Multi-Layer Neural Network Classification of Fashion MNIST Dataset

Anshuman Dewangan
A59001372
adewanga@ucsd.edu

Jinlong Huang
A53285117
jih002@ucsd.edu

Margot Wagner
A53279875
mwagner@ucsd.edu

Abstract

We implemented a multi-layer neural network to classify images of fashion products using the Zalando's Fashion MNIST Dataset in order to identify strategies that optimize performance in neural networks. Our baseline model consisted of 2 hidden layers of 50 nodes and tanh activation and a softmax output layer, utilized a learning rate of 1.2e-2, momentum gamma of 0.9, and no regularization, and was trained using stochastic gradient descent of mini-batch size 128 over 100 epochs with early stopping. The baseline achieved a high accuracy of 78.36%. Several variations were tested, including varying activation functions (sigmoid, ReLU, leakyReLU), number of hidden nodes, and number of hidden layers. The fact that the baseline outperformed other variations suggests parameters in a sweet spot that are not too low or too high for the task at hand result in the best performance overall.

1 Introduction

In this work, we implemented a multi-layer neural network to classify images of fashion products using the Zalando's Fashion MNIST Dataset using softmax outputs for classification in order to identify strategies that optimize performance in neural networks. The multi-layer neural network included inputs, hidden layers, and outputs. The goal was to classify 28x28 greyscale images into one of 10 classes: top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. The Fashion MNIST dataset is a more complicated problem to solve compared to the classical MNIST dataset.

First, a neural network was implemented containing 2 hidden layers, each with 50 units, and tanh activation for the hidden layers. Cross-entropy loss was used as the objective function. We used mini-batch stochastic gradient descent for training throughout. Here we compared different learning rates and minibatch sizes to find the best model.

Next, we performed various experiments on the multi-layer neural network with hopes of optimizing the performance of the network. We experimented with regularization where we added a weight decay, L_2 regularization, in the update scheme. Additionally, we tested various activation functions for the hidden units to compare to the initial tanh function that was used. Lastly, we varied the neural network topology. In each scenario, we analyzed network performance in order to optimize classification and minimize loss.

Figure 1 summarizes the results from our network optimization experiments. The rest of this work goes into the findings and the resulting intuition gleaned in detail.

Experiment type	Test loss	Test accuracy
$\lambda = 0$	**1.54**	**78.36%**
$\lambda = 1e-2$	1.17	19.98%
$\lambda = 1e-3$	1.87	63.16%
$\lambda = 1e-4$	7.88	73.25%
Leaky ReLU	**1.84**	**80.07%**
ReLU	1.84	77.93%
Sigmoid	1.08	77.43%
Tanh	**1.54**	**78.36%**
25×50	1.58	72.14%
50×25	1.42	68.22%
50×50	**1.54**	**78.36%**
50×100	1.48	63.84%
100×50	1.51	63.98%
1 layer (53 nodes)	1.61	71.38%
2 layers (50 nodes)	**1.54**	**78.36%**
3 layers (47 nodes)	1.50	70.03%

Figure 1: Summary of the effects of various network optimization strategies on performance.

2 Methods

Images were loaded and preprocessed from the provided Fashion MNIST dataset. Each image, originally 28x28 pixels, was flattened to a 1D vector of pixels of length 784 and normalized. The images for each category were then stacked into a data matrix with dimensions of the number of examples by the flattened number of pixels. Output labels were transformed using one hot encoding. The training data was loaded from the provided file and split 80/20 into a training and validation set. The testing data was used as provided. The neural network parameters were loaded from a config file and implemented as detailed below.

2.1 Multi-Layer Neural Network Implementation

The multi-layer neural network was originally built with the specifications provided in the assignment. We used an input layer, two hidden layers each with 50 units and tanh activation, and a softmax output layer. The neural network comprised of three classes: Activation, Layer, and Neuralnetwork.

The Activation class defined all the activation functions and their gradients as well as forward and backward pass functions. The activation functions were treated as an additional layer on top of a linear layer that computed the weight sum of inputs to that unit.

The Layer class was for standard linear layers with both forward and backward passes. The forward function converted the input vector "x" to an output variable "a" to then be passed to the activation function. The backward function used the weighted sum of deltas and computed the weight and bias gradients. For each layer, weights and bias were initialized according to normal distributions with zero mean and unit standard deviation. They were updated by gradient descent with momentum and $L2$ regularization. Parameters (inputs, outputs, weights and gradients) for every layer were stored here.

Lastly, the Neuralnetwork class created a list of Layers. Forward and backward functions defined in this class recursively call the forward and backward functions in Activation and Layer classes. Loss function and initial delta were normalized by number of classification classes (10 for Fashion MNIST) and number of examples in each mini-batch. This class was initialized with specifications given in the configuration file. These three classes were used for later tasks: numerical approximation, training, testing and experimentation.

For training the neural network, we used mini-batch stochastic gradient descent as well as momentum in our update rule weighted by a term γ. We also implemented early stopping to use the weights

with the best performance on the validation set during the training period; however, for the purposes of generating graphs for this report, we allowed the model to run for the full number of epochs and stored the weights that best performed on the validation set. We used an initial learning rate of $1.0e-3$, mini-batch size of 128, number of epochs of 100, and momentum gamma of 0.9. After trying other values for these features in order to optimize the model, we found that changing the learning rate of $1.2e-2$ while leaving the other parameters the same gave the best performance.

2.2 Numerical Approximation

In order to confirm that our code was working, we computed the gradients using numerical approximation:

$$\frac{d}{d\omega}E(\omega) \approx \frac{E(w+\epsilon) - E(w-\epsilon)}{2\epsilon} \tag{1}$$

with $\epsilon = 10^{-2}$ and compared the results with answers obtained from backpropagation. We used 10 examples, one in each category, and added 10 numbers in the end for comparison. Each final answer is computed with only one weight changed in the network for numerical approximation.

2.3 Network Optimization Strategies

2.3.1 Regularization

In order to further optimize our network, we experimented with a variety of aspects of the model. We initially implemented L2 (ridge) regularization, which works to penalize large weights by adding a sum of the weights squared to the loss term. This influences the weight update rule by adding a $2W$ term weighted by a regularization factor, 2λ. We varied the regularization term to optimize the model performance.

2.3.2 Activation Functions

Next, using the best performing regularization rate, we implemented various different hidden unit activation functions to compare to the loss and accuracy obtained by the tanh function. These included sigmoid, ReLU, and leakyReLU activation functions. In order to obtain good performance and avoid overflow errors with these activation functions, we decreased the learning rate by 2 orders of magnitude (1.2e-4) and decreased the batch size to one, leading to slower training.

2.3.3 Network Topology

Lastly, we experimented by changing the neural network topology. This involved halving and doubling the number of hidden units in each layer to analyze the effect on performance. In addition to 50×50, we used the combinations of 50×100, 100×50, 25×50, and 50×25. We also changed the number of hidden layer to 1 and 3 instead of 2 while maintaining approximately the same number of parameters. This means we used 53 nodes in the hidden layer for the network with one hidden layer and 47 nodes in each layer in the 3 layer network.

3 Results & Discussion

3.1 Multi-Layer Neural Network Implementation

In our initial neural network model, we found the best performance using a learning rate of 1.2e-2 and a mini-batch size of 128 after testing multiple different learning rates and mini-batch sizes. We achieved a test accuracy of 78.36% and a test loss of 1.54. The accuracy increases over the training period of 100 epochs, but levels off towards the end with the holdout accuracy being slightly below the training accuracy. While the loss decreases over time for both the training and holdout sets, uncharacteristically, the holdout loss was lower than the training loss. We believe that this was a result of the specific train/validation split; if we were to run the experiment again using a different train/validation split, we will likely see the train loss lower than that of the validation as expected. Both the training and holdout loss drop significantly between the 10th and 20th epoch and again between the 30th and 40th training epoch. The high test accuracy signifies a network that is able to classify between the output classes quite well.

Figure 2: Performance of neural network consisting of 2 hidden layers of 50 nodes and tanh activation and a softmax output layer. Utilized a learning rate of 1.2e-2, momentum gamma of 0.9, and no regularization. Trained using stochastic gradient descent of mini-batch size 128 over 100 epochs with early stopping.

3.2 Numerical Approximation

Numerical Approximation		
Layers	Weights/bias	Differences
Output	Bias(5)	4.49×10^{-4}
Hidden	Bias(2)	3.17×10^{-4}
Hidden	Bias(9)	-1.44×10^{-4}
Hidden to output	Weight(6, 9)	-4.18×10^{-4}
Hidden to output	Weight(31, 2)	-4.25×10^{-4}
Input to hidden	Weight(21, 9)	1.24×10^{-4}
Input to hidden	Weight(13,1)	-5.81×10^{-5}

Figure 3: Checking validity of gradient calculations using numerical approximation.

We observed that the differences between numerical approximation and backpropagation $\approx 10^{-4}$, which is at the order of $O(\epsilon^2)$ since we choose $\epsilon = 10^{-2}$. Numbers in the second column indicate which weight/bias is changed to do the approximation. This confirms that our code for calculating gradients during backpropagation works soundly.

3.3 Network Optimization Strategies

3.3.1 Regularization

Regularization acts to penalize large weights, with the goal of creating a less complex network. Therefore, we experimented with different amounts of regularization to analyze the effect it had on performance. The extent of regularization is captured by the regularization constant, λ, where a larger constant signifies that large weights are being penalized more, and the network will be less complex.

While maintaining a learning rate of 1.2e-2 and mini-batch size of 128 as in our initial experiment, we found that networks with larger regularization constant performed worse. With a regularization constant of $\lambda = 1e-4$, the test accuracy was 73.25% and the loss 7.88; for $\lambda = 1e-3$, the test accuracy was 73.25% and the loss was 1.87; and for $\lambda = 1e-2$, the test accuracy was 19.98% and the loss 1.17. Thus, with these regularization constants, we perform worse than the network with no regularization, which had an accuracy of 78.36% and loss of 1.54 as previously mentioned.

From the loss plots, we see the network with no regularization and that with 1e-2 perform rather similarly. When λ=1e-3, the loss was originally above 20, but dropped exponentially in the first few training epochs. The accuracy plots show that the λ=1e-2 accuracy drops significantly at the start of training, as the weights that are larger but likely useful in classification become penalized. The

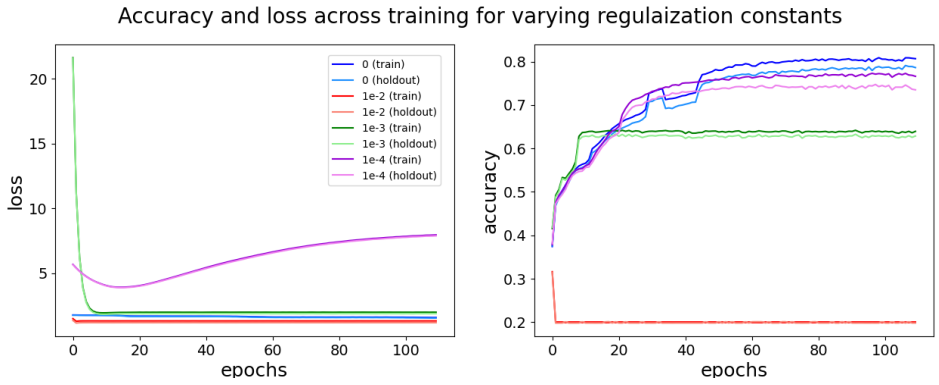

Figure 4: Comparing performance of neural network across varying regularization constants. Consists of 2 hidden layers of 50 nodes and tanh activation and a softmax output layer. Utilized a a learning rate of 1.2e-2 and momentum gamma of 0.9. Trained using stochastic gradient descent of mini-batch size 128 over 100 epochs with early stopping.

small regularization constants as well as the network with no regularization all increase in accuracy over training; however when λ=1e-3, the accuracy plateaus after about 10 training epochs, again suggesting that larger weights to obtain higher accuracy are not able to be utilized in the network. Both the network with no regularization and with λ=1e-4 see increases in accuracy throughout the majority of training, although less towards the end.

We conjecture that higher learning rates did not improve performance because the neural network is not showing signs of overfitting. In our initial model in Section 3.1, the training and holdout loss as well as training and holdout accuracy closely track each other. Consequently, by introducing a penalty in our objective function, we are just hurting the performance of the overall model without benefitting from reduced overfitting.

If network complexity were more of a concern, λ=1e-4 would be a good alternative to the network with no regularization as the performance only drops a small amount, and the network appears to still learn much of the internal structure of the data. However, we continue forward with a network without regularization in order to obtain the best performance.

3.3.2 Activation Functions

Different activation functions will allow the network to learn in a different manner. Here we test different activation functions for the hidden layers and analyze their performance on image classification. We compare the default tanh function to sigmoid, ReLU, and leaky ReLU.

The sigmoid function is similar to the tanh function, although it is not symmetric around the origin in the way the tanh function is. Therefore, the range of values is 0 to 1 rather than -1 to 1. The ReLU function puts a floor on ”x” such that negative values are brought up to zero. The leakyReLU function is similar to the ReLU function, but allows some negative values proportional to ”x” as opposed to a hard floor at zero.

Maintaining the same learning rate of 1.2e-2, batch size of 128, and zero regularization while changing the activation function of the hidden layers greatly varied the performance of the model. In particular, ReLU and leakyReLU were subject to overflow errors that would result in extremely poor performance. Consequently, we decreased the learning rate by 2 orders of magnitude (1.2e-4) and decreased the batch size to 1 for these two activation functions.

With the sigmoid activation function, the test accuracy was 77.43% and the loss 1.08; for the ReLU activation function, the test accuracy was 77.93% and the loss was 1.84; and for the leakyReLU activation function, the test accuracy was 80.07% and the loss 1.84. Thus, all activation functions performed similarly to the tanh activation function, which had an accuracy of 78.36% and loss of 1.54 as previously mentioned, except the leakyReLU function, which performed slightly better.

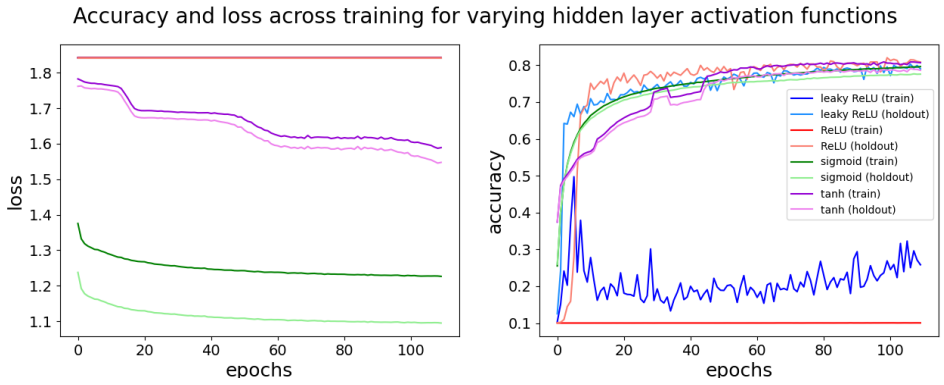

Figure 5: Comparing performance of neural network across varying hidden layer activation functions. Consists of 2 hidden layers of 50 nodes and a softmax output layer. Trained using mini-batch stochastic gradient descent over 100 epochs with early stopping, momentum gamma of 0.9, and no regularization. Tanh and sigmoid utilized a learning rate of 1.2e-2 and batch size of 128. ReLU and leakyReLU utilized a learning rate of 1.2e-4 and batch size of 1.

The sigmoid activation function loss plots behave as expected, slowly decreasing across the number of epochs. There are some signs of overfitting since the validation loss is much lower than the training loss. The sigmoid accuracy plots are also well-behaved, increasing in accuracy over the number of epochs. As the graph of the sigmoid activation function is similar to that of the tanh activation function, it is no surprise that their performance on the data at hand is also similar.

On the other hand, the ReLU and leakyReLU loss plots are ill-behaved, with the loss staying nearly constant throughout the training. The accuracy plots paint an even muddier picture, with the training accuracy for both ReLU and leakyReLU very low (10%-20%) but the validation accuracy well-behaved and quite high (80%).

We are unsure of the cause of the anomolous ReLU and leakyReLU plots. One interpretation is that the learning rate is too low, which prevent the model from learning quickly and results in a constant loss across epochs. Another interpretation is that the learning rate is too high, causing the loss to bounce around local minima and resulting in illogical behavior. A third conjecture is that there is a bug in the code when calculating loss and/or training accuracy. Overall, we conclude that ReLU and leakyReLU are not suitable for this prediction task.

Even though the leakyReLU activation function resulted in a slighly higher test accuracy than the rest, we continued to use the tanh activation function in the rest of the report due to the unstable nature of the leakyReLU loss and training accuracy plots.

3.3.3 Network Topology

Increasing the number of hidden units increases the complexity of the model and allows it to learn representations that might better capture the data. However, this is at the cost of overfitting to the training data and making it less generalizable to unseen data. Here we test different numbers of hidden units within two hidden layers of the neural network.

While maintaining the same learning rate of 1.2e-2, mini-batch size of 128, zero regularization, and tanh activation function as in our initial experiment, we found that networks with different architectures vary widely in performance. With a neural network architecture of 25x50, the test accuracy was 72.14% and the loss 1.58; for 50x25, the test accuracy was 68.22% and the loss 1.42; for 50x100, the test accuracy was 63.84% and the loss 1.48; and for 100x50, the test accuracy was 63.98% and the loss 1.51. None of these architectures performed better than our initial model of 50x50 hidden units, which had an accuracy of 78.36% and loss of 1.54 as previously mentioned.

The loss plots are relatively similar, decreasing throughout the number of epochs with some periods of rapid decline in between. The only outlier is loss plot for 50x100 hidden units, which uncharacteristically increased at first and then decreased faster than any of the other architectures.

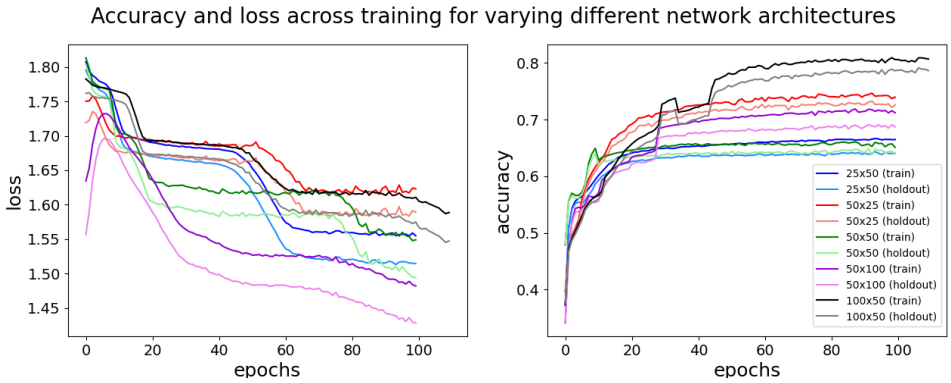

Figure 6: Comparing performance of neural network across varying number of hidden units. Consists of 2 hidden layers and tanh activation and a softmax output layer. Utilized a a learning rate of 1.2e-2, momentum gamma of 0.9, and no regularization. Trained using stochastic gradient descent of mini-batch size 128 over 100 epochs with early stopping.

The accuracy plots are also similar and extremely well-behaved, increasing almost monotonically throughout the number of epochs. Architectures that had a higher number of hidden units in the first layer than the second (ie. 100x50, 50x25) fared better than the reverse architecture (ie. 50x100, 25x50) in terms of training and validation accuracy.

However, these findings were not translated to the test set. We were surprised to find that the 100x50 architecture did not perform well on the test set given its high performance on the training and validation sets. This might suggest a difference between the test and training data that the neural network learned and overfitted for. Consequently, our results suggest that the number of hidden units should not be too large or too less, but just right to get the best performance. If the number of hidden units is too low, there may be some underfitting and the model is not learning as much as it can from the training data. If the number of hidden units is too high, there may be some overfitting preventing the model from generalizing well to the test set.

We can also change the number of hidden layers while maintaining the same number of overall parameters. Deeper neural network structures can learn more representations of the data, but are also prone to overfitting.

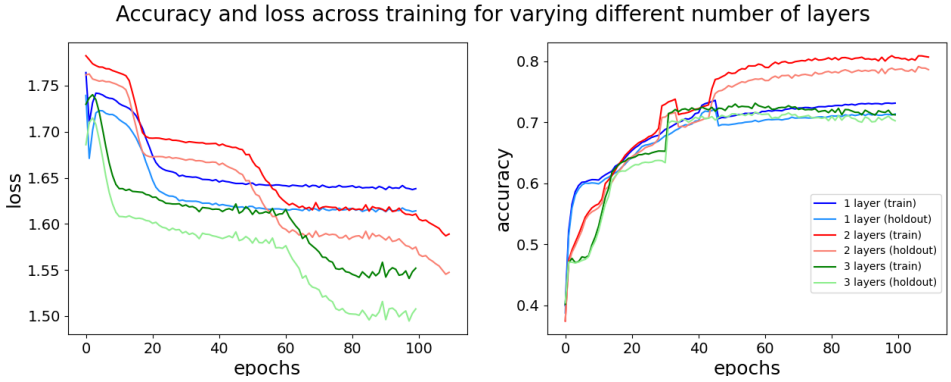

Figure 7: Comparing performance of neural network across varying number of hidden layers. Consists of tanh activation and a softmax output layer. Utilized a a learning rate of 1.2e-2, momentum gamma of 0.9, and no regularization. Trained using stochastic gradient descent of mini-batch size 128 over 100 epochs with early stopping.

While maintaining the same learning rate of 1.2e-2, mini-batch size of 128, zero regularization, and tanh activation function as in our initial experiment, we found that two hidden layers seems to be

the sweet spot in performance between one layer and three layers. With one hidden layer, the test accuracy was 71.38% and the loss 1.61; for three hidden layers, the test accuracy was 70.03% and the loss 1.50. These models perform worse than the network with two hidden layers, which had an accuracy of 78.36% and loss of 1.54 as previously mentioned.

The number of hidden layers seems to follow a similar principle as the number of hidden units, with too many or too few hidden layers performing poorly. Based on the loss and accuracy plots, however, it does not seem that overfitting seems to play as much of a role, but that the three-layer network is simply unable to learn appropriate representations in the data. A moderate number of hidden layers finds the sweet spot that maximizes performance on unseen data.

4 Conclusion

Experimenting with different network optimization strategies offered insight into how to tune neural networks to get the best performance. In general, all parameters seem to follow the "Goldilocks principle" in that the parameter should not be too high or too low, but in a sweet spot that is optimal for the nature of the problem.

A baseline neural network architecture can perform really well on Fashion MNIST Dataset, achieving a test accuracy of 78%. While regularization introduces a penalty in the loss function that prevents the overfitting of data, we observed that increasing the regularization constant actually decreased performance because our model was not overfitting in the first place.

Our experiments varying the activation functions in the hidden layers were largely inconclusive. While tanh and sigmoid performed similarly, ReLU and leakyReLU demonstrated extreme instability in their loss plots and training accuracies, despite respectable test accuracies. Consequently, we assert that ReLU and leakyReLU are not well suited for this prediction task.

Varying the number of hidden nodes and hidden layers also revealed a sweet spot. Too many or too few hidden nodes or hidden layers resulted in poor performance. Our baseline model of 50 units in each of two hidden layers ended up being most optimal.

In future work, we look to test if these principles hold for other datasets. We also aim to run these experiments across a greater sample of learning rates and batch sizes to determine if the model can be even better tuned.

5 Contributions

Anshuman implemented the training and test functions, including updating weights of the neural network, momentum, early stopping, and handling overflow errors. He also assisted in the implementation of backpropagation. He wrote the abstract, conclusion, half the Results & Discussion, and proofread the report prior to submission.

Jinlong implemented Activation, Layer and Neuralnetwork classes, including activation functions, their gradients, forward and backward propagation. He also carried out numerical approximation and proofread the code. He wrote the methods for numerical approximation.

Margot implemented regularization and carried out experiments including the addition of regularization, experimenting with the different activation functions, and experimenting with different network topologies. She also wrote all the code used for plotting. Lastly, she wrote the introduction, methods, and half the results, and helped with proofreading.

We all affirm that the others' contributions were equal in effort and outcome.