Author: Benjamin Smidt
Created: September 11, 2022
Last Updated: September 11, 2022

Assignment 2: Two Layer Neural Net

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.

1 Mathematics

2 Programming

2.1 NN Forward Pass

For this function we simply want to compute the forward pass through our two layer, fully connected network. Our input X has shape NxD, our first weight matrix W_1 shape DxH, first bias vector b_1 length H, second weight matrix W_2 shape HxC, and second bias vector b_2 length C. N is the number of examples, D the dimension of each example, H the number of layers in our hidden layer, and C the number of classes to classify our examples into.

Moving forward through the network is pretty simply. We begin by matrix multiplying XW_1 yielding the matrix H (for hidden) of shape NxH. Each column represents the computation that a given hidden layer does for *every single example*. Said differently, each row indicates the computations for *all the hidden layers* for a given example in N. Because of this we add our bias vector b_1 to every single row in our H matrix (using broadcasting).

Next, we compute our ReLU function using a mask and pass H off to W_2 to produce our final S matrix (for scores). Similar to our hidden layer computation, we matrix multiply HW_2 and add b_2 using broadcasting. This final S matrix has shape NxC as you would expect. Each example has a score for each class.

2.2 NN Forward Backward

Loss

This function has two parts. First we compute the loss from our forward pass (previous function). Then we find the gradient using backpropagation. The loss is fairly simply so the first part will be brief. For this neural network we're using softmax as our final classifier and adding L2 regularization for every weight matrix (just two, W_1 and W_2). As such, our loss function is as follows.

$$L = 1/N \sum_{i=1}^{N} -log(\frac{e^{f_{y_i}}}{\sum_{j=1}^{C} e^{f_j}}) + \lambda \sum_{k}^{D} \sum_{l}^{C} (w_1)_{l,k}^2 + \lambda \sum_{k}^{D} \sum_{l}^{C} (w_2)_{l,k}^2 \quad (1)$$

For more information about Softmax and regularization see A2-Softmax, the second part of the "Linear Classifiers" assignment in A2. The *important detail* here is that I did normalize the output scores that we performed the data loss on (using softmax) to prevent numeric instability. For more info on numeric instability check out CS 231N: Linear Classifiers

Gradient

Now time for the fun part, backpropagating the gradient through our network! This section may be quite long and tedious because I want to work through the computation from beginning to end. Before we begin, remember back propagation takes advantage of the chain rule. Recall that a neural network is really a chain of functions, each one using the output of the previous function as the input of its own until we reach the end of our network and make a prediction.

What we're actually doing in our backpropagation algorithm is finding the gradient or jacobian with respect to a variable (often a weight matrix W) for a modular portion of the network, a single function. Since the chain rule simply multiplies the derivatives (or gradients or jacobians depending on your circumstance) of each function in the chain, we're left with a nice, modular method of computing the gradient from beginning to end by just multiplying gradients.

To make this more clear, let's walk through the loss function of the two layer neural network in A2 using a single example x_i. You may wonder why

I'm using a lot of row vectors. This is because when we generalize this process to N examples, each row will be an example. So for now many vectors are 1xk (k just represents some dimension, not a specific dimension used in the neural network) but in practice they will be Nxk.

x_i: 1xD, W_1: DxH, b_1: 1xH

$$H = x_i W_1 + b_1 \tag{2}$$

H: 1xH

$$H = max(0, H) \tag{3}$$

W_2: HxC, b_2: 1xC

$$S = HW_2 + b_2 \tag{4}$$

S: 1xC

$$L_i = -log(\frac{e^{S_y}}{\sum_{j=1}^{C} e^{S_j}}) + \lambda \sum_{k}^{D} \sum_{l}^{C} (w_1)_{l,k}^2 + \lambda \sum_{k}^{D} \sum_{l}^{C} (w_2)_{l,k}^2 \tag{5}$$

This is our forward propagation through the network for our loss function. If we wanted to find the prediction instead of the loss, we'd replace our last equation $L_i = ...$ with

$$x_i Prediction = max(\frac{e^{S_j}}{\sum_{j=1}^{C} e^{S_j}} \forall j \in C) \tag{6}$$

However, we're trying to compute the gradient (of the loss function L) so we need to focus on the output of L_i. I do quickly want to point out that the graphical representation of this network looks very different (at least to me) from the algebraic plug and chug we see here. They are the same though. The number of nodes in our first layer, our hidden layer, is defined by the number of columns H in W_1. Similarly, the number of nodes in our second layer, which passes scores off to our softmax function, is defined by the number of columns in W_2. As you would expect, the number of nodes is C since we're trying to classify our data into 1 of C categories.

3 References

1. CS 231N: Neural Networks