want to optimize a function $J(\mathbf{w})$ of the form $J(\mathbf{w}) = \mathbb{E}[g(X; \mathbf{w})]$ where X is a random variable.
- We assume that we can sample from the distribution w.r.t. which the expectation is taken.
- Stochastic gradient descent uses samples to approximate the gradient of $J(\mathbf{w})$ using just one sample (*SGD can also use a mini-batch of multiple samples but we will not consider it now for simplicity*) and estimates the gradient of J as:

$$\nabla J(\mathbf{w}) \approx \nabla g(X; \mathbf{w})$$

(instead of $\nabla \mathbb{E}[g(X; \mathbf{w})]$).

- *Assuming that we can exchange the order of expectation and taking gradients (which we can when g is well-behaved), the expected SGD step is the same as the full gradient of J.*

Warm-Up: Learning to "Compress" $V^\pi(s)$, (1/3)

- Suppose that we know $V^\pi(s)$ and can query it but yet want to learn an approximation of it… using a parametric function $\hat{V}^\pi(s; \mathbf{w})$…

- We will use mean-squared error to measure how good the approximation is, i.e.:

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[\left(V^\pi(X) - \hat{V}^\pi(X; \mathbf{w}) \right)^2 \right].$$

- How could we train the approximation using SGD?

Warm-Up: Learning to "Compress" $V^{\pi}(s)$, (2/3)

- We will use mean-squared error to measure how good the approximation is, i.e.:

$$J(\mathbf{w}) = \mathbb{E}_{\pi}\left[\left(V^{\pi}(X) - \hat{V}^{\pi}(X; \mathbf{w})\right)^2\right].$$

- How could we train the approximation using SGD (using 1 example in each minibatch)?

```
While (some stopping condition):
    Sample a state $s$ and compute the gradient of $\hat{J}_s(\mathbf{w}) = (V^{\pi}(s) - V(s; \mathbf{w}))^2$:
    $\nabla \hat{J}_s(\mathbf{w}) = -2(V^{\pi}(s) - V^{\pi}(s; \mathbf{w})) \cdot \nabla V^{\pi}(s; \mathbf{w}) = 2(V^{\pi}(s; \mathbf{w}) - V^{\pi}(s)) \cdot \nabla V^{\pi}(s; \mathbf{w})$
    Take the gradient step:
        $\mathbf{w} := \mathbf{w} - \alpha \cdot 2(V^{\pi}(s; \mathbf{w}) - V^{\pi}(s)) \cdot \nabla V(s; \mathbf{w})$
```

Warm-Up: Learning to "Compress" $V^{\pi}(s)$, (3/3)

- But in reality **we will not have access** to $V^{\pi}(s)$!
- So we cannot compute the gradient step: $\mathbf{w} := \mathbf{w} - \alpha \cdot 2(V^{\pi}(s; \mathbf{w}) - V^{\pi}(s)) \cdot \nabla V(s; \mathbf{w})\ldots$
- We will therefore need to combine SGD with what we saw in the previous lectures…

Part 4: Let’s Make It Concrete (Some Parametric Families)

State Representation

- States will be represented by feature vectors.
- The feature vector of a state s will be denoted as $\mathbf{x}(s)$ and we can think of it as a function mapping states to some vector space, e.g. $\mathbb{R}^d$, i.e. $\mathbf{x}(s) = (x_1(s), x_2(s), \ldots, x_d(s))^T$.
- **Examples:**
  - Atari: the feature vector can, e.g., contain the intensities of the pixels (concatenated).
  - Pole balancing: physical features such as velocities, angles…

Linear Functions

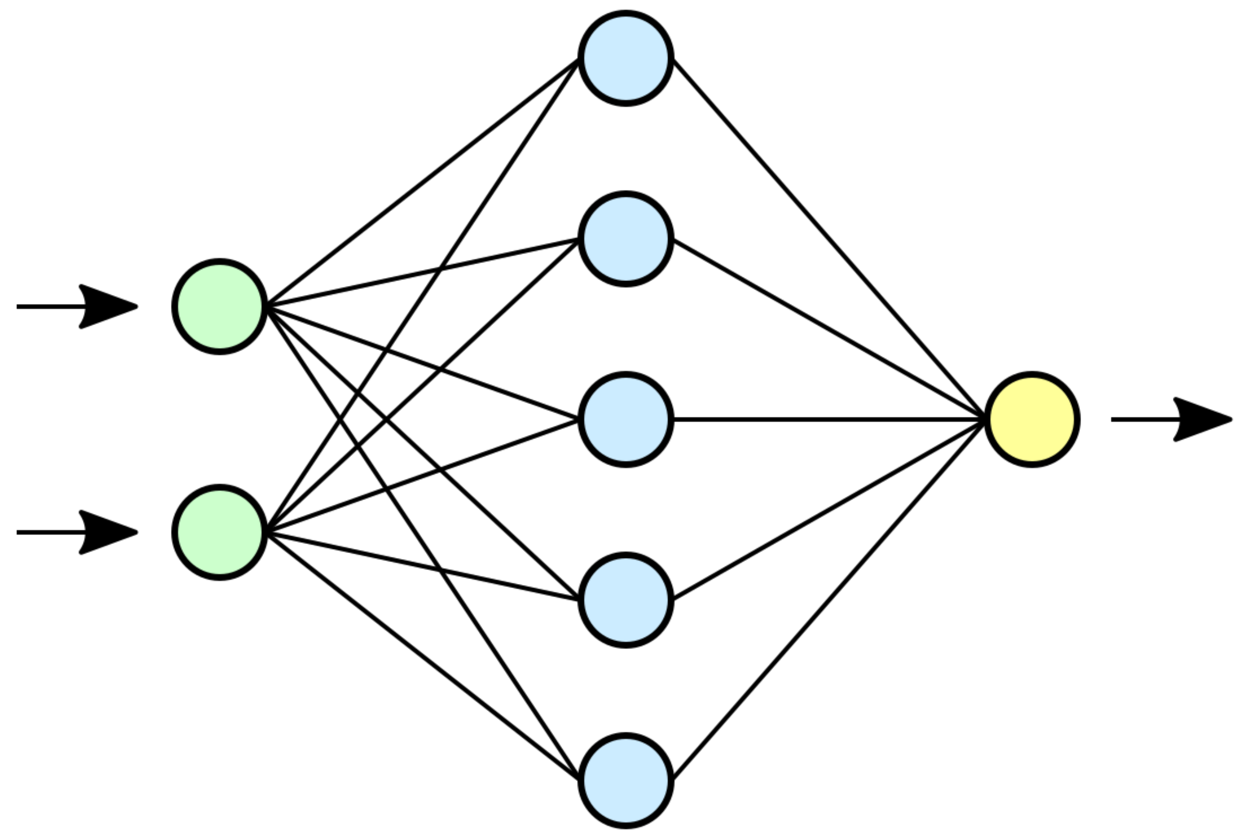
- Scalar product of a weight vector with the feature vector, which represents the state:

$$\hat{V}^{\pi}(s; \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s).$$

- Linear function approximations can work well but need good features (which requires feature engineering).

Neural Networks

- Neural network (*well, you know them*):



- In this lecture we will think of neural networks simply as blackboxes $g(\mathbf{x}; \mathbf{w})$ which we can evaluate and for which we can compute the gradients $\nabla_{\mathbf{w}} g(\mathbf{x}; \mathbf{w})$ efficiently *(we will usually omit the subscript* $\mathbf{w}$ *from* $\nabla_{\mathbf{w}}$ *when it is clear from the context)*.

- In particular, the approximation will have the form $V^{\pi}(s; \mathbf{w}) = g(\mathbf{x}(s); \mathbf{w})$, where g is some neural network…

Part 5: Policy Evaluation with Function Approximation

Monte-Carlo Value Function Approximation

We know from the previous lectures that the return $G_t^\pi = R(X_t, \pi(X_t)) + \gamma \cdot R(X_{t+1}, \pi(X_{t+1})) + \ldots$ is an unbiased estimate (but with high variance) of $V^\pi(X_t)$… *Conditioning on the value of* X_t, *we then arrive at the Monte-Carlo estimators, i.e. we have* $V^\pi(s) = \mathbb{E}[G_t \mid X_t = s]$.

Basic Idea (*not yet complete… wait for the next slide*)**:** We can frame the value function approximation problem as a supervised learning problem under MSE loss:

Sample an episode under policy π**:** $s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T$

Training examples: $[s_1, g_1], [s_2, g_2], \ldots, [s_{T-1}, g_{T-1}]$, where g_i denotes the return from the episode from time i.

First-visit or every-visit? See next slide.

First/Every-Visit Monte-Carlo Value Function Approximation

```
Initialize: $\mathbf{w} =$ some initialization....
For $i = 1,\ldots,N$:
    Sample episode $e_i := s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \ldots, s_{i,T_i}$.
    For each time step $1 \le t \le T_i$:
        If $t$ is the first occurrence of state $s$ in the episode $e_i$ /* This is for first-visit MC */
            $s$ is the state visited at time $t$ in the episode $e_i$
            $g_{i,t} := r_{i,t} + \gamma \cdot r_{i,t+1} + \gamma^2 \cdot r_{i,t+2} + \ldots + \gamma^{T_i - t} \cdot r_{i,T_i}$
             /* SGD step */
            $\mathbf{w} := \mathbf{w} - \alpha \cdot (V^{\pi}(\mathbf{x}(s_t); \mathbf{w}) - g_t) \cdot \nabla V(\mathbf{x}(s_t); \mathbf{w})$
```

MC Value Function Evaluation with Linear Function Approximation

We have

$$\hat{V}^{\pi}(s; \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s)$$

$$\nabla \hat{V}^{\pi}(s; \mathbf{w}) = \mathbf{x}(s).$$

```
Initialize: $\mathbf{w} =$ some initialization....
For $i = 1, \dots, N$:
    Sample episode $e_i := s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \dots, s_{i,T_i}$.
    For each time step $1 \le t \le T_i$:
        If $t$ is the first occurrence of state $s$ in the episode $e_i$ /* This is for first-visit MC */
            $s$ is the state visited at time $t$ in the episode $e_i$
            $g_{i,t} := r_{i,t} + \gamma \cdot r_{i,t+1} + \gamma^2 \cdot r_{i,t+2} + \dots + \gamma^{T_i - t} \cdot r_{i,T_i}$
            /* SGD step */
            $\boxed{\mathbf{w} := \mathbf{w} - \alpha \cdot (\mathbf{w}^T \mathbf{x}(s_t) - g_t) \cdot \mathbf{x}(s_t)}$
```

Intuition About Why It Works

- Recall that what we want to estimate is $V^{\pi}(s) = \mathbb{E}[G_t | X_t = s]$, i.e. expected value.
- When using first-visit MC, each of the training examples $[s_t, g_t]$ is an unbiased (but very noisy!) estimate of $V^{\pi}(s)$. But when we use these examples and try to find a best mean-squared-error fit then we are estimating their expectation which equals $V^{\pi}(s)$. And that is why it works…

Convergence of MC VFA (1/3)

- **Definition (On-Policy Distribution):** Given an MDP and a policy π, we define on-policy distribution P^{π}_{onp} as follows.
    - **In non-episodic settings:** P^{π}_{onp} is the stationary distribution of the MRP that is given by the MDP and the policy (*recall MDP + policy = MRP*).
    - **In episodic settings:** P^{π}_{onp} depends also on the distribution of the initial states P_{init} *(see Sutton's book for details)*.
- In what follows, we denote the on-policy distribution by P^{π}_{onp}.

Convergence of MC VFA (2/3)

- **Definition:** Mean squared error of value function approximation is defined as

$$MSVE_{\pi}(\mathbf{w}) = \sum_{s \in S} P^{\pi}_{onp}(s) \cdot \left(V^{\pi}(s) - \hat{V}^{\pi}(s; \mathbf{w}) \right)^2,$$

which is the same as

$$MSVE_{\pi}(\mathbf{w}) = \mathbb{E}_{X \sim P^{\pi}_{onp}} \left[\left(V^{\pi}(s) - \hat{V}^{\pi}(s; \mathbf{w}) \right)^2 \right].$$

Convergence of MC VFA (3/3)

- **Theorem:** Assume that $\hat{V}^{\pi}(s; \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s)$, i.e. **we are assuming linear function approximation**. Then MC VFA converges to weights that are optimal in the sense that they minimize $MSVE_{\pi}(\mathbf{w})$.

- **Caution:** This theorem holds for **linear** function approximation, not for general functions! We do not have such guarantees for, e.g., arbitrary neural networks.

Temporal Difference VFA (1/5)

- For temporal difference learning, we had the following update rule:

$$V^\pi(s_t) := V^\pi(s_t) + \alpha \cdot \left(\underbrace{r_t + \gamma \cdot V^\pi(s_{t+1})}_{\text{TD-target}} - V^\pi(s_t) \right).$$

- Now, we will want to have a similar update rule but for the case where $V^\pi(s)$ is only approximated by $V^\pi(s; \mathbf{w})$.

Temporal Difference VFA (2/5)

Recall the Bellman equation (*for simplicity, we are showing it for deterministic policy*):

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \cdot \sum_{s' \in S} P(s' \,|\, s, \pi(s)) \cdot V(s')$$

which is the same as:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \cdot \mathbb{E}\left[V^{\pi}(X_{t+1}) \,|\, X_t = s\right].$$

We can turn the system of equations above into the following minimization problem:

$$\min_{\mathbf{V}^{\pi}} \sum_{s \in S} P_{stat}(s) \cdot \mathbb{E}\left[\left(R(s, \pi(s)) + \gamma \cdot V^{\pi}(X_{t+1}) - V^{\pi}(s)\right)^2 \middle| \, X_t = s\right].$$

Temporal Difference VFA (3/5)

Next we replace $V^{\pi}(s)$ by its approximation $\hat{V}^{\pi}(s;\mathbf{w})$, yielding:

$$\min_{\mathbf{V}^{\pi}} \sum_{s \in S} P_{stat}(s) \cdot \mathbb{E}\left[\left(R(s,\pi(s)) + \gamma \cdot \hat{V}^{\pi}(X_{t+1};\mathbf{w}) - \hat{V}^{\pi}(s;\mathbf{w})\right)^2 \middle| X_t = s\right].$$

Now, instead of the stationary distribution, we will just take the distribution of states as they come in an episode and instead of the expectation we will use the tuple (s_t, a_t, r_t, s_{t+1}) we get in the current episode *(as is common in TD-learning)*. That will lead us to the minimization problem:

$$\min_{\mathbf{w}} \left(R(s_t, r_t) + \gamma \cdot \hat{V}^{\pi}(s_{t+1};\mathbf{w}) - \hat{V}^{\pi}(s_t;\mathbf{w})\right)^2$$

Temporal Difference VFA (4/5)

We need to solve:

$$\min_{\mathbf{w}} \left(R(s_t, r_t) + \gamma \cdot \hat{V}^{\pi}(s_{t+1}; \mathbf{w}) - \hat{V}^{\pi}(s_t; \mathbf{w}) \right)^2$$

. Denoting

$$J(\mathbf{w}) = \left(R(s_t, r_t) + \gamma \cdot \hat{V}^{\pi}(s_{t+1}; \mathbf{w}) - \hat{V}^{\pi}(s_t; \mathbf{w}) \right)^2$$

we have

$$\nabla J(\mathbf{w}) = 2 \left(R(s_t, r_t) + \gamma \cdot \hat{V}^{\pi}(s_{t+1}; \mathbf{w}) - \hat{V}^{\pi}(s_t; \mathbf{w}) \right) \cdot (\gamma \cdot \nabla \hat{V}^{\pi}(s_{t+1}; \mathbf{w}) - \nabla \hat{V}^{\pi}(s_t; \mathbf{w}))$$

But this is not what TD with function approximation does! TD VFA is a so-called semigradient method. It does not consider the contribution of $\nabla \hat{V}^{\pi}(s_{t+1}; \mathbf{w})$ and considers it fixed.

Temporal Difference VFA (4/5)

We need to solve:

$$\min_{\mathbf{w}} \left(R(s_t, r_t) + \gamma \cdot \hat{V}^{\pi}(s_{t+1}; \mathbf{w}) - \hat{V}^{\pi}(s_t; \mathbf{w}) \right)^2 .$$ Denoting

$$J(\mathbf{w}) = \left(R(s_t, r_t) + \gamma \cdot \hat{V}^{\pi}(s_{t+1}; \mathbf{w}) - \hat{V}^{\pi}(s_t; \mathbf{w}) \right)^2$$

we have

$$\nabla J(\mathbf{w}) = 2 \left(R(s_t, r_t) + \gamma \cdot \hat{V}^{\pi}(s_{t+1}; \mathbf{w}) - \hat{V}^{\pi}(s_t; \mathbf{w}) \right) \cdot \left(\gamma \cdot \nabla \hat{V}^{\pi}(s_{t+1}; \mathbf{w}) - \nabla \hat{V}^{\pi}(s_t; \mathbf{w}) \right)$$

But this is not what TD with function approximation does! TD VFA is a so-called semigradient method. It does not consider the contribution of $\nabla \hat{V}^{\pi}(s_{t+1}; \mathbf{w})$ and considers it fixed.

Temporal Difference VFA (5/5)

The TD update rule for value function approximation is:

$$\mathbf{w} := \mathbf{w} + \alpha \left(r_t + \gamma \cdot \hat{V}^{\pi}(s_{t+1}; \mathbf{w}) - \hat{V}^{\pi}(s_t; \mathbf{w}) \right) \cdot \nabla \hat{V}^{\pi}(s_t; \mathbf{w})$$

TD VFA with Linear Functions

We have

$$\hat{V}^{\pi}(s; \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s),$$

$$\nabla \hat{V}^{\pi}(s; \mathbf{w}) = \mathbf{x}(s).$$

The TD update rule for value function approximation with **linear function** is:

$$\mathbf{w} := \mathbf{w} + \alpha \left(r_t + \gamma \cdot \mathbf{w}^T \mathbf{x}(s_{t+1}) - w^T \mathbf{x}(s_t) \right) \mathbf{x}(s_t).$$

Convergence of TD VFA with Linear Functions

- As for MC VFA, we will use the on-policy distribution P^{π}_{onp} and define the mean squared error w.r.t. it, that is…

$$MSVE_{\pi}(\mathbf{w}) = \sum_{s \in S} P^{\pi}_{onp}(s) \cdot \left(V^{\pi}(s) - \hat{V}^{\pi}(s; \mathbf{w}) \right)^2$$

- **Theorem:** Let $\mathbf{w}_{TD}$ be the weight vector to which TD VFA converges. Then it holds:

$$MSVE_{\pi}(\mathbf{w}_{TD}) \leq \frac{1}{1-\gamma} \cdot \min_{\mathbf{w}} MSVE_{\pi}(\mathbf{w}).$$

- Recall that for MC VFA with linear functions we had convergence of mean squared error to $\min_{\mathbf{w}} MSVE_{\pi}(\mathbf{w})$.

Part 6: Control with Function Approximation

Basic Idea

- Same ideas, just plugging them into what we were doing in the last lecture, but there are caveats…
- Instead of approximating V^π, we need to approximate $Q^\pi(s, a)$ from which we can then extract the best action to take in a given state s.
- The algorithms are similar to those we saw last week (MC, SARSA, Q-Learning). **Important: the idea of using ε-greedy policies.** The motivation is the same but we use $Q^\pi(s, a; \mathbf{w})$.

Representing State-Action Pairs

- For control RL problems, we need to encode both states and actions together.
- The feature vector of a state-action pair (s, a) will be denoted as $\mathbf{x}(s, a)$ and we can think of it as a function mapping state-action pairs to some vector space, e.g. $\mathbb{R}^d$, i.e. $\mathbf{x}(s, a) = (x_1(s, a), x_2(s, a), \ldots, x_d(s, a))^T$.

Approximation of Q-Function

- **Linear function approximation:** Scalar product of a weight vector with the feature vector, which represents the state-action pair:

$$\hat{Q}^{\pi}(s, a; \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s, a).$$

- **Neural network function approximation:**

$$\hat{Q}^{\pi}(s, a; \mathbf{w}) = g(\mathbf{x}(s, a); \mathbf{w})$$

where g is a function represented as a neural network.

Weight Updates

- MC:

$$\mathbf{w} := \mathbf{w} + \alpha \cdot \left(g_t - \hat{Q}(s_t, a_t; \mathbf{w})\right) \cdot \nabla \hat{Q}(s_t, a_t; \mathbf{w})$$

- SARSA:

$$\mathbf{w} := \mathbf{w} + \alpha \cdot \left(r + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \mathbf{w}) - \hat{Q}(s_t, a_t; \mathbf{w})\right) \cdot \nabla Q(s_t, a_t; \mathbf{w})$$

- Q-Learning:

$$\mathbf{w} := \mathbf{w} + \alpha \cdot \left(r + \gamma \max_{a \in A} \hat{Q}(s_{t+1}, a; \mathbf{w}) - \hat{Q}(s_t, a_t; \mathbf{w})\right) \cdot \nabla Q(s_t, a_t; \mathbf{w})$$

Convergence

- Next time…