HW 3

Nicole Sullivan

```
knitr::opts_chunk$set(echo = T, message = F, warning = F)
```

1

> Derive the update equations of the regularized MLP using the given activation functions.

The error function is:

$$E(W,v|X) = \sum r^t log y^t + (1-r^t) log(1-y^t) + \sum ||w_h||_m^2$$

First, we'll derive the update equations for the parameters nearest the output (y) layer: the weights for the hidden layer, v_h and v_0 .

v_h & v_0

To derive the gradient of v_h , we can use chain rule: $\Delta v_h = \frac{\partial E}{\partial v_h} \frac{\partial y}{\partial a} \frac{\partial a}{\partial v_h}$. In the case of v_h and v_0 , w_h is merely a constant; therefore, the derivative of the last term in the error function (which gets treated as a constant) is 0.

First, taking the derivative of E w.r.t. y: $\frac{\partial E}{\partial y} = \frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}$

Next, we take the derivative of y w.r.t a, which is actually given to us in Hint 1. $\frac{\partial y}{\partial a} = y^t(1-y^t)$

Finally, we take the derivative of a w.r.t to v_h . From the textbook, we know that $a = \sum v_{jh} z_h + v_0$ the term inside the sigmoid function that outputs y). So the derivative is just z_{jh}^t.

Putting that all together, we get:

$$\begin{aligned}\Delta v_h &= \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial v_h} \\ &= -\sum\left[\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}\right] y^t(1-y^t) z_h^t \\ &= -\sum (r^t - y^t) z_h^t\end{aligned}$$

For v_0, the only difference from the procedure followed above is that the derivative of a w.r.t. v_0 is **1** (recalling that the equation of a is: $a = \sum v_{jh} z_h + v_0$).

Therefore, we have:

$$\begin{aligned}\Delta v_0 &= \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial v_0} \\ &= -\sum (r^t - y^t)\end{aligned}$$

The full update equations for the hidden layer parameters, then, are:

- $v_h = v_h - \eta \Delta v_h = v_h - \eta \sum (r^t - y^t) z_h^t$
- $v_0 = v_0 - \eta \Delta v_0 = v_0 - \eta \sum (r^t - y^t)$

Note that η is the learning rate, which is a hyperparameter set by the user. Using an η that's too small will result in slow convergence, while an η that's too large will result in "bouncing" around the gradient, oftentimes over the minimum, causing a failure to converge.

w_h & w_0

Next, we'll derive the gradient for w_h .

To derive the gradient of w_h , we can again use the chain rule, in conjunction with a simple derivative for the second term in the error function: $\Delta w_h = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial \beta}\frac{\partial \beta}{\partial w_h}$. Since the activation function for the hidden layer, z, is a reLU, we can write the gradients using an if-then statement (which we're given in Hint 2) to represent $\frac{\partial z}{\partial \beta}$, rather than including that term directly in our derivation.

Fortunately, we can also start directly at calculating $\frac{\partial a}{\partial w_h}$ because we've already calculated through the first 2 terms in our derivation of the gradient of v_h .

The derivative of a w.r.t. w_h is just x^t. Next, the derivative of the second term w.r.t w_h is just $\sum 2w_h$. We know this because the L-2 norm is the square root of the sum of squares of the terms within. Since the L-2 norm is squared, that means the term just represents the sum of squares. Writing that out, we get: $w_1^2 + w_2^2 + w_3^2 ... w_d^2$. If we take the derivative w.r.t w_h of that expanded expression, we get: $2w_1 + 2w_2 + 2w_3 ... 2w_d$.

Putting that all together, we get the gradient of w_h to be:

$$\begin{aligned}&\text{if } w^T x^t + w_0 > 0: \\ \Delta w_h &= \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial \beta}\frac{\partial \beta}{\partial w_h} \\ &= -\sum (r^t - y^t) v_h x^t + \sum 2w_h \\ &else: \;\; \Delta w_h = 0\end{aligned}$$

In the case of w_0, the only differences in the gradient from w_h are:

1. The derivative of z w.r.t β would be 1
2. w_h would be treated as a constant, and thus the $\sum 2w_h$ term would zero out

That means that the gradient of w_0 is:

$$\begin{aligned}&\text{if } w^T x^t + w_0 > 0: \\ \Delta w_0 &= \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial \beta}\frac{\partial \beta}{\partial w_0} \\ &= -\sum (r^t - y^t) v_h \\ &else: \;\; \Delta w_0 = 0\end{aligned}$$

Thus, the full update equations for w_h and w_0 are:

- *if* $w^T x^t + w_0 > 0$:
  - $w_h = w_h - \eta \Delta w_h = w_h - \eta \sum (r^t - y^t) v_h x^t + \sum 2w_h$
  - $w_0 = w_0 - \eta \Delta w_0 = w_0 - \eta \sum (r^t - y^t) v_h$
- else:
  - $w_h = w_h - \eta \times 0 = w_h$
  - $w_0 = w_0 - \eta \times 0 = w_0$

Summary of gradients for all parameters

$$\Delta v_h = -\sum (r^t - y^t) z_h^t$$

$$\Delta v_0 = -\sum (r^t - y^t)$$

$$\Delta W = \begin{cases} \Delta w_h = \sum (r^t - y^t) v_h x^t + \sum 2w_h; & \Delta w_0 = -\sum (r^t - y^t) v_h, \quad \text{if } w^T x^t + w_0 > 0 \\ \Delta w_h = \Delta w_0 = 0, & \text{otherwise} \end{cases}$$

Summary of update equations for all parameters

- $v_h = v_h - \eta \sum (r^t - y^t) z_h^t$
- $v_0 = v_0 - \eta \sum (r^t - y^t)$
- *if* $w^T x^t + w_0 > 0$:
  - $w_h = w_h - \eta \sum (r^t - y^t) v_h x^t + \sum 2w_h$
  - $w_0 = w_0 - \eta \sum (r^t - y^t) v_h$
- else:
  - $w_h = w_h$ (no change)
  - $w_0 = w_0$ (no change)

2a

> Determine the vector of coefficients $w = [w0, w1, w2]^T$ for a single layer perceptron of the form in Figure 1 to recognize the area in Figure 2 and again for Figure 3 shaded blue.

Weights for Fig. 2

1. Write the equation of the line that separates the classes:

$$x_2 = -1$$

2. Rearrange to equal 0:

$$x_2 + 1 = 0$$

3. Write options for the input to the activation function (step(o)):

- $o = x_2 + 1$
  OR
- $o = -x_2 - 1$

We want (0, 0) to yield (+). Plugging in:

- $o = 0 + 1 = +1$
- $o = 0 \cdot 1 = -1 \therefore o = x_2 + 1$

and **w** = $[1, 0, 1]^T$, where w = $[w_0, w_1, w_2]^T$.

Weights for Fig. 3

> Determine coefficients W, v in the 2-layer Perceptron of the form in Figure 4 to recognize the shaded region in Figure 5.

1. Write the equation of the line that separates the classes:

$$x_2 = -x_1 + 1$$

2. Rearrange to *equal zero*:

$$x_2 + x_1 - 1 = 0$$

3. Write options for the input to the activation function (step(o)):

- $o = x_2 + x_1 - 1$ **OR**
- $o = -x_2 - x_1 + 1$ We want (0, 0) to yield (-). Plugging in:
- $o = 0 + 0 - 1 = -1$
- $o = 0 + 0 + 1 = +1 \therefore o = x_2 + x_1 - 1$

and **w** = $[-1, 1, 1]^T$, where w = $[w_0, w_1, w_2]$.

2b

1. Write the equation of the line that separates the classes:

- $o = x_2 + 1$
  AND
- $o = x_2 + x_1 - 1$

  which is the same as:
  $z_1 = step(x_2 + 1)$
  &
  $z_2 = step(x_2 + x_1 - 1)$

  That means that $y = 1$ iff:

- $z_1 = 1$
  AND
- $z_2 = 1$

$\therefore 1.5 = x_1 + x_2$
Note: we use 1.5 as the middle point between 1 and 2, but could've picked another point between 1 and 2 if we wanted to.

2. Rearrange to equal to equal zero:

$\implies 0 = z_1 + z_2 - 1.5$

3. Write options for the input to the activation function (step(o)):

- $o = z_1 + z_2 - 1.5$
  OR
- $o = -z_1 - z_2 + 1.5$

We want (0, 0) to correspond to (-) **and** we want (1, 1) to correspond to (+).

$$\therefore v = [-1.5, 1, 1]^T$$

and

$$W = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 1 \end{bmatrix}$$

3a

> Report the validation accuracy by the number of hidden units. How many hidden units should we use? Report the accuracy on the test set using this number of hidden units.

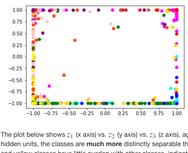
- 4 hidden units: 0.854
- 16: 0.916
- 20: 0.912
- 24: 0.916
- 32: 0.921
- 48: 0.914

If we care about accuracy alone, we'd want to use 32 hidden units, as that has the highest validation accuracy of all the # of hidden units we used (92.1%). The test accuracy for 32 hidden units was 0.919.

3b

> *Train* two MLPs with 2 and 3 hidden units, respectively. Visualize the data by values of their hidden units. Compare the 2-D and 3-D plots and explain the results in the report. The code for plotting is included in visualization.py, and you do not need to modify the file.

The plot below shows z_1 (x axis) vs. z_2 (y axis), colored by their respective classes (using the ground-truth labels, r^t). With just 2 hidden units, we can see that classes are already starting to form into distinct "clusters". Clusters that have very little overlap with other clusters of colors will mean higher correct classification within that cluster. At this point, clusters aren't perfectly defined so we'd expect a fair amount of misclassification within each class.



The plot below shows z_1 (x axis) vs. z_2 (y axis) vs. z_3 (z axis), again colored by their respective classes (using the ground-truth labels, r^t). With 3 hidden units, the classes are **much more** distinctly separable than with the 2 hidden-unit case. We can see that the blue, orange, maroon, pink, and yellow classes have little overlap with other classes, indicating lower misclassification rates for these classes than when compared with the 2-unit MLP, where there is greater overlap amongst these classes. Altogether, we'd expect much higher overall accuracy in this 3-unit case than in the 2-unit case visualized above.