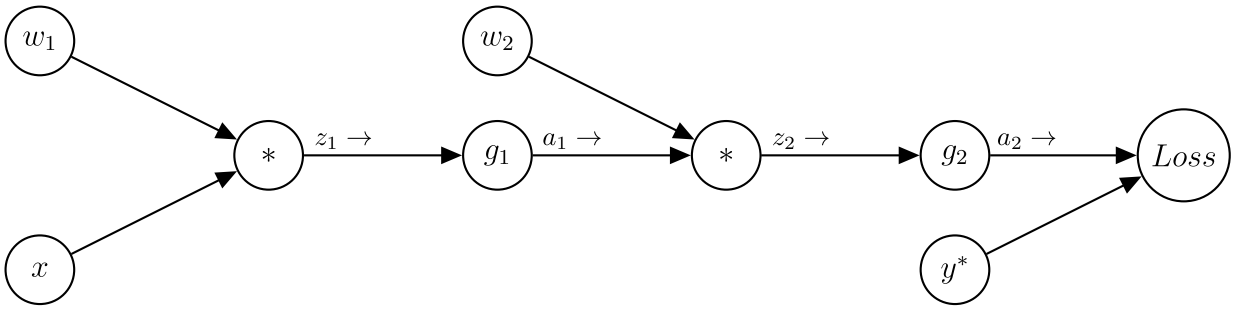
CS 188
Fall 2021

Final Review: ML II Solutions

Q1. Neural Nets

Consider the following computation graph for a simple neural network for binary classification. Here x is a single real-valued input feature with an associated class $y^\star$ (0 or 1). There are two weight parameters w_1 and w_2, and non-linearity functions g_1 and g_2 (to be defined later, below). The network will output a value a_2 between 0 and 1, representing the probability of being in class 1. We will be using a loss function $Loss$ (to be defined later, below), to compare the prediction a_2 with the true class $y^\star$.



1. Perform the forward pass on this network, writing the output values for each node z_1, a_1, z_2 and a_2 in terms of the node's input values:

$$z_1 = x * w_1$$
$$a_1 = g_1(z_1)$$
$$z_2 = a_1 * w_2$$
$$a_2 = g_2(z_2)$$

2. Compute the loss $Loss(a_2, y^\star)$ in terms of the input x, weights w_i, and activation functions g_i:

Recursively substituting the values computed above, we have:

$$Loss(a_2, y^\star) = Loss(g_2(w_2 * g_1(w_1 * x)), y^\star)$$

3. Now we will work through parts of the backward pass, incrementally. Use the chain rule to derive $\frac{\partial Loss}{\partial w_2}$. Write your expression as a product of partial derivatives at each node: i.e. the partial derivative of the node's output with respect to its inputs. (Hint: the series of expressions you wrote in part 1 will be helpful; you may use any of those variables.)

$$\frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

4. Suppose the loss function is quadratic, $Loss(a_2, y^\star) = \frac{1}{2}(a_2 - y^\star)^2$, and g_1 and g_2 are both sigmoid functions $g(z) = \frac{1}{1+e^{-z}}$ (note: it's typically better to use a different type of loss, *cross-entropy*, for classification problems, but we'll use this to make the math easier).

   Using the chain rule from Part 3, and the fact that $\frac{\partial g(z)}{\partial z} = g(z)(1 - g(z))$ for the sigmoid function, write $\frac{\partial Loss}{\partial w_2}$ in terms of the values from the forward pass, $y^\star$, a_1, and a_2:
   First we'll compute the partial derivatives at each node:

$$\begin{aligned}\frac{\partial Loss}{\partial a_2} &= (a_2 - y^\star)\\ \frac{\partial a_2}{\partial z_2} &= \frac{\partial g_2(z_2)}{\partial z_2} = g_2(z_2)(1 - g_2(z_2)) = a_2(1 - a_2)\\ \frac{\partial z_2}{\partial w_2} &= a_1\end{aligned}$$

   Now we can plug into the chain rule from part 3:

$$\begin{aligned}\frac{\partial Loss}{\partial w_2} &= \frac{\partial Loss}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial w_2}\\ &= (a_2 - y^\star) * a_2(1 - a_2) * a_1\end{aligned}$$

5. Now use the chain rule to derive $\frac{\partial Loss}{\partial w_1}$ as a product of partial derivatives at each node used in the chain rule:

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

6. Finally, write $\frac{\partial Loss}{\partial w_1}$ in terms of $x, y^\star, w_i, a_i, z_i$: The partial derivatives at each node (in addition to the ones we computed in Part 4) are:

$$\begin{aligned}\frac{\partial z_2}{\partial a_1} &= w_2\\ \frac{\partial a_1}{\partial z_1} &= \frac{\partial g_1(z_1)}{\partial z_1} = g_1(z_1)(1 - g_1(z_1)) = a_1(1 - a_1)\\ \frac{\partial z_1}{\partial a_1} &= x\end{aligned}$$

   Plugging into the chain rule from Part 5 gives:

$$\begin{aligned}\frac{\partial Loss}{\partial w_1} &= \frac{\partial Loss}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1}\\ &= (a_2 - y^\star) * a_2(1 - a_2) * w_2 * a_1(1 - a_1) * x\end{aligned}$$

7. What is the gradient descent update for w_1 with step-size α in terms of the values computed above?

$$w_1 \leftarrow w_1 - \alpha(a_2 - y^\star) * a_2(1 - a_2) * w_2 * a_1(1 - a_1) * x$$

Q2. Deep "Blackjack"

To celebrate the end of the semester, you visit Las Vegas and decide to play a good, old fashioned game of "Blackjack"!

Recall that the game has states 0,1,...,8, corresponding to dollar amounts, and a *Done* state where the game ends. The player starts with \$2, i.e. at state 2. The player has two actions: Stop ($a = 0$) and Roll ($a = 1$), and is forced to take the Stop action at states 0,1,and 8.

When the player takes the Stop action ($a = 0$), they transition to the *Done* state and receive reward equal to the amount of dollars of the state they transitioned from: e.g. taking the stop action at state 3 gives the player \$3. The game ends when the player transitions to *Done*.

The Roll action ($a = 1$) is available from states 2-7. The player rolls a **biased** 6-sided die. If the player Rolls from state s and the die lands on outcome o, the player transitions to state $s + o - 2$, as long as $s + o - 2 \leq 8$ (s is the amount of dollars of the current state, o is the amount rolled, and the negative 2 is the price to roll). If $s + o - 2 > 8$, the player busts, i.e. transitions to Done and does NOT receive reward.

As the bias of the dice **is unknown**, you decided to perform some good-old fashioned reinforcement learning (RL) to solve the game. However, unlike in the midterm, you have decided to flex and solve the game using approximate Q-learning. Not only that, you decided not to design any features - the features for the Q-value at (s, a) will simply be the vector $[s\ a]$, where s is the state and a is the action.

(a) First, we will investigate how your choice of features impacts whether or not you can learn the optimal policy. Suppose the unique optimal policy in the MDP is the following:

State	2	3	4	5	6	7
$\pi^*(s)$	Roll	Roll	Roll	Stop	Stop	Stop

For each of the cases below, select "Possible with large neural net" if the policy can be expressed by using a large neural net to represent the Q-function using the features specified as input. (That is, the greedy policy with respect to some Q-function representable with a large neural network is the optimal policy: $Q(s, \pi^*(s)) > Q(s, a)$ for all states s and actions $a \neq \pi^*(s)$.) Select "Possible with weighted sum" if the policy can be expressed by using a weighted linear sum to represent the Q-function. Select "Not Possible" if expressing the policy with given features is impossible no matter the function.

(i) Suppose we decide to use the state s and action a as the features for $Q(s, a)$.

[✓ Possible with large neural network] ☐ Possible with linear weighted sum of features ○ Not possible

A sufficiently large neural network could represent the true optimal Q-function using this feature representation. The optimal Q-function satisfies the desired property (there are no ties as the optimal policy is unique). Alternatively, a sufficiently large neural network could represent a function that is 1 for the optimal action in each state, and 0 otherwise, which also suffices.

No linear weighted sum of features can represent this optimal policy. To see this, let our linear weighted sum be $w_0 s + w_1 a + w_3$. We need $Q(4, 1) > Q(4, 0)$ and $Q(5, 0) > Q(5, 1)$. Plugging in the expression for Q, the former inequality requires that $w_1 > 0$. The second inequality requires that $w_1 < 0$, a contradiction. So we cannot represent the policy with a weighted sum of features.

(ii) Now suppose we decide to use $s + a$ as the feature for $Q(s, a)$.

[✓ Possible with large neural network] ☐ Possible with linear weighted sum of features ○ Not possible

Indeed, it's possible that no neural network can represent the optimal Q-function with this feature representation, as $Q(4,1)$ does not have to equal $Q(5,0)$. However, the question is not asking about representing the optimal Q-function, but instead the optimal policy, which merely requires that $Q(s,1) > Q(s,0)$ for $s \leq 4$ and $Q(s,0) > Q(s,1)$ for $s \geq 5$. This can be done with the feature representation. For example the neural network in part (d) can represent the following function (using $w_0 = -5$ and $w_1 = -2$) that represents the optimal policy:

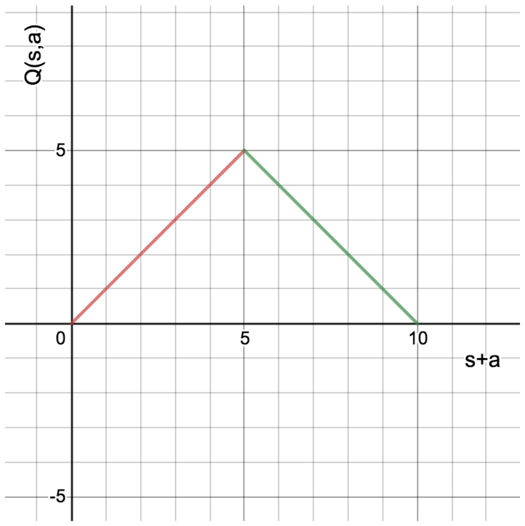


Figure 1: A Q-function from $s + a$, that represents the optimal policy.

Again, no linear weighted sum of features can represent this optimal policy. To see this, let our linear weighted sum be $w_0 s + w_0 a + w_1$. This is a special case of the linear weighted sums in part (i), which we know cannot represent the optimal policy.

(iii) Now suppose we decide to use a as the feature for $Q(s,a)$.

☐ Possible with large neural network ☐ Possible with linear weighted sum of features [✓ Not possible]

This isn't possible, regardless of what function you use. With this representation, we will have $Q(4,1) = Q(5,1)$ and $Q(4,0) = Q(5,0)$. So we cannot both have $Q(4,1) > Q(4,0)$ and $Q(5,0) > Q(5,1)$.

(iv) Now suppose we decide to use $\text{sign}(s - 4.5) \cdot a$ as the feature for $Q(s,a)$, where $\text{sign}(x)$ is -1 if $x < 0$, 1 if $x > 0$, and 0 if $x = 0$.

[✓ Possible with large neural network] [✓ Possible with linear weighted sum of features] ○ Not possible

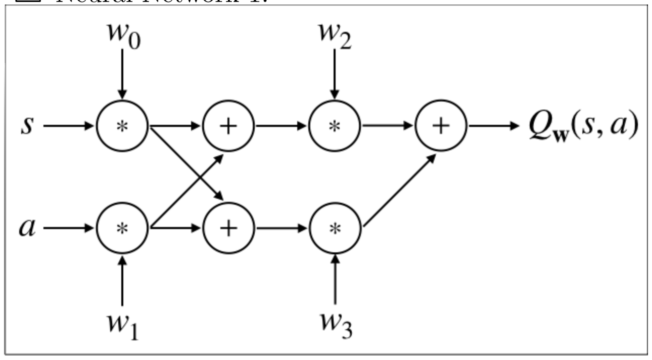
$Q(s,a) = -\text{sign}(s - 4.5) \cdot a$ is sufficient to represent the optimal policy, as we have $Q(s,0) = 0$ for all s, $Q(s,1) = 1 > Q(s,0)$ for $s \leq 4$, and $Q(s,1) = -1 < Q(s,0)$ for $s \geq 5$. This is a linear function of the input, and so can also be represented using a neural network.

(b) Next, we investigate the effect of different neural network architectures on your ability to learn the optimal policy. Recall that our features for the Q-value at (s, a) will simply be the vector $[s\ a]$, where s is the state and a is the action. In addition, suppose that the unique optimal policy is the following:

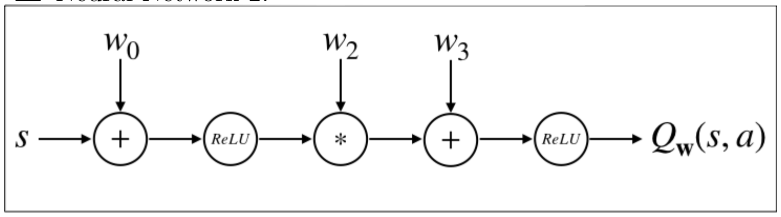
State	2	3	4	5	6	7
$\pi^*(s)$	Roll	Roll	Roll	Stop	Stop	Stop

Which of the following neural network architectures can express Q-values that represent the optimal policy? That is, the greedy policy with respect to some Q-function representable with the given neural network is the optimal policy: $Q(s, \pi^*(s)) > Q(s, a)$ for all states s and actions $a \neq \pi^*(s)$. *Hint: Recall that* $ReLU(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$

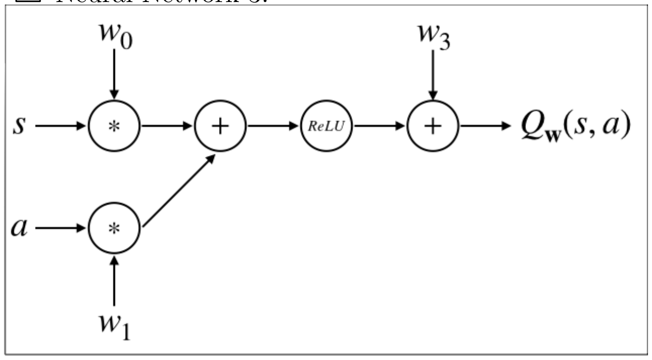
☐ Neural Network 1:



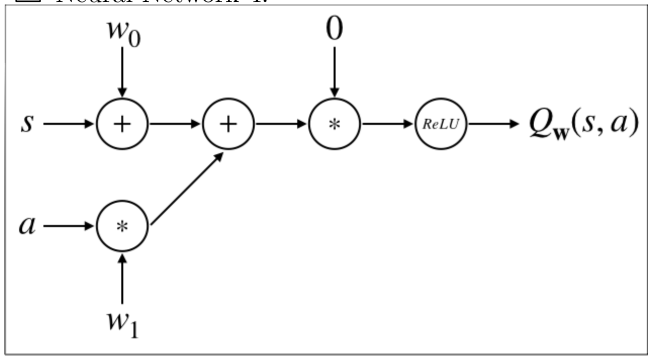
☐ Neural Network 2:



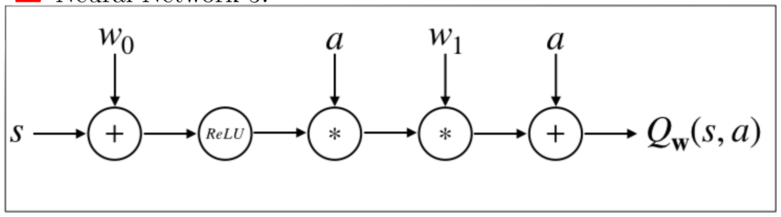
☐ Neural Network 3:



☐ Neural Network 4:



🟥 Neural Network 5:



○ None of the above.

Recall from the previous question that no linear function of the features $[s\ a]$ can represent the optimal policy. So network 1, which is linear (as it has no activation function), cannot represent the optimal policy.

Network 2 cannot represent the optimal function, as it does not take as input the action. So $Q(s,0) = Q(s,1)$ for all states s.

Network 3 cannot simultaneously satisfy $Q(4,0) < Q(4,1)$ and $Q(5,0) > Q(5,1)$. This is because the rectified linear unit is a monotonic function: if $x \geq y$, then $ReLU(x) \geq ReLU(y)$. Since we cannot represent the optimal policy using a linear function of s, a, we cannot represent it with a ReLU of a linear function of s, a.

Network 4 is always 0, so it cannot represent the (unique) optimal policy.

Network 5 can represent the optimal policy. For example, $w_0 = -4$, $w_1 = -2$ represents the optimal policy.

(c) As with the linear approximate q-learning, you decide to minimize the squared error of the Bellman residual. Let $Q_{\mathbf{w}}(s,a)$ be the approximate Q-values of s, a. After taking action a in state s and transitioning to state s' with reward r, you first compute the target target $= r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a')$. Then your loss is:

$$\text{loss}(\mathbf{w}) = \frac{1}{2}\left(Q_{\mathbf{w}}(s,a) - \text{target}\right)^2$$

You then perform gradient descent to **minimize** this loss. Note that we will **not** take the gradient through the target - we treat it as a fixed value.

Which of the following updates represents one step of gradient descent on the weight parameter w_i with learning rate $\alpha \in (0,1)$ after taking action a in state s and transitioning to state s' with reward r? [Hint: which of these is equivalent to the normal approximate Q-learning update when $Q_{\mathbf{w}},(s,a) = \mathbf{w} \cdot \mathbf{f}(s,a)$?]

○ $w_i = w_i + \alpha\left(Q_{\mathbf{w}}(s,a) - (r + \gamma \max_{a'} Q_{\mathbf{w}}(s',a'))\right) \frac{\partial Q_{\mathbf{w}}(s,a)}{\partial w_i}$

● $w_i = w_i - \alpha\left(Q_{\mathbf{w}}(s,a) - (r + \gamma \max_{a'} Q_{\mathbf{w}}(s',a'))\right) \frac{\partial Q_{\mathbf{w}}(s,a)}{\partial w_i}$

○ $w_i = w_i + \alpha\left(Q_{\mathbf{w}}(s,a) - (r + \gamma \max_{a'} Q_{\mathbf{w}}(s',a'))\right) s$

○ $w_i = w_i - \alpha\left(Q_{\mathbf{w}}(s,a) - (r + \gamma \max_{a'} Q_{\mathbf{w}}(s',a'))\right) s$

○ None of the above.

(d) and (e) are on the next page.

Note that the gradient of the loss with respect to the parameter, via the chain rule, is:

$$\left(Q_{\mathbf{w}}(s,a) - \left(r + \gamma \max_{a'} Q_{\mathbf{w}}(s',a')\right)\right) \frac{\partial Q_{\mathbf{w}}(s,a)}{\partial w_i}$$

The second option performs gradient descent, the first option is gradient ascent, and the other options compute the gradient incorrectly.

(d) While programming the neural network, you're getting some bizarre errors. To debug these, you decide to calculate the gradients by hand and compare them to the result of your code.

Suppose your neural network is the following:

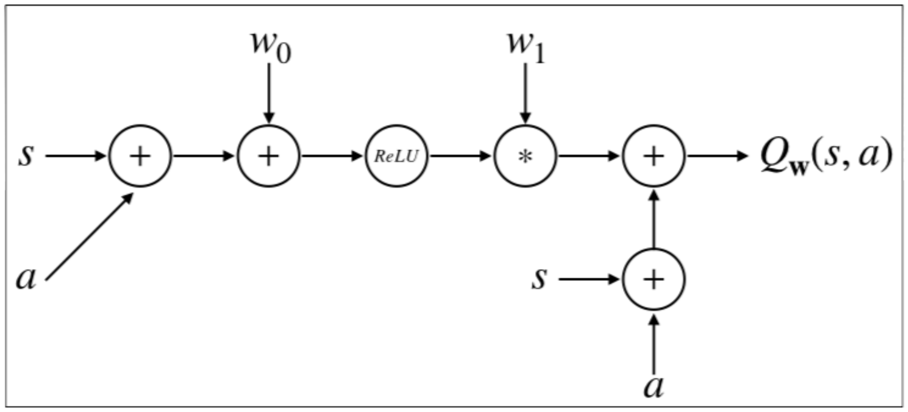


Figure 2: Neural Network 6

That is, $Q_{\mathbf{w}}(s,a) = s + a + w_1\ ReLU(w_0 + s + a)$.

You are able to recall that $\frac{d}{dx}ReLU(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$.

(i) Suppose $w_0 = -4$, and $w_1 = -1$. What is $Q_{\mathbf{w}}(5,0)$?

$Q_{\mathbf{w}}(5,0) = \boxed{4}$

Plugging in values, we get

$$Q_{\mathbf{w}}(5,0) = 5 - ReLU(-4+5) = 4.$$

(ii) Suppose $w_0 = -4$, and $w_1 = -1$. What is the gradient with respect to w_0, evaluated at $s = 5, a = 0$?

$\frac{\partial}{w_0}Q_{\mathbf{w}}(5,0) = \boxed{-1}$

Since the input to the ReLU is positive, the gradient of the ReLU is 1. Applying the chain rule, we get that $\frac{\partial}{w_0}Q_{\mathbf{w}}(5,0) = w_1 = -1$

(iii) Suppose $w_0 = -4$, and $w_1 = -1$. What is the gradient with respect to w_0, evaluated at $s = 3, a = 0$?

$\frac{\partial}{w_0}Q_{\mathbf{w}}(3,0) = \boxed{0}$

Since the input to the ReLU is negative, the gradient of the ReLU is 0. Applying the chain rule, the gradient with respect to w_0 has to be 0.

(e) After picking a feature representation, neural network architecture, and update rule, as well as calculating the gradients, it's time to turn to the age old question... will this even work?

(i) Without any other assumptions, is it guaranteed that your approximate Q-values will converge to the optimal policy, *if* each s, a pair is observed an infinite amount of times?

○ Yes [✓ No]

(ii) Without any other assumptions, is it guaranteed that your approximate Q-values will converge to the optimal policy, *if* each s, a pair is observed an infinite amount of times and there exists some w such that $Q_{\mathbf{w}}(s,a) = Q^*(s,a)$?

○ Yes [✓ No]

Note that there's no guarantee that your neural network will converge in this case. For example, the learning rate can be too large! (As in the RL Blackjack question on midterm.)