Reinforcement Learning: Q-learning and AlphaGo

BIOE 498/598 PJ

Spring 2022

Review

- 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.

Review

- 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.
- **Today:**
  - Review SARSA
  - Q-learning
  - AlphaGo

Learning Q-factors

Using Q-factors, the policy problem at state s_i

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

becomes

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

Learning Q-factors

Using Q-factors, the policy problem at state s_i

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

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_i

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

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:

$$\begin{aligned} \hat{Q}(s_i, a_i) &= r_i + \gamma Q(s_{i+1}, a_{i+1}) && \text{target} \\ Q(s_i, a_i) &\leftarrow Q(s_i, a_i) + \alpha\left[\hat{Q}(s_i, a_i) - Q(s_i, a_i)\right] && \text{update} \end{aligned}$$

This approach is called *SARSA*.

SARSA follows a trajectory, not an optimal path

The SARSA update equation is

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[\underbrace{r_i + \gamma Q(s_{i+1}, a_{i+1})}_{\text{target}} - Q(s_i, a_i) \right].$$

SARSA follows a trajectory, not an optimal path

The SARSA update equation is

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \Big[\underbrace{r_i + \gamma Q(s_{i+1}, a_{i+1})}_{\text{target}} - Q(s_i, a_i) \Big].$$

Our estimate of $Q(s_i, a_i)$ is based on

- The reward r_i experienced by selecting action a_i in state s_i.
- The future reward $Q(s_{i+1}, a_{i+1})$ based on the action a_{i+1} from the trajectory.

SARSA follows a trajectory, not an optimal path

The SARSA update equation is

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \Big[\underbrace{r_i + \gamma Q(s_{i+1}, a_{i+1})}_{\text{target}} - Q(s_i, a_i) \Big].$$

Our estimate of $Q(s_i, a_i)$ is based on

- The reward r_i experienced by selecting action a_i in state s_i.
- The future reward $Q(s_{i+1}, a_{i+1})$ based on the action a_{i+1} from the trajectory.

The policy that generated the trajectory is not optimal, so it is likely that a_{i+1} was not the best action to take.

Selecting a suboptimal action underestimates the reward to go, and therefore the value $Q(s_i, a_i)$.

Q-learning

The Q-learning algorithm changes the SARSA update

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[r_i + \gamma Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i)\right]$$

to use the optimal action in state s_{i+1}:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[r_i + \gamma \max_a Q(s_{i+1}, a) - Q(s_i, a_i)\right].$$

Q-learning

The Q-learning algorithm changes the SARSA update

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[r_i + \gamma Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i)\right]$$

to use the optimal action in state s_{i+1}:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[r_i + \gamma \max_a Q(s_{i+1}, a) - Q(s_i, a_i)\right].$$

Q-learning can converge faster to an optimal policy. However, it has two drawbacks:

1. If the number of available actions is large, the maximization operator can be expensive to evaluate.
2. The maximization operator is biased.

Records were meant to be broken.

- Imagine that the quality of professional basketball players was fixed over time.
- In this case, scoring records would still be broken.
- Basketball includes stochastic elements, so as more games are played the chance of observing outliers increases.

Records were meant to be broken.

- Imagine that the quality of professional basketball players was fixed over time.
- In this case, scoring records would still be broken.
- Basketball includes stochastic elements, so as more games are played the chance of observing outliers increases.

Any algorithm with a $\max$ operator will drift upwards over time, *even if the mean value remains fixed*.

For Q-learning, we need to combat the bias in the $\max$ operator.

Double Q-learning

One solution to the $\max$ bias is using two separate Q functions (networks), called Q_1 and Q_2.

Both Q_1 and Q_2 are trained with separate experiences. (Or, one network can *lag* behind the other in experiences.)

Double Q-learning

One solution to the $\max$ bias is using two separate Q functions (networks), called Q_1 and Q_2.

Both Q_1 and Q_2 are trained with separate experiences. (Or, one network can *lag* behind the other in experiences.)

When updating, we use one network to select the action, and the other network to compute its value.

$$Q_1(s_i, a_i) \leftarrow Q_1(s_i, a_i) + \alpha \left[r_i + \gamma \, Q_2(s_{i+1}, a_1) - Q_1(s_i, a_i) \right]$$
$$a_1 \equiv \arg\max_a Q_1(s_{i+1}, a)$$

$$Q_2(s_i, a_i) \leftarrow Q_2(s_i, a_i) + \alpha \left[r_i + \gamma \, Q_1(s_{i+1}, a_2) - Q_2(s_i, a_i) \right]$$
$$a_2 \equiv \arg\max_a Q_2(s_{i+1}, a)$$

Double Q-learning

One solution to the $\max$ bias is using two separate Q functions (networks), called Q_1 and Q_2.

Both Q_1 and Q_2 are trained with separate experiences. (Or, one network can *lag* behind the other in experiences.)

When updating, we use one network to select the action, and the other network to compute its value.

$$Q_1(s_i, a_i) \leftarrow Q_1(s_i, a_i) + \alpha \left[r_i + \gamma\, Q_2(s_{i+1}, a_1) - Q_1(s_i, a_i)\right]$$
$$a_1 \equiv \arg\max_a Q_1(s_{i+1}, a)$$

$$Q_2(s_i, a_i) \leftarrow Q_2(s_i, a_i) + \alpha \left[r_i + \gamma\, Q_1(s_{i+1}, a_2) - Q_2(s_i, a_i)\right]$$
$$a_2 \equiv \arg\max_a Q_2(s_{i+1}, a)$$

Even if a_1 was selected because $Q_1(s_{i+1}, a_1)$ was aberrantly high, the value $Q_2(s_{i+1}, a_1)$ will not share this bias.

Summary

- Q-learning is a state-of-the-art technique for RL.
- Double Q-learning counteracts the bias in the $\max$ operator.