

# Reinforcement Learning: Rollout

BIOE 498/598 PJ

Spring 2021

## Review

The *value function* is the expected sum of future rewards

$$V(s_i) = \mathbb{E}\{r_i + r_{i+1} + \cdots + r_T\}.$$

At any state  $s_i$ , the optimal policy follows the objective

$$\begin{aligned} & \max_{a_i} \mathbb{E}\{r_i + r_{i+1} + \cdots + r_T\} \\ &= \max_{a_i} \mathbb{E}\{r_i\} + \mathbb{E}\{r_{i+1} + \cdots + r_T\} \\ &= \max_{a_i} \mathbb{E}\{r_i\} + V(s_{i+1}) \end{aligned}$$

Optimal policies can be found with a value function.

The optimal RL policy balances the immediate reward  $r_i$  with future rewards  $V(s_{i+1})$ :

$$\pi^*(s_i) = \arg \max_{a_i} \{ \mathbb{E}[r_i] + V(s_{i+1}) \} .$$

## Optimal policies can be found with a value function.

The optimal RL policy balances the immediate reward  $r_i$  with future rewards  $V(s_{i+1})$ :

$$\pi^*(s_i) = \arg \max_{a_i} \{ \mathbb{E}[r_i] + V(s_{i+1}) \}.$$

If we know the value function, the optimal policy is to be greedy with respect to it. However, we rarely know  $V$ :

- ▶ Monte Carlo estimation of  $V$  is only approximate even with many simulations.
- ▶ If the state space is very large, we may never visit every state to estimate  $V$  by tabular methods.

## Optimal policies can be found with a value function.

The optimal RL policy balances the immediate reward  $r_i$  with future rewards  $V(s_{i+1})$ :

$$\pi^*(s_i) = \arg \max_{a_i} \{ \mathbb{E}[r_i] + V(s_{i+1}) \}.$$

If we know the value function, the optimal policy is to be greedy with respect to it. However, we rarely know  $V$ :

- ▶ Monte Carlo estimation of  $V$  is only approximate even with many simulations.
- ▶ If the state space is very large, we may never visit every state to estimate  $V$  by tabular methods.

Instead, we use an *approximate value function*  $\tilde{V}(s)$  to find a sub-optimal policy  $\pi(s)$ .

## Approximate value functions.

There are two classes of methods for approximating a value function.

1. **Parametric approximation** trains a model that predicts value from previous visits to states. The model predicts the value for *all* states, even those that have not been visited.
  - ▶ Any type of model can be used to predict value: Linear models, Gaussian Process Regression, Artificial Neural Networks (“Deep RL”).
  - ▶ Parametric methods are often *offline*; the model is trained before the agent uses the model to navigate an MDP.

# Approximate value functions.

There are two classes of methods for approximating a value function.

1. **Parametric approximation** trains a model that predicts value from previous visits to states. The model predicts the value for *all* states, even those that have not been visited.
  - ▶ Any type of model can be used to predict value: Linear models, Gaussian Process Regression, Artificial Neural Networks (“Deep RL”).
  - ▶ Parametric methods are often *offline*; the model is trained before the agent uses the model to navigate an MDP.
2. **Monte Carlo approximation** uses a model of the MDP to simulate the rewards following a state.
  - ▶ Monte Carlo methods work well *online* by simulating states just ahead of the agent in the MDP.
  - ▶ These methods are *sample efficient* but require a computational model of the MDP.

# Rollout

- ▶ Rollout is a Monte Carlo method frequently used for online RL.
- ▶ Rollout “looks ahead” to estimate the value of states the agent is likely to visit next.
- ▶ Rollout is robust and works with many RL problems. Variants of rollout (e.g. MCTS) power AlphaGo, AlphaZero, and other top game engines.
- ▶ Rollout can be used for policy iteration, but we will limit our discussion to a single pass through an MDP.



# The rollout algorithm

Rollout requires

- ▶ A *simulator* that generates sequences

$$s_i, a_i, r_i, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$$

given a policy  $\pi$ .

- ▶ A *base policy*  $\pi_{\text{base}}$  to use with the simulator. A random base policy works.

# The rollout algorithm

Rollout requires

- ▶ A *simulator* that generates sequences

$$s_i, a_i, r_i, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$$

given a policy  $\pi$ .

- ▶ A *base policy*  $\pi_{\text{base}}$  to use with the simulator. A random base policy works.

$s$

# The rollout algorithm

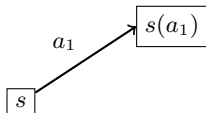
Rollout requires

- ▶ A *simulator* that generates sequences

$$s_i, a_i, r_i, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$$

given a policy  $\pi$ .

- ▶ A *base policy*  $\pi_{\text{base}}$  to use with the simulator. A random base policy works.



# The rollout algorithm

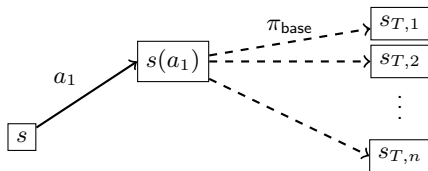
Rollout requires

- ▶ A *simulator* that generates sequences

$$s_i, a_i, r_i, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$$

given a policy  $\pi$ .

- ▶ A *base policy*  $\pi_{\text{base}}$  to use with the simulator. A random base policy works.



# The rollout algorithm

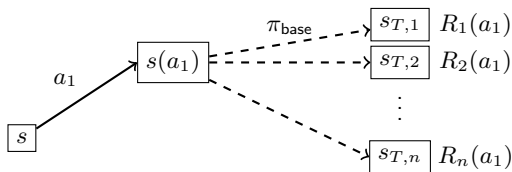
Rollout requires

- ▶ A *simulator* that generates sequences

$$s_i, a_i, r_i, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$$

given a policy  $\pi$ .

- ▶ A *base policy*  $\pi_{\text{base}}$  to use with the simulator. A random base policy works.



# The rollout algorithm

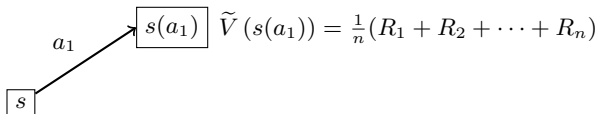
Rollout requires

- ▶ A *simulator* that generates sequences

$$s_i, a_i, r_i, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$$

given a policy  $\pi$ .

- ▶ A *base policy*  $\pi_{\text{base}}$  to use with the simulator. A random base policy works.



# The rollout algorithm

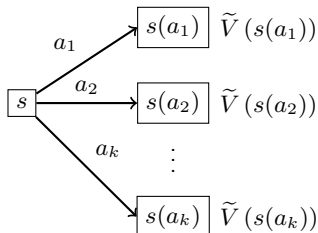
Rollout requires

- ▶ A *simulator* that generates sequences

$$s_i, a_i, r_i, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$$

given a policy  $\pi$ .

- ▶ A *base policy*  $\pi_{\text{base}}$  to use with the simulator. A random base policy works.



# The rollout algorithm

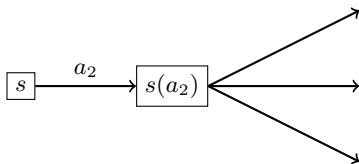
Rollout requires

- ▶ A *simulator* that generates sequences

$$s_i, a_i, r_i, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$$

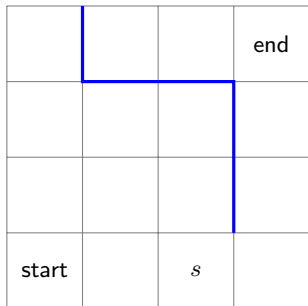
given a policy  $\pi$ .

- ▶ A *base policy*  $\pi_{\text{base}}$  to use with the simulator. A random base policy works.



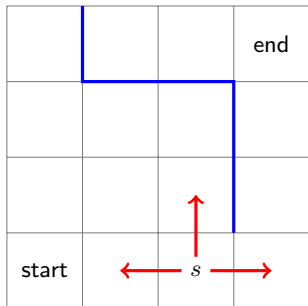


## Rollout in Gridworld



- Imagine we are in the middle of the maze at state  $s$ .

## Rollout in Gridworld



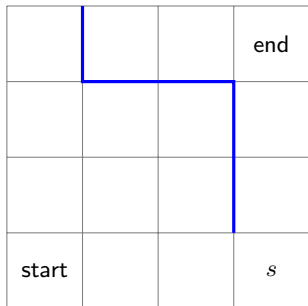
- ▶ Imagine we are in the middle of the maze at state  $s$ .
- ▶ There are three available actions: {left, up, down}.

## Rollout in Gridworld

			end
		-33.3	
start	-32.8	<i>s</i>	-18.6

- ▶ Imagine we are in the middle of the maze at state  $s$ .
- ▶ There are three available actions: {left, up, down}.
- ▶ We use random walks to estimate  $\tilde{V}$  after each action.

## Rollout in Gridworld



- ▶ Imagine we are in the middle of the maze at state  $s$ .
- ▶ There are three available actions: {left, up, down}.
- ▶ We use random walks to estimate  $\tilde{V}$  after each action.
- ▶ We select the action that leads to the best estimated value.

## Policy improvement with rollout

- ▶ The online policy we are finding is called the *rollout policy*.
- ▶ The rollout policy has the *policy improvement property* — it will be equal to or better than the base policy.
- ▶ We can repeat the process with another trip through the MDP using the rollout policy as the new base policy.
- ▶ However, iterating in this way requires us to add exploration to our policies.

## Summary

- ▶ **Rollout** is an online method that reduces simulation by focusing on local starts.
- ▶ A single pass with a random base policy provides good, but not necessarily optimal, behavior.
- ▶ Iteration and exploration are required to find optimal policies.