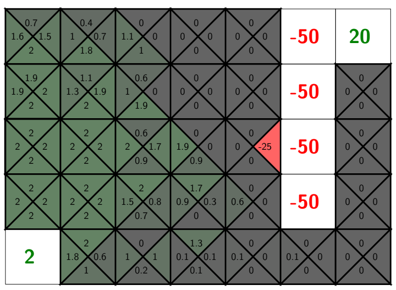
MDPs: function approximation

Generalization

Problem: large state spaces, hard to explore



a	r	s
		(3,1)
N	0	(2,1)
N	0	(1,1)
E	0	(1,2)
W	0	(1,1)
S	0	(2,1)
E	0	(2,2)
S	0	(3,2)
W	0	(3,1)
S	0	(4,1)
S	2	(5,1)

Average (lifetime) utility: 1.48

- Now we turn to another problem with vanilla Q-learning.
- In real applications, there can be millions of states, in which there's no hope for epsilon-greedy to explore everything in a reasonable amount of time.

Q-learning

Stochastic gradient update:

$$\hat{Q}_{\text{opt}}(s,a) \leftarrow \hat{Q}_{\text{opt}}(s,a) - \eta[\underbrace{\hat{Q}_{\text{opt}}(s,a)}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}]$$

This is **rote learning**: every $\hat{Q}_{\text{opt}}(s,a)$ has a different value

Problem: doesn't generalize to unseen states/actions

- If we revisit the Q-learning algorithm, and think about it through the lens of machine learning, you'll find that we've just been memorizing Q-values for each (s,a), treating each pair independently.
- In other words, we haven't been generalizing, which is actually one of the most important aspects of learning!

Function approximation

Key idea: linear regression model

Define **features** $\phi(s, a)$ and **weights** $\mathbf{w}$:
$\hat{Q}_{\text{opt}}(s, a; \mathbf{w}) = \mathbf{w} \cdot \phi(s, a)$

Example: features for volcano crossing

$\phi_1(s, a) = \mathbf{1}[a = \mathsf{W}]$ $\qquad$ $\phi_7(s, a) = \mathbf{1}[s = (5, *)]$
$\phi_2(s, a) = \mathbf{1}[a = \mathsf{E}]$ $\qquad$ $\phi_8(s, a) = \mathbf{1}[s = (*, 6)]$
... $\qquad$ ...

- **Function approximation** fixes this by parameterizing $\hat{Q}_{\text{opt}}$ by a weight vector and a feature vector, as we did in linear regression.
- Recall that features are supposed to be properties of the state-action (s, a) pair that are indicative of the quality of taking action a in state s.
- The ramification is that all the states that have similar features will have similar Q-values. For example, suppose ϕ included the feature $\mathbf{1}[s = (*, 4)]$. If we were in state $(1, 4)$, took action E, and managed to get high rewards, then Q-learning with function approximation will propagate this positive signal to all positions in column 4 taking any action.
- In our example, we defined features on actions (to capture that moving east is generally good) and features on states (to capture the fact that the 6th column is best avoided, and the 5th row is generally a good place to travel to).

Function approximation

Algorithm: Q-learning with function approximation

On each (s, a, r, s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}]\phi(s, a)$$

Implied objective function:

$$(\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}})^2$$

- We now turn our linear regression into an algorithm. Here, it is useful to adopt the stochastic gradient view of RL algorithms, which we developed a while back.
- We just have to write down the least squares objective and then compute the gradient with respect to $\mathbf{w}$ now instead of $\hat{Q}_{\text{opt}}$. The chain rule takes care of the rest.