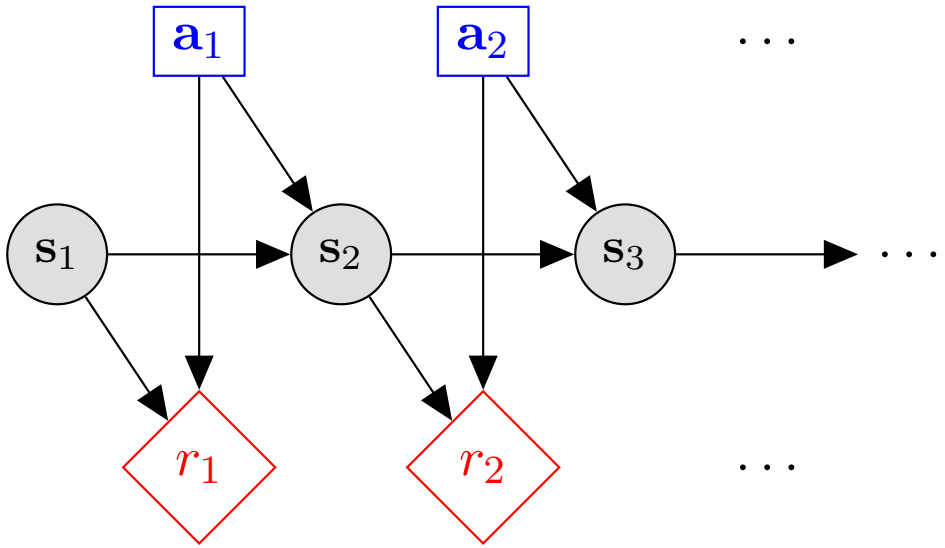
Machine Learning (CS 181): 21. Reinforcement Learning

David C. Parkes and Sasha Rush

Spring 2017

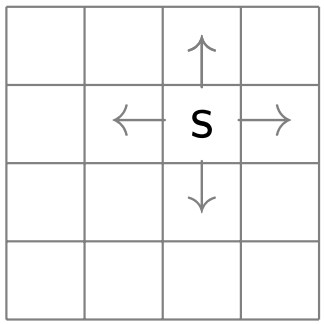
Contents

Markov Decision Process



S	States
A	Actions
$r : S \times A \mapsto \mathbb{R}$	Reward Function
$p(s' \mid s, a)$	Transition Model

Running Illustration: MDP on Gridworld



S	Location of the grid (x_1, x_2)
A	Local movements $\leftarrow, \rightarrow, \uparrow, \downarrow$
$r : S \times A \mapsto \mathbb{R}$	Reward function, e.g. make it to goal
$p(s' \mid s, a)$	Transition model, e.g deterministic or slippages

Policy Evaluation

- Policy function: $\pi : S \to A$
- Value Function: expected discounted reward

$$V^{\pi}(s) = \underbrace{r(s, \pi(s))}_{\text{reward now}} + \underbrace{\gamma \sum_{s' \in S} p(s' \mid s, \pi(s)) V^{\pi}(s)}_{\text{expected, discounted future reward}} \tag{1}$$

- Q-Function: expected discounted reward of state and action (new)

$$Q^{\pi}(s, a) = \underbrace{r(s, a)}_{\text{reward now}} + \underbrace{\gamma \sum_{s' \in S} p(s' \mid s, a) Q^{\pi}(s', \pi(a))}_{\text{expected, discounted future reward}} \tag{2}$$

- Can compute Value function from Q-Function

$$V^{\pi}(s) = Q^{\pi}(s, \pi(a))$$

Policy Evaluation

- Policy function: $\pi : S \to A$
- Value Function: expected discounted reward

$$V^{\pi}(s) = \underbrace{r(s, \pi(s))}_{\text{reward now}} + \underbrace{\gamma \sum_{s' \in S} p(s' \mid s, \pi(s)) V^{\pi}(s)}_{\text{expected, discounted future reward}} \tag{1}$$

- Q-Function: expected discounted reward of state and action (new)

$$Q^{\pi}(s, a) = \underbrace{r(s, a)}_{\text{reward now}} + \underbrace{\gamma \sum_{s' \in S} p(s' \mid s, a) Q^{\pi}(s', \pi(a))}_{\text{expected, discounted future reward}} \tag{2}$$

- Can compute Value function from Q-Function

$$V^{\pi}(s) = Q^{\pi}(s, \pi(a))$$

Working with MDPs

An MDP is a general probabilistic framework, and can be utilized in many different scenarios.

- Planning:
  - Full access to the MDP, compute an optimal policy.
  - "How do I act in a known world?"
- Policy Evaluation:
  - Full access to the MDP, compute the 'value' of a fixed policy.
  - "How will this plan perform under uncertainty?"
- Reinforcement Learning (today):
  - Limited access to the MDP.
  - "Can I learn to act in an uncertain world?"

Working with MDPs

An MDP is a general probabilistic framework, and can be utilized in many different scenarios.

- Planning:
  - Full access to the MDP, compute an optimal policy.
  - "How do I act in a known world?"
- Policy Evaluation:
  - Full access to the MDP, compute the 'value' of a fixed policy.
  - "How will this plan perform under uncertainty?"
- Reinforcement Learning (today):
  - Limited access to the MDP.
  - "Can I learn to act in an uncertain world?"

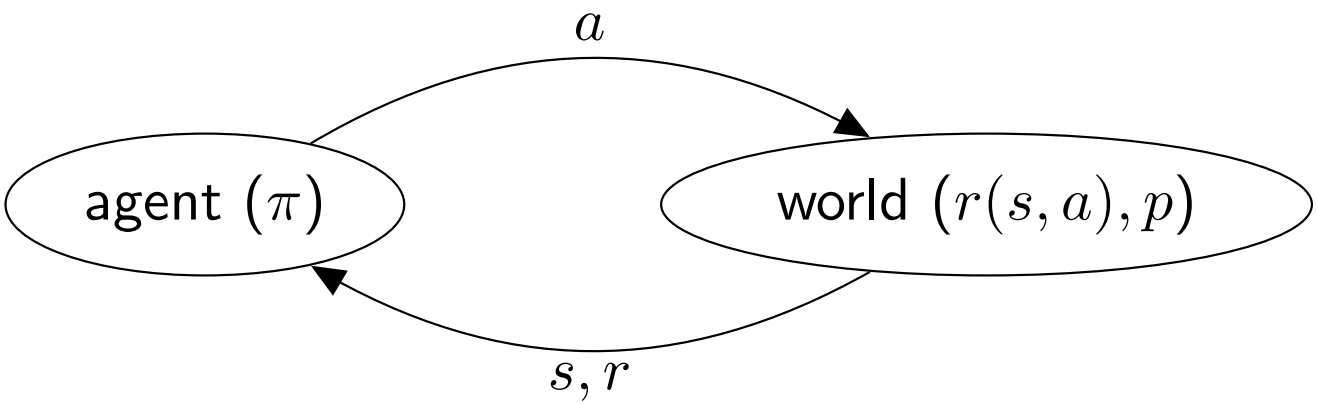
Working with MDPs

An MDP is a general probabilistic framework, and can be utilized in many different scenarios.

- Planning:
  - Full access to the MDP, compute an optimal policy.
  - "How do I act in a known world?"
- Policy Evaluation:
  - Full access to the MDP, compute the 'value' of a fixed policy.
  - "How will this plan perform under uncertainty?"
- Reinforcement Learning (today):
  - Limited access to the MDP.
  - "Can I learn to act in an uncertain world?"

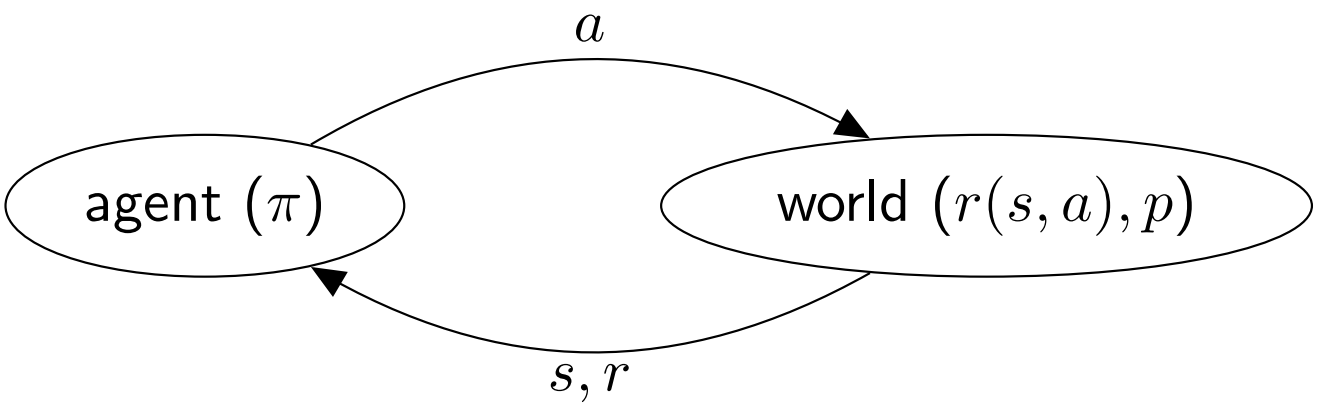
Contents

Reinforcement Learning



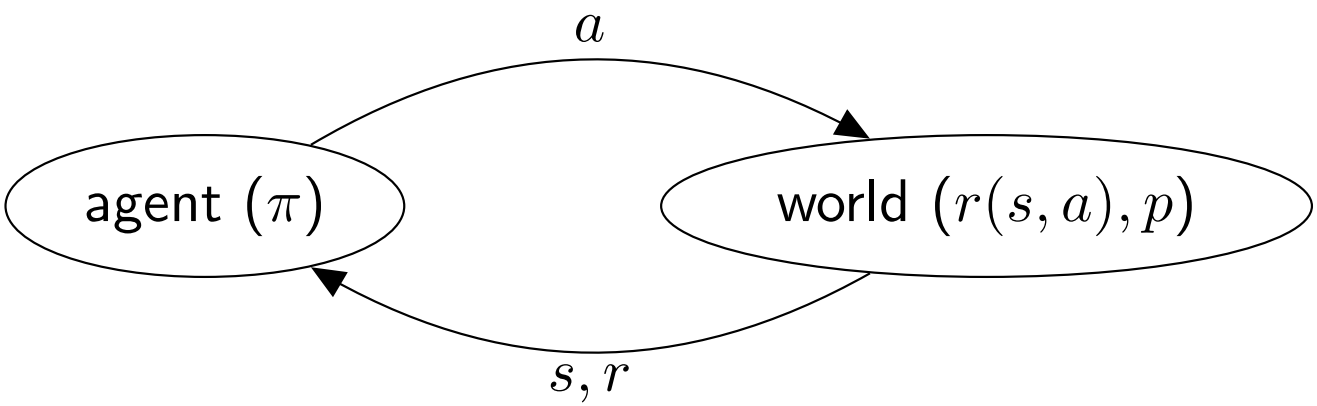
- Agent knows current state s takes actions a, and gets reward r.
- **No access** to reward model $r(s,a)$ or transition model $p(s'|s,a)$, only see outcome reward r and next state s'
- Very challenging problem to learn π while uncertain about model of the world, (contrast with last class).

RL Example: Medical Diagnostics



- States: patient symptoms
- Actions: prescribe drugs, change diet, do nothing, ...
- Reward: $+5$ if health improves, -1 if costly, ...
- Transition model: update of symptoms health based on actions

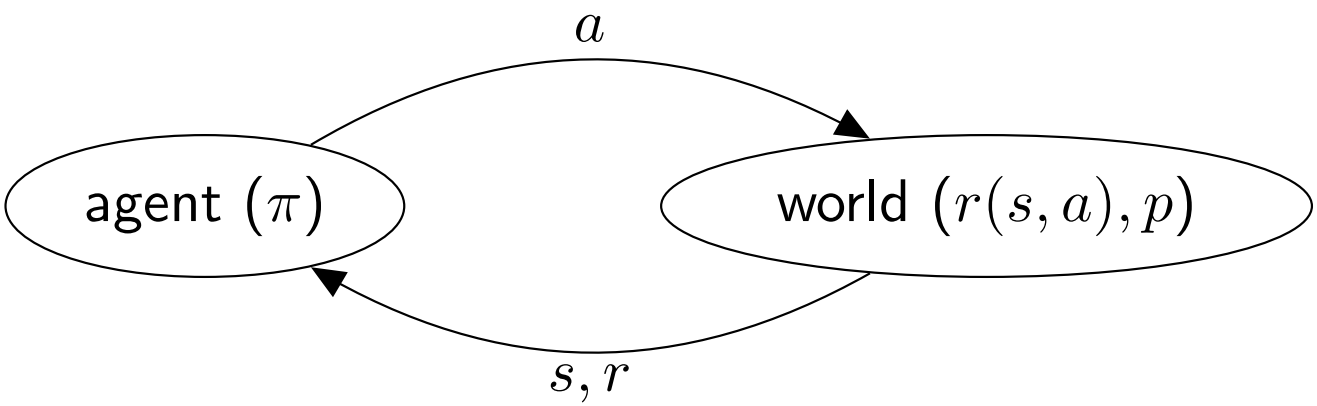
RL Example: Ad Market



- States: current knowledge of user's preferences
- Actions: show particular ad ...
- Reward: $+100$ if user clicks, -1 if otherwise, ...
- Transition model: user remains on site or leaves

Note: transition model is probabilistic in both planning and RL. The difference is that in planning we **know** the probabilities.

RL Example: Ad Market



- States: current knowledge of user's preferences
- Actions: show particular ad ...
- Reward: $+100$ if user clicks, -1 if otherwise, ...
- Transition model: user remains on site or leaves

Note: transition model is probabilistic in both planning and RL. The difference is that in planning we **know** the probabilities.

Types of RL

- Model-Based RL:
  - Estimate world models $r(s, a; \mathbf{w})$ and $p(s'|s, a; \mathbf{w})$.
  - Utilize planning (value or policy iteration) to develop policy π.
- Model-Free (our focus):
  - Directly learn the policy π from samples of the world.

When might you prefer one over the other?

Types of RL

- Model-Based RL:
  - Estimate world models $r(s, a; \mathbf{w})$ and $p(s'|s, a; \mathbf{w})$.
  - Utilize planning (value or policy iteration) to develop policy π.
- Model-Free (our focus):
  - Directly learn the policy π from samples of the world.

When might you prefer one over the other?

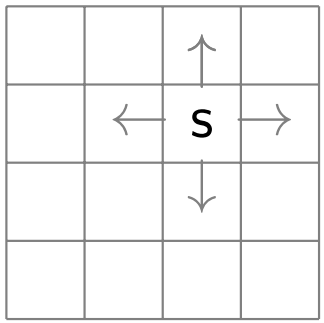
Types of RL

- Model-Based RL:
  - Estimate world models $r(s, a; \mathbf{w})$ and $p(s'|s, a; \mathbf{w})$.
  - Utilize planning (value or policy iteration) to develop policy π.
- Model-Free (our focus):
  - Directly learn the policy π from samples of the world.

When might you prefer one over the other?

Reinforcement Learning Setup

Learning is performed **online**, we learn as we interact with the world.



Contrast with supervised learning:

- No train/test, reward accumulated over interactions.
- Not learning from fixed data, more information acquired as we go.
- Able to influence the training distribution by action decisions.

High-Level Challenges of RL

1. **Exploration/Exploitation**: Trade-off between taking actions with high expected future reward [exploitation], and taking less explored actions to improve estimation [exploration].
2. **Asynchronous Samples**: In previous approaches we had a fixed set of samples, in RL samples come in on the fly based on interaction with the world.

High-Level Challenges of RL

1. **Exploration/Exploitation**: Trade-off between taking actions with high expected future reward [exploitation], and taking less explored actions to improve estimation [exploration].
2. **Asynchronous Samples**: In previous approaches we had a fixed set of samples, in RL samples come in on the fly based on interaction with the world.

Contents

Review: Bellman equations

The planning problem for an MDP is:

$$\pi^* \in \arg\max_{\pi} V^{\pi}(s).$$

(exists a solution that is optimal for every state s).

Definition (Bellman equations)

For an optimal policy π^*, we have

$$V^*(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s, a) V^*(s') \right], \quad \forall s \tag{3}$$

Alternate Form: Bellman equations

Alternate form of the Bellman operator using the Q-Function using:

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

Definition (Bellman equations)

For an optimal policy π^*, we have

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s' \mid s, a) \max_{a' \in A} \left[Q^*(s', a') \right], \quad \forall s, a \tag{4}$$

Model-Free Estimation Strategy

Observe:

- $Q^*(s, a)$ is just a function from $S \times A \mapsto \mathbb{R}$
- If we had Q^* then $\pi^*(s) = \arg\max_a Q^\pi(s, a)$

Strategy:

- Learn the value of a Q-function to estimate Q^*
- Use a parameter table, $\mathbf{w} \in \mathbb{R}^{|S||A|}$:

$$Q(s, a; \mathbf{w}) \triangleq w_{s,a}$$

Model-Free Estimation Strategy

Observe:

- $Q^*(s, a)$ is just a function from $S \times A \mapsto \mathbb{R}$
- If we had Q^* then $\pi^*(s) = \arg\max_a Q^\pi(s, a)$

Strategy:

- Learn the value of a Q-function to estimate Q^*
- Use a parameter table, $\mathbf{w} \in \mathbb{R}^{|S||A|}$:

$$Q(s, a; \mathbf{w}) \triangleq w_{s,a}$$

Learning Objective

But how do we learn the Q-function without supervision.

- Use **Bellman** equations.
- We know that upon convergence, we should have π.

$$Q(s, a; \mathbf{w}) = r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) \max_{a'} Q(s', a'; \mathbf{w})$$

- Enforce this directly as a **loss function**,

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{s,a} ||Q(s, a; \mathbf{w}) - [r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) \max_{a'} Q(s', a'; \mathbf{w})]||_2^2$$

- "Find a Q for which the Bellman equation holds."

Learning Objective

But how do we learn the Q-function without supervision.

- Use **Bellman** equations.
- We know that upon convergence, we should have π.

$$Q(s, a; \mathbf{w}) = r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) \max_{a'} Q(s', a'; \mathbf{w})$$

- Enforce this directly as a **loss function**,

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{s,a} ||Q(s, a; \mathbf{w}) - [r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) \max_{a'} Q(s', a'; \mathbf{w})]||_2^2$$

- "Find a Q for which the Bellman equation holds."

Learning Objective

But how do we learn the Q-function without supervision.

- Use **Bellman** equations.
- We know that upon convergence, we should have π.

$$Q(s, a; \mathbf{w}) = r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) \max_{a'} Q(s', a'; \mathbf{w})$$

- Enforce this directly as a **loss function**,

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{s,a} ||Q(s, a; \mathbf{w}) - [r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) \max_{a'} Q(s', a'; \mathbf{w})]||_2^2$$

- "Find a Q for which the Bellman equation holds."

Sketch of SGD Approach

Loss at a sampled state-action pair s, a

$$\mathcal{L}^{(s,a)}(\mathbf{w}) = \frac{1}{2}||Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q(s',a';\mathbf{w})]||_2^2$$

Take the gradient of the sampled loss (assuming $s' \neq s$),

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} = Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q(s',a';\mathbf{w})]$$

Approximate this by further sampling the next state to approximate the expectation $s' \sim p(s' \mid s,a)$,

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} \approx Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \max_{a'} Q(s',a';\mathbf{w})]$$

Sketch of SGD Approach

Loss at a sampled state-action pair s, a

$$\mathcal{L}^{(s,a)}(\mathbf{w}) = \frac{1}{2}||Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q(s',a';\mathbf{w})]||_2^2$$

Take the gradient of the sampled loss (assuming $s' \neq s$),

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} = Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q(s',a';\mathbf{w})]$$

Approximate this by further sampling the next state to approximate the expectation $s' \sim p(s' \mid s,a)$,

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} \approx Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \max_{a'} Q(s',a';\mathbf{w})]$$

Sketch of SGD Approach

Loss at a sampled state-action pair s, a

$$\mathcal{L}^{(s,a)}(\mathbf{w}) = \frac{1}{2}||Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q(s',a';\mathbf{w})]||_2^2$$

Take the gradient of the sampled loss (assuming $s' \neq s$),

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} = Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q(s',a';\mathbf{w})]$$

Approximate this by further sampling the next state to approximate the expectation $s' \sim p(s' \mid s,a)$,

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} \approx Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \max_{a'} Q(s',a';\mathbf{w})]$$

Contents

Temporal Difference Meta-Algorithm

General update will be to use update (for $s' \sim p(s' \mid s, a)$)

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} \approx Q(s, a; \mathbf{w}) - [r(s, a) + \gamma \max_{a'} Q(s', a'; \mathbf{w})]$$

Issues:

- Need to decide how to obtain s, a
- Want to act in a real-world (possibly non-reversible).

General approach:

- Start at state $s \in S$, select action $a \in A$, collect r and s' from world, consider new action a', update with gradients, repeat at s'.
- How we pick actions a and a' will give us different algorithms.

Temporal Difference Meta-Algorithm

General update will be to use update (for $s' \sim p(s' \mid s, a)$)

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} \approx Q(s, a; \mathbf{w}) - [r(s, a) + \gamma \max_{a'} Q(s', a'; \mathbf{w})]$$

Issues:

- Need to decide how to obtain s, a
- Want to act in a real-world (possibly non-reversible).

General approach:

- Start at state $s \in S$, select action $a \in A$, collect r and s' from world, consider new action a', update with gradients, repeat at s'.
- How we pick actions a and a' will give us different algorithms.

TD Algorithmic Framework

procedure TD-RL(s)

Select action a based on $Q(s, a)$

Collect $r \leftarrow r(s, a)$ from environment

Sample $s' \sim p(s'|s, a)$ from environment

Consider possible future actions a'*

Update estimate of $Q(s, a)$ function through $w_{s,a}$

return s'

One step of RL on Gridworld

procedure TD-RL(s)

Select action a based on $Q(s, a)$

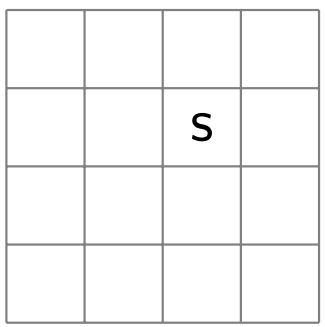
Collect $r \leftarrow r(s, a)$ from environment

Sample $s' \sim p(s'|s, a)$ from environment

Consider possible future actions a'

Update estimate of $Q(s, a)$ function through $w_{s,a}$

return s'



One step of RL on Gridworld

procedure TD-RL(s)

Select action a based on $Q(s, a)$

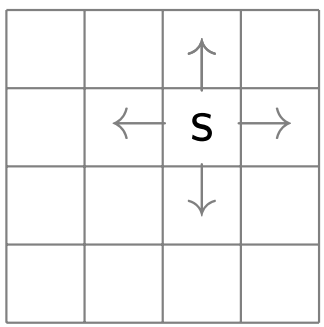
Collect $r \leftarrow r(s, a)$ from environment

Sample $s' \sim p(s'|s, a)$ from environment

Consider possible future actions a'

Update estimate of $Q(s, a)$ function through $w_{s,a}$

return s'



One step of RL on Gridworld

procedure TD-RL(s)

Select action a based on $Q(s, a)$

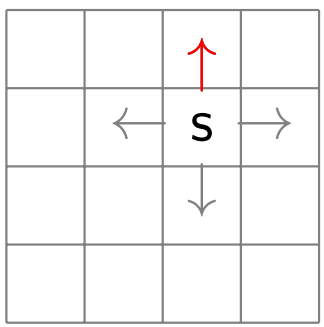
Collect $r \leftarrow r(s, a)$ from environment

Sample $s' \sim p(s'|s, a)$ from environment

Consider possible future actions a'

Update estimate of $Q(s, a)$ function through $w_{s,a}$

return s'



One step of RL on Gridworld

procedure TD-RL(s)

Select action a based on $Q(s, a)$

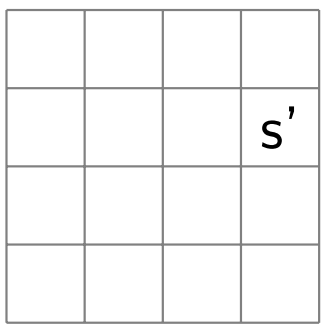
Collect $r \leftarrow r(s, a)$ from environment

Sample $s' \sim p(s'|s, a)$ from environment

Consider possible future actions a'

Update estimate of $Q(s, a)$ function through $w_{s,a}$

return s'



One step of RL on Gridworld

procedure TD-RL(s)

Select action a based on $Q(s, a)$

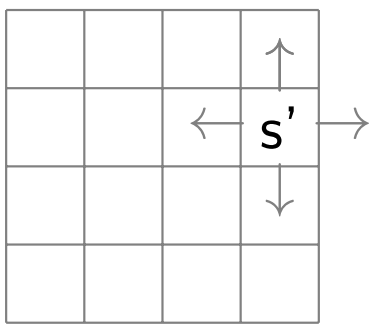
Collect $r \leftarrow r(s, a)$ from environment

Sample $s' \sim p(s'|s, a)$ from environment

Consider possible future actions a'

Update estimate of $Q(s, a)$ function through $w_{s,a}$

return s'



TD Choices

procedure TD-RL(s)

Select action a based on $Q(s, a)$ [1]

Collect $r \leftarrow r(s, a)$ from environment

Sample $s' \sim p(s'|s, a)$ from environment

Consider possible future actions a' [2]

Update estimate of $Q(s, a)$ function through $w_{s,a}$

return s'

1. How should we select with which action a to take to move to the next step?
2. How should we select which action a' should we the gradient ?

(1) Agent Policies

If we are **exploitative**, it is clear which action a to take. Just take highest expected reward.

$$a \leftarrow \arg\max_a Q(s, a; \mathbf{w})$$

(Why might this be bad?)

For **exploration**, an alternative is an **epsilon greedy** approach.

$$a \leftarrow \begin{cases} \arg\max_a Q(s, a; \mathbf{w}) & \text{with prob } 1 - \epsilon \\ \text{random } a \in A & \text{with prob } \epsilon \end{cases}$$

Many other approaches for exploration.

(1) Agent Policies

If we are **exploitative**, it is clear which action a to take. Just take highest expected reward.

$$a \leftarrow \arg\max_a Q(s, a; \mathbf{w})$$

(Why might this be bad?)

For **exploration**, an alternative is an **epsilon greedy** approach.

$$a \leftarrow \begin{cases} \arg\max_a Q(s, a; \mathbf{w}) & \text{with prob } 1 - \epsilon \\ \text{random } a \in A & \text{with prob } \epsilon \end{cases}$$

Many other approaches for exploration.

(2) Update Policies

Bellman equation assumes we are acting greedily, however if we are exploring, how do we select the a' to update towards?

Two different approaches:

1. On-Policy: Compute expected reward using a' from same policy that we are acting with.
2. Off-Policy: Update towards with a different policy than we are acting with.

SARSA

- SARSA is an **on-policy update** rule. Instead of max a', use next decision from policy π, e.g. epsilon greedy.

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} \approx Q(s,a;\mathbf{w}) - [r(s,a) + \gamma Q(s', \pi(s');\mathbf{w})]$$

- SGD update at each time step becomes

$$w_{s,a} \leftarrow Q(s,a;\mathbf{w}) + \eta[r + \gamma Q(s', \pi(s');\mathbf{w}) - Q(s,a;\mathbf{w})]$$

- Can be seen as taking an avg.

$$w_{s,a} \leftarrow (1-\eta) w_{s,a} + \eta(r + \gamma Q(s', \pi(s');\mathbf{w}))$$

- Better convergence, however updates towards worse policy. In practice, let $\epsilon \to 0$.

Q-Learning

- Q-Learning is an **off-policy update** rule, where a' is always max action and a is agent policy.
- SGD update at each time step becomes

$$w_{s,a} \leftarrow Q(s, a; \mathbf{w}) + \eta[r + \gamma \max_{a'} Q(s', a'; \mathbf{w}) - Q(s, a; \mathbf{w})]$$

- Off-policy since π may be an exploration policy, even if a' is always greedy. Worse convergence, but can learn Q^* even with epsilon-greedy policy.

Contents

Issues with Q-Learning

Q-Learning has so far assumed a very simple **parameterization** where our function estimator $Q(s, a; \mathbf{w})$ had a single parameter for each $S \times A$ i.e. $w_{s,a}$.

This is fine for grid-world but runs into issues on real-data:

- There are likely many states and actions that we will never see, how do we learn these parameters?
- Even if we do see everything, there will be way too many parameters in the model itself.

Luckily, we have spent all semester studying **function approximation**

Issues with Q-Learning

Q-Learning has so far assumed a very simple **parameterization** where our function estimator $Q(s, a; \mathbf{w})$ had a single parameter for each $S \times A$ i.e. $w_{s,a}$.

This is fine for grid-world but runs into issues on real-data:

- There are likely many states and actions that we will never see, how do we learn these parameters?
- Even if we do see everything, there will be way too many parameters in the model itself.

Luckily, we have spent all semester studying **function approximation**

Linear Basis for Q-Learning

Instead of an independent parameterization, we can learn a linear model over a **feature basis** $\boldsymbol{\phi} : (S \times A) \to \mathbb{R}^d$.

Then define our **weights** to be $\mathbb{R}^d$. The Q function becomes

$$Q(s, a) = \mathbf{w}^\top \boldsymbol{\phi}(s, a)$$

We can use (roughly) the same approach as above to learn the weight matrix.

Learning RL

$$Q(s, a; \mathbf{w}) = \mathbf{w}^\top \boldsymbol{\phi}(s, a)$$

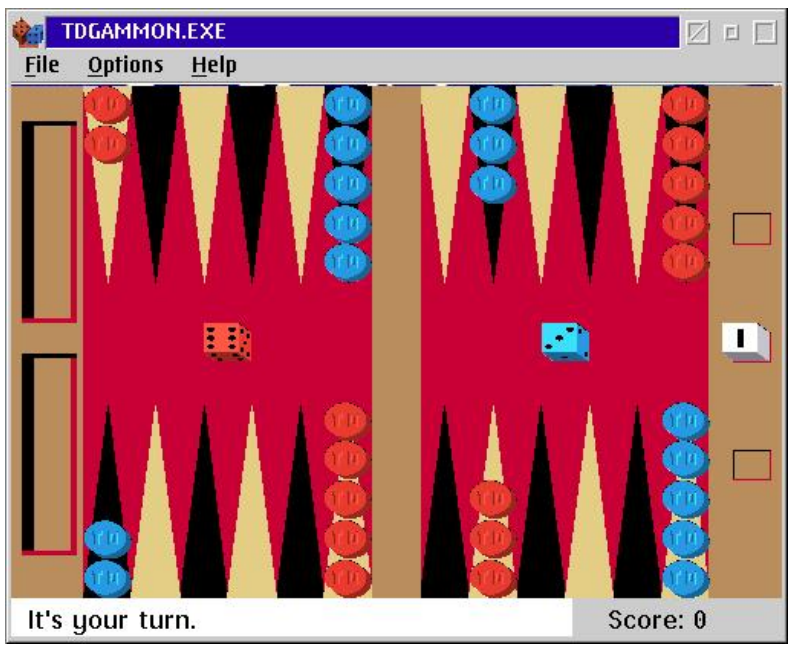
- Utilizing basis functions allows us to **generalize** to even unseen states and to share information between similar states.
- If we can learn states are similar, then we can better estimate their expected future reward, and inherently learn a better π.

Backgammon

TD-Gammon (1992)

- Computer backgammon system utilizing temporal difference learning and neural network.

$$Q(s, a; \mathbf{w}) = \mathbf{w}^\top \boldsymbol{\phi}(s, a; \mathbf{W})$$



Research Frontier

- What can be said about the convergence properties of improved function approximators with RL?
- Are there better ways to train these models beyond SGD?
- What if we had even better models for state representations? How could this improve the use of RL techniques in practice?

The last bullet will be the focus of our next class.