COMP551 Assignment 2

Alif Naufal Farrashady (261121584) Calvin Chan (261121702)
Katherine Meyers (260987652)

December 6, 2022

Abstract
In this paper, we implemented a neural network model from scratch and experimented with various parameters through adjusting the gradient descent methods, L2 regularization rates and the architecture of the model layers for the Fashion-MNIST dataset from the Keras library. Since the dataset contains a multi-class dependent variable with 10 categories, the model output activation in the last layer was defaulted to softmax.

The best accuracy was attained when training a model consisting of 1 hidden layer with 256 hidden units using the ReLU activation function, a learning rate of 0.005 and L2 regularization. After training this model over 50 epochs, we obtained an accuracy of 82.44%. When comparing our implementation against Keras' convolutional neural network model performance with similar parameters, we achieved a test set accuracy of 90.07%. When training our model, we found that the factor that made the biggest difference in the resulting accuracy was the number of hidden layers used. We also noted that normalization and L2 regularization both allowed us to improve the accuracy of our models, though only by a few percent each.

Introduction
In this paper, we explore the use of neural networks as a classification technique. We developed a custom implementation of a multi-layer perceptron, similar to that provided by existing libraries, such as Keras. We then conducted various experiments with our custom implementation on the Fashion-MNIST dataset, which is a multi-class classification problem with 10 categories. Interestingly, we noted that an MLP with only 1 hidden layer performed better than one with more hidden layers with this dataset. Additionally, we found that the ReLU activation function produced the highest accuracy, and that using normalized data and L2 regularization allowed us to increase the training accuracy. Overall, the best accuracy was attained when training an MLP with 1 hidden layer of 256 hidden units while using the ReLU activation function, a learning rate of 0.005 and L2 regularization. This model converged after 50 epochs, resulting in a testing accuracy of 82.44%. However, we were able to achieve an accuracy of 90.07% when using Keras' convolutional neural network model with similar parameters, indicating that ConvNets are better suited to image classification problems than MLPs. When exploring further, we noted that more hidden units per layer increases accuracy, but that the benefits taper off when using more than 100 hidden units. Finally, we observed that in order to successfully perform image classification, at least 1000 training points are needed, but ideally more than 10000 would be used.

Datasets
We loaded the dataset from the Keras library and vectorized the data using flatten to turn it into a 2D image of 28x28 size before standardizing it in an array. Standardizing the data helps with the model training as data which are on a smaller scale and are more uniform speeds up the training of the model. There was 60,000 train set data and 10,000 test set data. Each

point of data represents an image corresponding to the numbers 0 to 9 and are categorized with the number they represent accordingly.

Results
Our custom implementation of the multi-layer perceptron is able to use mini-batch training, whereby in each epoch, parameters are only updated after an iteration of each mini-batch. We will be able to experiment with different mini-batch sizes to see which can improve performance. Additionally, our implementation supports various activation functions in the intermediate layers, including the Rectified Linear Unit (ReLU), Leaky Rectified Linear Unit (Leaky ReLU), as well as the Hyperbolic Tangent (Tanh) function. We will experiment with various activation functions in our second experiment. For the output layers, only the Softmax activation function is used. The Softmax activation output layer is suited for our dataset as it is a multiclass classification problem, whereby it will be able to generate probabilities of a single data point belonging to each of the 10 classes that sum up to 1.

We have also included methods to verify gradients, as well as plotting the cross entropy loss training loss as a function of the number of epochs, in order to visualize how the model has progressed during the training phase. In order to support experiment 3, we have also included the ability to apply L2 regularization to the model, with a choice of lambda as a tunable hyperparameter.

For our first experiment, we built 3 models, an MLP with no hidden layer, an MLP with a single hidden layer of 128 units with the ReLU activation function, as well as an MLP with 2 hidden layers, each of 128 units and the ReLU activation functions. All 3 models have an output layer that utilizes the Softmax activation function. For ease of comparison, all 3 models will apply mini-batch training with a batch size of 128 and the models will run for 10 epochs.

The first model achieved a test accuracy of 0.75, while the second model had 0.81, while the third model achieved 0.76. Generally, it does seem that higher network depth improves performance, as seen by how the second and third model is better than the first. However, interestingly, the second model with only 1 hidden layer did better than the model with 2 hidden layers. Generally, models with more layers are able to learn at higher levels of abstraction, but for this dataset, it seems that a model with a single layer is sufficient and a second hidden layer does not improve performance.

For experiment 2, we found that the Leaky ReLU activation function performed similarly to the ReLU activation function, although slightly worse (72.83% vs 76.35%). However, our model failed to produce any meaningful predictions when trained with the Tanh activation function, potentially due to the vanishing gradient problem.

For experiment 3, after L2 regularization was added, it resulted in a decrease in trainset accuracy by 0.50% but increased test set accuracy by 1.56%.

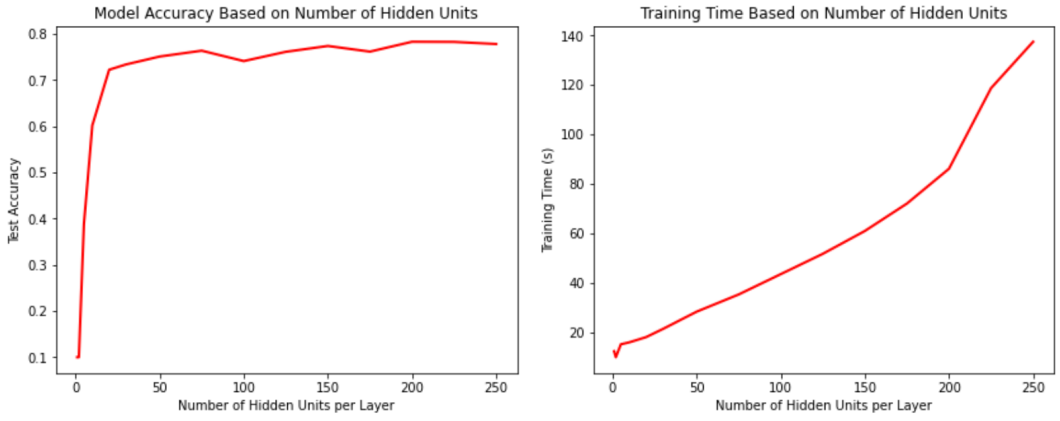
For experiment 4, when training with unnormalized images, the testing accuracy decreased slightly (76.4% vs 72.4%). This makes sense, since when handling images each pixel uses roughly the same scale (0-255). We also needed to use a smaller learning rate and train over more epochs to ensure that the model converges.

In experiment 5, the convolutional neural network model implemented using the Keras library had a much better train and test set accuracy - with the test set represented as val_accuracy in the figure below. The test set accuracy fluctuated around 90-91% and was stable after around 15 epochs as the decrease in loss was becoming minimal.

```
Epoch 1/20
1875/1875 [==============================] - 53s 26ms/step - loss: 0.6437 - accuracy: 0.8254 - val_loss: 0.4067 - val_accuracy: 0.8552
Epoch 2/20
1875/1875 [==============================] - 39s 21ms/step - loss: 0.3283 - accuracy: 0.8807 - val_loss: 0.3331 - val_accuracy: 0.8805
Epoch 3/20
1875/1875 [==============================] - 38s 20ms/step - loss: 0.2856 - accuracy: 0.8955 - val_loss: 0.3124 - val_accuracy: 0.8885
Epoch 4/20
1875/1875 [==============================] - 38s 20ms/step - loss: 0.2569 - accuracy: 0.9051 - val_loss: 0.2901 - val_accuracy: 0.8939
Epoch 5/20
1875/1875 [==============================] - 39s 21ms/step - loss: 0.2346 - accuracy: 0.9131 - val_loss: 0.2939 - val_accuracy: 0.8921
Epoch 6/20
1875/1875 [==============================] - 38s 20ms/step - loss: 0.2128 - accuracy: 0.9201 - val_loss: 0.2974 - val_accuracy: 0.8987
Epoch 7/20
1875/1875 [==============================] - 40s 22ms/step - loss: 0.1960 - accuracy: 0.9273 - val_loss: 0.3045 - val_accuracy: 0.9002
Epoch 8/20
1875/1875 [==============================] - 40s 21ms/step - loss: 0.1808 - accuracy: 0.9331 - val_loss: 0.3021 - val_accuracy: 0.9012
Epoch 9/20
1875/1875 [==============================] - 41s 22ms/step - loss: 0.1668 - accuracy: 0.9377 - val_loss: 0.2926 - val_accuracy: 0.9076
Epoch 10/20
1875/1875 [==============================] - 39s 21ms/step - loss: 0.1587 - accuracy: 0.9419 - val_loss: 0.3065 - val_accuracy: 0.9028
Epoch 11/20
1875/1875 [==============================] - 40s 21ms/step - loss: 0.1428 - accuracy: 0.9463 - val_loss: 0.3186 - val_accuracy: 0.9021
Epoch 12/20
1875/1875 [==============================] - 40s 21ms/step - loss: 0.1380 - accuracy: 0.9485 - val_loss: 0.3111 - val_accuracy: 0.9081
Epoch 13/20
1875/1875 [==============================] - 42s 22ms/step - loss: 0.1273 - accuracy: 0.9522 - val_loss: 0.3437 - val_accuracy: 0.8988
Epoch 14/20
1875/1875 [==============================] - 38s 20ms/step - loss: 0.1211 - accuracy: 0.9543 - val_loss: 0.3529 - val_accuracy: 0.9033
Epoch 15/20
1875/1875 [==============================] - 38s 20ms/step - loss: 0.1162 - accuracy: 0.9570 - val_loss: 0.3663 - val_accuracy: 0.9030
Epoch 16/20
1875/1875 [==============================] - 38s 20ms/step - loss: 0.1037 - accuracy: 0.9614 - val_loss: 0.3906 - val_accuracy: 0.9051
Epoch 17/20
1875/1875 [==============================] - 40s 21ms/step - loss: 0.1020 - accuracy: 0.9611 - val_loss: 0.4058 - val_accuracy: 0.9044
Epoch 18/20
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0989 - accuracy: 0.9644 - val_loss: 0.4025 - val_accuracy: 0.9050
Epoch 19/20
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0933 - accuracy: 0.9668 - val_loss: 0.4485 - val_accuracy: 0.9041
Epoch 20/20
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0895 - accuracy: 0.9678 - val_loss: 0.4371 - val_accuracy: 0.9007
<keras.callbacks.History at 0x7fbba91fad90>
```

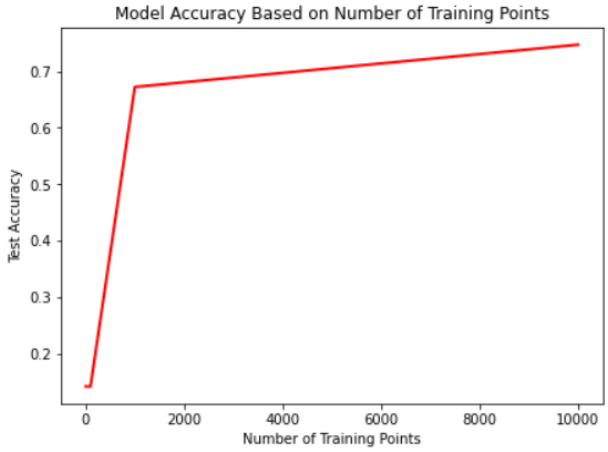
For experiment 6, we determined the optimal model to consist of 1 hidden layer with 256 hidden units, the ReLU activation function, a learning rate of 0.005 and L2 regularization. After training this model over 20 epochs, we obtained an accuracy of 82.44%. However, this model's accuracy is still almost 10% less than that of the ConvNet.

To explore further, we looked at the correlation between the number of hidden units per layer and the model's accuracy and training time. As can be seen in the plots below, as long as there are at least around 25 hidden units, the model will achieve a reasonable accuracy. Beyond this point, the increase in accuracy is very small. However, the training time consistently increases with the number of hidden units. Therefore, if training time is not a concern, roughly 200 hidden units per layer would be ideal, but if training time is a concern, you can achieve a similar accuracy with roughly 75 hidden units per layer.



We also looked at the accuracy as a function of the number of training points used, which revealed that at least 1000 training points are necessary to produce meaningful predictions on the fashion-MNIST dataset, but upwards of 10000 training points is ideal. For a more complex

image classification task, even more training points would be needed. This emphasizes the importance of abundant sample data in machine learning.



Discussion and Conclusion

To conclude, we were able to achieve significant success when performing image classification on the fashion-MNIST dataset with our MLP implementation. In general, we found that more hidden layers is better, but for this particular problem we had the best results with only one hidden layer. We also noted that the ReLU activation function, normalization and L2-regularization helped to increase the testing accuracy of our model. However, even after carefully tuning our MLP hyperparameters, we were only able to achieve an accuracy of 82.44%, as compared to the 90.07% that we achieved using Keras' ConvNet implementation. This indicates that while the MLP is capable of generating reasonable predictions, a ConvNet is significantly better equipped for image classification tasks. Further experimentation could be done on the effects of the various ConvNet hyperparameters, in order to determine a model that is truly optimal for solving the fashion-MNIST image classification problem.

Statement of Contributions

Alif implemented the neural network model along with the different gradient descent methods, and Katherine processed the MNIST data. All members helped with the experimentation and all members contributed to the report.

References

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.