# Fully-connected Multi-layer Neural Networks for Fashion MNIST Classification

**Shubham Chaudhary**
UCSD
shchaudh@ucsd.edu

**Allen Zeng**
UCSD
azeng@ucsd.edu

## Abstract

We trained multi-layer neural networks for the classification of 28x28 images of clothing items belonging to ten unique classes. In our first model, we trained a 2-hidden layer network without weight regularization and used a *tanh* activation function for all hidden layers. This model achieved a test accuracy of 84.30%. For a 2-hidden layer network with weight regularization, we compared the use of different activation functions after all hidden layers: *tanh*, *sigmoid*, *ReLU*, and *leakyReLU*. We achieved test accuracies of 85.57%, 87.03%, 84.52%, and 85.66% respectively using these activation functions. Using weight regularization and a *sigmoid* activation function for all hidden layers, we test the perform of neural networks with varied number of hidden layer units and varied number of layers. Between one, two and three layers, the 2-hidden layer network achieved the best test accuracy of 87.36%. We observe that for this data set, a comparable performance (87.32%) was achieved by the network with a single hidden layers as well as a network with 3 hidden layers (87.19%).

## 1 Introduction

This report explores the use of multi-layer neural networks [1] for the classification of images of fashion products from Zalando's Fashion MNIST dataset [2][3]. A modular neural network framework is implemented such that the number of layers, the number of units in each hidden layer, and the layer activation functions can be customized. We explore the use of activation functions such as *tanh*, *sigmoid*, *ReLU*, and *leakyReLU*. A softmax function is applied to the outputs of the last layer to produce prediction class probabilities. Furthermore, we experiment with weight decay via $L_2$ regularization. For our training procedure, we employ mini-batch stochastic gradient descent and momentum-based weight updates. The implementation of the gradients calculations in our network is verified by a method of numerical approximation [1]. We primarily utilize a 2-hiddden layer neural network, but the effectiveness of 1-and 3-hidden layer networks is also tested for this classification problem.

The Fashion MNIST dataset is provided with a training and test dataset already separated [2]. We further divide the initial training dataset into a new training dataset and validation dataset using an 80-20 split. The input image data is normalized to have 0 mean and unit variance. The labels are one-hot encoded. For all of the experiments, we train the models for all epochs but implement an early-stopping procedure. After each epoch we check the validation loss and accuracy of the trained model, and save the model if validation loss has improved. The model corresponding to the lowest validation loss is run on the test dataset to determine the test accuracy. We observe that the sigmoid activation produced the highest test accuracy for a 2-hidden layer neural network. And we observe that for networks with an equal number of layers and parameters, a greater number of units in early hidden layers generally leads to better performance.

## 2 Verifying Gradient Calculations in Neural Networks

It is known [1][2] that the gradient of the loss function with respect to one specific weight in a neural network can be numerically approximated as:

$$\frac{d}{dw}E(w) \approx \frac{E(w+\epsilon) - E(w-\epsilon)}{2\epsilon}$$

where $\epsilon$ is a small constant. We used $\epsilon = 10^{-2}$ in our approximations. This approximation agrees, in big-O notation, within $O(\epsilon^2)$, [1][2]. We were able to verify our implementation of a neural network and its gradient calculations using this approximation.

In the verification procedure, we performed the following steps. First, initialize the model with random weights. Two copies of the model and its weights are created. For a specific weight in the first copied model, $\epsilon$ is added to that weight. Using 10 training examples, one from each class of the dataset, we perform a forward pass and calculate the loss $E(w+\epsilon)$. Then for the same specific weight in the second copied model, $\epsilon$ is subtracted from that weight. Using the same 10 training examples, we perform a forward pass and calculate the loss $E(w-\epsilon)$. Then for the original model, using the same 10 training examples, we perform a forward and backward pass so that the network calculates all of its gradients with respect to the loss. The gradient corresponding to the same specific weight as before is $\frac{d}{dw}E(w)$. Finally, the expression above is evaluated, we verify that the gradient and the approximation agrees within $\epsilon^2 = 10^{-4}$.

We repeated the verification procedure for weights throughout our network, resetting the weights every time. We tabulate results for select weights in Table 1 below. For this experiment, we use the default network described in [2]: 2 hidden layers, each with 50 units and using $\tanh$ activation, and one output layer with 10 labels. Softmax is applied after the output layer and we used cross-entropy as our loss function.

| Weight Description | $dE(w)/dw$ | Approximate Gradient | in $O(\epsilon^2)$? |
|---|---|---|---|
| Output Layer Bias Weight Index [0] | 0.00380513 | 0.00380517 | True |
| Hidden Layer #2 Bias Weight Index [0] | 0.01311730 | 0.01311720 | True |
| Hidden Layer #1 Bias Weight Index [0] | 0.00328705 | 0.00328706 | True |
| Hidden Layer #2 Weight Index [0,0] | -0.01350273 | -0.01350263 | True |
| Hidden Layer #2 Weight Index [1,1] | 0.00289995 | 0.00289973 | True |
| Hidden Layer #1 Weight Index [0,0] | -0.00313974 | -0.00313975 | True |
| Hidden Layer #1 Weight Index [1,1] | -0.00015532 | -0.00015534 | True |

Table 1: Gradients and approximate gradients for select weights in our neural network.

## 3 Multi layer Perceptrons

In this section we describe the structure and training procedure of our neural network for multi-class classification. The data used for this purpose is a collection of images that belong to 10 different classes of clothing articles [3]. Each element of this set is a 28x28 grayscale image, that is labeled with one of the 10 classes. Our Neural Network consumes a flattened image as input i.e. the size of input layer is $28^2 = 784$ units. This input is forwarded to a sequence of 2 hidden layers that contain 50 perceptrons each. Each perceptron applies a non linearity on the weighted sum of the inputs. We have used $tanh()$ as the non linear function. The output of second hidden layer is then forwarded to a softmax layer that maps the output of hidden layers to the range between $[0, 1]$. We intend to interpret the output of softmax layer as the probability of input article belonging to 10 different classes. We use cross-entropy loss to estimate the accuracy of the classification performed by the network. To train the neural network, we have used mini-batch stochastic gradient descent with momentum. The value of momentum parameter used was $0.9$. After experimenting with different learning rates and batch sizes, we found that the best validation accuracy is achieved when learning rate is set to 0.5 and batch size is 64. The training statistics are presented in Figure 1. With this trained model we were able to obtain the test accuracy of $84.30\%$

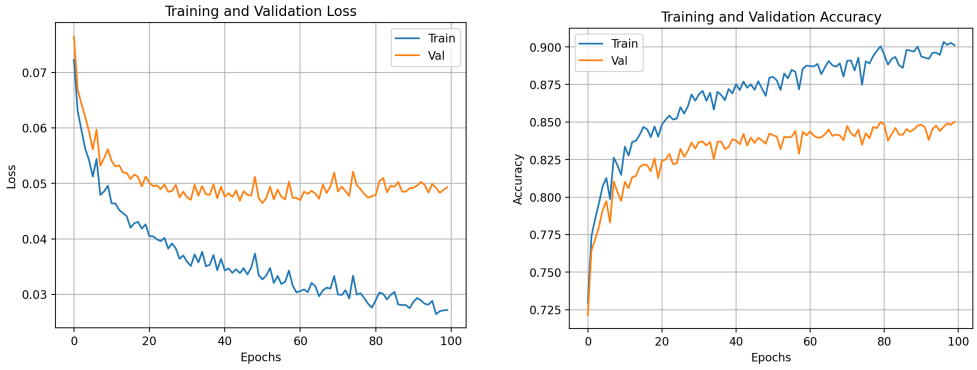



Figure 1: Training and validation losses, training validation accuracies.

## 4 Regularization

Beginning with the hyper parameters used in previous section, we added the L2 penalty on the weights of the model. The number of epochs were also increased from 100 to 110. Experimenting with the values mentioned in [2], we observed that the best validation accuracy was obtained with regularization factor $\lambda = 0.0001$. Presented in Figure 2 are the training statistics. The test accuracy obtained with trained model was $85.57\%$ which is slightly higher than that obtained without any regularization. One of the possible reason for marginal improvement in the performance could be that the model is not large enough to over-fit the data. Even though the improvement in classification accuracy is not much, we can observe that validation losses are now closer to training losses as compared to the case of no regularization.

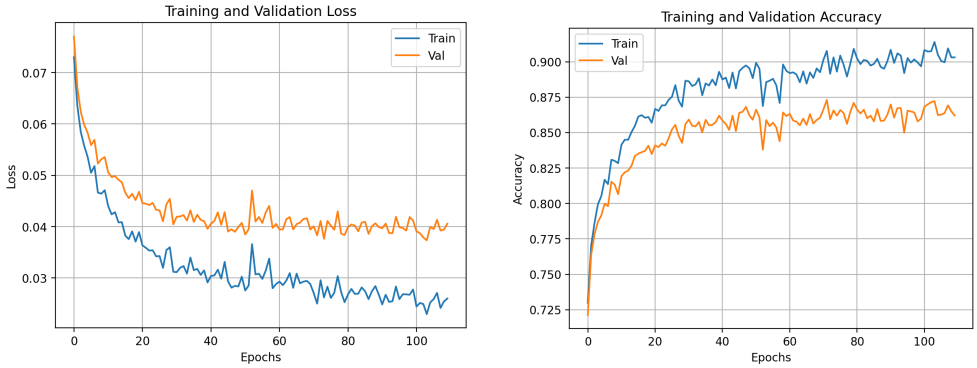



Figure 2: Regularization : Training and validation losses, training validation accuracies.

## 5 Activation Functions

In this section we present the results obtained for different activation function. We used $sigmoid(z) = \frac{1}{1+e^{-z}}$, $ReLU = max(0, z)$ and $leakyReLU(z) = max(0.1z, z)$. The hyper-parameters chosen for $tanh$ activation function namely, learning rate of $0.5$, batch size of $64$ and L2 regularization factor $\lambda = 0.0001$, work well for $sigmoid$ activation function as well but not for $ReLU$ and $leakyReLU$. Figure 3 presents the learning statistics for $sigmoid$ activation function. Test accuracy of $87.03\%$ was obtained using $sigmoid$ activation, which is better than that obtained with $tanh$.

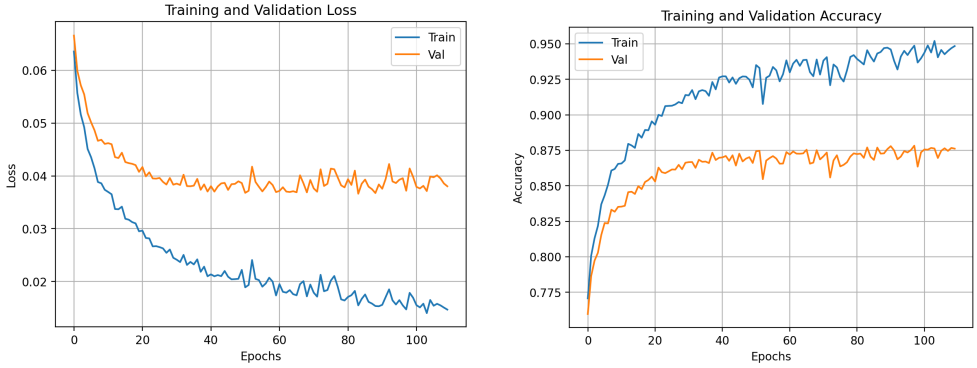



Figure 3: Sigmoid : Training and validation losses, training validation accuracies.

Learning rate of $0.5$, when used with $ReLU$ and $LeakyReLU$ turned out to be very high and resulted in very poor performance. Lowering the learning rate to $0.1$ while keeping rest of the hyper-parameters unchanged, led to good performance with both these activation functions. Figure 4 and 5 present the training statistics of the model using $ReLU$ and $LeakyReLU$ activation functions respectively. We were able to obtain the test accuracy of $84.52\%$ for $ReLU$ and $85.66\%$ for $LeakyReLU$. We observed that even though the accuracy using $ReLU$ activation is close to the accuracy obtained with other function, the validation loss in very close to training loss which led us to the belief that $ReLU$ somehow acts as an implicit regularizer as well.

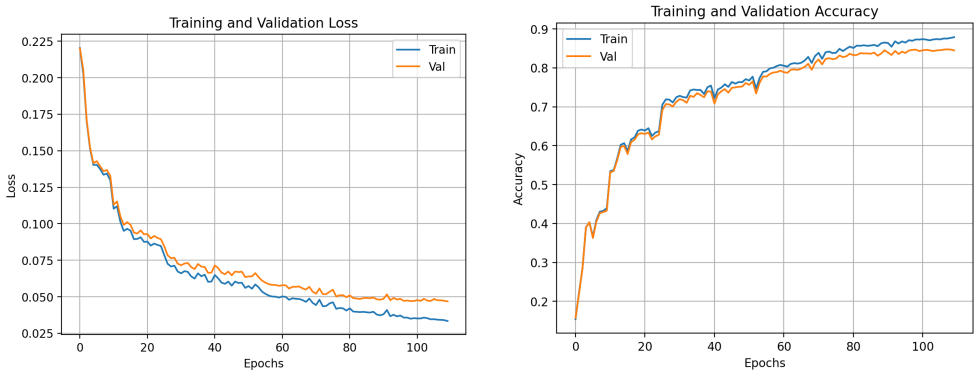



Figure 4: ReLU : Training and validation losses, training validation accuracies.

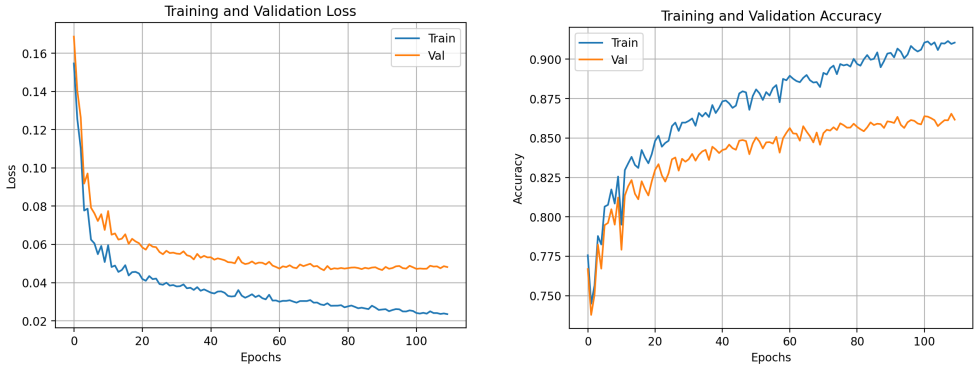
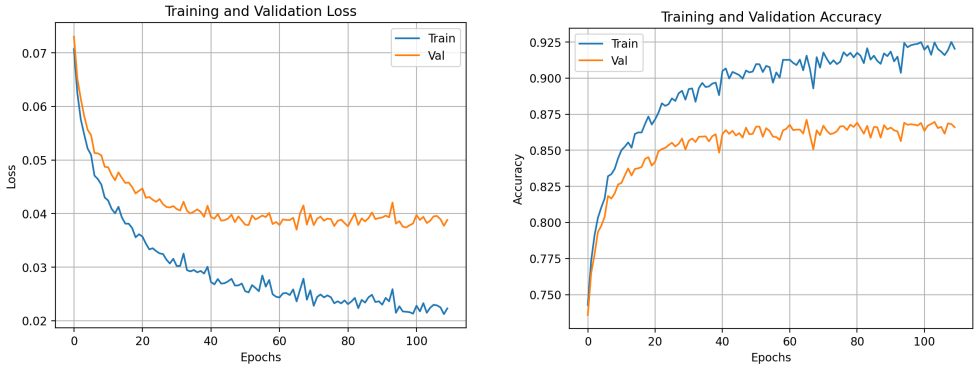


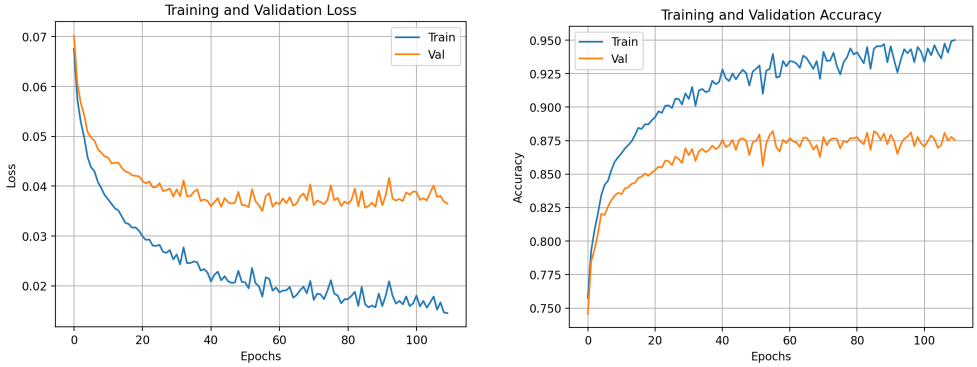

Figure 5: LeakyReLU : Training and validation losses, training validation accuracies.

## 6 Network topology

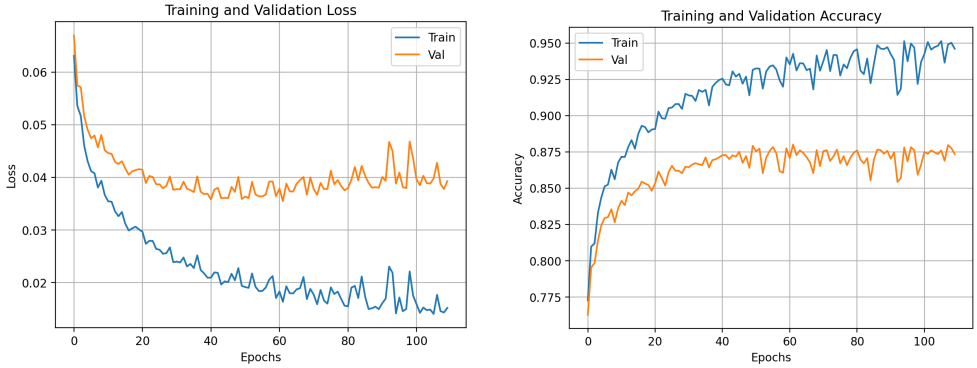
In this section we report the performance of network with different architecture. For these experiments we have used the set of hyper parameters that have given us the best results so far, specifically we have used $sigmoid$ activation function with learning rate $0.5$, batch size $64$, L2 penalty factor $\lambda = 0.0001$. Keeping the number of hidden layers as 2, we performed experiments with different number of neural units in each hidden layer. Presented in the Figure 6 are the training statistics and accuracy of these models. We denote the number of units per hidden layer using the notaion "$a$X$b$", where $a$ and $b$ are the number of units for hidden layer one and two respectively. (We similarly denote the number of hidden units for three hidden layers as "$a$X$b$X$c$".) Testing accuracy obtained corresponding to neural networks with hidden layer size 25X50, 50X25, 50X100 and 100X50 were 85.71%, 86.66%, 86.89% and 87.36% respectively. We can observe that reducing the number of hidden units leads to a slight dip in the test accuracy (from 87.03% to 85.71%), and increasing the number of hidden units led to an even slighter improvement in test accuracy (from 87.03% to 87.36%). Also notice that by increasing the number of parameters, validation loss just begins to climb up gradually, which is a sign of over-fitting. We infer from these findings that 50X50 hidden units are most suitable for training on this data. For the cases where the number of parameters we identical i.e. in 25X50 and 50X25, and in 50X100 and 100X50, we notice that the network with larger layer first, perform slightly better. We explore this further in the next set of experiments.



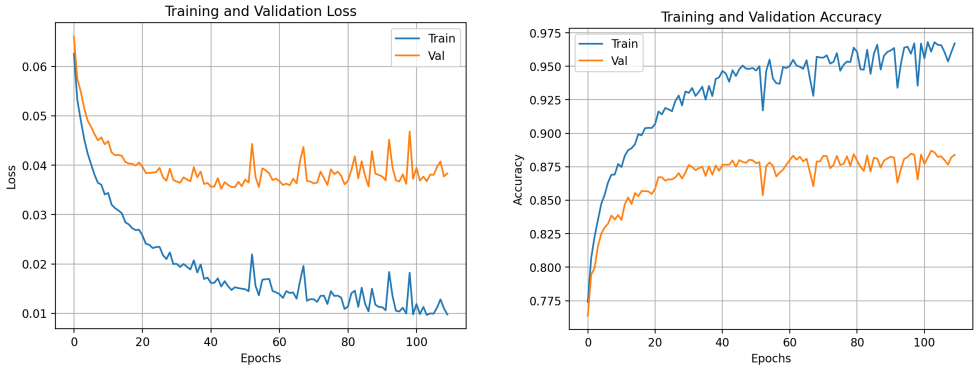

(a) 25X50 Test Accuracy 85.71%




(b) 50X25 Test Accuracy 86.66%



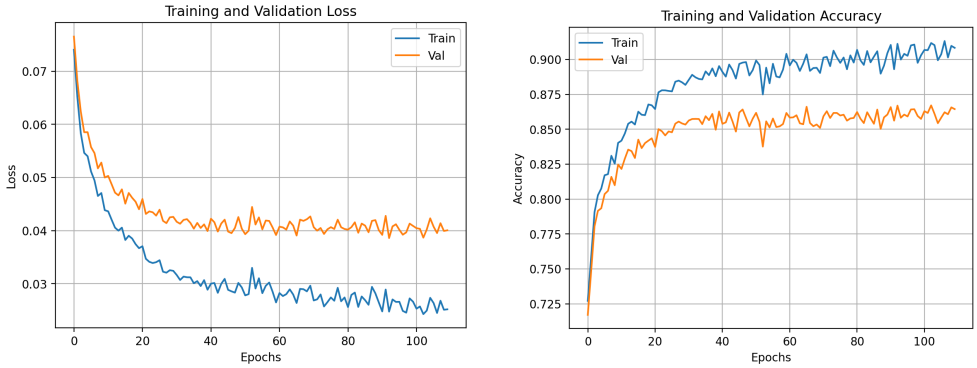

(c) 50X100 Test Accuracy 86.89%



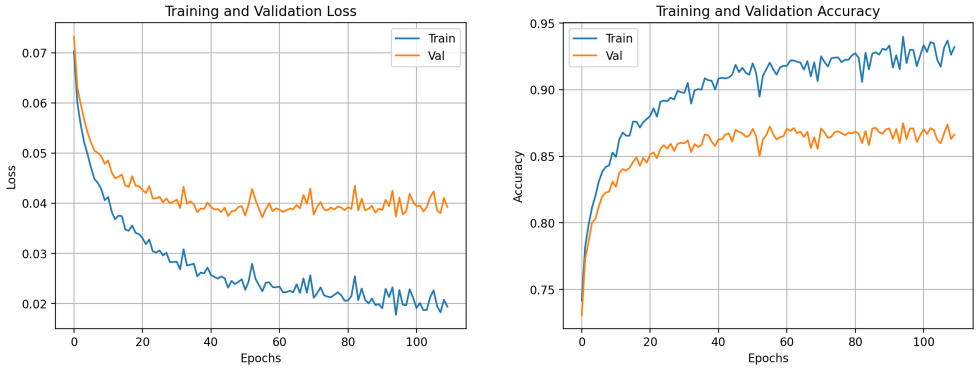

(d) 100X50 Test Accuracy 87.36%

Figure 6: Variation of hidden layer size

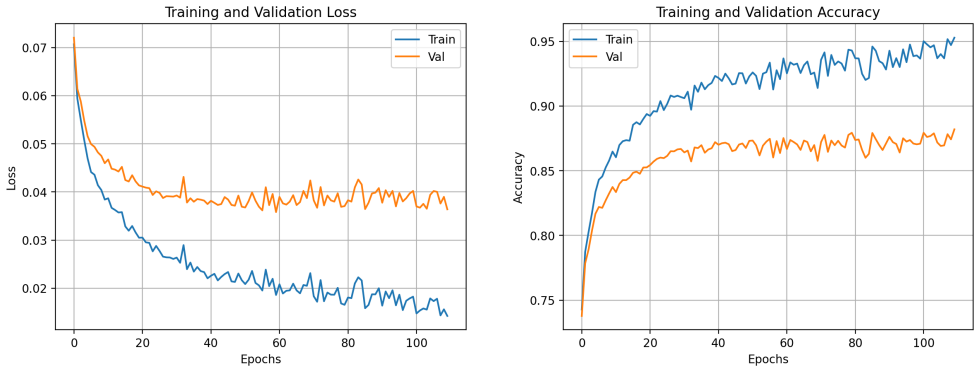
In the next set of experiments we change the number of hidden layers to 3. We chose 3 different sizes 20X30X50, 33X33X33 and 50X30X20 that resulted in test accuracy of 85.53%, 86.43% and 87.19%. The leaning statistics for these experiments are presented in Figure 7. We observe that it is better use more neural units for the earlier hidden layers. A probable reason could be the loss of data, at initial stages, due to the deficit of parameters, leads to significant information loss and hence more performance penalty.




(a) 20X30X50 Test Accuracy 85.53%




(b) 33X33X33 Test Accuracy 86.43%




(c) 50X30X20 Test Accuracy 87.19%

Figure 7: Networks with 3 hidden layers

In this last section we present the results using only one hidden layer with 100 neurons. The training statistics are demonstrated in 8. Using this network we were able to achieve 87.32% test accuracy which is very close to the highest accuracy achieved so far (87.36%).

Observing the similar performance of networks with different typologies, we have come to the conclusion that all these networks have enough parameters to separate the classes equally well. Also, not much gain in performance was achieved by adding more parameters (in the form of layers as well as hidden units) and we could begin to observe the signs of overfitting. All these findings led us to conclude that the dataset is simple enough to be modelled by a network with 2 hidden layers with 50 units each, when trained using batch gradient descent with momentum and regularization.

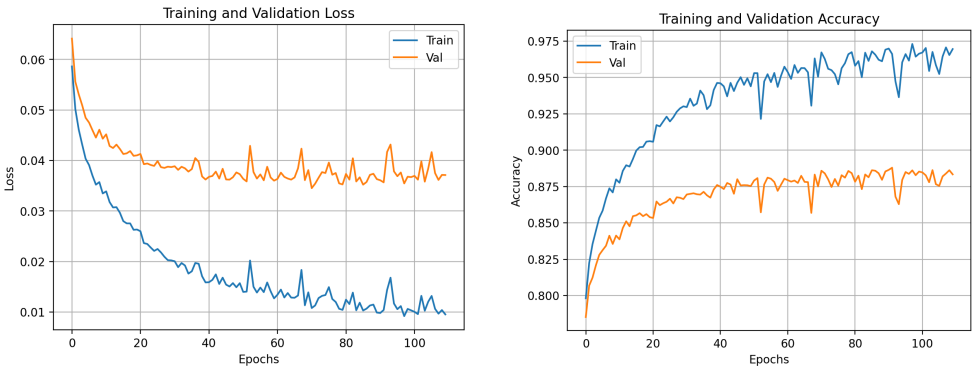



Figure 8: Network with 1 hidden layer

## 7 Individual Contributions

Shubham and Allen worked on the code asynchronously and synchronously during scheduled meetings. Once the training procedure was in place and the sanity checks passed, we were able to run the different experiments individually and record them into this report. We wrote the report sections individually at first, and then reviewed each other's work.

Specifically, Shubham worked on implementing the activation functions and gradients in 'neuralnet.py', including 'softmax()', 'Activation.sigmoid()', 'Activation.tanh()', 'Activation.ReLU()', 'Activation.leakyReLU()', 'Activation.grad_sigmoid()', 'Activation.grad_tanh()', 'Activation.grad_ReLU()', 'Activation.grad_leakyReLU()'. He also implemented 'Layer.forward()', 'Layer.backward()'. For the report, he wrote the sections for experimenting with no regularization, with regularization, with different activation functions, and different network topologies.

Specifically, Allen worked on 'gradient_checker.py' for numerically approximating the gradients in part 3(b) of [2]. In 'neuralnet.py', he implemented the train-validation split, the plotting code, the training loop in 'train()', 'test()', 'calculate_accuracy_from_logits()', 'Neuralnetwork.forward()', 'Neuralnetwork.loss()', 'Neuralnetwork.backward()', 'Neuralnetwork.update()'. He also implemented $L_2$ regularization and momentum in 'Layer.update()'. He wrote the 'readme.txt' file. For the report, he wrote the abstract, introduction, and section on verifying gradients calculations.

### Acknowledgments

We used the code written by Manjot Bilkhu and the CSE 251B Staff to to load in the Fashion-MNIST dataset and run sanity checks on our neural network implementation.

### References

[1] Bishop, C. (1995) *Neural Networks for Pattern Recognition*

[2] UCSD CSE 251B Staff (2021) *Programming Assignment 2, CSE 251B: Neural Networks for Pattern Recognition, Winter 2021*, pp. 3-5. San Diego, CA.

[3] Xiao, H., Rasul, K. & Vollgraf, R. (2015) *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* arXiv:1708.07747 [cs.CV].