CS 181 LECTURE 4/16/24

Scribe Notes

Contents

1 Admin

Today's lecture is the last lecture of content that will be covered on the midterm.

There are midterm review sessions today and Saturday. We will release practice problems and an example midterm. Anything covered in lecture is fair game and exercises in lecture are good practice as well! So, make sure to not overfit to the example midterm.

Midterm 2 will not be cumulative. The second midterm will only cover everything after the first midterm.

HW6 is due the Friday after the exam.

2 Reinforcement Learning

Objective design is a real-world challenge. The question is how do we assign rewards?

3 Story of the World: RL with Human Feedback

The idea is that people give feedback, $\{+1, -1\}$, on the agent's behavior.

Before, we created the diagram where the agent interacts with the world and the world sends rewards and state information back.

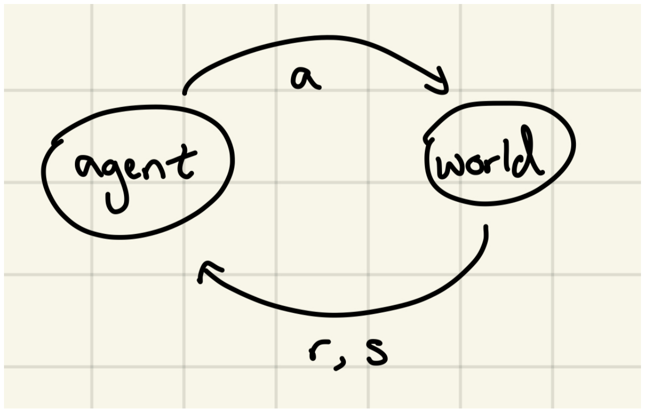


Figure 1: Agent interacting with the world.

The key element is that the world is sending back rewards, but where do these rewards come from? That is the reward design problem. For example, if you are getting to the goal you get a certain number of points. But if you get injured along the way then we need to subtract points.

Consider LLM's where the goal is to predict the next word. We need to be wary of racist rhetoric or other words that we do not want the LLM to output. This is where human feedback comes in. We take the human feedback and we train a $\hat{r}$. We are using human feedback to improve the quality of our reward system.

4 Learning From Experience

Last week, we assumed access to S, A, γ, R, T. If we had all that information to us - aka we know how the world works - then we solved the planning problem to find the optimal policy. There may be uncertainty in the world, but we know the distribution.

We are transitioning this week to learning from experience. For example, we experience something and we change what we do in the future. We have access to history, $h = \{s_0, a_0, r_0, s_1, a_1, ...s_t, a_t, r_t\}$.

In order to learn more, we need to explore. For instance, when trying to find the best route to work, you have to try different routes. To exploit is to then take advantage of what you know. So, if you know the fastest route to work, then you should use that route. You should not give up exploring but the more you understand, the more that you should exploit what you know.

One of the key ideas that drives RL is how to optimize between exploring and exploiting. **Optimism under uncertainty** is a key principle that says if you are unsure, presume that it will work well. Say you have gone down one route and you know how long it will take. You are considering a different route and you are unsure of how long it will take. Then, with optimism under uncertainty, you should choose the new route. This works because either you learn something new and squash the uncertainty or you find a better path that you can then exploit.

5 Value-Based Methods for RL

1. Updating/tracking our action-value function $Q(s, a)$.
2. Using $Q(s, a)$ to pick actions.

The first step represents a knowledge update. After doing something, we see what it teaches us. The second step is a decision making problem.

Let us start with step 1.

5.1 Updating Action Value Function

We can think of this as asynchronous updates. We only get data about a particular state, action, and reward, which we use to make our updates.

5.1.1 Approach 1: SARSA

SARSA stands for state, action, reward, state, action.

Given some new experience, (s, a, r, s', a'), we update $Q(s, a)$ to the following:

$$\underbrace{Q(s,a)}_{\text{original value}} + \underbrace{\alpha_t}_{\text{learning rate}} \underbrace{[r + \gamma Q(s', a') - Q(s,a)]}_{\text{temporal difference error}}$$

We can further breakdown the temporal difference error.

$$\underbrace{r + \gamma Q(s', a')}_{\text{one-sample estimate of original value}} - \underbrace{Q(s,a)}_{\text{original value}}$$

Recall that $Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a)V^{\pi}(s)$. Therefore, $r + \gamma Q(s',a')$ is an estimate for $Q(s,a)$.

If we follow some policy π, and we decrease α_t such that $\sum \alpha_t \to \infty$ and $\sum \alpha_t^2$ is bounded, then we will get $Q^{\pi}(s,a)$. We want the α's to get smaller over time but not too quickly. In practice, α is normally set to a certain value for awhile and then set to 0.

SARSA is called an on-policy technique because we assume we continue with policy π. In other words, a' is taken from π.

5.1.2 Approach 2: Q-Learning

Here is our update for Q-Learning:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

The only difference between q-learning and SARSA is that we have $\max_{a'} Q(s',a')$. SARSA is evaluating one policy. In q-learning, after we take action a with reward r, we will do the best thing we can possibly do. In the future, we do the best possible action given our history. T

Suppose every s, a is visited infinitely often in an infinite run. If we run with policy π, SARSA will converge to Q^{π}. On the other hand, q-learning will converge to Q^*. SARSA is evaluating your policy while q-learning is evaluating the best policy.

Therefore, q-learning is an off-policy method. In healthcare, the doctors are doing something. But we want to make guesses on what they could have done. Hence, off-policy methods are quite valuable in a real-world application.

5.1.3 SARSA vs Q-Learning

SARSA is very simple. However, SARSA's convergence is based on the policy being followed, which may lead to suboptimal policies if the exploration strategy is not sufficiently explorative. SARSA also tends to converge more slowly compared to Q-learning. So, Q-learning is faster but Q-learning's off-policy nature may lead to policy instability.

5.2 Picking Actions

Now, we move onto the second step where we pick actions.

5.2.1 ϵ-Greedy Method

One method is the ϵ-greedy method:

- Take action argmax $Q(s,a)$ (which is exploiting) with probability 1-ϵ .
- Take a random action (which is exploring) with probability ϵ.

6 Other Methods

So, we have talked about value-based methods which involve learning the value of state-action pairs or states and selecting actions based on their associated values. But there are other methods we can use.

Model-based methods involve learning an explicit model of the environment, which typically includes transition dynamics (probabilities of transitioning between states) and rewards. With this model, the agent can simulate possible trajectories and use them to optimize its policy.

Another method is direct policy optimization. Unlike value-based methods, direct policy optimization methods directly parameterize and optimize the policy function. Instead of estimating the value of actions or states, these methods directly adjust the parameters of the policy to maximize expected returns.

7 Continuous Space

We need function approximation.

Say that we are given a batch of (s, a, r, s').

Fitted-q iteration:

1. Regress to learn $\hat{r}(s, a)$. We take examples of s, a and we learn r.
2. Regress to learn $\hat{q}_1(s, a) : s, a \to \gamma \max_{a'} \hat{r}(s', a')$
3. $\hat{q}_2(s, a) : s, a \to r + \gamma \max_{a'} \hat{q}_2(s', a')$

Another method is deep-q networks where we directly optimize the following loss function:

$$\mathcal{L} = \sum_n (q(s_n, a_n) - (r_n + \gamma q(s'_n, a'_n)))^2.$$

8 Concept Check

Consider the following grid world.

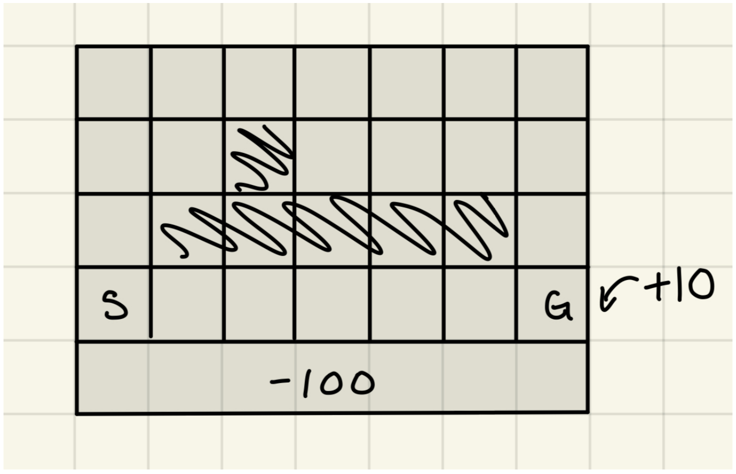


Figure 2: Grid World.

There is a cliff and you do not want to fall off the bottom of the cliff.

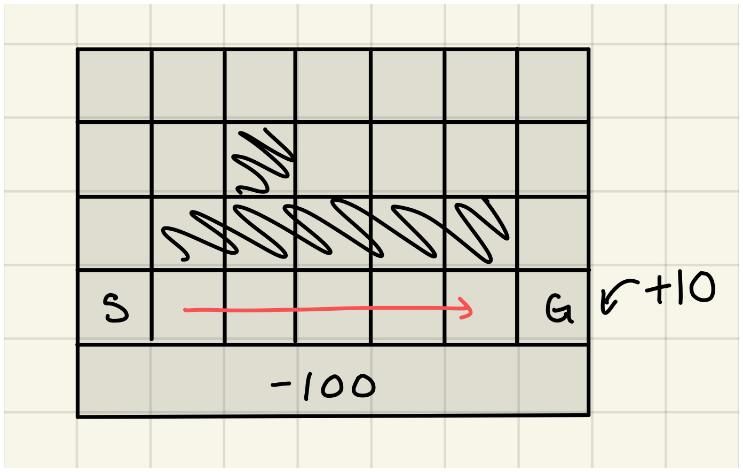
Uncertainty is coming from two places: 1) is the world and 2) is our choice to use the ϵ-greedy method.

Say that our actions err with probability δ. Here are the questions:

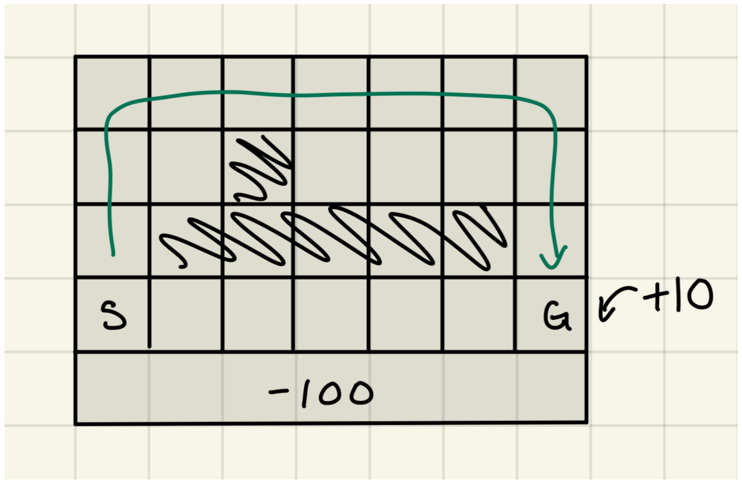
1. If $\delta \approx 0$, what is π^*?
2. If δ is moderate, what is π^*?
3. If ϵ is moderate, what does SARSA learn?
4. If ϵ is moderate, what does Q-learning learn?

9 Concept Check Solution

1. If the agent never screws up, then going across is the best action.



2. But if δ is moderate, then it is better to go around, so that you do not fall off the cliff.



3. SARSA is evaluating based on the next a'. If we are using ϵ-greedy, then every so often, it will make the move of hopping off the cliff. So, SARSA will learn the safe route.

4. Q-learning will know that you should not hop off the cliff.