Project 3 CMSC426

Andrew Lee
April 22, 2021

1 Define the Network

The weight matrices are initialized with a size of 64x784 for W1 and 10x64 for W2 The original image has dimensions 28x28 but is flattened to a 784x1 column vector, which is then stored into self.x. Multiplying the weight matrix W1 with the image x gives us W1x. Adding b1 to W1x gives a1, the final state of the 64 neurons in the hidden layer of the MLP. Applying the sigmoid function to a1 gives f1. Multiplying the matrix W2 with f1 gives W2f1. Adding b2 to W2f1 gives a2, the output layer consisting of 10 neurons. Finally applying the softmax function to a2 gives y_hat. The softmax function used in the code is borrowed from a link that a TA posted on piazza, found here: https://stackoverflow.com/questions/54880369/implementation-of-softmax-function-returns-nan-for-high-inputs

2 Back-propagation

2.1 Deriving the Jacobians

For deriving the Jacobian matrices, the identities from https://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf were used.

$$\frac{\partial a_2}{\partial b_2} = \boldsymbol{I}$$

Using identity 3 in the gradient notes, deriving $\frac{\partial a_2}{\partial b_2}$ results in a 10x10 identity matrix. Because multiplying by this matrix would have no effect, it is omitted from calculations in the code.

$$\frac{\partial a_2}{\partial \mathbf{W}_2} = f_1^T$$

I derived $a2 = \mathbf{W}_2 f_1 + b_2$ with respect to $\mathbf{W}_2$, which gives f_1^T. I know this should be a 10x640 matrix, but I don't know how to make it have those dimensions. The only idea I have would be to outer multiply with the 10x10 identity matrix we would derive from b_2.

$$\frac{\partial a_2}{\partial f_1} = \mathbf{W}_2^T$$

Using identity 1 in the gradient notes, deriving $a2 = \mathbf{W}_2 f_1 + b_2$ with respect to $\mathbf{f}_1$ results in the weight matrix $\mathbf{W}_2$.

$$\frac{\partial f_1}{\partial a_1} = diag(\sigma'(x))$$

Using identity 4 from the gradient notes shows us that the Jacobian here is a 64x64diagonal matrix with the entry at (i, i) is the derivative of the sigmoid function applied to x_i. With help from https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e we find $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. In the code, this is represented as a column vector containing the diagonal elements of $diag(\sigma'(x))$ to optimize for speed doing element-wise multiplication.

$$\frac{\partial a_1}{\partial b_1} = \boldsymbol{I}$$

Similar to $\frac{\partial a_2}{\partial b_2}$, we use identity 3 from the gradient notes and find that $\frac{\partial a_1}{\partial b_1}$ derives to a 64x64 identity matrix.

$$\frac{\partial a_1}{\partial \mathbf{W}_1} = x^T$$

Similar to $\frac{\partial a_2}{\partial \mathbf{W}_2}$, I'm sure this should be a 64x50176 matrix, but I don't know how to get to those dimensions.

2.2 Compute the gradients

The rest of the Jacobian matrices were derived with the help of the gradient notes, with

$$\frac{\partial L}{\partial \mathbf{W}_2} = (\hat{\boldsymbol{y}} - \boldsymbol{y})^T f_1^T$$

reshaped to 1x640,

$$\frac{\partial L}{\partial \mathbf{b}_2} = (\hat{\boldsymbol{y}} - \boldsymbol{y})^T$$

$$\frac{\partial L}{\partial \mathbf{W}_1} = (\hat{\boldsymbol{y}} - \boldsymbol{y})^T \mathbf{W}_2 diag(\sigma'(x)) x^T$$

reshaped to 1x50176, and

$$\frac{\partial L}{\partial \mathbf{b}_1} = (\hat{\boldsymbol{y}} - \boldsymbol{y})^T \mathbf{W}_2 diag(\sigma'(x))$$

3 Training

Training is done by iterating through every image, assigning batches through a set of loops. The labels are translated into a 1x10 matrix with the label becoming the index where it has the value of 1. For calculating accuracy, finding matches is done by finding the index of the max value in y_hat and the index of 1 in y and then comparing them.

4 Overfitting?

The main differences that are apparent is that the training and validation lines are more separated in the 2000 image training graph and the 50000 image graph manages to reach a higher maximum accuracy than the 2000 image graph. Also, the 50000 image graph ramps up much quicker in accuracy at the beginning, although both seem to reach about the same rate of increase in accuracy by the end of the 100 epochs.

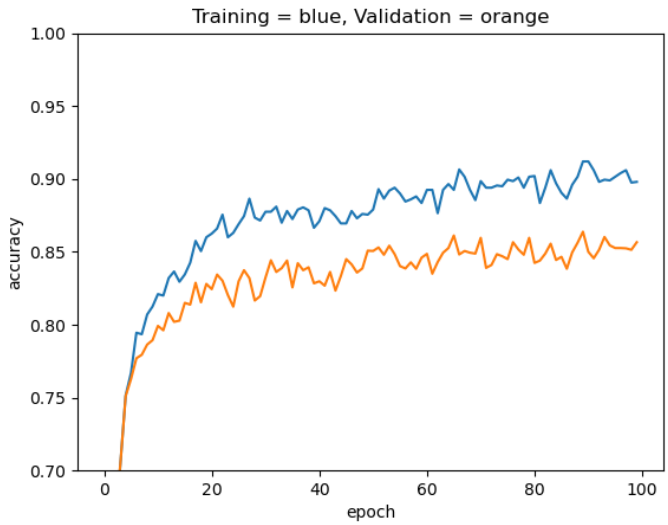


Figure 1: 2000 image training

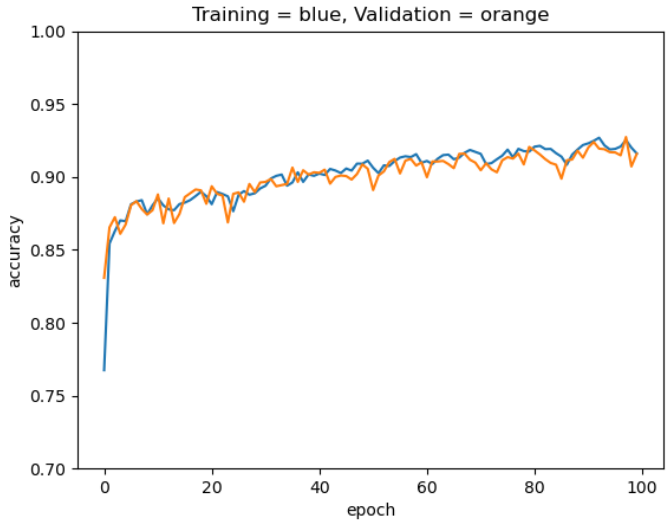


Figure 2: 50000 image training

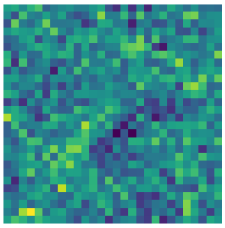
5 How well does it work?

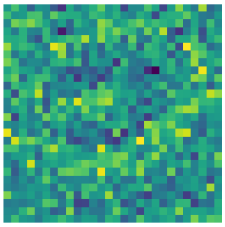
With the default settings, I was able to reach an accuracy of 90-91%. However, the best accuracy I was able to achieve was 94% by changing the learning rate to .0001. Attempting to change the learning rate to .001 or .00001

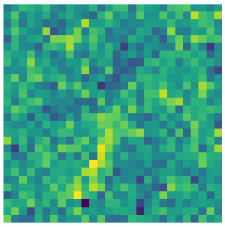
would result in a much lower accuracy in the 80-90%range. Note that the graphs of the 2000 image and 50000 image training correspond to default values, and not the modified learning rate that was used for my best accuracy.

6 Visualize the Weights

Its difficult to make out anything useful, but I can somewhat see a semblance of shapes in some parts of some images. The groupings of similarly shaded pixels makes me feel like I'm looking at a Rorschach test. Here are a few examples:







7 Convolutional Neural Networks

This part of the project is unfinished and the code does not fully function. However, I have still made attempts to acquire partial credit where I can.