MDPs: model-free methods

From model-based to model-free

$$\hat{Q}_{\text{opt}}(s,a) = \sum_{s'} \hat{T}(s,a,s')[\widehat{\text{Reward}}(s,a,s') + \gamma \hat{V}_{\text{opt}}(s')]$$

All that matters for prediction is (estimate of) $Q_{\text{opt}}(s,a)$.

Key idea: model-free learning

Try to estimate $Q_{\text{opt}}(s,a)$ directly.

- Taking a step back, if our goal is to just find good policies, all we need is to get a good estimate of $\hat{Q}_{\text{opt}}$. From that perspective, estimating the model (transitions and rewards) was just a means towards an end. Why not just cut to the chase and estimate $\hat{Q}_{\text{opt}}$ directly? This is called **model-free** learning, where we don't explicitly estimate the transitions and rewards.

Model-free Monte Carlo

Data (following policy π):

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots ; a_n, r_n, s_n$$

Recall:

$Q_\pi(s, a)$ is expected utility starting at s, first taking action a, and then following policy π

Utility:

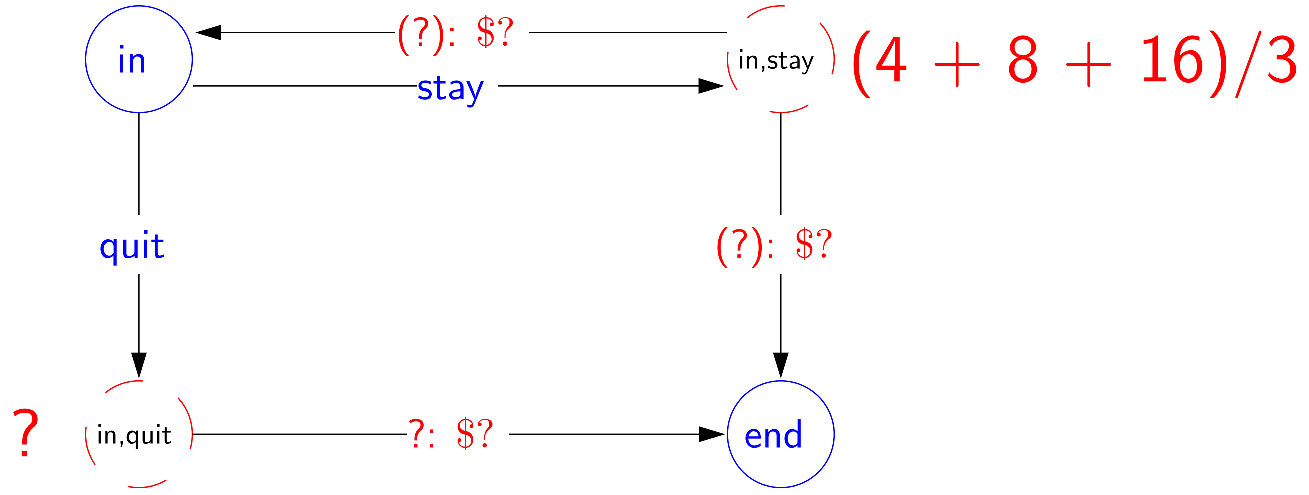
$$u_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots$$

Estimate:

$\hat{Q}_\pi(s, a) =$ average of u_t where $s_{t-1} = s, a_t = a$

(and s, a doesn't occur in $s_0, \cdots, s_{t-2}$)

Model-free Monte Carlo



Data (following policy $\pi(s) =$ stay):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

Note: we are estimating Q_π now, not Q_{opt}

Definition: on-policy versus off-policy

On-policy: estimate the value of data-generating policy

Off-policy: estimate the value of another policy

- Recall that $Q_\pi(s, a)$ is the expected utility starting at s, taking action a, and the following π.
- In terms of the data, define u_t to be the discounted sum of rewards starting with r_t.
- Observe that $Q_\pi(s_{t-1}, a_t) = \mathbb{E}[u_t]$; that is, if we're at state s_{t-1} and take action a_t, the average value of u_t is $Q_\pi(s_{t-1}, a_t)$.
- But that particular state and action pair (s, a) will probably show up many times. If we take the average of u_t over all the times that $s_{t-1} = s$ and $a_t = a$, then we obtain our Monte Carlo estimate $\hat{Q}_\pi(s, a)$. Note that nowhere do we need to talk about transitions or immediate rewards; the only thing that matters is total rewards resulting from (s, a) pairs.
- One technical note is that for simplicity, we only consider $s_{t-1} = s, a_t = a$ for which the (s, a) doesn't show up later. This is not necessary for the algorithm to work, but it is easier to analyze and think about.
- Model-free Monte Carlo depends strongly on the policy π that is followed; after all it's computing Q_π. Because the value being computed is dependent on the policy used to generate the data, we call this an **on-policy** algorithm. In contrast, model-based Monte Carlo is **off-policy**, because the model we estimated did not depend on the exact policy (as long as it was able to explore all (s, a) pairs).

Model-free Monte Carlo (equivalences)

Data (following policy π):

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$$

Original formulation

$\hat{Q}_\pi(s,a) =$ average of u_t where $s_{t-1} = s, a_t = a$

Equivalent formulation (convex combination)

On each (s,a,u):

$$\eta = \frac{1}{1 + (\#\text{ updates to } (s,a))}$$

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$

[whiteboard: u_1, u_2, u_3]

- Over the next few slides, we will interpret model-free Monte Carlo in several ways. This is the same algorithm, just viewed from different perspectives. This will give us some more intuition and allow us to develop other algorithms later.
- The first interpretation is thinking in terms of **interpolation**. Instead of thinking of averaging as a batch operation that takes a list of numbers (realizations of u_t) and computes the mean, we can view it as an iterative procedure for building the mean as new numbers are coming in.
- In particular, it's easy to work out for a small example that averaging is equivalent to just interpolating between the old value $\hat{Q}_\pi(s, a)$ (current estimate) and the new value u (data). The interpolation ratio η is set carefully so that u contributes exactly the right amount to the average.
- Advanced: in practice, we would constantly improve the policy π constantly over time. In this case, we might want to set η to something that doesn't decay as quickly (for example, $\eta = 1/\sqrt{\text{\# updates to } (s, a)}$). This rate implies that a new example contributes more than an old example, which is desirable so that $\hat{Q}_\pi(s, a)$ reflects the more recent policy rather than the old policy.

Model-free Monte Carlo (equivalences)

Equivalent formulation (convex combination)

On each (s, a, u):

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$

Equivalent formulation (stochastic gradient)

On each (s, a, u):

$$\hat{Q}_\pi(s,a) \leftarrow \hat{Q}_\pi(s,a) - \eta[\underbrace{\hat{Q}_\pi(s,a)}_{\text{prediction}} - \underbrace{u}_{\text{target}}]$$

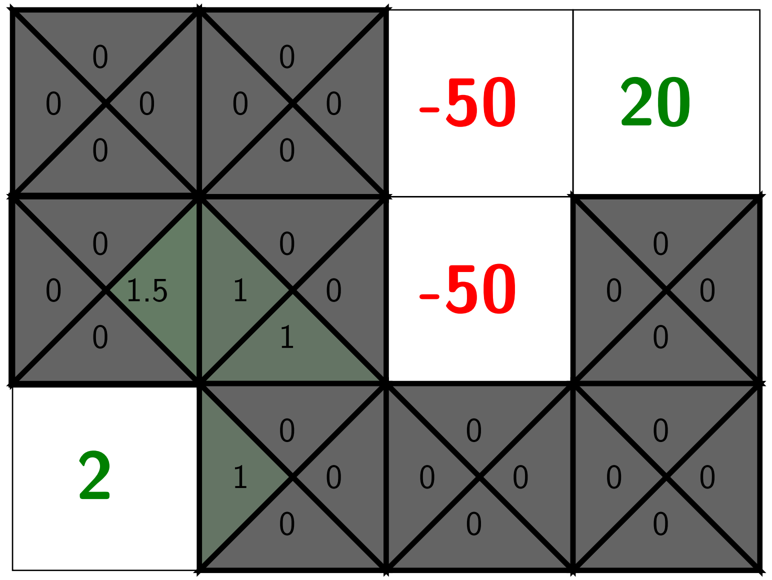
Implied objective: least squares regression

$$(\hat{Q}_\pi(s,a) - u)^2$$

- The second equivalent formulation is making the update look like a stochastic gradient update. Indeed, if we think about each (s, a, u) triple as an example (where (s, a) is the input and u is the output), then the model-free Monte Carlo is just performing stochastic gradient descent on a least squares regression problem, where the weight vector is $\hat{Q}_\pi$ (which has dimensionality SA) and there is one feature template "(s, a) equals ___".
- The stochastic gradient descent view will become particularly relevant when we use non-trivial features on (s, a).

Volcanic model-free Monte Carlo

Run (or press ctrl-enter)



a	r	s
		(2,1)
E	0	(2,2)
W	0	(2,1)
E	0	(2,2)
S	0	(3,2)
W	2	(3,1)

Utility: 2

- Let's run model-free Monte Carlo on the volcano crossing example. `slipProb` is zero to make things simpler. We are showing the Q-values: for each state, we have four values, one for each action.
- Here, our exploration policy is one that chooses an action uniformly at random.
- Try pressing "Run" multiple times to understand how the Q-values are set.
- Then try increasing `numEpisodes`, and seeing how the Q-values of this policy become more accurate.
- You will notice that a random policy has a very hard time reaching the 20.