6. Q-learning with linear function approximation

Extension Note: Project 5 due date has been extended by 1 **more** day to **September 6 23:59UTC** .

Since the state displayed to the agent is described in text, we have to choose a mechanism that maps text descriptions into vector representations. A naive way is to create one unique index for each text description, as we have done in previous part. However, such approach becomes infeasible when the state space becomes huge. To tackle this challenge, we can design some representation generator that does not scale as the original textual state space. In particular, a representation generator $\psi_R(\cdot)$ reads raw text displayed to the agent and converts it to a vector representation $v_s = \psi_R(s)$. One approach is to use a bag-of-words representation derived from the text description.

In large games, it is often impractical to maintain the Q-value for all possible state-action pairs. One solution to this problem is to approximate $Q(s,c)$ using a parametrized function $Q(s,c;\theta)$.

In this section we consider a linear parametric architecture:

$$Q(s,c;\theta) = \phi(s,c)^T\theta = \sum_{i=1}^{d} \phi_i(s,c)\,\theta_i,$$

where $\phi(s,c)$ is a fixed feature vector in $\mathbb{R}^d$ for state-action pair (s,c) with i-th component given by $\phi_i(s,c)$, and $\theta \in \mathbb{R}^d$ is a parameter vector that is shared across state-action pairs. The key challenge here is to design the feature vectors $\phi(s,c)$. Note that given a textual state s, we first translate it to a vector representation v_s using $\psi_R(s)$. So the question here is how to design a mapping function convert $(\psi_R(s), c)$ into a vector representation in $\mathbb{R}^d$. Assume that the size of action space is d_C, and the dimension of the vector space for state representation is d_R.

Feature engineering

1/1 point (graded)

Exercise: Consider the following feature engineering. Define a function $\psi_C : \mathcal{C} \to \mathbb{R}^{d_C}$ where the j-th component $\psi_{C,j}(c)$ is given as follows:

$$\psi_{C,j}(c) = \begin{cases} 1 & \text{if } j = c \\ 0 & \text{else} \end{cases}$$

The feature vector is defined as

$$\phi(s,c) = \begin{bmatrix} \psi_R(s) \\ \psi_C(c) \end{bmatrix}.$$

Will it work?

Yes [✓ No]

✓

Solution:

No. Let us fix the parameters θ. By the definition, for each state s:

$$\begin{aligned}\max_c Q(s,c,\theta) &= \max_c \sum_{i=1}^{d} \phi_i(s,c)\,\theta_i = \max_c \Big\{ \sum_{i=1}^{d_R} \psi_{R,i}(s)\,\theta_i + \sum_{j=1}^{d_C} \psi_{C,j}(c)\,\theta_{j+d_R} \Big\} \\ &= \max_c \left\{ \sum_{j=1}^{d_C} \psi_{C,j}(c)\,\theta_{j+d_R} \right\} \\ &= \max_c \theta_{c+d_R}.\end{aligned}$$

Note that the action that maximizes the Q-function for each state does not depend on the state s. In particular, for each learned parameter θ, the optimal action for all states are the same. Such policy is not optimal in our setting. Therefore, the above feature engineering is not sufficient to learn a good approximation of the optimal Q-function.

Submit You have used 1 of 4 attempts

Answers are displayed within the problem

Alternatively, consider the following feature map: $\phi(s,c) \in \mathbb{R}^{d_C \cdot d_R}$, where $\phi_i(s,c) = 0$ for all $i \notin [(c-1)\cdot d_R + 1,\ c \cdot d_R]$, and for $i \in [(c-1)\cdot d_R + 1,\ c\cdot d_R]$, $\phi_i(s,c) = \psi_{R,i-(c-1)d_R}(s)$. That is,

$$\phi(s,c) = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \psi_R(s) \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$$

You will implement this feature map in the next tab.

Computing theta update rule

1.0/1 point (graded)

The Q-learning approximation algorithm starts with an initial parameter estimate of θ. As the tabular Q-learning, upon observing a data tuple $(s, c, R(s,c), s')$ the target value y for the Q-value of (s,c) is defined as the sampled version of the Bellman operator,

$$y = R(s,c) + \gamma \max_{c'} Q(s', c', \theta).$$

Then the parameter θ is simply updated by taking a gradient step with respect to the squared loss

$$L(\theta) = \frac{1}{2}(y - Q(s,c,\theta))^2.$$

The negative gradient can be computed as follows:

(Enter your answer in terms of `y`, `Q(s, c, theta)`, and `phi(s, c)`.)

$g(\theta) = -\frac{\partial}{\partial\theta} L(\theta) =$ phi(s,c)*(y - Q(s,c,theta)) ✓ **Answer:** (y - Q(s, c, theta))*phi(s, c)

Solution:

The negative gradient can be computed as follows:

$$g(\theta) = -\frac{\partial}{\partial\theta}L(\theta) = (y - Q(s,c,\theta)) \cdot \frac{\partial}{\partial\theta}Q(s,c,\theta) = (y - Q(s,c,\theta))\,\phi(s,c)$$

Submit You have used 1 of 6 attempts

Answers are displayed within the problem

Hence the update rule for θ is :

$$\theta \quad \leftarrow \theta + \alpha g(\theta) = \theta + \alpha\left[R(s,c) + \gamma \max_{c'} Q(s',c',\theta) - Q(s,c,\theta)\right]\phi(s,c)\,,$$

where α is the learning rate.

Discussion

Show Discussion

Topic: Unit 5 Reinforcement Learning (2 weeks) :Project 5: Text-Based Game / 6. Q-learning with linear function approximation