Reinforcement Learning: Discounting, TD-learning, and Q-factors

BIOE 498/598 PJ

Spring 2022

Review

- **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.

Review

- **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.
- **Today:** Model-free learning with discounted rewards and Q-factors.

Discount factors

Let's extend our RL theory to incorporate *discounting* — reducing the present value of rewards from the future.

Discounting applies a horizon to the problem. The agent cares less and less about future states.

Discount factors

Let's extend our RL theory to incorporate *discounting* — reducing the present value of rewards from the future.

Discounting applies a horizon to the problem. The agent cares less and less about future states.

$$
\begin{aligned}
\text{Undiscounted:} \quad & \max_a \mathbb{E}\left\{r_i + V(s_{i+1})\right\} \\
\text{Discounted:} \quad & \max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}
\end{aligned}
$$

Discount factors

Let's extend our RL theory to incorporate *discounting* — reducing the present value of rewards from the future.

Discounting applies a horizon to the problem. The agent cares less and less about future states.

$$\text{Undiscounted:} \quad \max_a \mathbb{E}\left\{r_i + V(s_{i+1})\right\}$$
$$\text{Discounted:} \quad \max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}$$

The *discount factor* $\gamma \in [0, 1]$ determines the length of the horizon.

- $\gamma = 0$ makes the algorithms greedy; only the immediate reward r_i influences the agent.
- $\gamma = 1$ equally weights all rewards to the end of the trajectory.

Discounting in Gridworld

- For Gridworld we used a penalty (negative reward) to encourage the agent to finish the maze quickly.
- Instead of penalizing each action, we could discount and offer a terminal reward.

Discounting in Gridworld

- For Gridworld we used a penalty (negative reward) to encourage the agent to finish the maze quickly.
- Instead of penalizing each action, we could discount and offer a terminal reward.

Penalized stepping with $r_i = -1$, $r_T = 0$:

$$\begin{aligned} \text{reward} &= r_0 + r_1 + \cdots + r_{T-1} + r_T \\ &= \sum_{i=0}^{T-1} r_i + 0 \\ &= -T \end{aligned}$$

Discounting in Gridworld

- For Gridworld we used a penalty (negative reward) to encourage the agent to finish the maze quickly.
- Instead of penalizing each action, we could discount and offer a terminal reward.

Penalized stepping with $r_i = -1$, $r_T = 0$:

$$\begin{aligned} \text{reward} &= r_0 + r_1 + \cdots + r_{T-1} + r_T \\ &= \sum_{i=0}^{T-1} r_i + 0 \\ &= -T \end{aligned}$$

Discounting with $r_i = 0$, $r_T = 1$, $\gamma < 1$:

$$\begin{aligned} \text{reward} &= r_0 + \gamma(r_1 + \gamma(r_2 + \gamma(\cdots \gamma(r_{T-1} + \gamma(r_T))))) \\ &= r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \gamma^{T-1} r_{T-1} + \gamma^T r_T \\ &= \gamma^T \end{aligned}$$

Discounting in Gridworld

- For Gridworld we used a penalty (negative reward) to encourage the agent to finish the maze quickly.
- Instead of penalizing each action, we could discount and offer a terminal reward.

Penalized stepping with $r_i = -1$, $r_T = 0$:

$$\begin{aligned}\text{reward} &= r_0 + r_1 + \cdots + r_{T-1} + r_T \\ &= \sum_{i=0}^{T-1} r_i + 0 \\ &= -T\end{aligned}$$

Discounting with $r_i = 0$, $r_T = 1$, $\gamma < 1$:

$$\begin{aligned}\text{reward} &= r_0 + \gamma(r_1 + \gamma(r_2 + \gamma(\cdots\gamma(r_{T-1} + \gamma(r_T))))) \\ &= r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \gamma^{T-1} r_{T-1} + \gamma^T r_T \\ &= \gamma^T\end{aligned}$$

In both cases, the maximum reward is achieved by minimizing the number of steps T.

When to discount?

Almost all algorithms are written with a discount factor

$$\max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}.$$

- If you don't want to discount future rewards, set $\gamma = 1$.
- If you want to compare your algorithm to a greedy algorithm, set $\gamma = 0$.
- If you want the agent to terminate the process quickly, set $\gamma < 1$.

When to discount?

Almost all algorithms are written with a discount factor

$$\max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}.$$

- If you don't want to discount future rewards, set $\gamma = 1$.
- If you want to compare your algorithm to a greedy algorithm, set $\gamma = 0$.
- If you want the agent to terminate the process quickly, set $\gamma < 1$.

Discounting is also the key to solving non-episodic (infinite horizon) problems. While the MDP never terminates, the discounted rewards become so small that the agent stops caring after a finite number of steps.

Model-free learning

- Monte Carlo methods like rollout require a *model* to simulate ahead when estimating value functions.
- *Model-free* algorithms learn directly from experience. Their only method of sampling is to interact with the environment.
- Model-free algorithms try to maximize the information that can be extracted from every trajectory.

Temporal difference learning

- Model-free algorithms learn directly from experience.
- Each trajectory is "expensive" relative to a simulated trajectory.
- Ideally, we would update our estimates of the value function from every trajectory; however, a single trajectory is a noisy estimate of value.
- **Temporal difference (TD) learning** balances new experiences with previous results when updating $V(s)$.

The TD-learning algorithm

1. Initialize our value estimates $V(s)$ for all states s.

The TD-learning algorithm

1. Initialize our value estimates $V(s)$ for all states s.
2. Experience a new trajectory $s_0, a_0, r_0,\ s_1, a_1, r_1\ \ldots,\ s_T, r_T$.

The TD-learning algorithm

1. Initialize our value estimates $V(s)$ for all states s.
2. Experience a new trajectory $s_0, a_0, r_0,\ s_1, a_1, r_1\ \ldots,\ s_T, r_T$.
3. For each state s_i in the trajectory, calculate the *TD target*

$$\hat{V}(s_i) = r_i + \gamma V(s_{i+1})$$

   using the experienced reward r_i and the previous estimate for $V(s_{i+1})$.

The TD-learning algorithm

1. Initialize our value estimates $V(s)$ for all states s.
2. Experience a new trajectory $s_0, a_0, r_0,\ s_1, a_1, r_1\ \ldots,\ s_T, r_T$.
3. For each state s_i in the trajectory, calculate the *TD target*

   $$\hat{V}(s_i) = r_i + \gamma V(s_{i+1})$$

   using the experienced reward r_i and the previous estimate for $V(s_{i+1})$.
4. Incrementally update the value of state s_i using a learning rate α:

   $$V(s_i) = V(s_i) + \alpha \left[\hat{V}(s_i) - V(s_i) \right].$$

The TD-learning algorithm

1. Initialize our value estimates $V(s)$ for all states s.
2. Experience a new trajectory $s_0, a_0, r_0,\ s_1, a_1, r_1\ \ldots,\ s_T, r_T$.
3. For each state s_i in the trajectory, calculate the *TD target*

   $$\hat{V}(s_i) = r_i + \gamma V(s_{i+1})$$

   using the experienced reward r_i and the previous estimate for $V(s_{i+1})$.
4. Incrementally update the value of state s_i using a learning rate α:

   $$V(s_i) = V(s_i) + \alpha \left[\hat{V}(s_i) - V(s_i) \right].$$
5. Go to step #2 and repeat.

The TD-learning algorithm

1. Initialize our value estimates $V(s)$ for all states s.
2. Experience a new trajectory $s_0, a_0, r_0,\ s_1, a_1, r_1\ \ldots,\ s_T, r_T$.
3. For each state s_i in the trajectory, calculate the *TD target*

$$\hat{V}(s_i) = r_i + \gamma V(s_{i+1})$$

   using the experienced reward r_i and the previous estimate for $V(s_{i+1})$.

4. Incrementally update the value of state s_i using a learning rate α:

$$V(s_i) = V(s_i) + \alpha \left[\hat{V}(s_i) - V(s_i) \right].$$

5. Go to step #2 and repeat.

TD-learning is a *bootstrap* method since $V(s)$ is updated using $V(s_i)$ and $V(s_{i+1})$ from the previous iteration. New information only enters through r_i when estimating the TD target $\hat{V}(s_i)$.

Q-factors

Learning $V(s)$ is not the end. We still need to find a policy that solves

$$\max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}.$$

This requires knowing s_{i+1} given s_i and a, or at least the probability distribution for ending up in each state.

Q-factors

Learning $V(s)$ is not the end. We still need to find a policy that solves

$$\max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}.$$

This requires knowing s_{i+1} given s_i and a, or at least the probability distribution for ending up in each state.

Example: In chess, the state s_i is the arrangement of all the pieces on the board.

- We select a based on the reward r_i (which is usually zero) and the future value $V(s_{i+1})$.
- However, s_{i+1} is the state *after our opponent's turn!* We have no idea what move our opponent will make.

Q-factors

Learning $V(s)$ is not the end. We still need to find a policy that solves

$$\max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}.$$

This requires knowing s_{i+1} given s_i and a, or at least the probability distribution for ending up in each state.

Example: In chess, the state s_i is the arrangement of all the pieces on the board.

- We select a based on the reward r_i (which is usually zero) and the future value $V(s_{i+1})$.
- However, s_{i+1} is the state *after our opponent's turn!* We have no idea what move our opponent will make.

For many problems it is easier to learn the value of each state/action pair, called a Q-factor or $Q(s, a)$.

Learning Q-factors

Using Q-factors, the policy problem at state s

$$\max_a \mathbb{E}\{r_i + \gamma V(s_{i+1})\}$$

becomes

$$\max_a \mathbb{E}\{Q(s_i, a)\}.$$

Learning Q-factors

Using Q-factors, the policy problem at state s

$$\max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}$$

becomes

$$\max_a \mathbb{E}\left\{Q(s_i, a)\right\}.$$

- **Pro:** We do not need a model or a way to predict s_{i+1}.
- **Con:** We need to learn a Q-factor for every state/action pair.

Learning Q-factors

Using Q-factors, the policy problem at state s

$$\max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}$$

becomes

$$\max_a \mathbb{E}\left\{Q(s_i, a)\right\}.$$

- **Pro:** We do not need a model or a way to predict s_{i+1}.
- **Con:** We need to learn a Q-factor for every state/action pair.

We can learn Q-factors using a TD approach given a trajectory s_0, a_0, r_0, s_1, a_1, r_1 …, s_T, r_T:

$$\hat{Q}(s_i, a_i) = r_i + \gamma Q(s_{i+1}, a_{i+1}) \qquad \text{target}$$

$$Q(s_i, a_i) = Q(s_i, a_i) + \alpha\left[\hat{Q}(s_i, a_i) - Q(s_i, a_i)\right] \qquad \text{update}$$

Learning Q-factors

Using Q-factors, the policy problem at state s

$$\max_a \mathbb{E}\left\{r_i + \gamma V(s_{i+1})\right\}$$

becomes

$$\max_a \mathbb{E}\left\{Q(s_i, a)\right\}.$$

- **Pro:** We do not need a model or a way to predict s_{i+1}.
- **Con:** We need to learn a Q-factor for every state/action pair.

We can learn Q-factors using a TD approach given a trajectory s_0, a_0, r_0, s_1, a_1, r_1 …, s_T, r_T:

$$\hat{Q}(s_i, a_i) = r_i + \gamma Q(s_{i+1}, a_{i+1}) \qquad \text{target}$$

$$Q(s_i, a_i) = Q(s_i, a_i) + \alpha\left[\hat{Q}(s_i, a_i) - Q(s_i, a_i)\right] \qquad \text{update}$$

This approach is also called *SARSA*.

Summary

- Discount factors shorten the horizon of RL problems, causing the agent to focus on rewards in the near future.
- Temporal Difference (TD) learning incrementally updates value functions using a new experience.
- Learning Q-factors eliminates the need to predict the next state given an action; however, the number of Q-factors is much greater than the number of states.

Summary

- Discount factors shorten the horizon of RL problems, causing the agent to focus on rewards in the near future.
- Temporal Difference (TD) learning incrementally updates value functions using a new experience.
- Learning Q-factors eliminates the need to predict the next state given an action; however, the number of Q-factors is much greater than the number of states.
- **Next time:** AlphaGo! (watch the documentary this weekend)