

Author: Benjamin Smidt
Created: September 11, 2022
Last Updated: September 13, 2022

Assignment 2: Two Layer Neural Net

Note to reader.

This is my work for assignment two of Michigan's course EECS 498: Deep Learning for Computer Vision. This document is meant to be used as a reference, explanation, and resource for the assignment, not necessarily a comprehensive overview of Neural Networks. Furthermore, this document is well-informed and 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 Functions

1.1 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 $N \times D$, our first weight matrix W_1 shape $D \times H$, first bias vector b_1 length H , second weight matrix W_2 shape $H \times C$, 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 $N \times H$. 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 $N \times C$ as you would expect. Each example has a score for each class.

1.2 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 simple 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.

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

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 ensure clarity, let's first walk through the forward pass of computing 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 $1 \times k$ (k just represents some dimension, not a specific dimension used in the neural network) but in practice they will be $N \times k$.

x_i : $1 \times D$, W_1 : $D \times H$, b_1 : $1 \times H$

$$H = x_i W_1 + b_1 \quad (2)$$

H : $1 \times H$

$$H = \max(0, h_i) \quad \forall h_i \in H \quad (3)$$

W_2 : $H \times C$, b_2 : $1 \times C$

$$S = H W_2 + b_2 \quad (4)$$

S : $1 \times C$

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

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 $L_i = \dots$ with

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

However, we're trying to compute the gradient of the loss function L so we need to focus on the output of L_i . Before moving on, 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.

Alright, let's get backpropagate through this small network. Remember, because of the chain rule we're just multiply gradients together. For instance,

if we want to find $\frac{\partial L_i}{\partial W_2}$ we can just compute $\frac{\partial L_i}{\partial S} \frac{\partial S}{\partial W_2}$. This is much easier to do rather than substituting S for $HW_2 + b_2$ in the equation for L_i and then finding $\frac{\partial L_i}{\partial W_2}$ directly. Let's begin with $\frac{\partial L_i}{\partial S_y}$ and $\frac{\partial L_i}{\partial S_j}$

$$\frac{\partial L_i}{\partial S_y} = \frac{\partial}{\partial S_y} - \log\left(\frac{e^{S_y}}{\sum_{j=1}^C e^{S_j}}\right) + \frac{\partial}{\partial S_y} \lambda \sum_k^D \sum_l^C (w_1)_{l,k}^2 + \frac{\partial}{\partial S_y} \lambda \sum_k^D \sum_l^C (w_2)_{l,k}^2 \quad (7)$$

$$\frac{\partial L_i}{\partial S_y} = \frac{\partial}{\partial S_y} - \log\left(\frac{e^{S_y}}{\sum_{j=1}^C e^{S_j}}\right) \quad (8)$$

$$\frac{\partial L_i}{\partial S_y} = \frac{\partial}{\partial S_y} \log\left(\sum_{j=1}^C e^{S_j}\right) - \frac{\partial}{\partial S_y} S_y \quad (9)$$

$$\frac{\partial L_i}{\partial S_y} = \left(\frac{1}{\sum_{j=1}^C e^{S_j}}\right) \frac{\partial}{\partial S_y} (e^{S_1} + \dots + e^{S_y} + \dots + e^{S_C}) - 1 \quad (10)$$

$$\frac{\partial L_i}{\partial S_y} = \left(\frac{e^{S_y}}{\sum_{j=1}^C e^{S_j}}\right) - 1 \quad (11)$$

$$\frac{\partial L_i}{\partial S_j} = \frac{\partial}{\partial S_j} - \log\left(\frac{e^{S_y}}{\sum_{j=1}^C e^{S_j}}\right) + \frac{\partial}{\partial S_j} \lambda \sum_k^D \sum_l^C (w_1)_{l,k}^2 + \frac{\partial}{\partial S_j} \lambda \sum_k^D \sum_l^C (w_2)_{l,k}^2 \quad (12)$$

$$\frac{\partial L_i}{\partial S_j} = \frac{\partial}{\partial S_j} - \log\left(\frac{e^{S_y}}{\sum_{j=1}^C e^{S_j}}\right) \quad (13)$$

$$\frac{\partial L_i}{\partial S_j} = \frac{\partial}{\partial S_j} \log\left(\sum_{j=1}^C e^{S_j}\right) - \frac{\partial}{\partial S_y} S_y \quad (14)$$

$$\frac{\partial L_i}{\partial S_j} = \left(\frac{1}{\sum_{j=1}^C e^{S_j}}\right) \frac{\partial}{\partial S_y} (e^{S_1} + \dots + e^{S_j} + \dots + e^{S_C}) \quad (15)$$

$$\frac{\partial L_i}{\partial S_j} = \left(\frac{e^{S_j}}{\sum_{j=1}^C e^{S_j}}\right) \quad (16)$$

This is the gradient of the softmax function. For more information on it, check out Assignment 2: Softmax where I explain it more in depth. Notice that we have different derivatives depending on if the score is the correct

score or not. We will have to program this but for now, let's store both $\frac{\partial L_i}{\partial S_y}$ and $\frac{\partial L_i}{\partial S}$ in a single variable $\frac{\partial L_i}{\partial S}$ and continue backward through the network (and no I haven't forgotten the regularization term, I want to add it later).

$$\frac{\partial S}{\partial W_2} = \frac{\partial}{\partial W_2} HW_2 + b_2 \quad (17)$$

$$\frac{\partial S}{\partial W_2} = H \quad (18)$$

$$\frac{\partial L_i}{\partial W_2} = \frac{\partial L_i}{\partial S} \frac{\partial S}{\partial W_2} \quad (19)$$

$$\frac{\partial L_i}{\partial W_2} = \frac{\partial L_i}{\partial S} H \quad (20)$$

If you're attentive you may be looking at equation 20 with some skepticism. And you would certainly be correct in doing so. $\frac{\partial L_i}{\partial S}$ has shape 1xC since S has shape 1xC (actually it's technically just a column vector of length C but thinking about it as a row vector will generalize to more examples much more intuitively). H has shape 1xH. Clearly this matrix multiplication won't work. So what gives?

This will be a recurring theme in backpropagation of needing to analyze shapes. Remember that $\frac{\partial y}{\partial Z}$, where y is a scalar and Z is just some variable (scalar, vector, matrix, tensor, etc.), will *always* have the same shape as Z . So $\frac{\partial L_i}{\partial W_2}$ has the same shape as W_2 since L_i is (always) a scalar. Since W_2 has shape HxC, we actually need to take the outer product between $\frac{\partial L_i}{\partial S}$, shape 1xC, and H , shape 1xH. The following results (note that $H.T$ is the transpose of H).

$$\frac{\partial L_i}{\partial W_2} = H.T \frac{\partial L_i}{\partial S} \quad (21)$$

Although maybe unintuitive at first, thinking about what these gradients actually mean makes everything fall into place (at least for me). Each value at the position i, j in the matrix $\frac{\partial L_i}{\partial W_2}$ indicates the change in L_i given a change in $(W_2)_{ij}$.

$$\frac{\partial L_i}{\partial H} = \frac{\partial L_i}{\partial S} \frac{\partial S}{\partial H} \quad (22)$$

$$\frac{\partial S}{\partial H} = \frac{\partial}{\partial W_2} HW_2 + b_2 \quad (23)$$

$$\frac{\partial S}{\partial W_2} = H \tag{24}$$

2 References

1. CS 231N: Neural Networks