CS 4650: Natural Language Processing
Spring 2023
Problem Set 1 Solutions

Giulia Paggini

February 9, 2023

1 Logistic vs Softmax

1.1

First of all, we know that:

$$P_{log}(Y=1|X=x) = \frac{e^{w^T x}}{1+e^{w^T x}}$$

$$P_{soft}(Y=1|X=x) = \frac{e^{w_1^T x}}{e^{w_0^T x}+e^{w_1^T x}}$$

then we divide the latter expression by $e^{w_0^T x}$:

$$P_{soft}(Y=1|X=x) = \frac{e^{(w_1^T - w_0^T)x}}{1+e^{(w_1^T - w_0^T)x}}$$

As a final step, we equate the two to retrieve the value of w:

$$w^T x = (w_1^T - w_0^T)x$$

$$w^T x = (w_1 - w_0)^T x$$

$$w = (w_1 - w_0)$$

for any $x \in X$.

1.2

For the logistic regression we have the following:

$$h_\theta(x) = \frac{1}{1 + e^{w^T x}}$$

$$\hat{y} = \begin{cases} 1 & h_\theta(x) \geq 0.5 \\ 0 & h_\theta(x) < 0.5 \end{cases}$$

Instead, for the softmax function:

$$\hat{y} = argmax_y \frac{e^{w^T x}}{\sum_{y' \in Y} e^{w_{y'}^T x}}$$

where $\hat{y}$ is the predicted class, namely the one with the highest probability among the two. As mentioned above, the sigmoid and the softmax function are similar, the first one takes a scalar, whereas the latter a vector. Softmax is commonly used in the implementation of neural networks, especially in the last layer to convert scores in probabilities, so that they are more interpretable; in fact, its main advantage is that the outputs are interrelated and sum up to one. The sigmoid instead is generally used for binary classification, tough the outputs do not sum up to one, therefore we need to compare them to a threshold in order to make a classification.

2 Multiclass Naive Bayes with Bag of Words

2.1

By looking at the list of label in the Y column, we obtain:

$$p(\theta_{warm}) = \frac{4}{8} = \frac{1}{2}$$

$$p(\theta_{cool}) = \frac{2}{8} = \frac{1}{4}$$

$$p(\theta_{neutral}) = \frac{2}{8} = \frac{1}{4}$$

2.2

The first step is to compute $\phi_{y,j}$ for each label and color indicated in the vector x, namely crimson (c), orange (o), purple (p) and blue (b).

$$\phi_{warm,c} = \frac{1}{10} \ \phi_{warm,o} = \frac{3}{10} \ \phi_{warm,p} = \frac{3}{10} \ \phi_{warm,b} = \frac{1}{10}$$

$$\phi_{cool,c} = \frac{1}{10} \ \phi_{cool,o} = \frac{1}{10} \ \phi_{cool,p} = 0 \ \phi_{cool,b} = \frac{2}{10}$$

$$\phi_{neutral,c} = \frac{1}{10} \ \phi_{neutral,o} = \frac{1}{10} \ \phi_{neutral,p} = \frac{1}{10} \ \phi_{neutral,b} = \frac{1}{10}$$

then we compute the probability for each label:

$$p(x, warm, \theta, \phi) = \frac{1}{2} * \frac{1}{10} * \frac{3}{10} * \frac{3}{10} * \frac{1}{10} = \frac{9}{20000}$$

$$p(x, cool, \theta, \phi) = \frac{1}{4} * 0 = 0$$

$$p(x, neutral, \theta, \phi) = \frac{1}{4} * \frac{1}{10} * \frac{1}{10} * \frac{1}{10} * \frac{1}{10} = \frac{1}{40000}$$

as a last step we need to take the log_{10} of these probabilities and the output $\hat{y}$ will be the class with the highest probability.

$$log_{10}(p(x, warm, \theta, \phi)) = -3.347$$

$$log_{10}(p(x, neutral, \theta, \phi)) = -4.602$$

therefore the predicted class is warm.

2.3

For the add-1 smoothing we will follow the same procedure as in the previous point. The difference is in the definition of the count to take care of possible disappearance of features in the dataset.

$$\phi_{warm,c} = \frac{1}{9} \ \phi_{warm,o} = \frac{2}{9} \ \phi_{warm,p} = \frac{2}{9} \ \phi_{warm,b} = \frac{1}{9}$$

$$\phi_{cool,c} = \frac{1}{9} \ \phi_{cool,o} = \frac{1}{9} \ \phi_{cool,p} = \frac{1}{18} \ \phi_{cool,b} = \frac{1}{6}$$

$$\phi_{neutral,c} = \frac{1}{9} \ \phi_{neutral,o} = \frac{1}{9} \ \phi_{neutral,p} = \frac{1}{9} \ \phi_{neutral,b} = \frac{1}{9}$$

then:

$$p(x, warm, \theta, \phi) = \frac{1}{2} * \frac{1}{9} * \frac{2}{9} * \frac{2}{9} * \frac{1}{9} = \frac{2}{9^4}$$

$$p(x, cool, \theta, \phi) = \frac{1}{4} * \frac{1}{9} * \frac{1}{9} * \frac{1}{18} * \frac{1}{6} = \frac{1}{2^4 * 3^7}$$

$$p(x, neutral, \theta, \phi) = \frac{1}{4} * \frac{1}{9} * \frac{1}{9} * \frac{1}{9} * \frac{1}{9} = \frac{1}{4 * 9^4}$$

and finally the logs:

$$log_{10}(p(x, warm, \theta, \phi)) = -3.156$$

$$log_{10}(p(x, cool, \theta, \phi)) = -4.544$$

$$log_{10}(p(x, neutral, \theta, \phi)) = -4.419$$

here, the predicted class $\hat{y}$ is warm.

3 Perceptron: Linear Separability and Weight Scaling

3.1

By introducing a third feature, we obtain the following table:

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	-1

Generally speaking, the XOR function with two inputs only is not linearly separable. The new data, including the additional column x_3, is linearly separable in a 3-dim setting by an hyperplane, since three points out of four have the same label. Tough, we are not guaranteed that a single perceptron is able to learn how to separate them correctly. This is why for more complex tasks, that take into account relations among features, we use Neural Networks rather than the simple Perceptron.

3.2

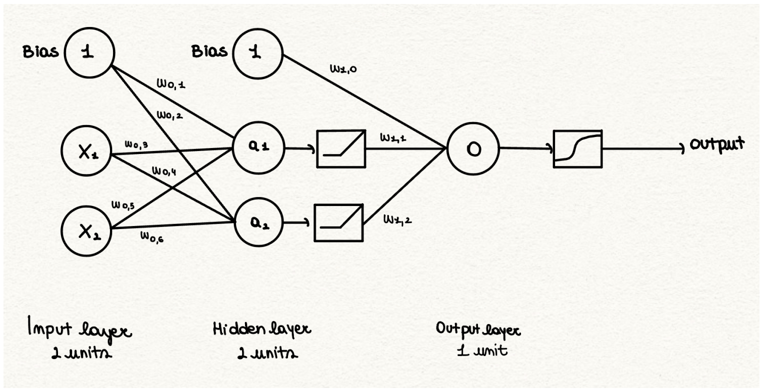
In the case in which the bias of the decision boundary equals zero, scaling the weights by a positive factor does not produce any effect and the final predictions won't be affected either. Instead, if the bias term is different from zero, scaling by a positive constant the set of weights changes the predictions of the model.

3.3

By operating a translation of the weights by a constant factor, the predictions are affected too, no matter the value of the bias included in the model. The decision boundary is shifted and the predictions on our data could be different.

4 Feedforward Neural Network

The Perceptron architecture will be composed of an input layer, an hidden layer as required and an output layer with the output of the XOR function. The input layer consists of two neurons, namely x_1 and x_2, and a bias term, that are all connected to the hidden layer. This latter takes the binary values as input and uses a ReLU as activation function on the weighted sum of each input. Instead, the output layer takes the past activations and applies a tanh function to produce an output that ranges between -1 and 1, consistently with Table 1. In general, weights and biases are parameters that the NN has to learn throughout the training to behave correctly. The idea behind is to map binary inputs to the correct output of the XOR by tuning weights and biases.



5 Dead Neurons

5.1

A neuron is said to be dead when it never activates during training, therefore it is pretty similar to a natural dropout. To be the case, the activation function should return zero so the argument must be negative. Then, the condition for the node is:

$$\theta_i^{(x \longrightarrow z)} * x + b_i \leq 0$$

5.2

To compute the two gradients, we can resort the chain rele as follows:

$$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial z} * \frac{\partial z}{\partial b_i} = \begin{cases} \theta_i^{(z \longrightarrow y)} & if\ z > 0 \\ 0 & otherwise \end{cases}$$

$$\frac{\partial L}{\partial \theta_{j,i}} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial z} * \frac{\partial z}{\partial \theta_i} = \begin{cases} \theta_i^{(z \longrightarrow y)} * x_j & if\ z > 0 \\ 0 & otherwise \end{cases}$$

5.3

In the case in which a neuron outputs zero, the gradient do not flow anymore during the back-propagation phase and consequently, weights are not updated and it becomes inactive. Since the slope of the ReLU in the negative semi-axes is zero, also its gradient is zero, then once the neuron is dead, it remains still. As long as some neurons stay alive, the network keeps learning as SGD considers multiple inputs at each iteration. The worst case scenario is when all the neurons are dead and we end up with a constant function. Some of the most common reasons why this may happen are : a too large initialization of the learning rate or a negative large bias in the model.