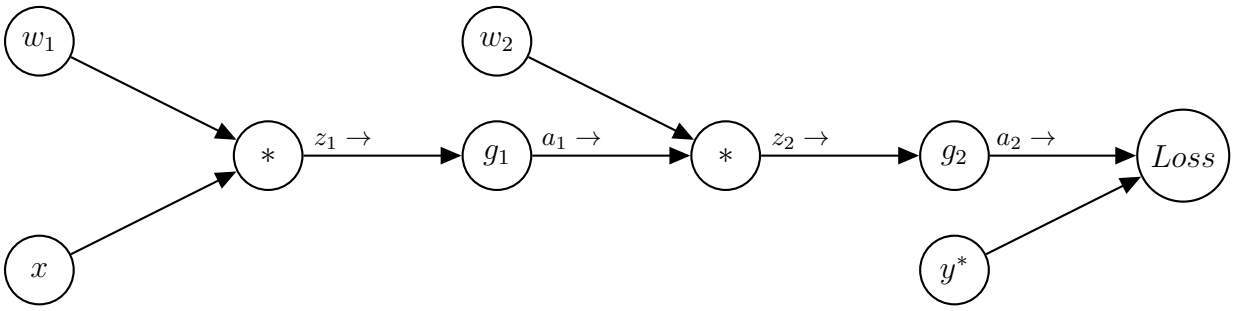
CS 188
Fall 2021

Final Review: Machine Learning II

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:

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

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.)

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:

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:

6. Finally, write $\frac{\partial Loss}{\partial w_1}$ in terms of $x, y^\star, w_i, a_i, z_i$:

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

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

(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

(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

(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 $x = 0$.

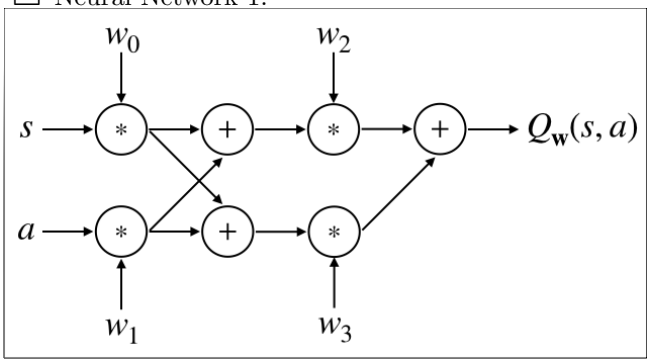
☐ Possible with large neural network ☐ Possible with linear weighted sum of features ◯ Not possible

(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:

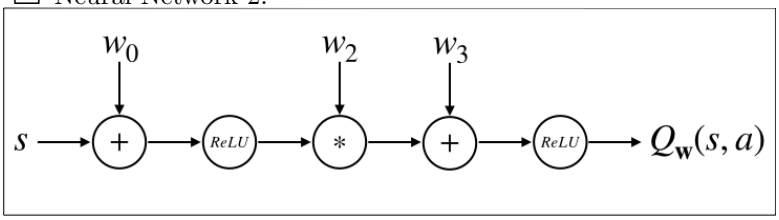
State	2	3	4	5	6	7
$\pi^*(s)$	Roll	Roll	Roll	Stop	Stop	Stop

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* $ReLU(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$

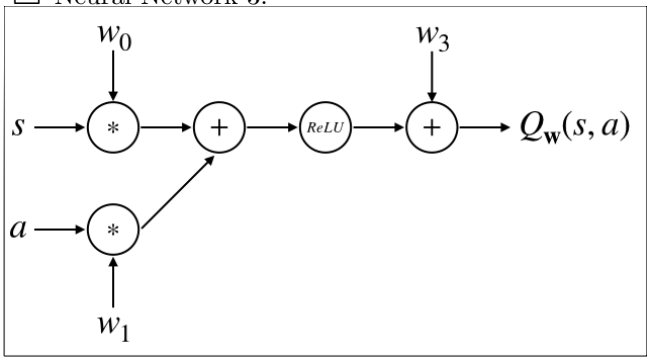
☐ Neural Network 1:



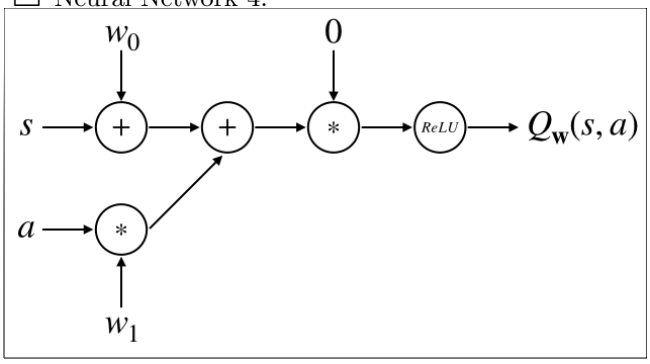
☐ Neural Network 2:



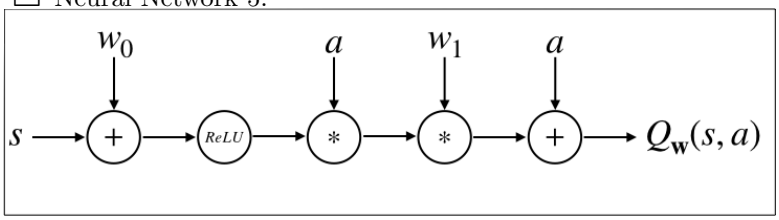
☐ Neural Network 3:



☐ Neural Network 4:



☐ Neural Network 5:



◯ None of the above.

(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.

(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:

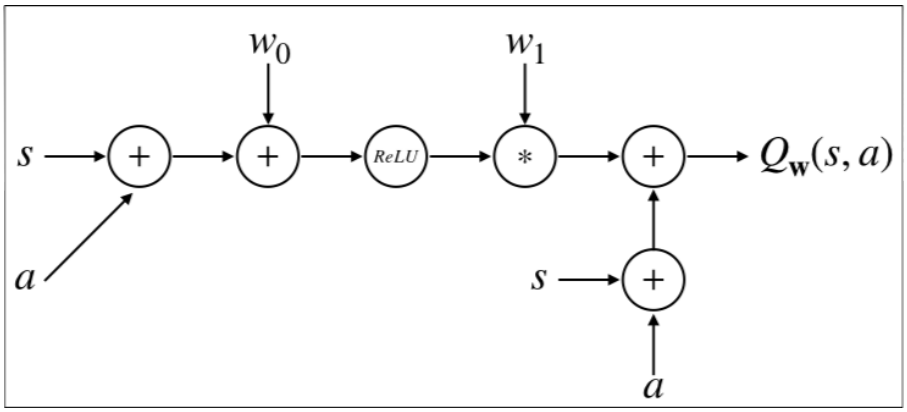


Figure 1: Neural Network 6

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 $Q_{\mathbf{w}}(5,0)$?

$Q_{\mathbf{w}}(5,0) =$ [ ]

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

$\frac{\partial}{w_0}Q_{\mathbf{w}}(5,0) =$ [ ]

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

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

(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