Applying Multi-layer Neural Network Classification on SVHN Dataset

Jacky Dam
jdam@ucsd.edu

Dylan Loe
djloe@ucsd.edu

Ali Zaidi
mazaidi@ucsd.edu

Abstract

In this project, we implemented backpropagation to create an algorithm which attempts to classify the correct number shown when given an image with handwritten numbers from the SVHN data set. From this data set, we imported a large set of training images which we used to train our algorithm, and a smaller set of testing images which we would eventually use to evaluate the efficiency of our algorithm. Through this process, one of the surprising bugs that we encountered was the fact that ignoring to update the bias weights resulted in an overall lower accuracy for all models. After developing our algorithm to learn how to classify numbers based on the training set, we found that the baseline model given to us achieved an accuracy of above 70% among all the different data sets.

1 Introduction

For our algorithm, we utilized Python to implement backpropagation to create a multilayer neural network which learns to classify the numbers shown when given a digit image from the SHVN data set. The implementation of the forward and backward passes through the network was abstracted out into multiple Layer, Activation, and Network classes for simplicity. After verifying backpropagation, our base model was trained with minibatch stochastic gradient descent and achieved training and validation accuracy above 70% and final test accuracy of 63%. Several variations of our model were tested individually, including regularization of the weights, using different activation functions, and changing the number of hidden layers and hidden units. Overall, 2 hidden layers with 256 hidden units each ended up yielding the best performance on the test set with 80.19%.

2 SVHN Dataset

The Street View House Numbers (SVHN) dataset consists of 32 x 32 pixel images of house address numbers obtained from Google Street View that have been processed in a way that centers the target digit in the image, similar to MNIST, with 73257 examples in the training set and 26032 in the test set [1]. A validation set is additionally constructed using an 80-20 split of the training set, with the final training set having 58606 images and the validation set with 14651.
The preprocessing for this dataset was minimal, comprising of flattening each 32 x 32 image into a 1024-dimensional vector, one-hot encoding all of the labels in each of the datasets, and standardizing all of the pixel values with the element-wise pixel means and standard deviations from the training data, to put their values within the range [-1, 1].

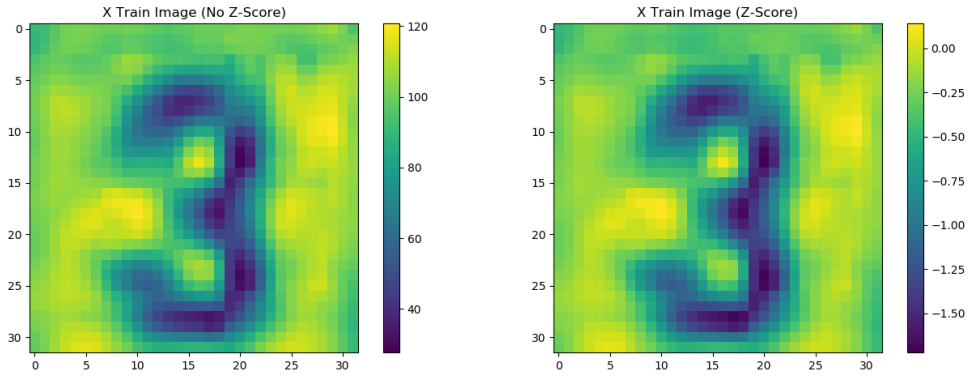

Figure 1: Examples of Training Images Before and After Preprocessing

3 Backpropagation

As our model is now a multilayer neural network, performing gradient descent requires computing gradients with respect to each layer's weights. The gradient error starts from the output layer and works its way from each hidden layer back to the input layer, changing each layer's weights, hence the term backpropagation. The update equation for the weights from layer i to j is [2]

$$w_{ij} = w_{ij} + \alpha \sum_n \delta_j^n z_i^n$$

where $\delta_j^n = y_j^n - \hat{y}_j^n$ if j is the output layer and $\delta_j^n = g'(a_j^n) \sum_k \delta_k^n w_{jk}$ if j is the hidden layer. We can check our implementation of backpropagation using a numerical approximation for the gradient:

$$\frac{\partial}{\partial w} E^n(w) \approx \frac{E^n(w+\epsilon) - E^n(w-\epsilon)}{2\epsilon}$$

We approximated the gradient using $\epsilon = 10^{-2}$ for two output to hidden weights, two hidden to input weights, one output bias weight, and one hidden bias weight.

	Approximation	Actual Gradient	Difference
Hidden to Input Weight 1	0.0002528789	0.000387876515	0.0001349976
Hidden to Input Weight 2	0.000461898	0.0006038747	0.0001419765
Output to Hidden Weight 1	0.677595318	.0.669406372	0.008188946
Output to Hidden Weight 2	0.05919962	0.0636618	0.0044622
Hidden Bias	0.0753188	0.01864303	0.005667578
Output Bias	1.1102e-15	0.07427821	0.0074278

Table 1: Summary of numerical approximation of gradients for all 6 weights.

For all the weights, the differences all in to the order of $O(\epsilon^2)$, and this marginal difference allows us to come to a conclusion that we implemented backpropagation correctly.

4 Base Model Training Procedure

Our base model uses an input layer with 1024 units, one for each pixel in the image, a hidden layer with 128 weights with a tanh activation function in between, and finally an output layer with 10 units, each corresponding to the 10 different digit labels, normalized into probabilities using softmax. We trained our model with minibatch stochastic gradient descent with momentum initialized to .9, batch size of 128, and learning rate of .05 for 100 epochs.

From the figure we can see the training and validation accuracy exceed 70% after 100 epochs and that cross-entropy loss is indeed dropping over time. The final test accuracy for the baseline model

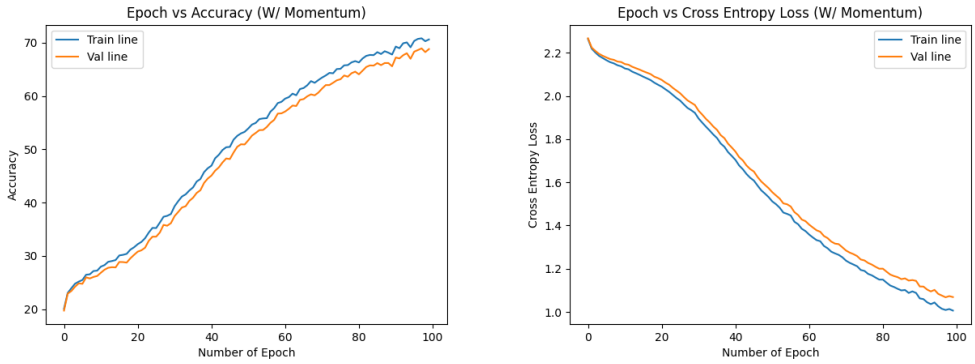

Figure 2: Train and Validation Accuracy and Loss for Tanh

is 63%, leaving some room for improvement with our other experimental models. The loss starts around 2 because initially all the weights are random, so all the classifications basically have equal probability, so the cross entropy loss function starts around 2 - 2.6.

5 Model Regularization

One way to improve upon this base model is to add regularization of the model's weights. Regularization is the process of introducing a penalty to the cross entropy loss that encourages small weight values that attempt to make the model simpler and more generalizable to unseen data. L2 and L1 regularization is achieved by adding the 2-norm $||w||_2^2$ and 1-norm $||w||_2^1$, respectively, to the model's loss function [3]. L2 regularization penalizes the model proportionally to the sum of the square of the weights while L1 regularization uses the sums of the absolute values of the weights. Because L1 regularization changes weights with a constant value each time, it tends to drive weights all the way to zero instead of a very small but nonzero amount like with L2 regularization. L1 is essentially a harsher variant of L2 regularization that would further decrease train accuracy but generally improve test accuracy.

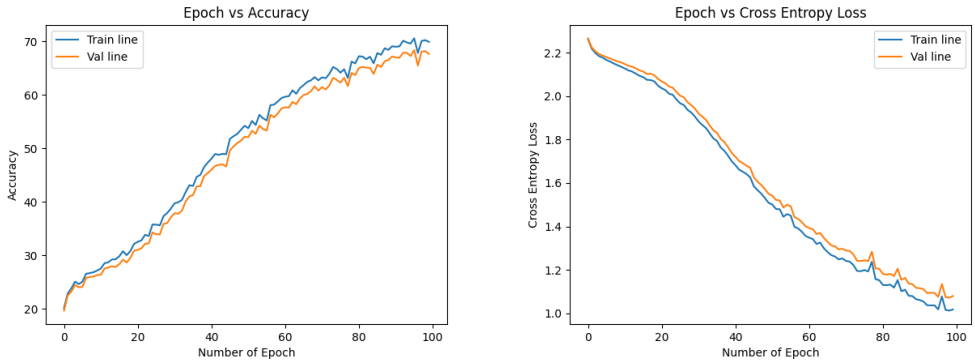

Figure 3: L2 regularization 1e-6, loss = 1.214, acc = 63.22%

6 Activation Functions

Another modification we can make to our neural network is our choice of activation function. We can run our baseline model with sigmoid and ReLU activation functions instead of tanh. Our baseline model with sigmoid activation function achieved 60% accuracy on the test set while the model with

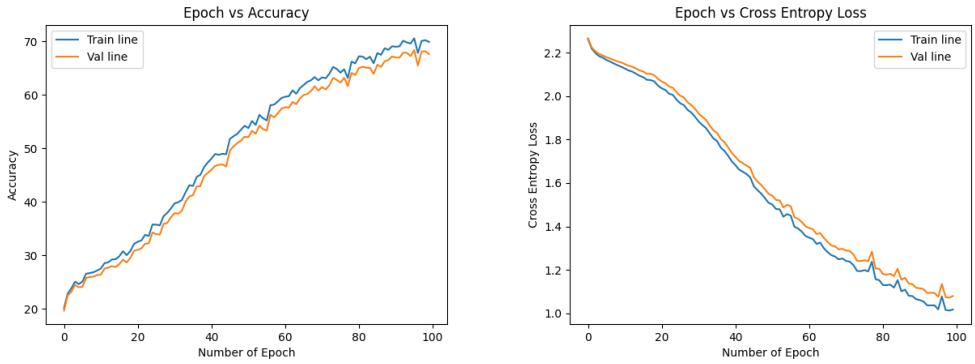

Figure 4: L1 regularization 1e-6, loss = 1.2136, acc = 63.218%

the ReLU activation function achieved results closer to 54%. In both cases, performance on training and validation sets was slightly better than test performance. ReLU is an activation function that is better suited for very deep networks, unlike our network with only one hidden layer. Tanh behaves similarly to the sigmoid but is found to achieve better results on networks with a small amount of hidden layers, such as our model. For this experiment, we utilized the baseline model that we used in tanh, with mini batches of size 128.

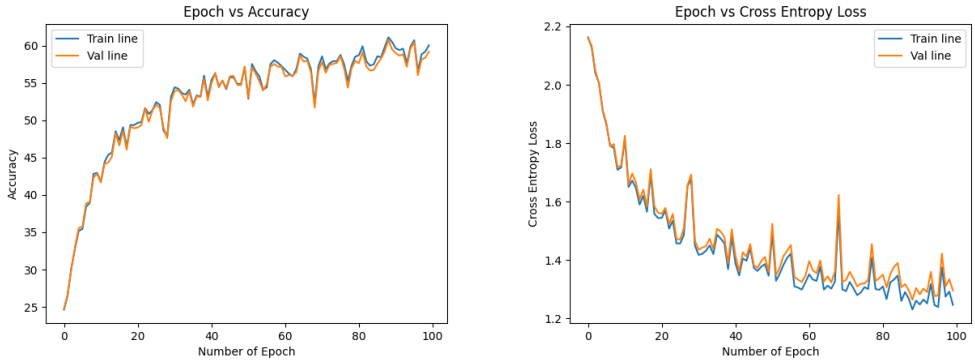

Figure 5: Train and Validation Accuracy and Loss for Model with Sigmoid Activation

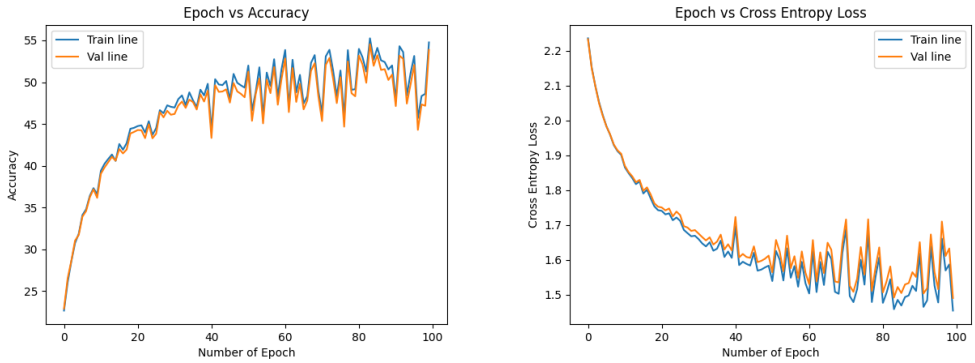

Figure 6: Train and Validation Accuracy and Loss for Model with ReLU Activation

7 Experiments With Network Topology

Taking a look at the plots shown in figures 8, 9, 10, we experimented with the number of hidden units and number of hidden layers to examine whether more we could affect the accuracy of the algorithm by manipulating these parameters. In our experiment, we found that increasing the number of hidden units (74.63%) performed better than reducing the number of hidden units (67.60%). This intuitively makes sense because the model is able to learn more complicated features of an image when given more hidden units to learn with. Additionally, when we increased the number of hidden layers, we found that the accuracy greatly increased to about (80.19%) compared to (74.63%) which makes sense for the same reason. In this process, we kept all parameters the same except as the base model for tanh for the number of hidden units and number of hidden layers.

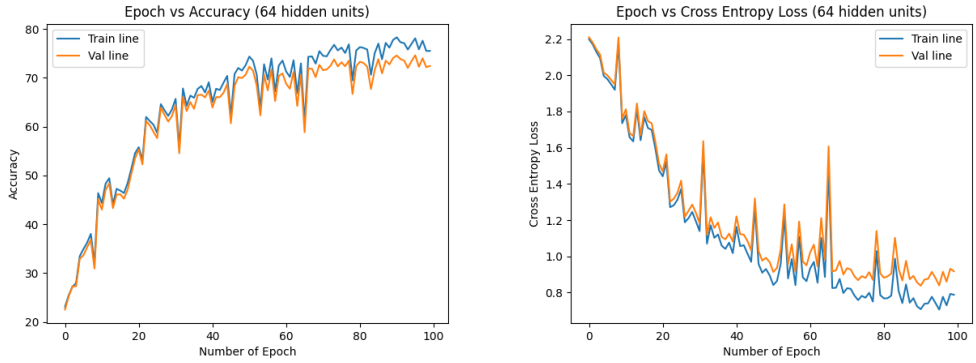

Figure 7: Reducing number of hidden units

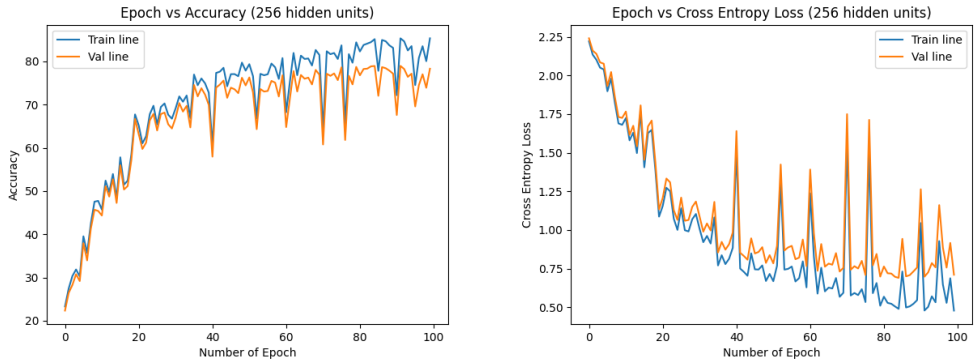

Figure 8: Adding more hidden units

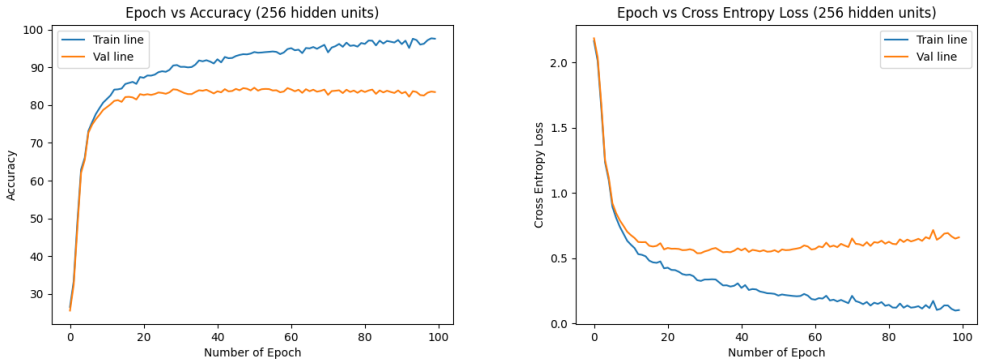

Figure 9: Using Two hidden layers

8 Team Contribution

8.1 Jacky Dam

In the project, I wrote parts of the code where we needed to manipulate, clean, organize the data. I also implemented parts of the code where we utilized mini-batch stochastic gradient descent to generate the losses and accuracies of the different activation functions.

8.2 Dylan Loe

I helped set up the pdf report for our team to collaborate on. I also wrote portions of the code that implemented the loss function and accuracy as well as some of the preprocessing for the data. I also helped implement the forward and backward passes through the neural network and wrote a good-sized portion of the report.

8.3 Ali Zaidi

In this project, I helped write code and debug the NN backward method, NN layer class, train.py, and main.py. Also I worked on the report and setting up some of the plots.

References/Related Works

[1] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng. *Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. (PDF). http://ufldl.stanford.edu/housenumbers/

[2] Cottrell, Gary, 2020. Lecture 3, lecture slides, *Backpropagation*, UCSD.

[3] Cottrell, Gary, 2020. Lecture 4A, lecture slides, *Improving Generalization*, UCSD.