Author: Benjamin Smidt
Created: September 7, 2022
Last Updated: September 9, 2022

Assignment 2: Softmax Classifier

Note to the reader: this is my work for assignment two of Michigan's course EECS 498: Deep Learning for Computer Vision. This document is thoroughly researched but may not be perfect. If there's a typo or a correction needs to be made, feel free to email me at benjamin.smidt@utexas.edu so I can fix it. Thank you! I hope you find this document helpful.

Contents

1 Mathematics

1.1 Loss

Resources

- CS 231N: SVM's and Softmax

The equation for the loss function of a single example of Multinomial Logistic Regression is:

$$L_i = -log(\frac{e^{f_{y_i}}}{\sum_{j=1}^{C} e^{f_j}}) = -f_{y_i} + log(\sum_j e^{f_j}) \tag{1}$$

$$f_j = f(x_i, W)_j = (Wx_i)_j \tag{2}$$

Thus, to find the loss for the training data, we simply need to average the loss L_i for each example. For SVM's, the loss was called *hinge loss*. The loss for a Softmax Classifier is known as *cross-entropy loss*.

1.2 Gradient

To find the gradient with respect to our weight matrix W, let's again focus on one example. We can rewrite our loss function as:

$$L_i = -log(\frac{e^{W_{y_i}x_i}}{\sum_j e^{W_j x_i}}) \tag{3}$$

where W_{y_i} represents the weights for the correct label for example i and W_j is the weights for any given class (including W_{y_i}) for example i. Note that since the shape of W is DxC, each W_j is a column of W.

Let's start by reformulating our loss function a bit.

$$L_i = -log(\frac{e^{W_{y_i}x_i}}{\sum_{j=1}^{C} e^{W_j x_i}}) \tag{4}$$

$$L_i = log(\frac{\sum_{j=1}^{C} e^{W_j x_i}}{e^{W_{y_i}x_i}}) \tag{5}$$

$$L_i = log(\sum_{j=1}^{C} e^{W_j x_i}) - W_{y_i}x_i \tag{6}$$

Then we find the gradient with respect to W_{y_i}

$$\frac{\partial L_i}{\partial W_{y_i}} = \frac{\partial}{\partial W_{y_i}} log(\sum_{j=1}^{C} e^{W_j x_i}) - \frac{\partial}{\partial W_{y_i}} W_{y_i} x_i \tag{7}$$

$$\frac{\partial L_i}{\partial W_{y_i}} = \frac{\partial}{\partial W_{y_i}} log(\sum_{j=1}^{C} e^{W_j x_i}) - x_i \tag{8}$$

$$\frac{\partial L_i}{\partial W_{y_i}} = \frac{1}{\sum_{j=1}^{C} e^{W_j x_i}} \frac{\partial}{\partial W_{y_i}} (e^{W_1 x_i} + ... + e^{W_{y_i} x_i} + ... + e^{W_C x_i}) - x_i \tag{9}$$

$$\frac{\partial L_i}{\partial W_{y_i}} = \frac{x_i e^{W_{y_i} x_i}}{\sum_{j=1}^{C} e^{W_j x_i}} - x_i \tag{10}$$

$$\frac{\partial L_i}{\partial W_{y_i}} = x_i (\frac{e^{W_{y_i} x_i}}{\sum_{j=1}^{C} e^{W_j x_i}} - 1) \tag{11}$$

The math works out similarly for the gradient with respect to W_j

$$\frac{\partial L_i}{\partial W_j} = \frac{\partial}{\partial W_j} log(\sum_{j=1}^{C} e^{W_j x_i}) - W_{y_i} x_i \tag{12}$$

$$\frac{\partial L_i}{\partial W_j} = \frac{\partial}{\partial W_j} log(\sum_{j=1}^{C} e^{W_j x_i}) \tag{13}$$

$$\frac{\partial L_i}{\partial W_j} = \frac{1}{\sum_{j=1}^{C} e^{W_j x_i}} \frac{\partial}{\partial W_j} (e^{W_1 x_i} + ... + e^{W_j x_i} + ... + e^{W_C x_i}) \tag{14}$$

$$\frac{\partial L_i}{\partial W_j} = \frac{x_i e^{W_j x_i}}{\sum_{j=1}^{C} e^{W_j x_i}} \tag{15}$$

2 Programming

2.1 Softmax Loss Naive

Per usual with the naive function, this one's again pretty straight forward. Although I did implement a hybrid version with some vectorization just because I didn't feel like writing out the for loops and all that jazz.

2.2 Softmax Loss Vectorized

We begin by matrix multiplying XW (X is NxD and W is DxC) to get our scores and store them in the matrix f. We then normalize the matrix for numerical stability purposes by making the largest value 0. For more information see CS 231N: SVM and Softmax. We then raise all of our (shifted) scores by e and stored those values in matrix ef.

For the loss function we sum all the scores in a given example for every example. Since matrix ef is NxC, we sum over the second dimension $dim = 1$. Finally we grab all the scores from the correct classes ($e - yi$ with shape Nx1) and divide each correct score by the sum of the all the scores for each example. We average those scores, add our regularization term, and viola, loss complete.

For the gradient, let's first work on the gradient with respect to w_j. We begin by cloning matrix ef (NxC) and setting all the correct class score values to zero. Then we divide all the scores by the sum of the scores in the same example and store it in matrix k. Finally, we just multiply k.T by X, divide everything by N and add it to the gradient. I can't explain with words why that multiplication works other than the fact that the shape of k.T is CxN and X is NxD. Since dW is CxD, matrix multiplying them gives the correct output. If you don't understand (I didn't at first), you need to draw out the shapes and trace how the matrix addition correctly sums elements to produce the proper matrix.

Now we can computer the gradient with respect to w_{y_i}. We use a mask to get rid of all the scores that aren't correct scores (set equal to 0). Then we divide the (correct) scores by the sum of the scores in the same example for each score. Next we subtract 1 from every correct score using the mask we created earlier since we want to leave all the incorrect scores as 0 still. Finally we matrix multiply k.T by X again and add it to the gradient. We finish off by adding L2 regularization.

2.3 Softmax Get Search Parameters

This function just returns different parameter combinations in seach of the best one.

3 References

1. CS 231N: SVM's and Softmax

2. EECS 498 Lecture 3: Linear Classifiers

3. EECS 498 Lecture 4: Optimization