Abstract

In this project, the performances of several neural network models were investigated on the Fashion MNIST dataset [1]. In particular, several variations of the multilayer perceptron (MLP) and the convolutional neural network (CNN) were used to make predictions on the dataset. It was found that the CNN was more accurate at generalizing than the MLP model by 2.43%, with a reported test accuracy of 90.83%; but was significantly slower to train. The best-performing MLP architecture obtained a lower test prediction accuracy of 88.4%; but was faster to train than the CNN model.

1. Introduction

The goal of this project was to use the multilayer perceptron (MLP) and the convolutional neural network (CNN) machine learning (ML) models to classify image data from the Fashion MNIST dataset [1]. The best performing model architecture was selected based on experiments conducted with various hyperparameter variations. The results of the tests concluded that the CNN architecture with 2 convolutional layers, 2 dense layers (width = 1000), Adam optimizer, ReLU activation, 2 strides, and 1 layer of padding outperformed all other models with a test prediction accuracy of 90.83%.The best performing MLP model with 2 hidden layers, 512 layer width, batch size of 60, SDG optimizer, and ReLU activation, had the highest generalization accuracy 88.4%. The detailed performance reports of each model can be found in the "Results" section of this document.

2. Relevant Work

To obtain a general idea of data pre-processing techniques, relevant ML models, and prediction accuracy results before starting any experiments, it was useful to review related projects. For the first reference, [2] was deemed relevant as it applied varying architectures of the CNN algorithm on the Fashion MNIST dataset. The authors of the paper evaluated the performance of the model with different activation functions (e.g. sigmoid, softmax, ReLU, and tanh), optimizers (e.g. Adam, Adagrad, Adadelta, SGD, and RMSprop), learning rates, and batch sizes. One of the highest reported accuracies in [2] (train acc. = 92.93% and test acc. = 89.21%) was generated with the Adam optimizer, sigmoid activation function, 100 Batch size, 60 epochs, and a 0.002 learning rate. Another experiment showed that the stochastic gradient descent (SGD) optimizer had a lower CNN performance accuracy: train accuracy = 87.99% and test accuracy = 83.53%.

[3] also experimented with varying CNN architectures on the Fashion MNIST dataset by using 60,000 training and 10,000 test gray scale images of 28x28 pixels on their two layer CNN implementation. One of the best training accuracies (99.90%) for this model were produced with the 2-layer CNN implementation, the Adam optimizer, and 40 epochs. A best test accuracy of 94.99% was achieved with the 4-layer CNN implementation at 60 epochs and the Adam optimizer.

[4] compared CNN models with traditional non-convolutive ML algorithms such as K-Nearest Neighbors (test acc. = 85.40%), Logistic Regression (test acc. = 84.20%), Decision Tree (test acc. = 79.80%), and Support Vector Classification (test acc. = 89.70%). [4] also mentioned test accuracies for the MLP (test acc. = 87.00%) and the CNN models (test acc. = 99.05%).

3. Datasets

The Fashion MNIST is a 10-class dataset containing 70,000 grayscale clothing images separated into 60,000 examples for training and 10,000 examples for testing. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels. After performing explorative analysis on the data, it was discovered that the classes are evenly distributed in both the train (6000 examples/class) and test (1000 examples/class) datasets. Additionally, after no NaN values were discovered in the data, the train and test datasets were reshaped from (60000, 28, 28) to (60000, 784) and (10000, 28, 28) to (10000, 784), respectively.

4. Results

Regarding the MLP implementations, 2 model variations were used: the 1st/main version was inspired by the code in the *Numpy-L-MLP* [5] course notebook, while the 2nd implementation is adapted from open-source *Neural Networks From Scratch* [6] textbook that one of the team members had read over the summer. The 2nd implementation was created and used as a part of an experiment to see how a different design would perform in comparison to the 1st/main one. Both versions conducted tests in "Part 1" through "Part 4", and the best-performing MLP model is mentioned in "Part 6".

Part 0 - Choosing hyperparameters

Learning Rate: CNN (5 epochs)

Learning Rate	0.001	0.01	0.05	0.1	0.2 (too large)	0.5 (too large)
Training Accuracy	80.90%	87.39%	91.03%	**92.77%**	~10%	~10%
Test Accuracy	80.66%	86.2%	88.8%	**90.64%**	~10%	~10%

Comment: A learning rate of 0.1 was chosen for its most accurate convergence.

Learning Rate: MLP Implementation 1

Learning Rate	0.001	0.01	0.05	0.1	0.5	1
Training Accuracy	75.41%	76.48%	77.95%	**79.23%**	78.12%	77.1%
Test Accuracy	74.85%	74.76%	75.58%	**77.82%**	76.79%	75.73%

Comment: A learning rate of 0.1 was chosen for its most accurate convergence.

Learning Rate: MLP Implementation 2 (20 epochs and 2 hidden layers)

Learning Rate	0.001	0.01	0.05	0.1	0.5	1
Training Accuracy	13.7%	80.05%	88.5%	**90.3%**	91%	20.1%
Test Accuracy	14.1%	57.1%	86.7%	**87.6%**	87.4%	20.0%

Comment: A learning rate of 0.1 was chosen for its most accurate convergence.

Part 1 - MLP variants and softmax classification

Using the chosen learning rates from Part 0, the reported accuracies for the three MLP variants (0, 1, 2 hidden layers) are summarized below. For Implementation 2, it can be concluded that a deeper MLP works better than a shallow one because they can learn features at different abstraction layers and hence generalize more accurately. In Implementation 1, the same behavior is not observed, it is therefore possible that the model is causing underfitting at 2 hidden layers. Additionally, using a non-linear ReLU activation function improves prediction because it helps to address the vanishing gradient problem. Note: the relevant graphs of the testing and training accuracy for Implementation 2 as a function of training epochs can be observed in FIGURES 1. The relevant figures for Implementation 1 can be observed in FIGURES 6-8.

- **MLP Implementation 1 (no hidden layer):** train accuracy = 74.20%; test accuracy = 73.64%
- **MLP Implementation 1 (1 hidden layer):** train accuracy = 80.99%; test accuracy = 79.11%

- **MLP Implementation 1 (2 hidden layers):** train accuracy = 74.30%; test accuracy = 72.49%
- **MLP Implementation 2 (no hidden layer):** train accuracy = 86.1%; test accuracy = 84.4%
- **MLP Implementation 2 (1 hidden layer):** train accuracy = 90.0%; test accuracy = 87.6%
- **MLP Implementation 2 (2 hidden layers): train accuracy = 90.43%; test accuracy = 87.7%**

Part 2 - MLP with 2 hidden layers

The tanh and Leaky ReLU (LReLU) are two additional activation functions used for transforming the summed weighted node input into an output that is fed to the next hidden layer. Much like the sigmoid activation function, tanh can be a source of the vanishing gradient problem; a problem mostly solved with the ReLU. Unlike the ReLU however, the LReLU uses a small slope (alpha) parameter to incorporate some information from the negative values. As a result, LReLU is expected to perform better than the tanh activation function and have a comparable performance with ReLU.

Observing the reported accuracies below, models using ReLUs perform better than the tanh activation function. The similar but slightly worse performance of LReLU in comparison with the ReLU in the 2nd MLP implementation can be explained by the fact that weights are initialized with random positive values - which decreases the probability of having negative activations throughout training and thereby produces a similar output to that of the ReLU. In implementation 1, the expected behavior is observed and LReLU outperforms ReLU because the weights are initialized to random positive/negative values. Note: the relevant graph of the testing and training accuracy for Implementation 2 as a function of training epochs can be observed in FIGURES 2. The relevant figures for Implementation 1 can be observed in FIGURES 9-10.

- **MLP Implementation 1 (tanh):** train accuracy = 66.77%; test accuracy = 41.3%
- **MLP Implementation 1 (LReLU; a=0.1):** train accuracy =77.0%; test accuracy = 74.91%
- **MLP Implementation 1 (ReLU):** train accuracy = 74.30%; test accuracy = 72.49%;
- **MLP Implementation 2 (tanh):** train accuracy = 89.3%; test accuracy = 87.2%
- **MLP Implementation 2 (LReLU; a=0.01):** train accuracy = 89.8%; test accuracy = 87.6%
- **MLP Implementation 2 (ReLU): train accuracy = 90.43%; test accuracy = 87.7%**

Part 3 - MLP with 2 hidden layers, ReLU activations, and L2 regularization

Increasing the depth of a network may reduce bias, but can increase the variance. The regularization technique, such as the one in Ridge/L2-regularized regression, is used to apply a penalty to the learning parameters in order to decrease overfitting. In this experiment, the weight and bias L2 regularizers were set to 0.01. The training and test prediction accuracies for MLP Implementation 2 are 84.7% and 84.2%, respectively. The training and test prediction accuracies for MLP Implementation 1 are 72.8% and 72.3%.

Regularization is often useful for data that contains more features than training data, which often leads to overfitting. In the case of the Fashion MNIST dataset, the significantly large 60,000 training samples and 784 features does not fall under that category - which explains why the reported accuracies are less than those without regularization. Note: the relevant graph of the testing and training accuracy for Implementation 2 as a function of training epochs can be observed in FIGURES 3. With regards to Implementation 1, FIGURES 11 to 15 display our results for the MLP without L2 Regularization (FIGURES 11 - 12) and with the L2 Regularization (FIGURES 14 - 15) which was optimized by testing various L2 Lambda values (FIGURE 13).

Part 4 - MLP with 2 hidden layers, ReLU activation, on unnormalized dataset

Normalizing is meant to put the feature values within a common small range. This is often important if one feature has values in the low negative hundreds, another feature has values in the high hundreds, and other features are centered around 0. In the case of the Fashion MNIST dataset, the feature values are all

within a range of [0, 255] and it is expected that normalizing the dataset should not significantly impact the prediction accuracies. After performing the experiment, the MLP Implementation 2 training prediction accuracy was 92.3% and the reported test accuracy was 87.1% which is comparable to the values generated in Part 1. It can therefore be concluded that using the Fashion MNIST dataset on Implementation 2, normalizing the data only slightly improves the generalization accuracy. For MLP Implementation 1, using the unnormalized dataset generated a training prediction accuracy of 59.8% and a test accuracy of 59.4% which was less than for the normalized dataset and therefore indicates that the implementation depends on a normalized dataset. Note: the relevant graph of the testing and training accuracy as a function of training epochs can be observed in FIGURES 4. With regards to Implementation 1, FIGURES 16 and 17 display the results for the unnormalized dataset.

Part 5 - CNN with 2 convolutional, 2 fully connected layers, and ReLU activations

Although both CNN and MLP are good models to use for general image classification tasks, the main reason for using CNN is the lack of scalability of MLP to high resolution images. Additionally, the MLP takes a flattened image vector as input, while the CNN takes an image tensor [8]. As a result, CNN can better understand the relations between nearby image pixels. It is therefore superior to the MLP for more complex image classification tasks. This generalization is confirmed with the prediction accuracy results of the experiment. Using 2 convolutional layers, 2 dense layers (with hidden units set to 128 and 10, respectively), ReLU activation, stochastic gradient descent, and softmax activation, the formerly stated architecture of the CNN makes predictions with a train accuracy of 99.07% and a test accuracy of 88.65%. Note: the relevant graph of the testing and training accuracy as a function of training epochs can be observed in FIGURES 5.

Based on the hyperparameter tuning in Part 7, the best performing CNN model appears to be the one with the Adam optimizer, ReLU activation, 1000 dense layers, 2 strides, and 1 layer of padding. The architecture generates training prediction accuracy of 95.31% and test prediction accuracy of 90.83%.

Part 6 - Best-performing MLP architecture

After conducting several experiment variations on both MLP Implementation 1 and 2 with the following parameters summarized in the Appendix Table 1, the best-performing MLP architecture was determined to be MLP Implementation 2. The highest performing architecture (test prediction accuracy 88.4%) has a learning rate of 0.1, a layer width of 512, batch sizes of 60, and activation function of ReLU.

Part 7 - CNN Experiments

Based on the reported prediction accuracies, the best performing model (MLP or CNN) in the above experiments was the CNN. Additionally, out of all the relevant work papers cited in section 2, the CNN was also reported to perform best on the Fashion MNIST dataset. It would therefore be relevant to test out different architecture variations of the model. Some changes were inspired by the best-performing architectures in the “Relevant Work” section of the document. The parameter changes and predicted accuracies can be observed from the table below. Note: the number of epochs was set to 5 for all experiments in order to minimize the time required to complete the CNN training and testing.

	Optimizer	**Dense Layer Width**	**Strides**	**Padding**	**Train Accuracy (%)**	**Test Accuracy (%)**	**Notes**
Experiment 1	Adam	128	1	Valid: no padding	95.59%	90.41%	Adam is the best optimizer because it has faster computation time and requires
Experiment 2	Adagrad	128	1	Valid: no padding	83.75%	82.77%	

Experiment 3	Adam	128	2	Same: zeros around	93.69%	90.62%	fewer parameters for tuning. It also uses the first and second moments while AdaGrad only uses the second moment [10].
Experiment 4	Adam	1000	2	Same: zeros around	95.31%	90.83%	Increasing width here does capture the variations of the data better and produces more accurate results.
Experiment 5	Adam	10	2	Same: zeros around	89.72%	88.48%	Decreasing the width to such a low number leads to slight underfitting.
Experiment 6	Adam	5000	2	Same: zeros around	95.73%	90.56%	Increasing the width to 5000 eventually leads to overfitting.
Experiment 7	Adam	1000	4	Same: zeros around	91.94%	89.95%	Using a stride of 4 downsampled the output.

Part 8 - MLP Implementation 2: Adam Optimizer Experiment

[1], [2], [3], [9], [10] all mention that Adam is also a great optimizer to apply to a neural network because it combines the advantages of Adagrad, RMSprop, and SGD with momentum. It was therefore useful to try the MLP Implementation 2 with the Adam optimizer. After conducting the experiment (learning rate set to 0.001), the accuracy of the Adam optimizer was around the same as the best-performing MLP, obtaining a training prediction score of 94.9% and a test prediction score of 88.2%.

5. Discussion and Conclusion

Although the best performing MLP and CNN model was found by trial and error of various architecture designs. In the next interaction of the project, it would be best to find the model hyperparameters through cross validation techniques. Despite any additional potential improvements, the overall task of testing the performance of the models was accomplished. The data was acquired, analyzed, and preprocessed. The ML models were then implemented and classifier performance was evaluated on the Fashion MNIST dataset.

6. Statement of Contributions

Everyone contributed an equal amount of effort. All team members initially implemented each ML model individually, however towards the end, each member took a specific model, improved it, and performed the testing experiments. The report was also written and previewed by everyone in the group.

APPENDIX

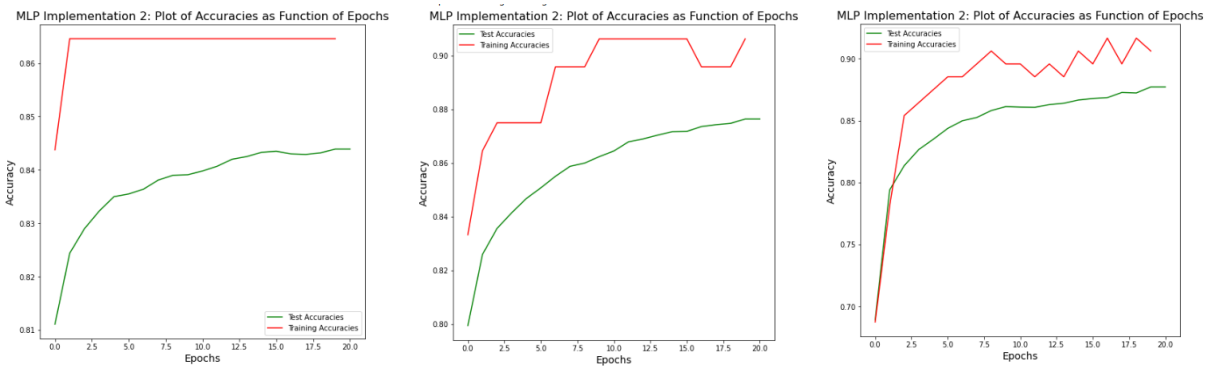
Tables

Table 1

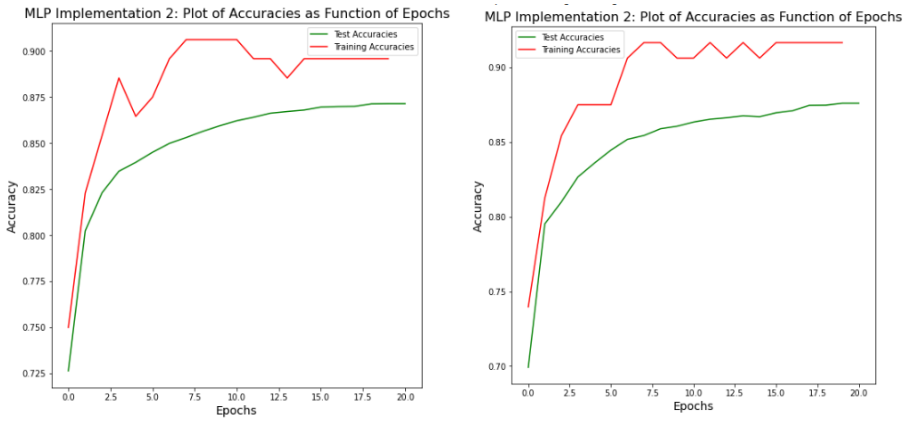
Hyperparameters	Dense Layer Width	Batch_size	Activation Function	Test Prediction Accuracy Implementation 1/2
Variation 1	64	64	ReLU	66.37%/87.4%
Variation 2	64	64	LReLU	68.95%/87.5%
Variation 3	256	128	ReLU	74.45%/87.9%
Variation 4	256	128	LReLU	75.06%/87.8%
Variation 5	512	128	ReLU	75.12%/88.1%
Variation 6	512	128	LReLU	**76.42%**/88.2%
Variation 7	512	60	ReLU	67.69%/**88.4%**
Variation 8	512	60	LReLU	72.33%/88.2%

Figures

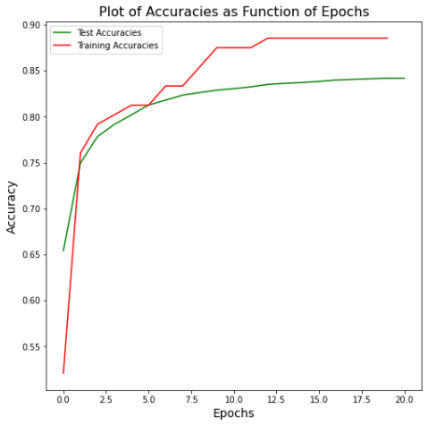
FIGURES 1 - MLP Implementation 2 with No Hidden Layers



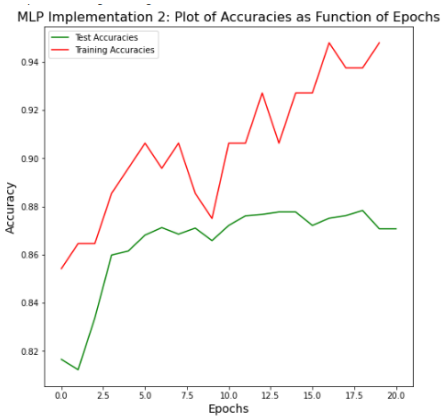
FIGURES 2 - MLP Implementation 2 with 1 Hidden Layer



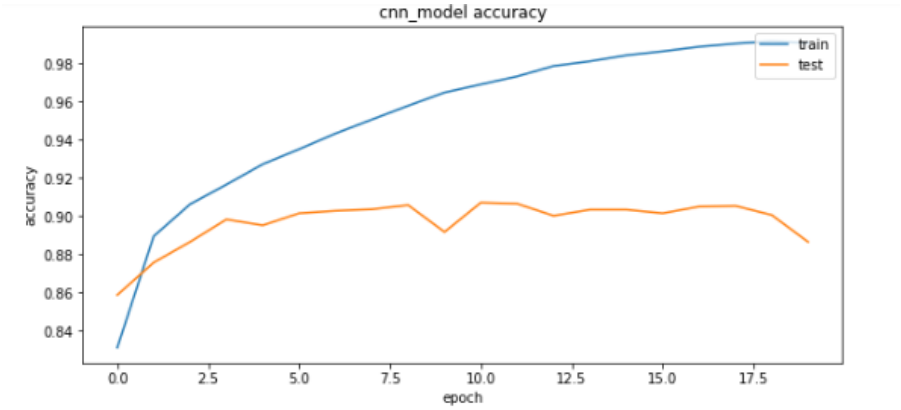
FIGURES 3 - MLP Implementation 2 with 2 Hidden Layers



FIGURES 4 - MLP Implementation 2 with 2 Hidden Layers + LeakyReLU



FIGURES 5 - CNN Accuracy for Various Numbers of Epochs



FIGURES 6 - MLP Implementation 1 with No Hidden Layers

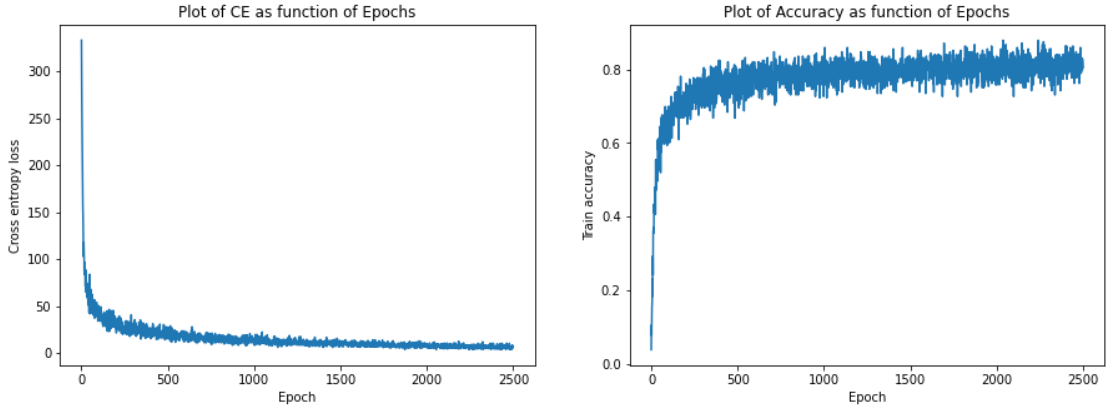


FIGURE 7 - MLP Implementation 1 with 1 Hidden Layer

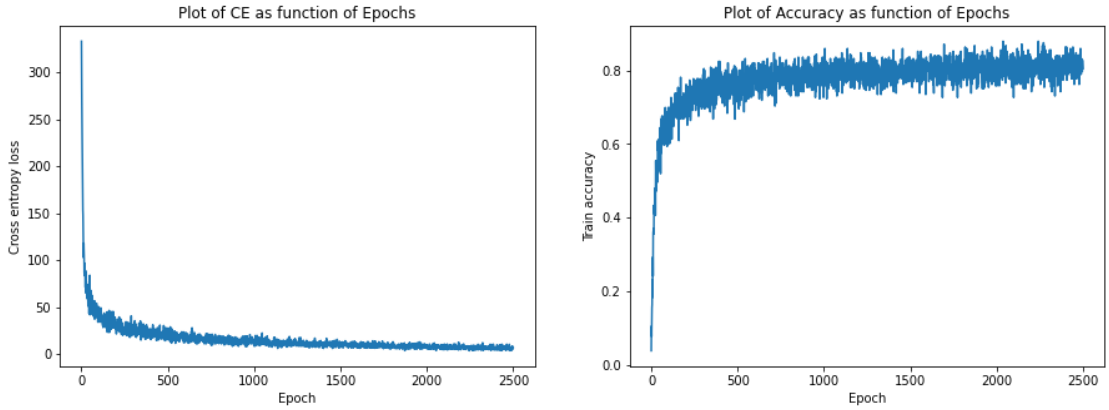


FIGURE 8 - MLP Implementation 1 with 2 Hidden Layers

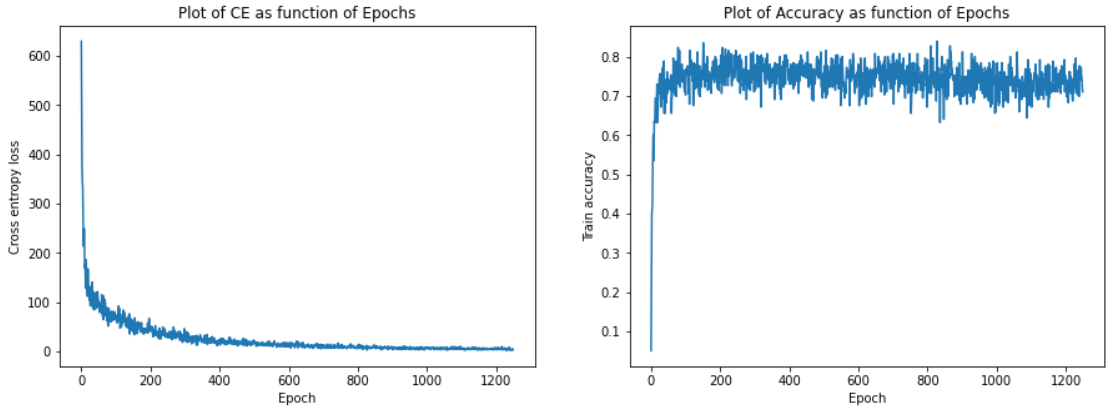


FIGURE 9 - MLP Implementation 1 with 2 Hidden Layers + LeakyReLU

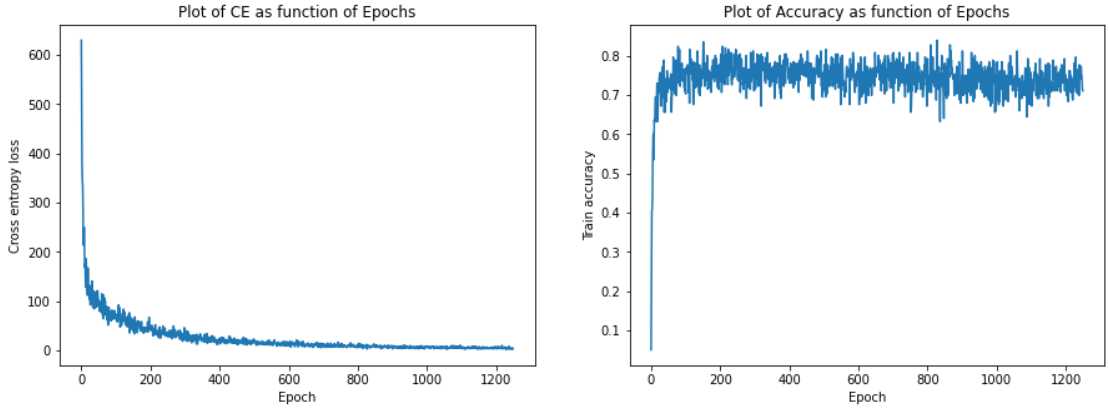


FIGURE 10 - MLP Implementation 1 with 2 Hidden Layers + Tanh

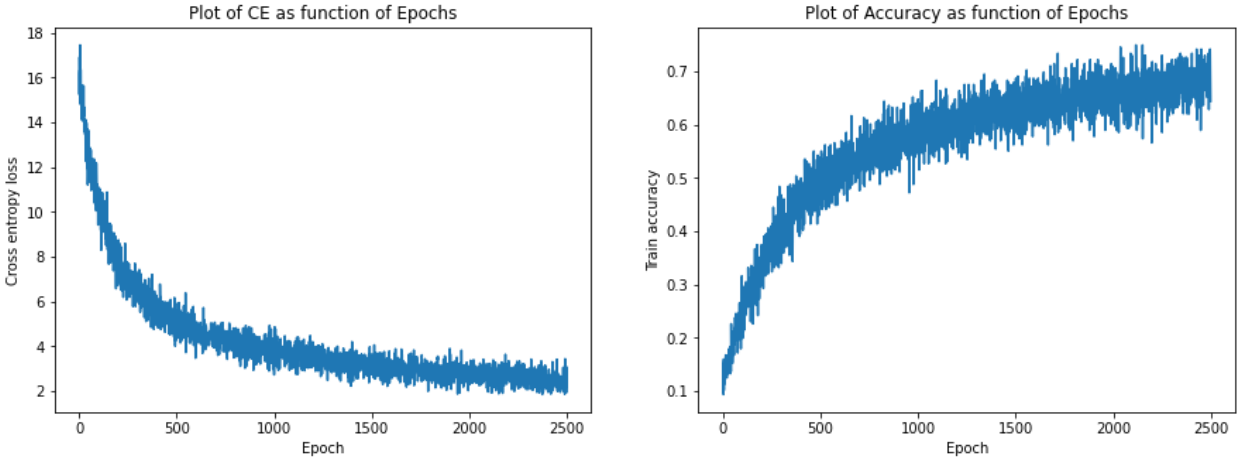


FIGURE 11 - MLP Implementation 1 with 2 Hidden Layers + RELU Activation for Various Epoch Values

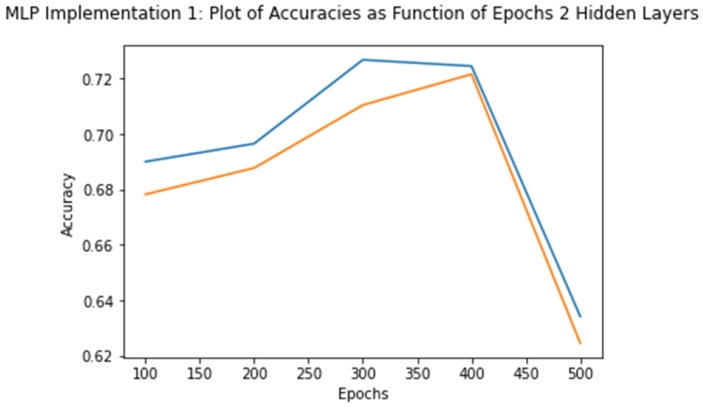


FIGURE 12 - MLP Implementation 1 with 2 Hidden Layers + RELU Activation for 350 Epochs

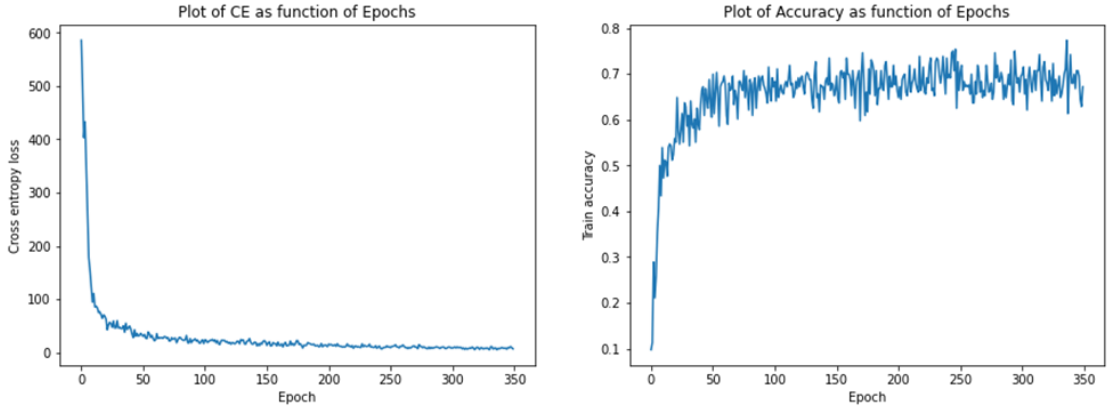


FIGURE 13 - MLP Implementation 1 with 2 Hidden Layers + RELU Activation for Various L2 Regularization Lambda Values

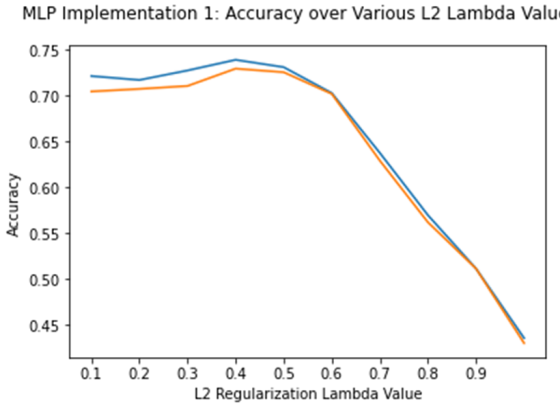


FIGURE 14 - MLP Implementation 1 with 2 Hidden Layers + RELU Activation with L2 Regularization for Various Epoch Values

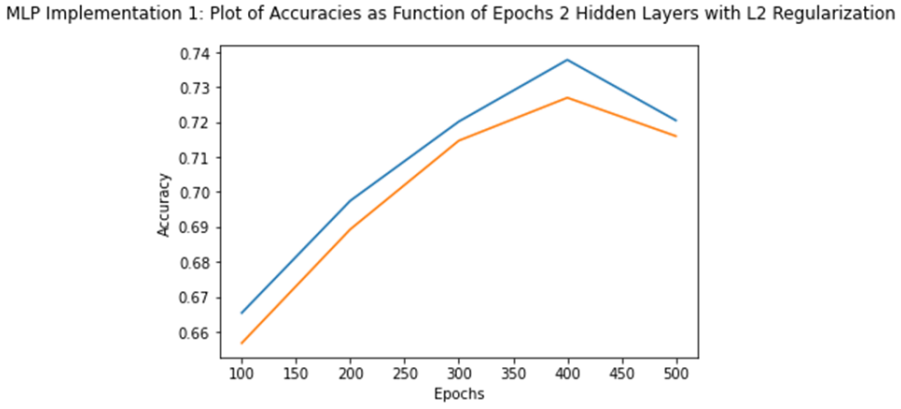


FIGURE 15 - MLP Implementation 1 with 2 Hidden Layers + RELU Activation with L2 Regularization for 400 Epochs

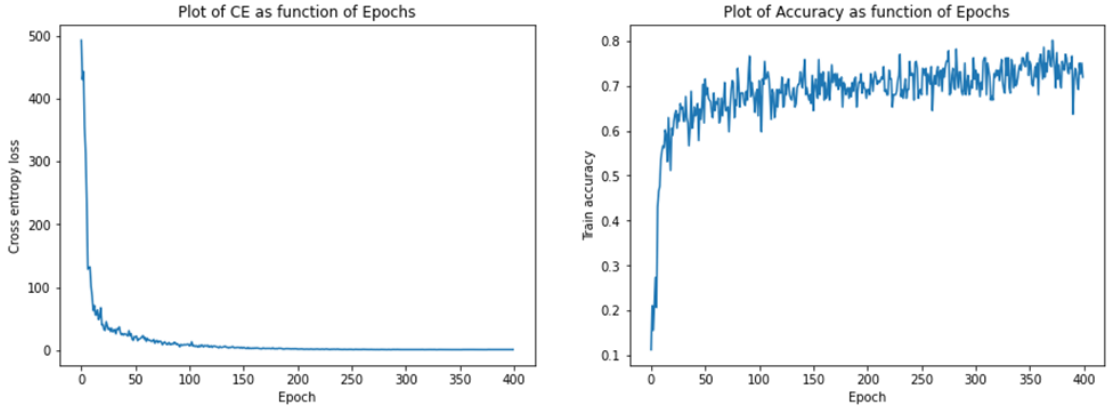


FIGURE 16 - MLP Implementation 1 with 2 Hidden Layers + RELU Activation for Unnormalized Dataset for Various Epoch Values

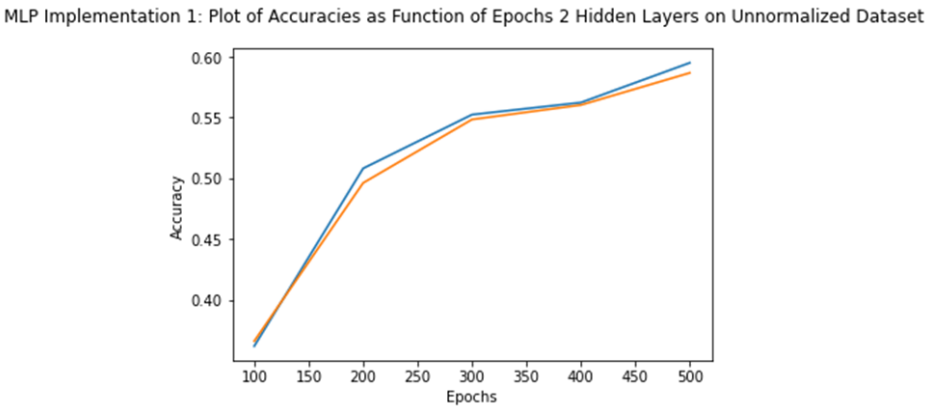
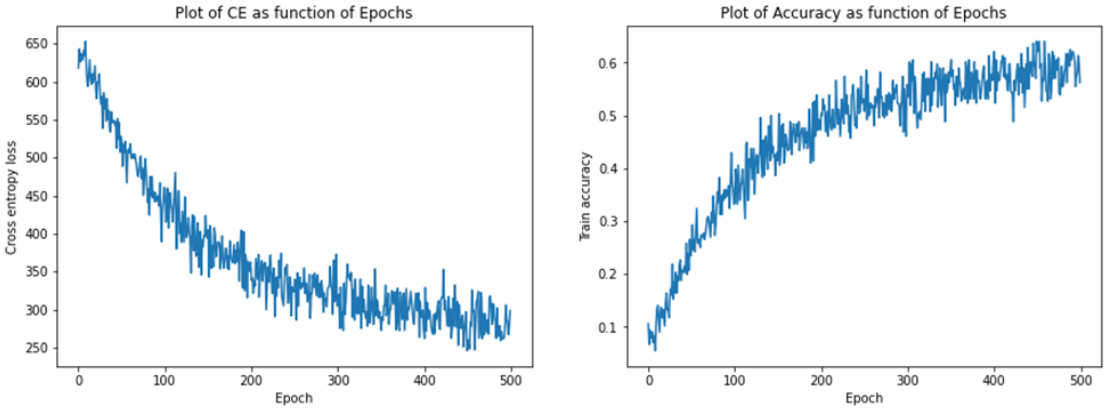


FIGURE 17 - MLP Implementation 1 with 2 Hidden Layers + RELU Activation for Unnormalized Dataset for 500 Epochs



References

[1] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST,” GitHub, 28-Aug-2017. [Online]. Available: https://github.com/zalandoresearch/fashion-mnist. [Accessed: 10-Nov-2022].

[2] Kadam, Shivam S., Amol C. Adamuthe, and Ashwini B. Patil. "CNN model for image classification on MNIST and fashion-MNIST dataset." *Journal of scientific research* 64.2 (2020): 374-384.

[3] Greeshma, K. V., and K. Sreekumar. "Hyperparameter optimization and regularization on fashion-MNIST classification." *International Journal of Recent Technology and Engineering (IJRTE)* 8.2 (2019): 3713-3719.

[4] LEITHARDT, VALDERI. "Classifying garments from fashion-MNIST dataset through CNNs." *Advances in Science, Technology and Engineering Systems Journal* 6.1 (2021): 989-994.

[5] https://colab.research.google.com/github/yueliyl/comp551-notebooks/blob/master/NumpyDeepMLP.ipynb

[6] Harrison Kinsley & Daniel Kukieła Neural Networks from Scratch (NNFS) https://nnfs.io

[7] L. Jiang. “A Visual Explanation of Gradient Descent Methods (Momentum, AdaGrad, RMSProp, Adam)” *towardsdatascience.* 7 June, 2020. [Online]. Available: https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c#:~:text=AdaGrad%20uses%20the%20second%20moment,is%20generally%20the%20best%20choice.

[8] M. Madiraju. “Deep Learning - Convolutional Neural Network.” *linkedin.* 24 Sept. 2020. [Online]. Available: https://www.linkedin.com/pulse/deep-learning-convolutional-neural-network-mounic-madiraju.

[9] D. Giordano. “7 tips to choose the best optimizer”. *towardsdatascience.* 25 July, 2020. [Online]. Available: https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e.

[10] V. Bushaev. “Adam — latest trends in deep learning optimization.” *towardsdatascience.* 22 Oct., 2018. [Online]. Available: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c.