10-701: Introduction to Machine Learning

Lecture 17 – Reinforcement Learning: Q-learning & Deep RL

Hoda Heidari

* Slides adopted from F24 offering of 10701 by Henry Chai.

Recall: Markov Decision Process (MDP)

- Assume the following model for our data:

1. Start in some initial state s_0
2. For time step t:
    1. Agent observes state s_t
    2. Agent takes action $a_t = \pi(s_t)$ $\pi : S \rightarrow \mathcal{A}$
    3. Agent receives reward $r_t \sim p(r \mid s_t, a_t)$
    4. Agent transitions to state $s_{t+1} \sim p(s' \mid s_t, a_t)$
3. Total reward is $\sum_{t=0}^{\infty} \gamma^t r_t$ $0 \leq \gamma < 1$ discount

- MDPs make the *Markov assumption*: the reward and next state only depend on the current state and action.

Recall: Value Function

- $V^\pi(s) = \mathbb{E}[$discounted total reward of starting in state s and executing policy π forever$]$

$$= \mathbb{E}[R(s_0, \pi(s_0)) + \gamma R(s_1, \pi(s_1)) + \gamma^2 R(s_2, \pi(s_2)) + \cdots \mid s_0 = s]$$

$$= R(s, \pi(s)) + \gamma \mathbb{E}[R(s_1, \pi(s_1)) + \gamma R(s_2, \pi(s_2)) + \ldots \mid s_0 = s]$$

$$= R(s, \pi(s)) + \gamma \sum_{s_1 \in \mathcal{S}} p(s_1 \mid s, \pi(s)) \big(R(s_1, \pi(s_1)) + \gamma \mathbb{E}[R(s_2, \pi(s_2)) + \cdots \mid s_1]\big)$$

Recursive Formula

$$\underbrace{V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s_1 \in \mathcal{S}} p(s_1 \mid s, \pi(s)) V^\pi(s_1)}_{\text{Bellman equations}} \quad \forall s \in S$$

Recall: Optimality

- Optimal value function:

Recursive formula

$$\underbrace{V^*(s)} = \underbrace{\max_{a \in \mathcal{A}}} \overbrace{\underbrace{R(s,a)}_{\text{immediate}} + \gamma \sum_{s' \in \mathcal{S}} \underbrace{p(s' \mid s,a) V^*(s')}_{\text{future}}}$$

  - System of $|\mathcal{S}|$ equations and $|\mathcal{S}|$ variables

- Optimal policy:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \boxed{\underbrace{R(s,a)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')}_{\text{(Discounted) Future reward}}}$$

Greedy policy w.r.t V^*

$F : \mathbb{R}^n \to \mathbb{R}^n$ is fixed point : $F(x) = x$ $\quad F(x_1, x_2, \ldots, x_n) = (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x}))$

Fixed Point Iteration

- Iterative method for computing fixed points of a fn
- Given some equations and initial values

$$\begin{cases} x_1 = f_1(x_1, \ldots, x_n) \\ \vdots \\ x_n = f_n(x_1, \ldots, x_n) \end{cases}$$

- Initialize $\quad x_1^{(0)}, \ldots, x_n^{(0)}$

- While not converged, do

next estimate

$$\begin{cases} x_1^{(t+1)} \leftarrow f_1\left(\underline{x_1^{(t)}, \ldots, x_n^{(t)}}\right) \\ \vdots \\ x_n^{(t+1)} \leftarrow f_n\left(\underline{x_1^{(t)}, \ldots, x_n^{(t)}}\right) \end{cases}$$

current estimate

Fixed Point Iteration

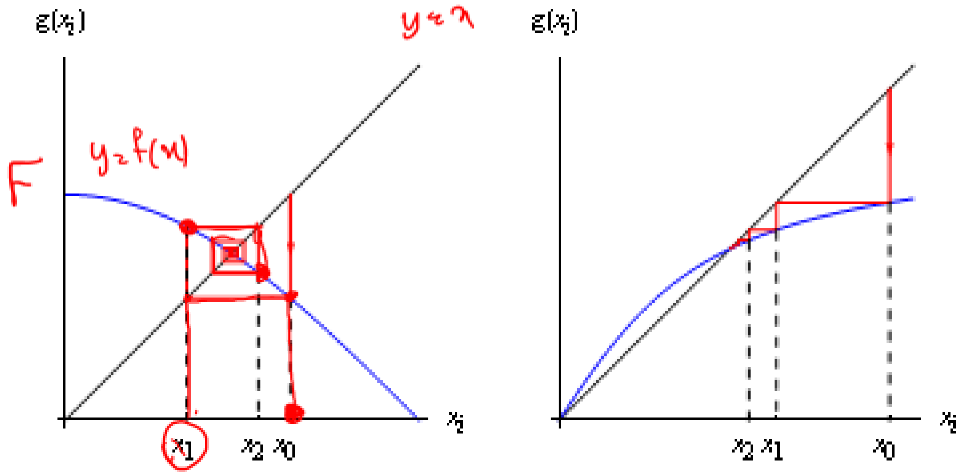


Diagram from: https://alexgs.co.uk/maths/unit-3-pure/3.8-numerical-methods/fixed-point-iteration

Say $S = \{s_1, \dots, s_n\}$

Optimal Values as Fixed Points

- Optimal value function:

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

$$V^*(s_1) = \max_{a \in \mathcal{A}} \quad (s_1, \dots, s_n)$$

$$\vdots$$

$$V^*(s_n) = \max_{a \in \mathcal{A}} \quad (s_1, \dots, s_n)$$

$$V^*(s_1) = f_1(V^*(s_1), V^*(s_2), \dots, V^*(s_n))$$

$$\vdots$$

$$V^*(s_n) = f_n(V^*(s_1), V^*(s_2), \dots, V^*(s_n))$$

F

Value Iteration

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

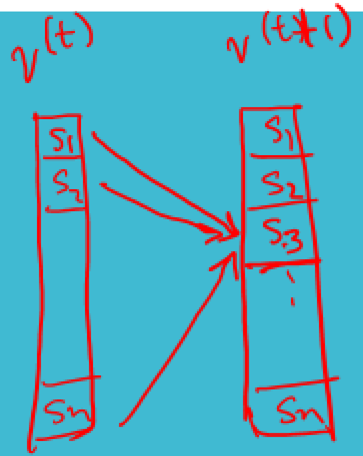
$$\underbrace{V^{(t+1)}(s)}_{\text{new estimate}} \leftarrow \underbrace{\max_{a \in \mathcal{A}} \overbrace{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \overset{\text{prev. estimates}}{\underbrace{V^{(t)}(s')}}}}_{Q(s,a)}$$

  - $t = t + 1$
- For $s \in \mathcal{S}$

$$\underbrace{\pi^*(s)} \leftarrow \operatorname*{argmax}_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) V^{(t)}(s')$$

- Return π^*

Greedy policy



Synchronous Value Iteration

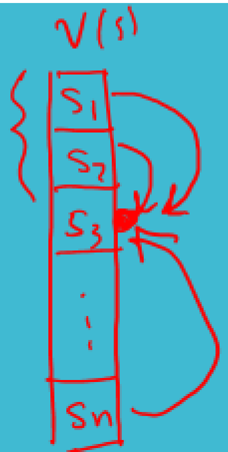
- Inputs: $R(s,a), p(s' \mid s,a)$
- Initialize $V^{(0)}(s) = 0 \ \forall \ s \in \mathcal{S}$ (or randomly) and set $t = 0$
- While not converged, do:
  - For $s \in \mathcal{S}$
    - For $a \in \mathcal{A}$

      $$Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(t)}(s')$$

    - $V^{(t+1)}(s) \leftarrow \max_{a \in \mathcal{A}} Q(s,a)$
  - $t = t + 1$
- For $s \in \mathcal{S}$

  $$\pi^*(s) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}} \ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(t)}(s')$$

- Return π^*



Asynchronous Value Iteration

- Inputs: $R(s,a)$, $p(s' \mid s,a)$
- Initialize $V^{(0)}(s) = 0 \ \forall\, s \in \mathcal{S}$ (or randomly)
- 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 $s \in \mathcal{S}$

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

- Return π^*

Value Iteration Theory

- **Theorem 1**: Value function convergence

  V will converge to V^* if each state is "visited" infinitely often (Bertsekas, 1989)

- **Theorem 2**: Convergence criterion

  if $\max_{s \in \mathcal{S}} \left| V^{(t+1)}(s) - V^{(t)}(s) \right| < \epsilon$,

  then $\max_{s \in \mathcal{S}} \left| V^{(t+1)}(s) - V^*(s) \right| < \frac{2\epsilon\gamma}{1-\gamma}$ (Williams & Baird, 1993)

- **Theorem 3**: Policy convergence

  The "greedy" policy, $\pi(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(s, a)$, converges to the optimal π^* in a finite number of iterations, often before the value function has converged! (Bertsekas, 1987)

Bellman Optimality Characterization

- A policy π is optimal if and only if it is greedy (optimal) w.r.t. its own value function V^{π}.
- Proof:
  - **(⇒) If π is optimal, then it must be greedy w.r.t V^{π}.** If π were not greedy at some state, there would exist an action with strictly higher expected return ⇒ we could improve the policy ⇒ π was not optimal. Contradiction.
  - **(⇐) If π is greedy w.r.t V^{π} ,then π is optimal.** Greedy w.r.t its own value solves the Bellman *optimality* fixed point, which is known to have a unique solution. So $V^{\pi} = V^{*}$ and π is optimal.

Policy Iteration

- Inputs: $R(s,a)$, $p(s' \mid s,a)$
- Initialize π randomly
- While not converged, do:
  - Solve the Bellman equations defined by policy π

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

  - Update π

Greedy policy w.r.t V^{π}

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

- Return π

Policy Iteration Theory

- In policy iteration, the policy improves in each iteration.
- Given finite state and action spaces, there are finitely many possible policies
  - Thus, the number of iterations needed to converge is bounded!
- Value iteration takes $O(|\mathcal{S}|^2|\mathcal{A}|)$ time / iteration
- Policy iteration takes $O(|\mathcal{S}|^2|\mathcal{A}| + |\mathcal{S}|^3)$ time / iteration
  - However, empirically policy iteration requires fewer iterations to converge

Key Takeaways

- In reinforcement learning, we assume our data comes from a Markov decision process
- The goal is to compute an optimal policy or function that maps states to actions
- Value function can be defined in terms of values of all other states; this is called the Bellman equations
- If the reward and transition functions are known, we can solve for the optimal policy (and value function) using value or policy iteration
    - Both algorithms are instances of fixed point iteration and are guaranteed to converge (under some assumptions)

Two big Q's

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

   $\mathcal{S}, \mathcal{A}$ known

   $r(s,a)$ & $p(\cdot \mid s,a)$ unknown

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

Value Iteration

- Inputs: $R(s,a), p(s' \mid s,a), \gamma$
- 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 $s \in \mathcal{S}$

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

- Return π^*

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

Recursive Formula

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

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

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

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

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

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

Q-learning: Overview

$Q^*(s,a) \quad \forall s \in S, \; a \in \mathcal{A}$

- Q-learning
  - Learn the optimal Q-table without requiring a model of the world, that is rewards and transition probabilities (**model-free**).
  - Utilizes *Monte Carlo sampling*:
    it approximates the expectation simply by taking more and more samples over time.
  - Is **off-policy** because it estimates the value of the greedy optimal policy while following a potentially different policy.

$Q^*(s,a)$ 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]

deterministic transition

$$= R(s, a) + \gamma V^*\big(\underbrace{\delta(s, a)}_{s'}\big)$$

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

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

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

- 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 $\epsilon \in [0, 1]$ exploration vs exploitation
- 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 $1 - \epsilon$, take the greedy action

    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s, a')$$

    Otherwise, with probability ϵ, 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 $1 - \epsilon$, take the greedy action
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s, a')$$
    Otherwise, with probability ϵ, 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 $Q(s, a)$:
    $$Q(s, a) \leftarrow (1 - \alpha)\underbrace{Q(s, a)}_{\text{Current value}} + \alpha\underbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}_{\text{Update w/ deterministic transitions}}$$

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 $Q(s,a)$:

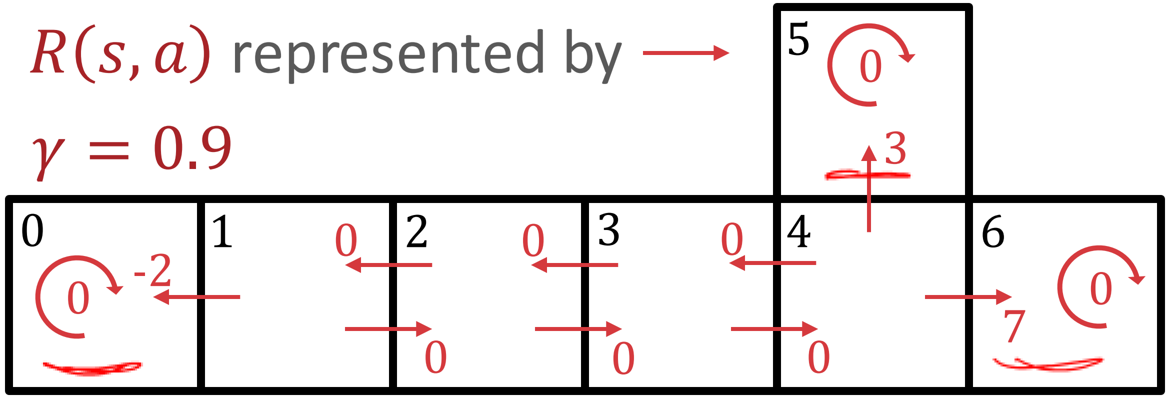
learning rate

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

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

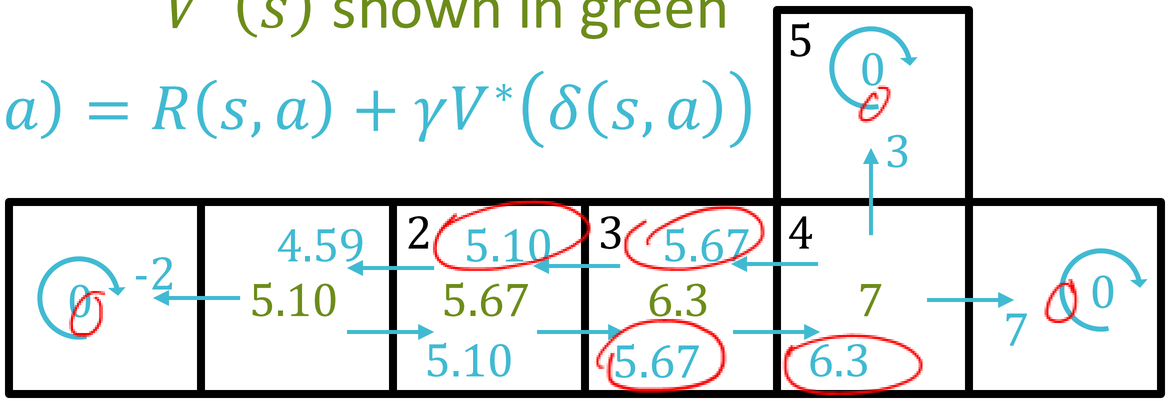
$R(s,a)$ represented by →

$\gamma = 0.9$



$V^*(s)$ shown in green

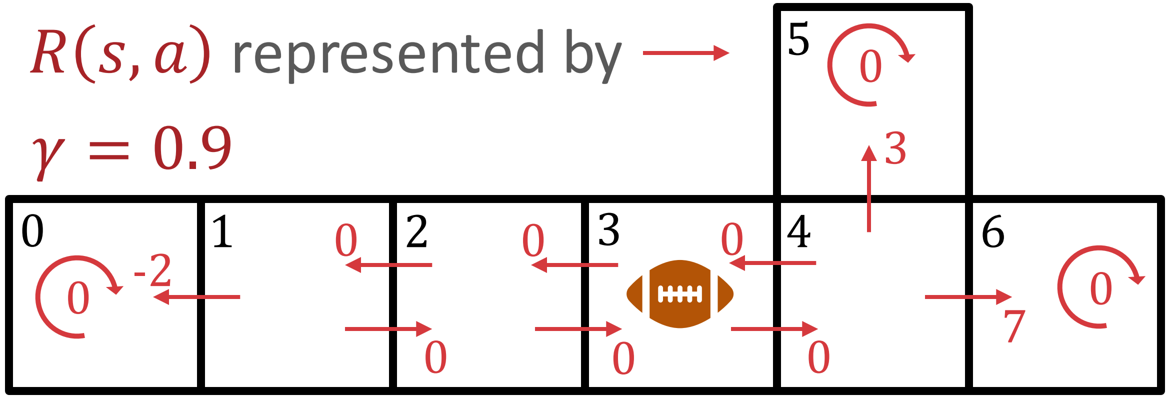
$Q^*(s,a) = R(s,a) + \gamma V^*(\delta(s,a))$



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

$R(s, a)$ represented by →

$\gamma = 0.9$

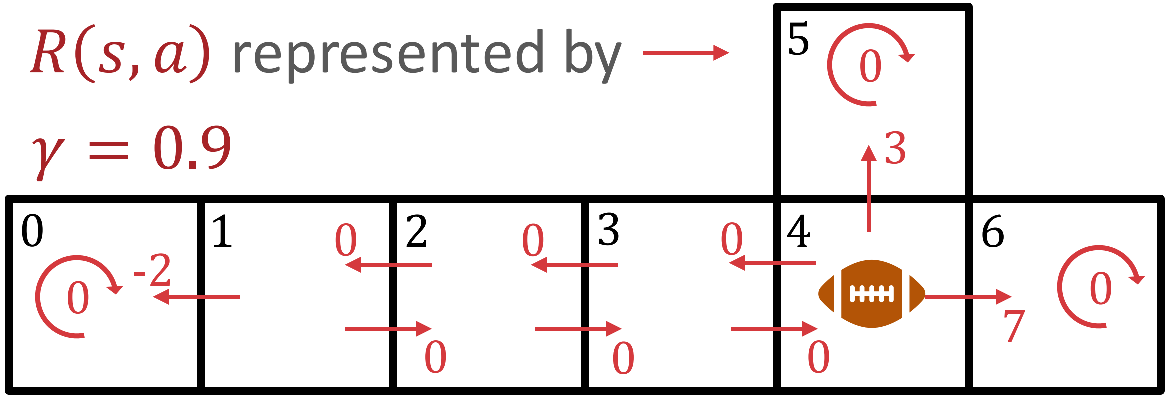


$Q(s, a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

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

$R(s, a)$ represented by $\longrightarrow$

$\gamma = 0.9$



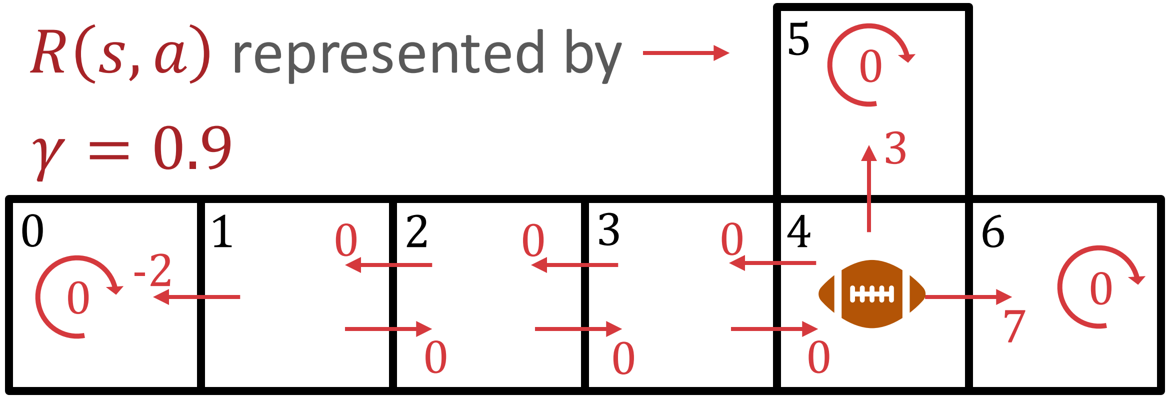
$$\underbrace{Q(\underbrace{3}, \underbrace{\rightarrow})} \leftarrow \underbrace{0} + \underbrace{(0.9)} \underbrace{\max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \circlearrowright\}} Q(4, a')} = 0$$

$Q(s,a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
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 →

$\gamma = 0.9$

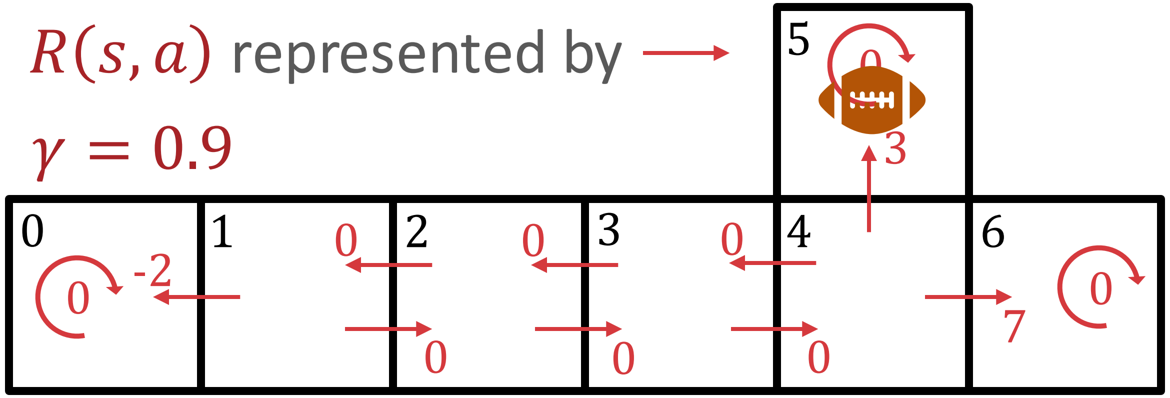


$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

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

$R(s,a)$ represented by $\longrightarrow$

$\gamma = 0.9$



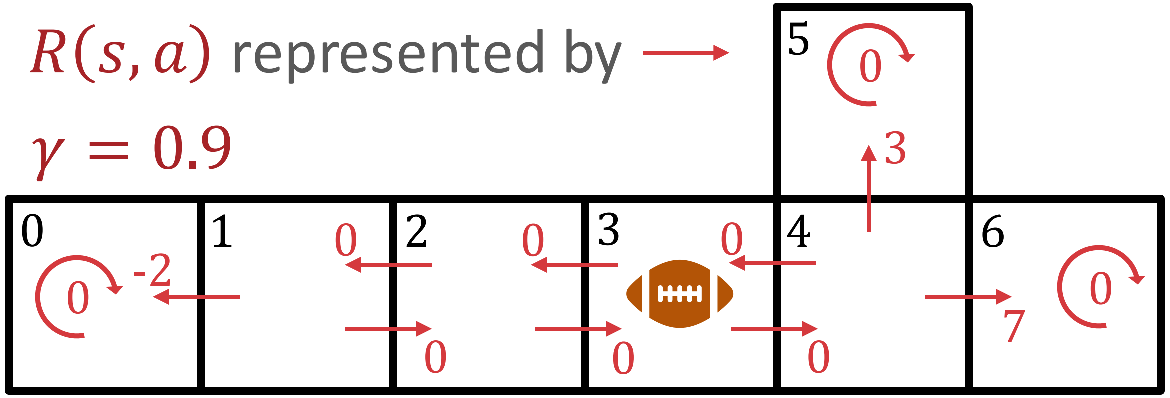
$$Q(4,\uparrow) \leftarrow 3 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(5,a') = 3$$

$Q(s,a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
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 →

$\gamma = 0.9$



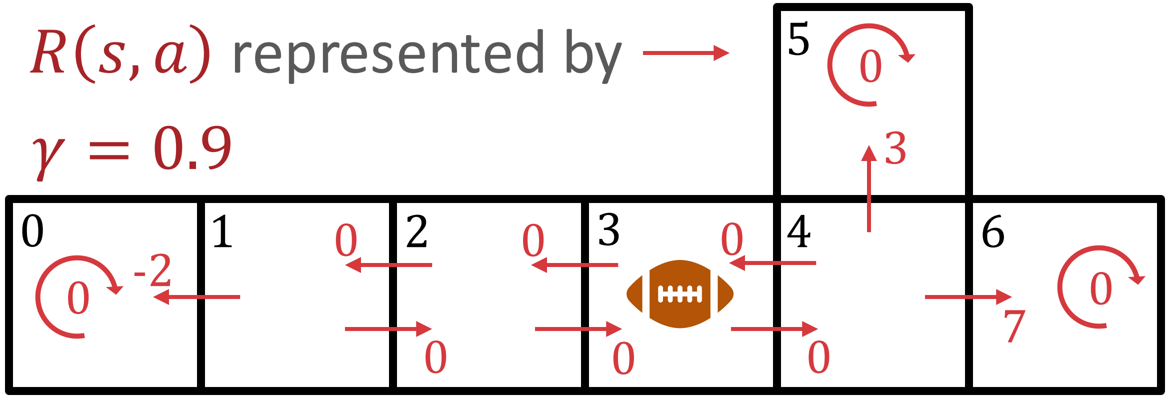
$$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \circlearrowright\}} Q(4, a') = 2.7$$

$Q(s,a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
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 $\longrightarrow$

$\gamma = 0.9$



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

$Q(s,a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
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 (initialized to 0)

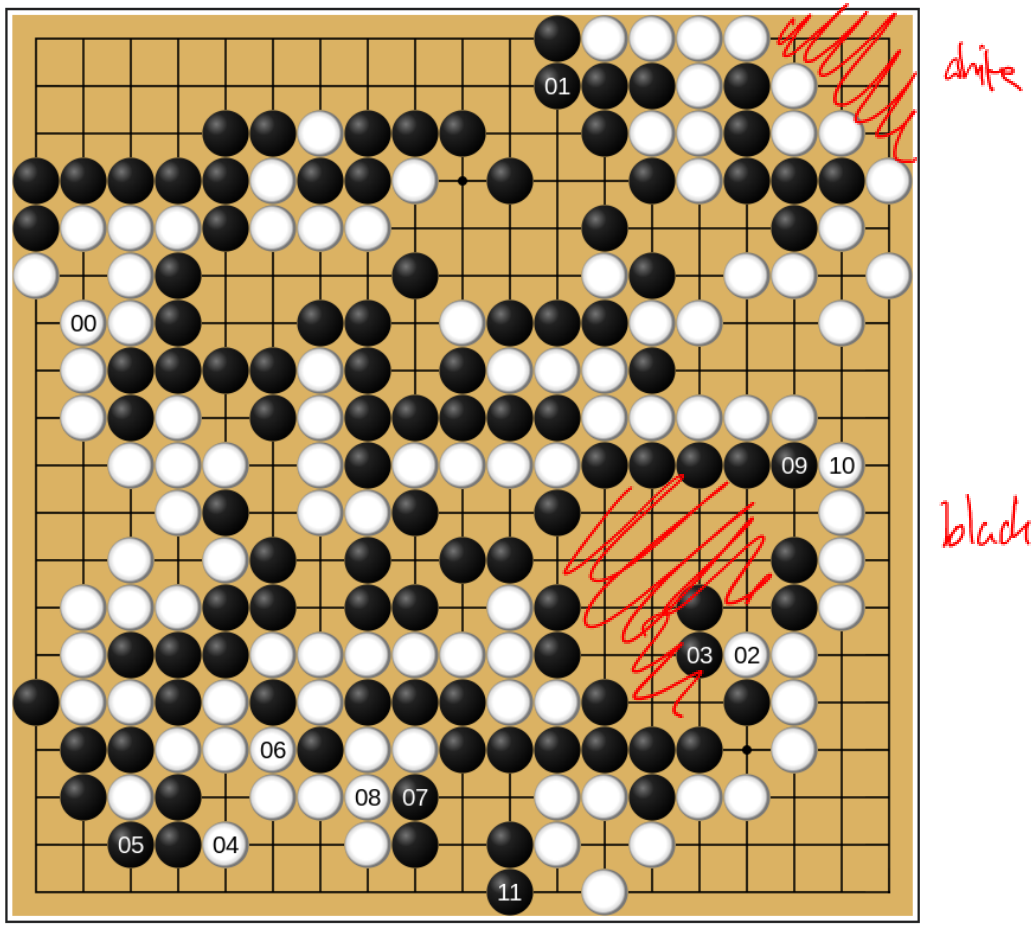
Learning $Q^*(s, a)$: Convergence

- 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. $\underbrace{\sum_{t=0}^{\infty} \alpha_t = \infty}$ and $\underbrace{\sum_{t=0}^{\infty} \alpha_t^2 < \infty}$ e.g., $\underbrace{\alpha_t = 1/_{t+1}}$

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)

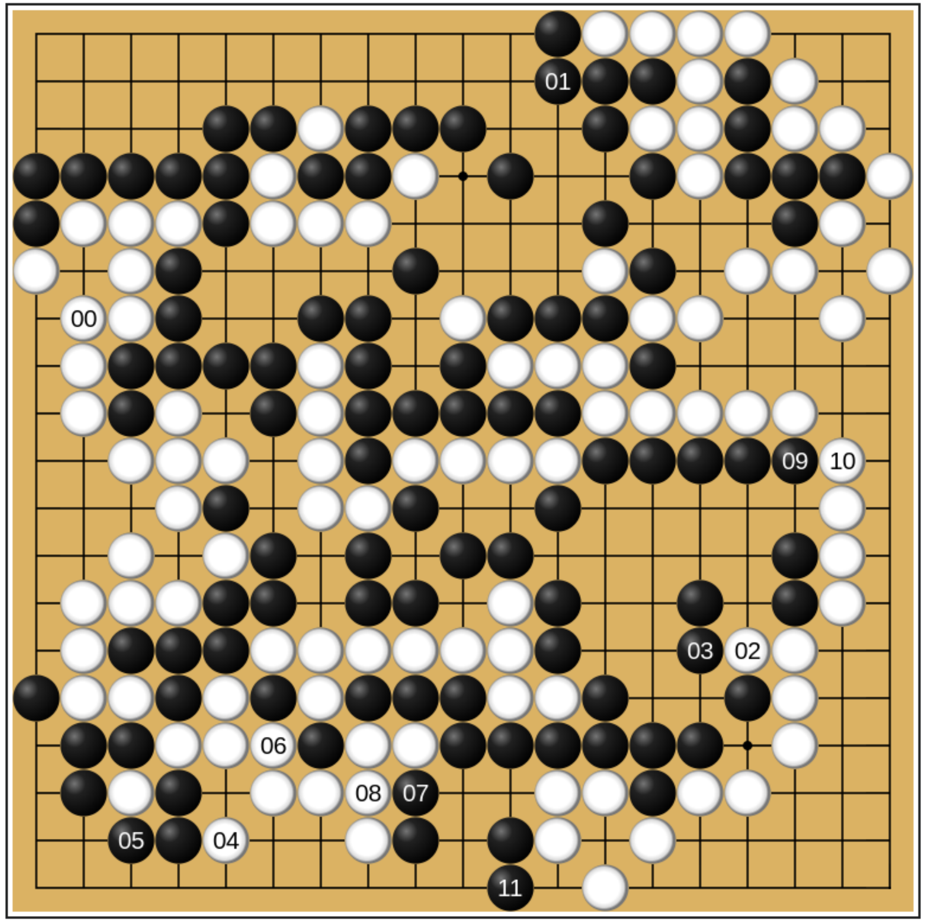


Source: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol

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

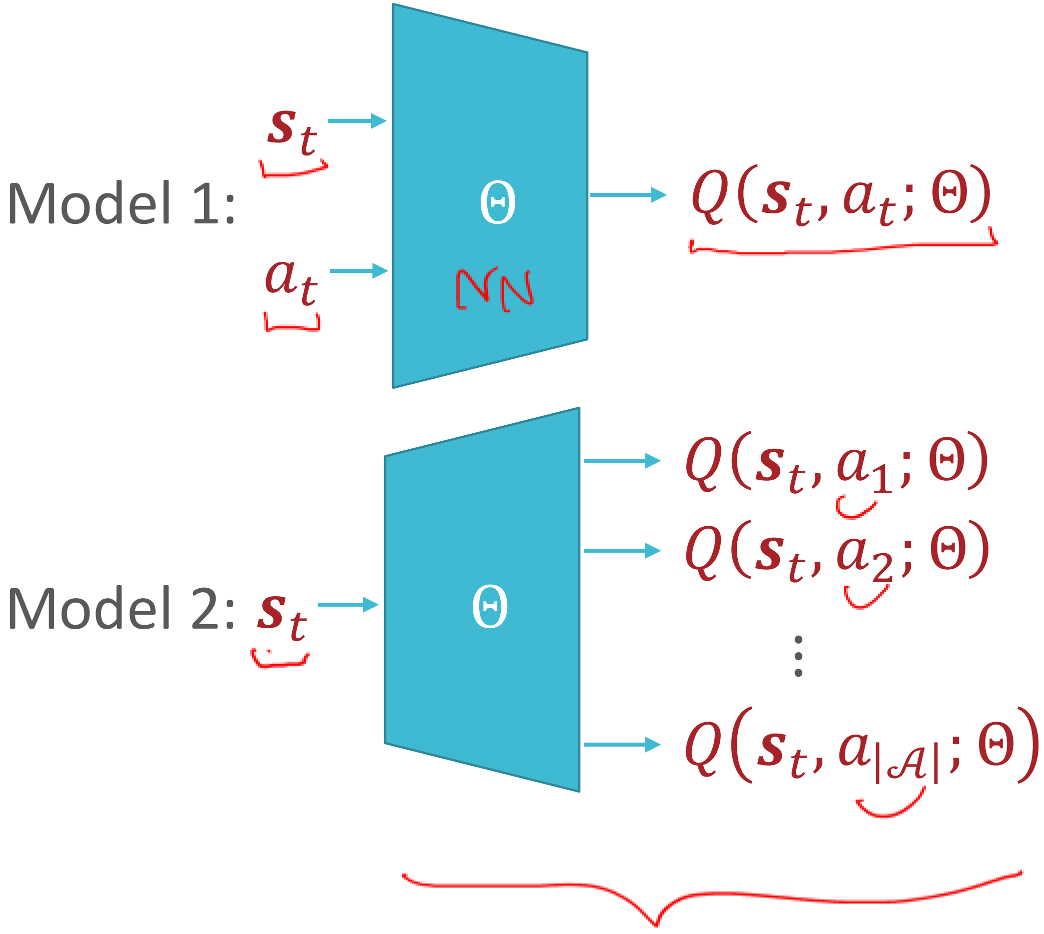
$Q^*(s, a) \; \forall s \in S, a \in A$

parameter

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

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

1. Use stochastic gradient descent: just consider one state-action pair in each iteration

2. Use temporal difference learning:
   - Given current parameters $\Theta^{(t)}$ 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 $\Theta^{(t+1)}$ such that $Q\left(s,a;\Theta^{(t+1)}\right) \approx y$

     $$\ell\left(\Theta^{(t)}, \Theta^{(t+1)}\right) = \left(y - Q\left(s,a;\Theta^{(t+1)}\right)\right)^2$$

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 $t = 0, 1, 2, \ \ldots$
  - 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)

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