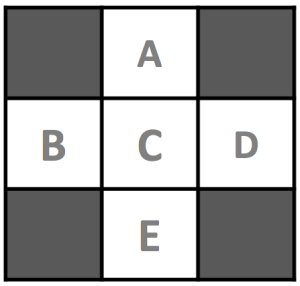
1 Temporal Difference Learning and Q-Learning

Consider the Gridworld example that we looked at in lecture. We would like to use TD learning to find the values of these states.



Suppose we observe the following $(s, a, s', R(s, a, s'))$* transitions and rewards:

(B, East, C, 2), (C, South, E, 4), (C, East, A, 6), (B, East, C, 2)

**Note that the $R(s, a, s')$ in this notation refers to observed reward, not a reward value computed from a reward function (because we don't have access to the reward function).*

The initial value of each state is 0. Let $\gamma = 1$ and $\alpha = 0.5$.

(a) What are the learned values for each state from TD learning after all four observations?

(b) In class, we presented the following two formulations for TD-learning:

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample \tag{1}$$

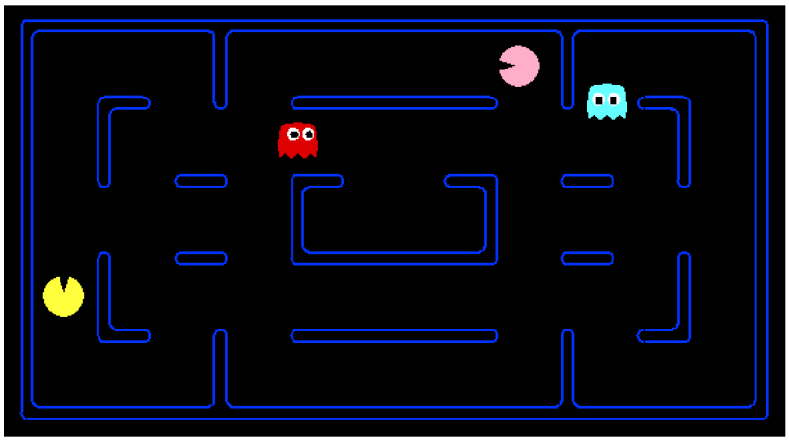
$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s)) \tag{2}$$

Mathematically, these two equations are equivalent. However, they represent two conceptually different ways of understanding TD value updates. How could we intuitively explain each of these equations?

(c) What are the learned Q-values from Q-learning after all four observations? Use the same $\alpha = 0.5, \gamma = 1$ as before.

2 Approximate Q-Learning

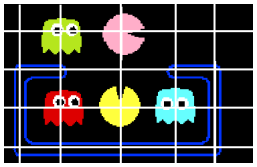
Maia and Claire are training agents to play AI eTag, which is totally different from Pacman. In this game, the player must find the other player in a maze. However, there are phantoms (note: these are not ghosts) that both players must avoid. However, it is unknown what the scores are for staying alive, for being caught by a phantom, or for finding the other player. Here's a sample board:



We want to apply Q-learning to this game to train our agents, but it takes a lot of memory to hold the entire grid. To remedy this, we switch to feature-based representation.

(a) What features would you extract from an eTag board to judge the expected outcome of the game?

(b) Say our two features are the number of phantoms in one step of our agent (F_p) and the Manhattan distance to our friend (F_d). Consider the state from the perspective of the yellow agent (the bottom agent). What are the feature values for the following board (note: the gridlines are drawn to help measure distance by eye):



(c) When we get to this state, we have learned weights $w_p = 100$ and $w_d = 10$. Calculate the Q value for the state above.

(d) We have received an episode, which is a start state s, an action a, and an end state s', and a reward $R(s, a, s')$. Now, we must update the values. The start state is the state above, and the next state has feature values $F_p = 3$ and $F_d = 1$, and the reward was 20. Assuming discount factor of 0.5, calculate the new estimate of the Q-value for s' **based on the sample**, i.e. $R(s, a, s') + \gamma max_{a'} Q(s', a')$.

(e) Now let's update the weights for each feature, given that our learning rate α is 0.5.

(f) So from a high-level view, what exactly did we learn here? Once we finish learning, how can we evaluate our agents' performances?

3 RL: Conceptual Questions

Recall that in Q-learning, we continually update the values of each Q-state by learning through a series of episodes, ultimately converging upon the optimal policy.

(a) What's the main shortcoming of TD learning that Q-learning resolves?

(b) We are given a pre-existing table of current estimate of Q-values (and its corresponding policy), and asked to perform ϵ-greedy Q-learning. Individually, what effect does setting each of the following constants to 0 have on this process?

(i) α:

(ii) γ:

(iii) ϵ:

(c) Consider a variant of the ϵ-greedy Q-learning algorithm that is changed such that instead of using the policy extracted from our current Q-values, we use a fixed policy instead. We still perform exploration with probability ϵ. If this fixed policy happens to be optimal, how does the performance of this algorithm compare to normal ϵ-greedy Q-learning?

(d) Let's revisit the CandyGrab code. What RL strategies does `AgentRL` employ? Does it evaluate states or Q-states?

(e) (Bonus) Using your knowledge of the game and potential strategies, think about some useful features an approximate Q-learning agent might for CandyGrab, and try coding up such an agent. Some starter code has been provided to you in `agentApprox.py`.

(f) Contrast the following pairs of reinforcement learning terms:

(i) Off-policy vs. on-policy learning

(ii) Model-based vs. model-free

(iii) Passive vs. active