10-701: Introduction to Machine Learning Lecture 11: Q-Learning and Policy Gradient RL

Henry Chai & Zack Lipton

10/04/23

Front Matter

- Announcements
  - HW2 released 9/20, due 10/4 (today!) at 11:59 PM
  - HW3 released 10/4 (today!), due 10/11 at 11:59 PM
  - Project details will be released on 10/13
    - You will have a choice between a more research-based project and a more implementation-focused project
    - You must work on the project in groups of 3 or 4; **you may not work on the project alone.**
- Recommended Readings
  - Mitchell, Chapter 13

Two big Q's

1. What can we do if the reward and/or transition functions/distributions are unknown?

2. How can we handle infinite (or just very large) state/action spaces?

Recall: Value Iteration

- Inputs: $R(s,a)$, $p(s' \mid s,a)$, γ
- Initialize $V^{(0)}(s) = 0 \; \forall \, s \in \mathcal{S}$ (or randomly) and set $t = 0$
- While not converged, do:
  - For $s \in \mathcal{S}$
    - For $a \in \mathcal{A}$

      $$Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s')$$

    - $V(s) \leftarrow \max_{a \in \mathcal{A}} Q(s,a)$
- For $s \in \mathcal{S}$

  $$\pi^*(s) \leftarrow \operatorname*{argmax}_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s')$$

- Return π^*

$Q^*(s,a)$ w/ deterministic rewards

- $Q^*(s,a) = \mathbb{E}[$total discounted reward of taking action a in state s, assuming all future actions are optimal$]$

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

$$V^*(s') = \max_{a' \in \mathcal{A}} Q^*(s',a')$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \left[\max_{a' \in \mathcal{A}} Q^*(s',a') \right]$$

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s,a)$$

- Insight: if we know Q^*, we can compute an optimal policy π^*!

$Q^*(s,a)$ w/ deterministic rewards and transitions

- $Q^*(s,a) = \mathbb{E}[$total discounted reward of taking action a in state s, assuming all future actions are optimal$]$

$$= R(s,a) + \gamma V^*\big(\delta(s,a)\big)$$

- $V^*\big(\delta(s,a)\big) = \max_{a' \in \mathcal{A}} Q^*(\delta(s,a),a')$

$$Q^*(s,a) = R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^*(\delta(s,a),a')$$

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s,a)$$

- Insight: if we know Q^*, we can compute an optimal policy π^*!

Learning $Q^*(s,a)$ w/ deterministic rewards and transitions

Algorithm 1: Online learning (table form)

- Inputs: discount factor γ, an initial state s
- Initialize $Q(s,a) = 0 \; \forall \, s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - Take a random action a
  - Receive reward $r = R(s,a)$
  - Update the state: $s \leftarrow s'$ where $s' = \delta(s,a)$
  - Update $Q(s,a)$:

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

Learning $Q^*(s, a)$ w/ deterministic rewards and transitions

Algorithm 2: ϵ-greedy online learning (table form)

- Inputs: discount factor γ, an initial state s,
  greediness parameter $\epsilon \in [0, 1]$
- Initialize $Q(s, a) = 0 \; \forall \; s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - With probability ϵ, take the greedy action
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s, a')$$
    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s, a)$
  - Update the state: $s \leftarrow s'$ where $s' = \delta(s, a)$
  - Update $Q(s, a)$:
    $$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

Learning $Q^*(s,a)$ w/ deterministic rewards

Algorithm 3: ϵ-greedy online learning (table form)

- Inputs: discount factor γ, an initial state s,
  greediness parameter $\epsilon \in [0,1]$,
  learning rate $\alpha \in [0,1]$ ("trust parameter")
- Initialize $Q(s,a) = 0 \; \forall \, s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - With probability ϵ, take the greedy action
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s,a')$$
    Otherwise, with probability $1-\epsilon$, take a random action a
  - Receive reward $r = R(s,a)$
  - Update the state: $s \leftarrow s'$ where $s' \sim p(s' \mid s,a)$
  - Update $Q(s,a)$:
    $$Q(s,a) \leftarrow (1-\alpha)\underbrace{Q(s,a)}_{\text{Current value}} + \alpha\left(\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{Update w/ deterministic transitions}}\right)$$

Learning $Q^*(s,a)$ w/ deterministic rewards

Algorithm 3: ϵ-greedy online learning (table form)

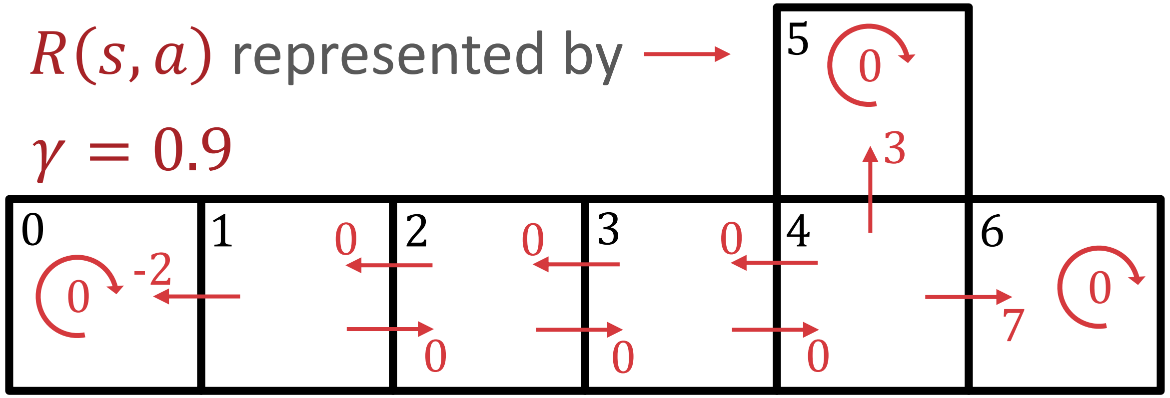
- Inputs: discount factor γ, an initial state s,
  greediness parameter $\epsilon \in [0,1]$,
  learning rate $\alpha \in [0,1]$ (“trust parameter”)
- Initialize $Q(s,a) = 0 \; \forall \, s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - With probability ϵ, take the greedy action
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s,a')$$
    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s,a)$
  - Update the state: $s \leftarrow s'$ where $s' \sim p(s' \mid s,a)$
  - Update $Q(s,a)$:

$$Q(s,a) \leftarrow \underbrace{Q(s,a)}_{\text{Current value}} + \alpha \left(\overbrace{\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{Temporal difference target}} - Q(s,a)}^{\text{Temporal difference}} \right)$$

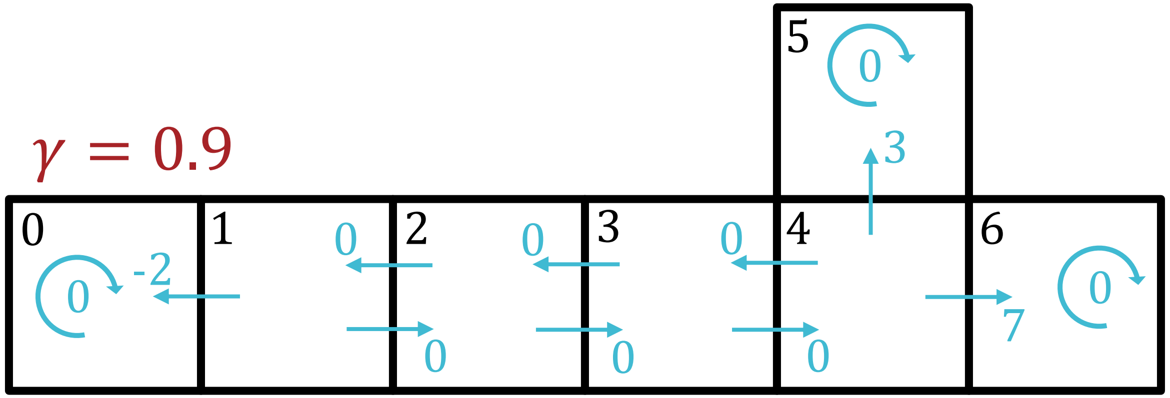
Learning $Q^*(s,a)$: Example

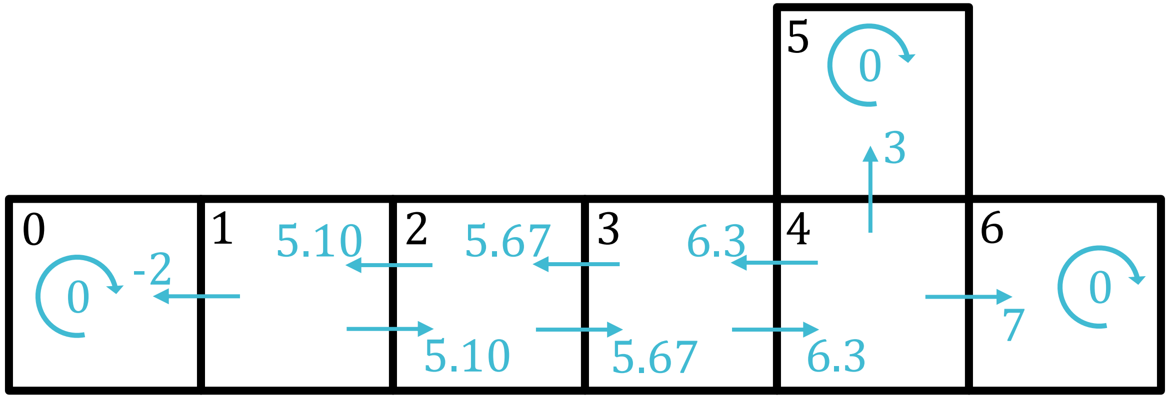
$R(s,a)$ represented by →

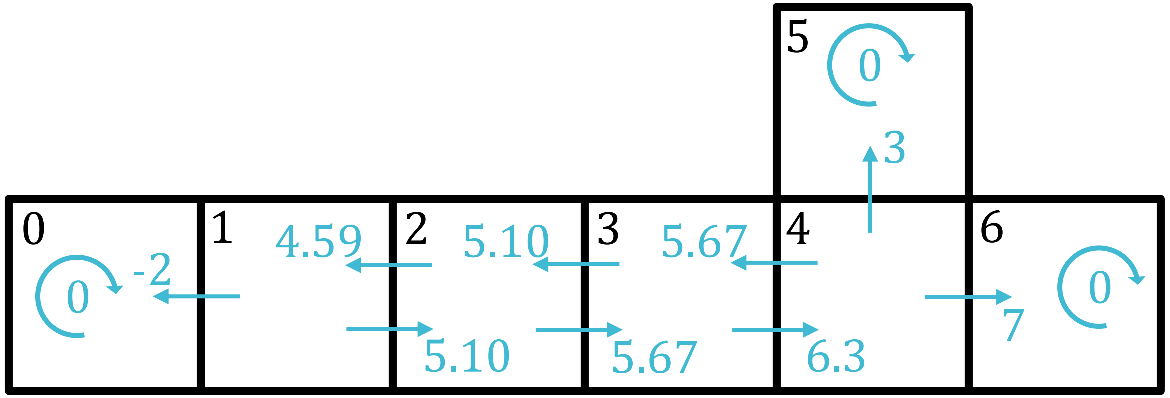
$\gamma = 0.9$



Which set of blue arrows (roughly) corresponds to $Q^*(s,a)$?

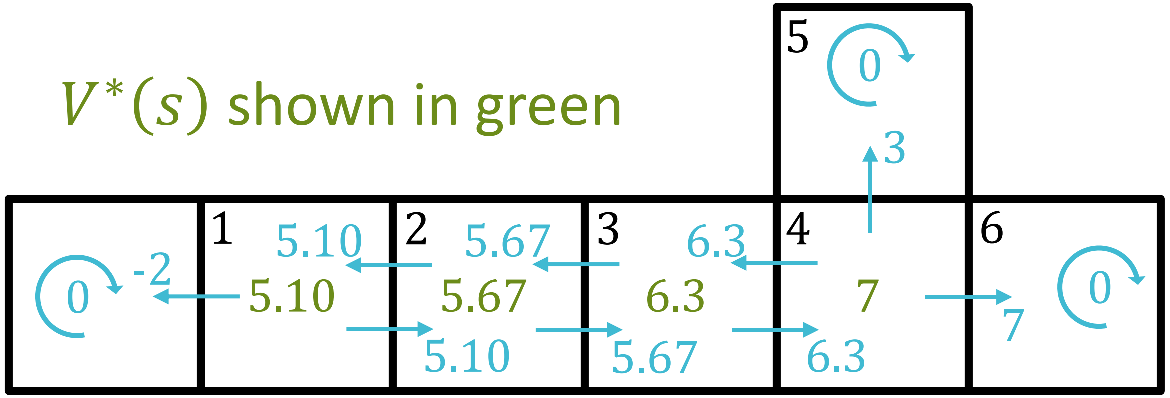




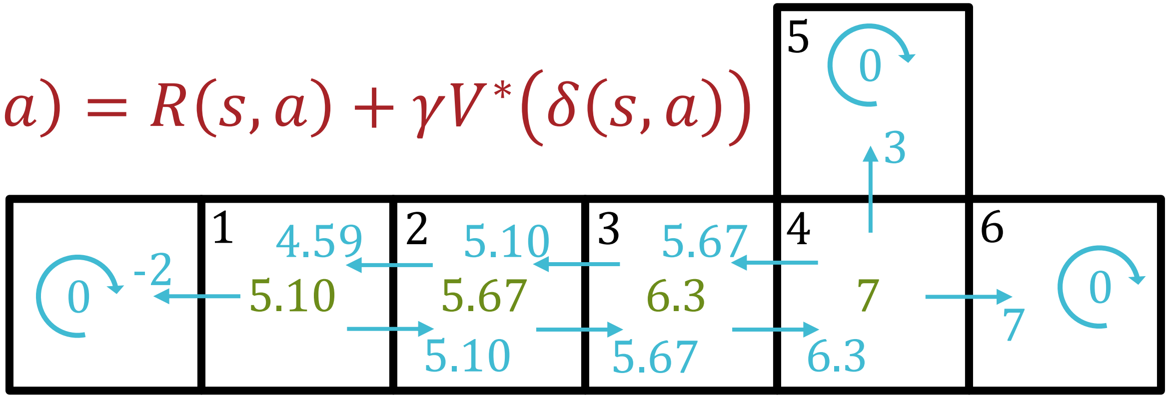


Which set of blue arrows (roughly) corresponds to $Q^*(s,a)$?

$V^*(s)$ shown in green



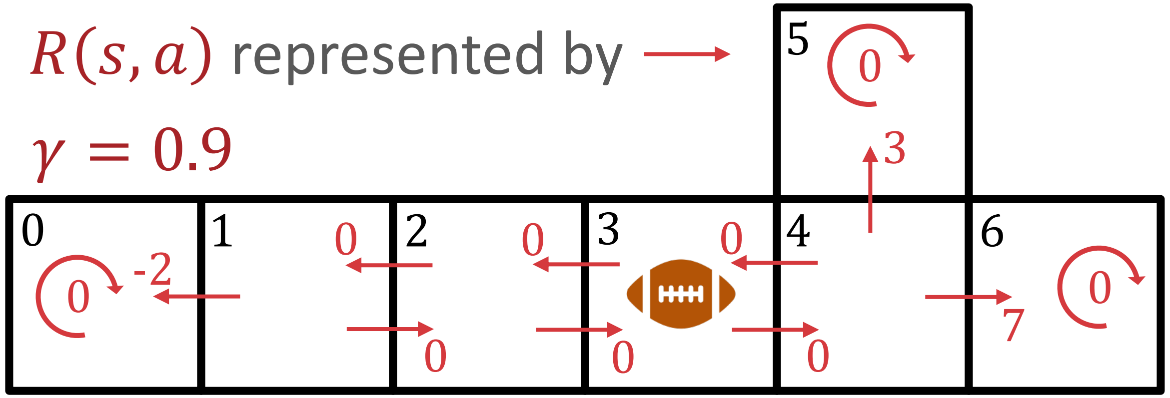
$$Q^*(s,a) = R(s,a) + \gamma V^*(\delta(s,a))$$



Learning $Q^*(s,a)$: Example

$R(s,a)$ represented by →

$\gamma = 0.9$

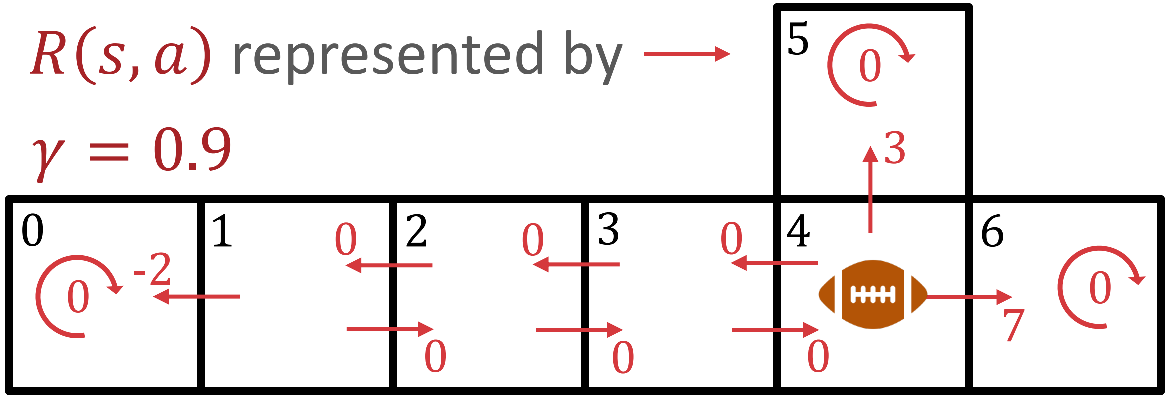


$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

Learning $Q^*(s,a)$: Example

$R(s,a)$ represented by →

$\gamma = 0.9$



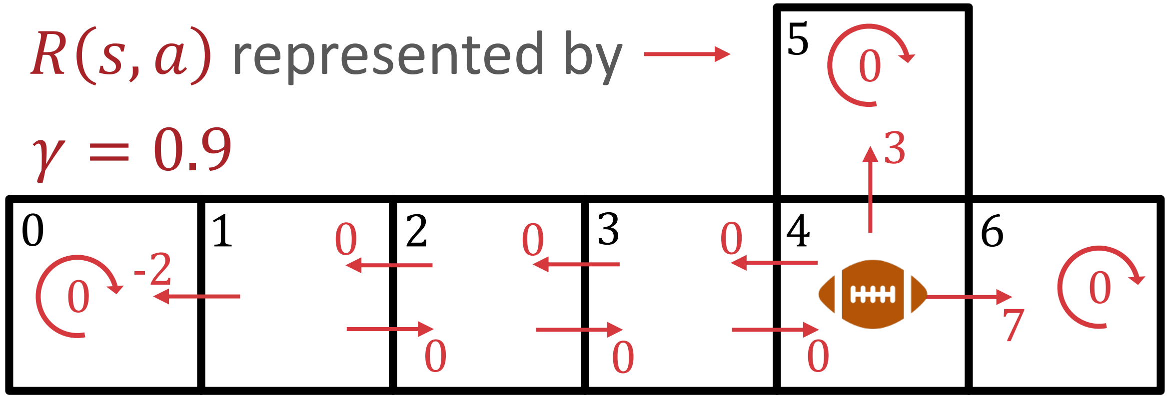
$$Q(3,\rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(4,a') = 0$$

$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

Learning $Q^*(s, a)$: Example

$R(s, a)$ represented by →

$\gamma = 0.9$

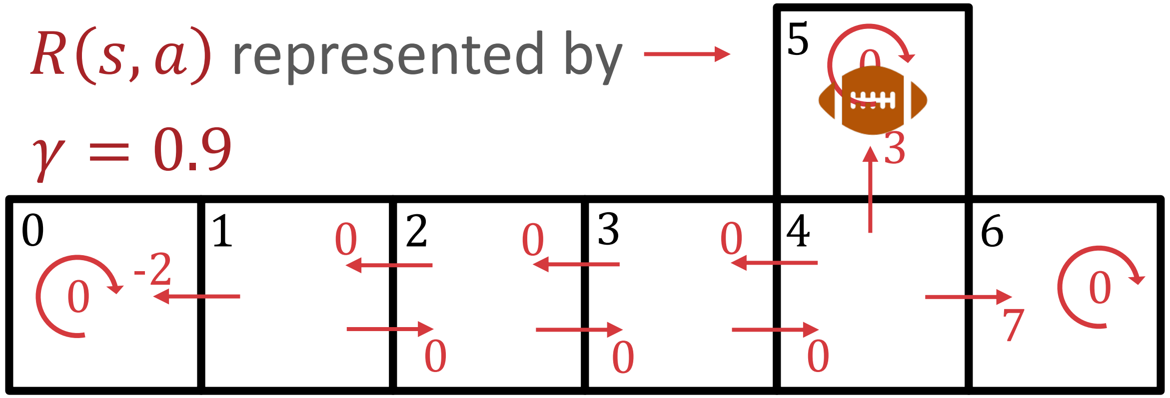


$Q(s, a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

Learning $Q^*(s,a)$: Example

$R(s,a)$ represented by →

$\gamma = 0.9$



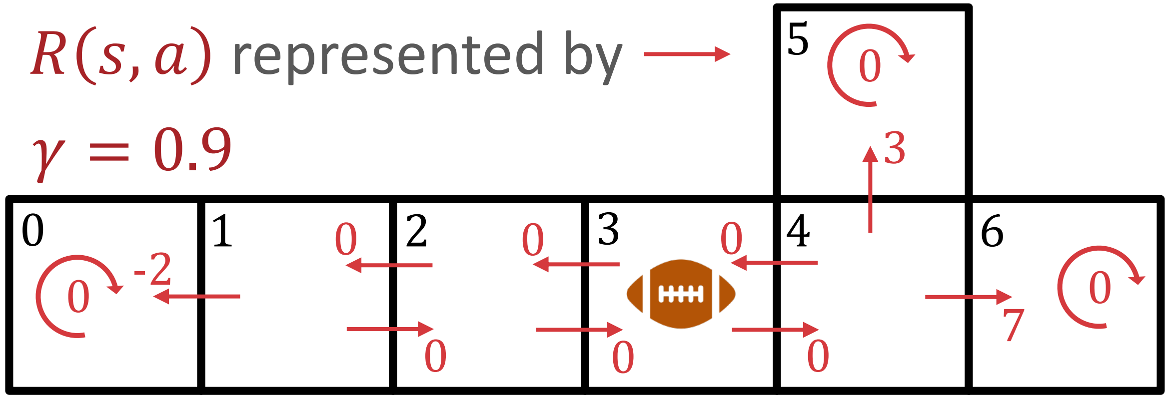
$$Q(4,\uparrow) \leftarrow 3 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(5,a') = 3$$

$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	**0**	0
5	0	0	0	0
6	0	0	0	0

Learning $Q^*(s,a)$: Example

$R(s,a)$ represented by →

$\gamma = 0.9$



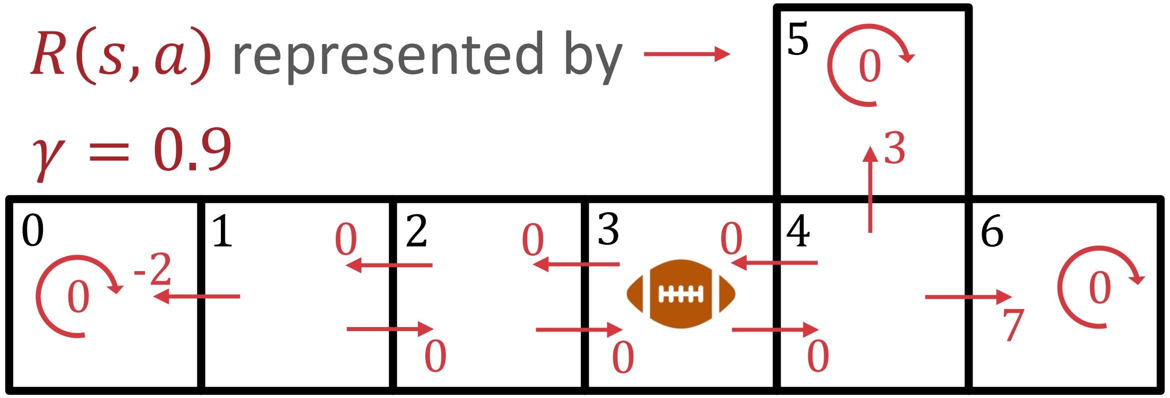
$$Q(3,\rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(4,a') = 2.7$$

$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	**0**	0	0	0
4	0	0	3	0
5	0	0	0	0
6	0	0	0	0

Learning $Q^*(s,a)$: Example

$R(s,a)$ represented by →

$\gamma = 0.9$



$$Q(3,\rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(4,a') = 2.7$$

$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	2.7	0	0	0
4	0	0	3	0
5	0	0	0	0
6	0	0	0	0

Learning $Q^*(s, a)$: Convergence

- For Algorithms 1 & 2 (deterministic transitions), Q converges to Q^* if
  1. Every valid state-action pair is visited infinitely often
     - Q-learning is exploration-insensitive: any visitation strategy that satisfies this property will work!
  2. $0 \leq \gamma < 1$
  3. $\exists\, \beta$ s.t. $|R(s, a)| < \beta\ \forall\, s \in \mathcal{S}, a \in \mathcal{A}$
  4. Initial Q values are finite

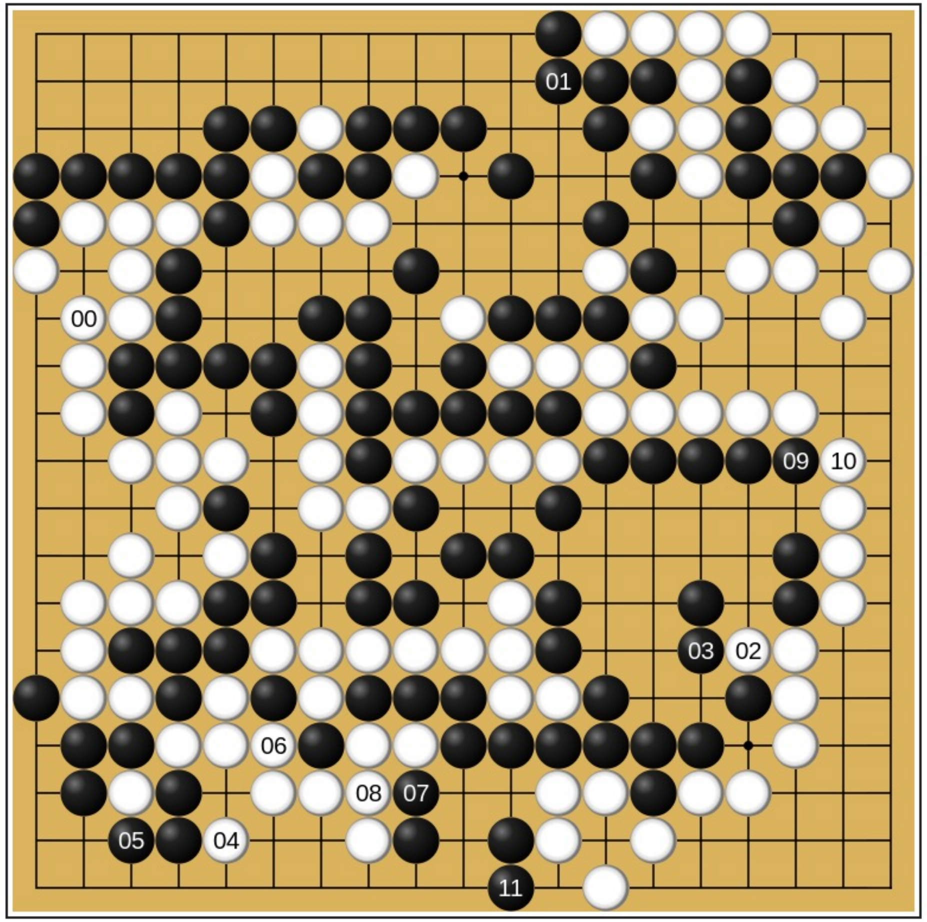
Learning $Q^*(s,a)$: Convergence

- For Algorithm 3 (temporal difference learning), Q converges to Q^* if
  1. Every valid state-action pair is visited infinitely often
     - Q-learning is exploration-insensitive: any visitation strategy that satisfies this property will work!
  2. $0 \leq \gamma < 1$
  3. $\exists\, \beta$ s.t. $|R(s,a)| < \beta\ \forall\, s \in \mathcal{S}, a \in \mathcal{A}$
  4. Initial Q values are finite
  5. Learning rate α_t follows some "schedule" s.t. $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$ e.g., $\alpha_t = {}^{1}/_{t+1}$

Two big Q's

1. What can we do if the reward and/or transition functions/distributions are unknown?
   - Use online learning to gather data and learn $Q^*(s, a)$
2. How can we handle infinite (or just very large) state/action spaces?

AlphaGo (Black) vs. Lee Sedol (White)
Game 2 final position (AlphaGo wins)

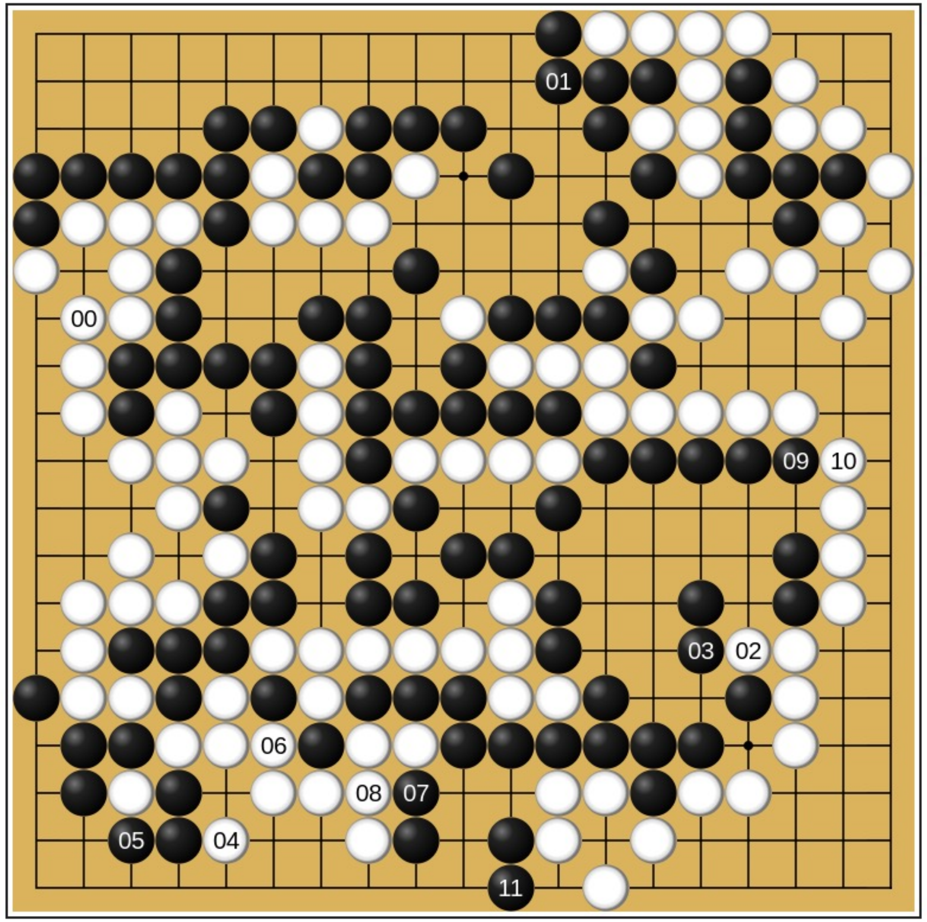


Source: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol

Playing Go

- 19-by-19 board
- Players alternate placing black and white stones
- The goal is claim more territory than the opponent
- How many legal Go board states are there?

AlphaGo (Black) vs. Lee Sedol (White)
Game 2 final position (AlphaGo wins)



Source: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol

Source: https://en.wikipedia.org/wiki/Go_and_mathematics

Playing Go

- 19-by-19 board
- Players alternate placing black and white stones
- The goal is claim more territory than the opponent
- There are ~10^{170} legal Go board states!

Two big Q's

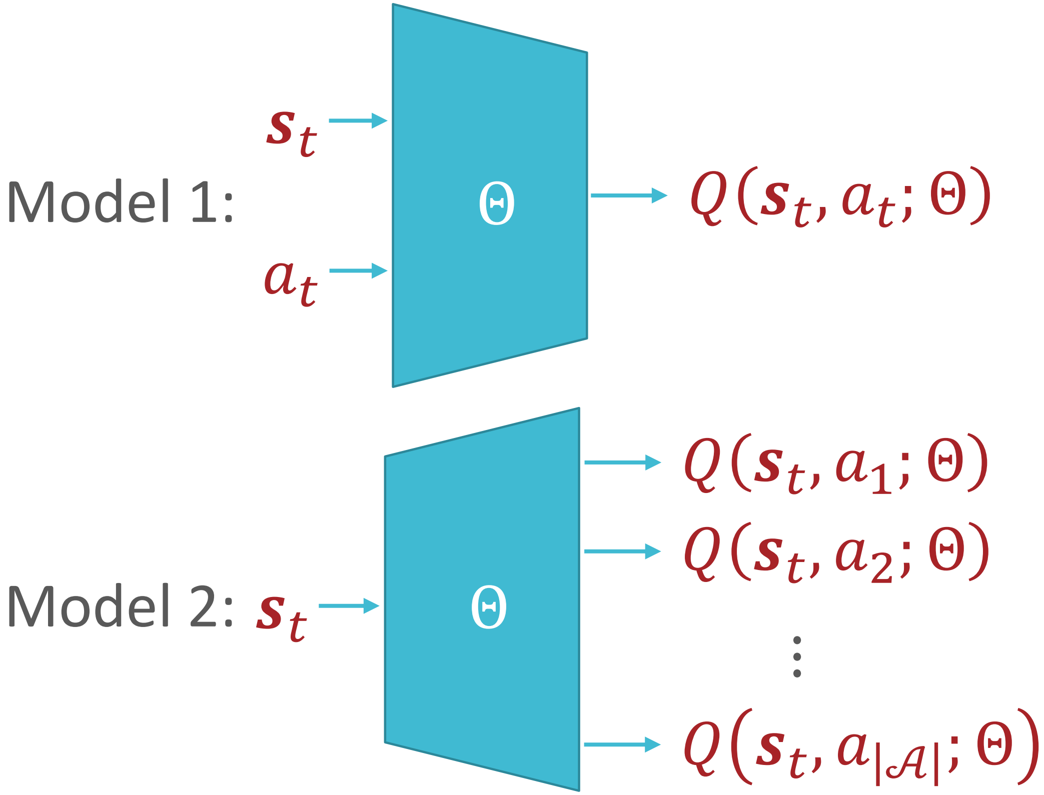
1. What can we do if the reward and/or transition functions/distributions are unknown?
    - Use online learning to gather data and learn $Q^*(s,a)$
2. How can we handle infinite (or just very large) state/action spaces?
    - Throw a neural network at it!

Deep Q-learning

- Use a parametric function, $Q(s, a; \Theta)$, to approximate $Q^*(s, a)$
  - Learn the parameters using *stochastic* gradient descent (SGD)
  - Training data $(\boldsymbol{s}_t, a_t, r_t, \boldsymbol{s}_{t+1})$ gathered online by the agent/learning algorithm

Deep Q-learning: Model

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$
e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a *differentiable* function that approximates Q



Deep Q-learning: Loss Function

- "True" loss

$$\ell(\Theta) = \sum_{\underbrace{s \in \mathcal{S}}_{\text{1. } \mathcal{S} \text{ too big to compute this sum}}} \sum_{a \in \mathcal{A}} \left(\overbrace{Q^*(s,a)}^{\text{2. Don't know } Q^*} - Q(s,a;\Theta)\right)^2$$

1. Use stochastic gradient descent: just consider one state-action pair in each iteration
2. Use temporal difference learning:
   - Given current parameters $\Theta^{(\mathrm{t})}$ the temporal difference target is

     $$Q^*(s,a) \approx r + \gamma \max_{a'} Q\left(s', a'; \Theta^{(t)}\right) := y$$

   - Set the parameters in the next iteration $\Theta^{(\mathrm{t+1})}$ such that $Q\left(s,a;\Theta^{(\mathrm{t+1})}\right) \approx y$

     $$\ell\left(\Theta^{(\mathrm{t})}, \Theta\right) = \left(y - Q(s,a;\Theta)\right)^2$$

Deep Q-learning

Algorithm 4: Online learning (parametric form)

- Inputs: discount factor γ, an initial state s_0,

  learning rate α
- Initialize parameters $\Theta^{(0)}$
- For $t = 0, 1, 2, \ \dots$
  - Gather training sample $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$
  - Update $\Theta^{(t)}$ by taking a step opposite the gradient

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \alpha \nabla_\Theta \ell\left(\Theta^{(t)}, \Theta\right)$$

where

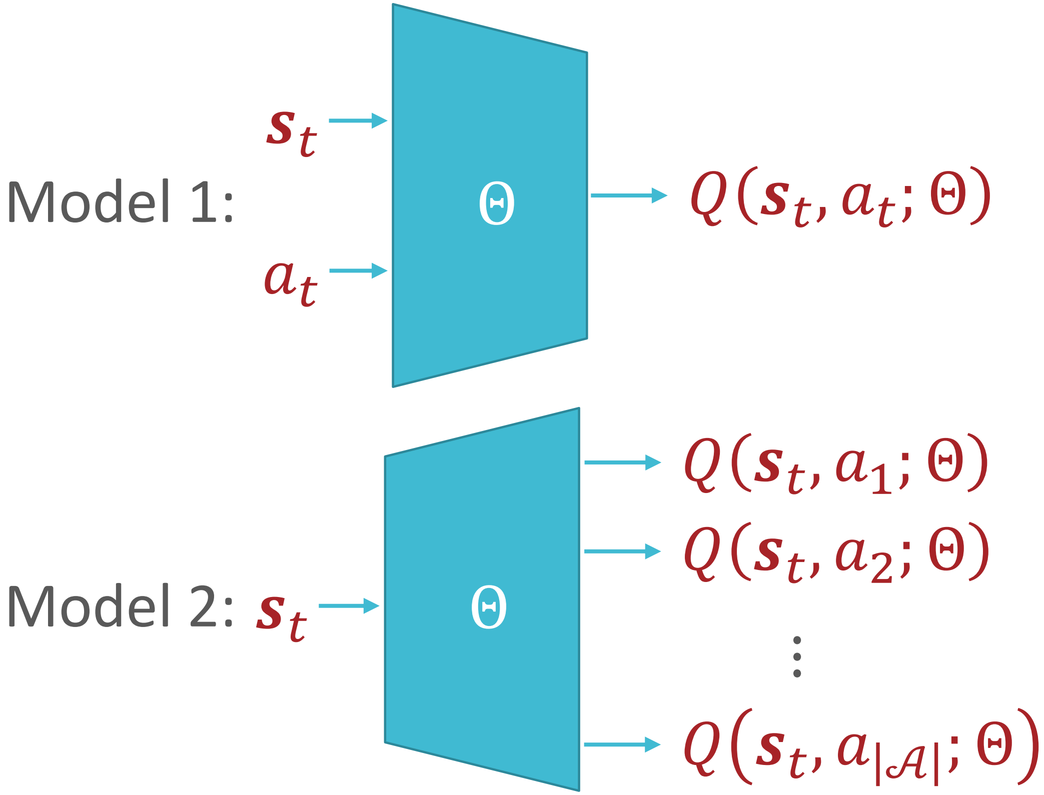
$$\nabla_\Theta \ell\left(\Theta^{(t)}, \Theta\right) = 2\big(y - Q(s, a; \Theta)\big)\nabla_\Theta Q(s, a; \Theta)$$

Deep Q-learning: Experience Replay

- SGD assumes i.i.d. training samples but in RL, samples are *highly* correlated
- Idea: keep a "replay memory" $\mathcal{D} = \{e_1, e_2, \dots, e_N\}$ of the N most recent experiences $e_t = (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ (Lin, 1992)
  - Also keeps the agent from "forgetting" about recent experiences
- Alternate between:
  1. Sampling some e_i uniformly at random from $\mathcal{D}$ and applying a Q-learning update (repeat T times)
  2. Adding a new experience to $\mathcal{D}$
- Can also sample experiences from $\mathcal{D}$ according to some distribution that prioritizes experiences with high error (Schaul et al., 2016)

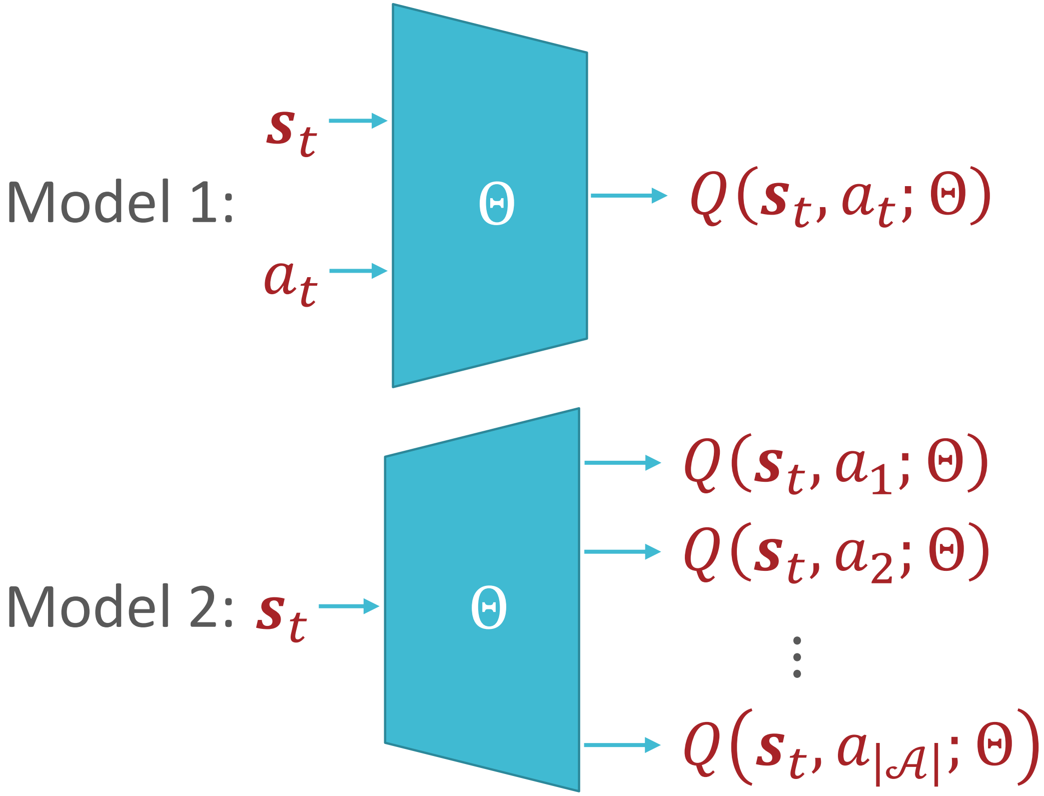
Deep Q-learning: Model

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$
  e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a *differentiable* function that approximates Q



What if instead of optimizing the Q-function, we could optimize the policy directly?

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$
  e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a *differentiable* function that approximates Q



Model 1: $\boldsymbol{s}_t$, a_t → Θ → $Q(\boldsymbol{s}_t, a_t; \Theta)$

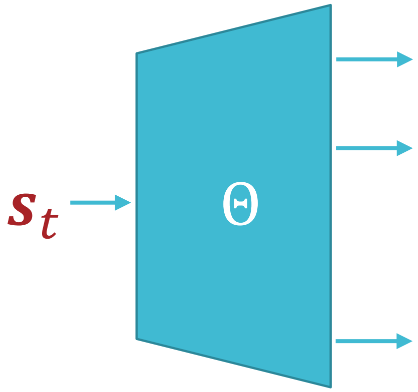
Model 2: $\boldsymbol{s}_t$ → Θ → $Q(\boldsymbol{s}_t, a_1; \Theta)$, $Q(\boldsymbol{s}_t, a_2; \Theta)$, $\vdots$, $Q(\boldsymbol{s}_t, a_{|\mathcal{A}|}; \Theta)$

Parametrized Stochastic Policies

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$ e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a *differentiable* function that specifies a *stochastic* policy π_Θ
- Minimize the negative expected total reward w.r.t. Θ

$$\ell(\Theta) = -\mathbb{E}_{\pi_\Theta}\left[\mathbb{E}_{p(s'|s,a)}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]\right]$$

Model:



$$\boldsymbol{s}_t \rightarrow \Theta$$

$$p(a_1|\boldsymbol{s}_t;\Theta) \coloneqq \pi_\Theta(a_1|\boldsymbol{s}_t)$$

$$p(a_2|\boldsymbol{s}_t;\Theta) \coloneqq \pi_\Theta(a_2|\boldsymbol{s}_t)$$

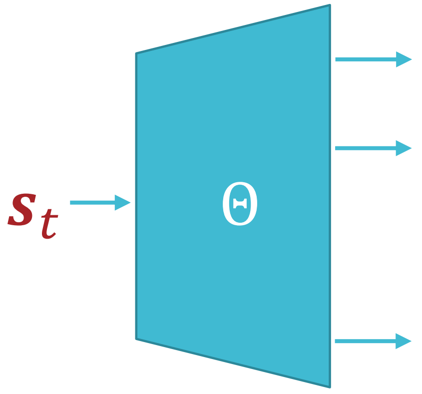
$$\vdots$$

$$p\left(a_{|\mathcal{A}|}\middle|\boldsymbol{s}_t;\Theta\right) \coloneqq \pi_\Theta\left(a_{|\mathcal{A}|}\middle|\boldsymbol{s}_t\right)$$

Okay… but how on earth do we compute the gradient of this thing?

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$ e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a *differentiable* function that specifies a stochastic policy π_Θ
- Minimize the negative expected total reward w.r.t. Θ

$$\ell(\Theta) = -\mathbb{E}_{\pi_\Theta}\left[\mathbb{E}_{p(s'|s,a)}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]\right]$$



Model: $\boldsymbol{s}_t \rightarrow \Theta$

$$p(a_1|\boldsymbol{s}_t;\Theta) \coloneqq \pi_\Theta(a_1|\boldsymbol{s}_t)$$

$$p(a_2|\boldsymbol{s}_t;\Theta) \coloneqq \pi_\Theta(a_2|\boldsymbol{s}_t)$$

$$\vdots$$

$$p\left(a_{|\mathcal{A}|}\middle|\boldsymbol{s}_t;\Theta\right) \coloneqq \pi_\Theta\left(a_{|\mathcal{A}|}\middle|\boldsymbol{s}_t\right)$$

Trajectories

- A *trajectory* $\mathrm{T} = \{\boldsymbol{s}_0, a_0, \boldsymbol{s}_1, a_1, \dots, \boldsymbol{s}_T\}$ is one run of an agent through an MDP ending in a terminal state, $\boldsymbol{s}_T$
- Our stochastic policy and the transition distribution induce a distribution over trajectories

$$p_\Theta(\mathrm{T}) = p(\{\boldsymbol{s}_0, a_0, \boldsymbol{s}_1, a_1, \dots, \boldsymbol{s}_T\})$$

$$= p(\boldsymbol{s}_0) \prod_{t=0}^{T-1} p(s_{t+1} | s_t, a_t)\, \pi_\Theta(a_t | \boldsymbol{s}_t)$$

  - Requires a distribution over initial states $p(\boldsymbol{s}_0)$ e.g., uniform over all states, fixed or deterministic, etc…
- If all runs end at a terminal state, then we can rewrite the negative expected total reward as

$$\ell(\Theta) = -\mathbb{E}_{p_\Theta(\mathrm{T}=\{\boldsymbol{s}_0, a_0, \dots, \boldsymbol{s}_T\})}\left[\sum_{t=0}^{T-1} \gamma^t R(\boldsymbol{s}_t, a_t)\right] \coloneqq -\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T})]$$

Likelihood Ratio Method a.k.a. REINFORCE (Williams, 1992)

$$\nabla_\Theta \ell(\Theta) = \nabla_\Theta\left(-\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T})]\right) = \nabla_\Theta\left(-\int R(\mathrm{T})p_\Theta(\mathrm{T})\,d\mathrm{T}\right)$$

$$= -\int R(\mathrm{T})\nabla_\Theta p_\Theta(\mathrm{T})\,d\mathrm{T}$$

$$= -\int R(\mathrm{T})\nabla_\Theta\left(p(\boldsymbol{s}_0)\prod_{t=0}^{T-1} p(s_{t+1}|s_t,a_t)\,\pi_\Theta(a_t|\boldsymbol{s}_t)\right)d\mathrm{T}$$

- Issues:
  - The transition probabilities $p(s_{t+1}|s_t,a_t)$ are unknown a priori
  - Computing $\nabla_\Theta p_\Theta(\mathrm{T})$ involves taking the gradient of a product

Source: https://citeseerx.ist.psu.edu/doc/10.1.1.129.8871

Likelihood Ratio Method a.k.a. REINFORCE (Williams, 1992)

$$\nabla_\Theta \ell(\Theta) = \nabla_\Theta\left(-\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T})]\right) = \nabla_\Theta\left(-\int R(\mathrm{T})\, p_\Theta(\mathrm{T})\, d\mathrm{T}\right)$$

$$= -\int R(\mathrm{T})\nabla_\Theta p_\Theta(\mathrm{T})\, d\mathrm{T}$$

$$= -\int R(\mathrm{T})\nabla_\Theta\left(p(\boldsymbol{s}_0)\prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\,\pi_\Theta(a_t|\boldsymbol{s}_t)\right) d\mathrm{T}$$

- Insight:

$$\nabla_\Theta p_\Theta(\mathrm{T}) = \frac{p_\Theta(\mathrm{T})}{p_\Theta(\mathrm{T})}\nabla_\Theta p_\Theta(\mathrm{T}) = p_\Theta(\mathrm{T})\nabla_\Theta(\log p_\Theta(\mathrm{T}))$$

$$\log p_\Theta(\mathrm{T}) = \log p(s_0) + \sum_{t=0}^{T-1} \log p(s_{t+1}|s_t, a_t) + \log \pi_\Theta(a_t|\boldsymbol{s}_t)$$

$$\nabla_\Theta(\log p_\Theta(\mathrm{T})) = \sum_{t=0}^{T-1} \nabla_\Theta \log \pi_\Theta(a_t|\boldsymbol{s}_t) \longleftarrow$$

No longer depends on $p(s_{t+1}|s_t, a_t)$!

Likelihood Ratio Method a.k.a. REINFORCE (Williams, 1992)

$$\nabla_\Theta \ell(\Theta) = \nabla_\Theta\left(-\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T})]\right) = \nabla_\Theta\left(-\int R(\mathrm{T})p_\Theta(\mathrm{T})\, d\mathrm{T}\right)$$

$$= -\int R(\mathrm{T})\nabla_\Theta p_\Theta(\mathrm{T})\, d\mathrm{T} = -\int R(\mathrm{T})\nabla_\Theta(\log p_\Theta(\mathrm{T}))p_\Theta(\mathrm{T})d\mathrm{T}$$

$$= -\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T})\nabla_\Theta(\log p_\Theta(\mathrm{T}))]$$

$$\approx -\frac{1}{N}\sum_{n=1}^{N} R\left(\mathrm{T}^{(n)}\right)\nabla_\Theta\left(\log p_\Theta\left(\mathrm{T}^{(n)}\right)\right)$$

(where $\mathrm{T}^{(n)} = \left\{\boldsymbol{s}_0^{(n)}, a_0^{(n)}, \boldsymbol{s}_1^{(n)}, a_1^{(n)}, \dots, \boldsymbol{s}_{T^{(n)}}^{(n)}\right\}$ is a sampled trajectory)

$$= -\frac{1}{N}\sum_{n=1}^{N}\left(\sum_{t=0}^{T^{(n)}-1}\gamma^t R\left(\boldsymbol{s}_t^{(n)}, a_t^{(n)}\right)\right)\left(\sum_{t=0}^{T^{(n)}-1}\nabla_\Theta \log \pi_\Theta\left(a_t^{(n)} \middle| \boldsymbol{s}_t^{(n)}\right)\right)$$

Likelihood Ratio Method a.k.a. REINFORCE (Williams, 1992)

- Practical considerations:
  - Sampled trajectories/rewards can be highly variable, which leads to unstable estimates of the expectation
    - Can compare sampled rewards against a *baseline* by subtracting some constant value from $R(\mathrm{T})$ (Peters and Schaal, 2008)
  - Policy gradient methods are *on-policy*: they require using the current (potentially bad) policy to sample (a lot of) trajectories…
    - Can use a surrogate policy and adjust gradient computation via *importance sampling*
  - Not compatible with deterministic policies (would require knowledge of the transition probabilities)

Key Takeaways

- We can use (deep) Q-learning when the reward/transition functions are unknown and/or when the state/action spaces are too large to be modelled directly
    - Also guaranteed to converge under certain assumptions
    - Experience replay can help address non-i.i.d. samples
- If our policy is parametrized, we can directly optimize the parameters using the policy gradient
    - The gradient can be expressed in a tractable way via the likelihood ratio method
    - We can approximate the gradient by sampling trajectories