CS 188
Spring 2019

Section Handout 5

Approximate Q-Learning

With feature vectors, we can treat values of states and q-states as **linear value functions**:

$$\begin{aligned} V(s) &= w_1 \cdot f_1(s) + w_2 \cdot f_2(s) + \ldots + w_n \cdot f_n(s) = \vec{w} \cdot \vec{f}(s) \\ Q(s,a) &= w_1 \cdot f_1(s,a) + w_2 \cdot f_2(s,a) + \ldots + w_n \cdot f_n(s,a) = \vec{w} \cdot \vec{f}(s,a) \end{aligned}$$

where $\vec{f}(s) = \begin{bmatrix} f_1(s) & f_2(s) & \ldots & f_n(s) \end{bmatrix}^T$ and $\vec{f}(s,a) = \begin{bmatrix} f_1(s,a) & f_2(s,a) & \ldots & f_n(s,a) \end{bmatrix}^T$ represent the feature vectors for state s and q-state (s,a) respectively and $\vec{w} = \begin{bmatrix} w_1 & w_2 & \ldots & w_n \end{bmatrix}$ represents a weight vector. Defining `difference` as

$$\texttt{difference} = [R(s,a,s') + \gamma \max_{a'} Q(s',a')] - Q(s,a)$$

approximate Q-learning works almost identically to Q-learning, using the following update rule:

$$w_i \leftarrow w_i + \alpha \cdot \texttt{difference} \cdot f_i(s,a)$$

CSPs

CSPs are defined by three factors:

1. *Variables* - CSPs possess a set of N variables $X_1, ..., X_N$ that can each take on a single value from some defined set of values.
2. *Domain* - A set $\{x_1, ..., x_d\}$ representing all possible values that a CSP variable can take on.
3. *Constraints* - Constraints define restrictions on the values of variables, potentially with regard to other variables.

CSPs are often represented as constraint graphs, where nodes represent variables and edges represent constraints between them.

- *Unary Constraints* - Unary constraints involve a single variable in the CSP. They are not represented in constraint graphs, instead simply being used to prune the domain of the variable they constrain when necessary.
- *Binary Constraints* - Binary constraints involve two variables. They're represented in constraint graphs as traditional graph edges.
- *Higher-order Constraints* - Constraints involving three or more variables can also be represented with edges in a CSP graph.

In **forward checking**, whenever a value is assigned to a variable X_i, forward checking prunes the domains of unassigned variables that share a constraint with X_i that would violate the constraint if assigned. The idea of

forward checking can be generalized into the principle of **arc consistency**. For arc consistency, we interpret each undirected edge of the constraint graph for a CSP as two directed edges pointing in opposite directions. Each of these directed edges is called an **arc**. The arc consistency algorithm works as follows:

- Begin by storing all arcs in the constraint graph for the CSP in a queue Q.
- Iteratively remove arcs from Q and enforce the condition that in each removed arc $X_i \longrightarrow X_j$, for every remaining value v for the tail variable X_i, there is at least one remaining value w for the head variable X_j such that $X_i = v, X_j = w$ does not violate any constraints. If some value v for X_i would not work with any of the remaining values for X_j, we remove v from the set of possible values for X_i.
- If at least one value is removed for X_i when enforcing arc consistency for an arc $X_i \longrightarrow X_j$, add arcs of the form $X_k \longrightarrow X_i$ to Q, for all unassigned variables X_k. If an arc $X_k \longrightarrow X_i$ is already in Q during this step, it doesn't need to be added again.
- Continue until Q is empty, or the domain of some variable is empty and triggers a backtrack.

We've delineated that when solving a CSP, we fix some ordering for both the variables and values involved. In practice, it's often much more effective to compute the next variable and corresponding value "on the fly" with two broad principles, **minimum remaining values** and **least constraining value**:

- *Minimum Remaining Values (MRV)* - When selecting which variable to assign next, using an MRV policy chooses whichever unassigned variable has the fewest valid remaining values (the *most constrained variable*).
- *Least Constraining Value (LCV)* - Similarly, when selecting which value to assign next, a good policy to implement is to select the value that prunes the fewest values from the domains of the remaining unassigned values.

Q1. Pacman with Feature-Based Q-Learning

We would like to use a Q-learning agent for Pacman, but the size of the state space for a large grid is too massive to hold in memory. To solve this, we will switch to feature-based representation of Pacman's state.

(a) We will have two features, F_g and F_p, defined as follows:

$$F_g(s,a) = A(s) + B(s,a) + C(s,a)$$
$$F_p(s,a) = D(s) + 2E(s,a)$$

where

$$\begin{aligned} A(s) &= \text{number of ghosts within 1 step of state } s \\ B(s,a) &= \text{number of ghosts Pacman touches after taking action } a \text{ from state } s \\ C(s,a) &= \text{number of ghosts within 1 step of the state Pacman ends up in after taking action } a \\ D(s) &= \text{number of food pellets within 1 step of state } s \\ E(s,a) &= \text{number of food pellets eaten after taking action } a \text{ from state } s \end{aligned}$$

For this pacman board, the ghosts will always be stationary, and the action space is $\{left, right, up, down, stay\}$.

calculate the features for the actions $\in \{left, right, up, stay\}$

(b) After a few episodes of Q-learning, the weights are $w_g = -10$ and $w_p = 100$. Calculate the Q value for each action $\in \{left, right, up, stay\}$ from the current state shown in the figure.

(c) We observe a transition that starts from the state above, s, takes action up, ends in state s' (the state with the food pellet above) and receives a reward $R(s,a,s') = 250$. The available actions from state s' are $down$ and $stay$. Assuming a discount of $\gamma = 0.5$, calculate the new estimate of the Q value for s based on this episode.

(d) With this new estimate and a learning rate (α) of 0.5, update the weights for each feature.

Q2. MDPs and RL

Recall that in approximate Q-learning, the Q-value is a weighted sum of features: $Q(s,a) = \sum_i w_i f_i(s,a)$. To derive a weight update equation, we first defined the loss function $L_2 = \frac{1}{2}(y - \sum_k w_k f_k(x))^2$ and found $dL_2/dw_m = -(y - \sum_k w_k f_k(x)) f_m(x)$. Our label y in this set up is $r + \gamma \max_a Q(s', a')$. Putting this all together, we derived the gradient descent update rule for w_m as $w_m \leftarrow w_m + \alpha \left(r + \gamma \max_a Q(s', a') - Q(s,a)\right) f_m(s,a)$.

In the following question, you will derive the gradient descent update rule for w_m using a different loss function:

$$L_1 = \left| y - \sum_k w_k f_k(x) \right|$$

(a) Find dL_1/dw_m. Ignore the non-differentiable point.

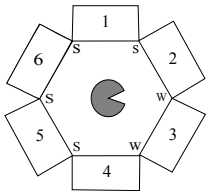
(b) Write the gradient descent update rule for w_m, using the L_1 loss function.

Q3. CSPs: Trapped Pacman

Pacman is trapped! He is surrounded by mysterious corridors, each of which leads to either a pit (P), a ghost (G), or an exit (E). In order to escape, he needs to figure out which corridors, if any, lead to an exit and freedom, rather than the certain doom of a pit or a ghost.

The one sign of what lies behind the corridors is the wind: a pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce any breeze at all. Unfortunately, Pacman cannot measure the strength of the breeze at a specific corridor. Instead, he can stand *between* two adjacent corridors and feel the max of the two breezes. For example, if he stands between a pit and an exit he will sense a strong (S) breeze, while if he stands between an exit and a ghost, he will sense a weak (W) breeze. The measurements for all intersections are shown in the figure below.

Also, while the total number of exits might be zero, one, or more, Pacman knows that two neighboring squares will *not* both be exits.



Pacman models this problem using variables X_i for each corridor i and domains P, G, and E.

(a) State the binary and/or unary constraints for this CSP (either implicitly or explicitly).

(b) Suppose we assign X_1 to E. Perform forward checking after this assignment. Also, enforce unary constraints.

X_1			E
X_2	P	G	E
X_3	P	G	E
X_4	P	G	E
X_5	P	G	E
X_6	P	G	E

(c) Suppose forward checking returns the following set of possible assignments:

X_1	P		
X_2		G	E
X_3		G	E
X_4		G	E
X_5	P		
X_6	P	G	E

According to MRV, which variable or variables could the solver assign first?

(d) Assume that Pacman knows that $X_6 = G$. List all the solutions of this CSP or write *none* if no solutions exist.