# Comp551 Assignment3

Jaylene Zhang, Keyu Yao, Yusen Tang

December 6, 2022

## 1 Abstract

Neural network is a powerful model to for image recognition. In this assignment, we implemented a multi-layer perceptron which was used to perform the multi-class classification on the Fashion-MNIST data. To improve the efficiency of computation and save ram, we used the mini-batch gradient descent method to compute the optimized weights, which were used to predict class. Also, in later experiments the L2 regularization was added to minimize the impact of large weights. In each layer we had a forward pass taking charge of predicting the class probability from the input features as well as a back function used later for the back-propagation method to update weights in each epoch.

Also, in order to explore the consequences of important decisions made while training neural networks, we also compared the original model with several variations. In our experiments, we modified the number of hidden layers to be 0, 1 and 2 respectively, and replaced the activation functions with tanh and LeakyReLU. Moreover, the model was trained with unnormalized data. Finally, we compared the our implementation with other models like CNN imported from Tensorflow.

## 2 Introduction

The goal of this project is to implement a MLP from scratch, and train and test it on the Fashion-MNIST Dataset. In our experiments, we trained our model with a variety of different activation functions, learning rates, number of hidden layers and so on. At the time of this assignment, it is believed that Convolutional Neural Network (CNN) gives the most rigorous effects in solving real-world problems. In a paper published by Kayed et al., they build a CNN based LeNet-5 architecture which is proposed to train parameters of the CNN on Fashion-MNIST dataset. Experimental results show that LeNet-5 model achieved accuracy over 0.98. Therefore, it outperforms both the classical CNN model and the other existing models.

### 2.1 Related Work

We found that there exists a variety of neural networks which can be applied to all kinds of data. For example, we found a paper by Boughara et al., which discussed the use of constructive training algorithm for a MLP that was developed by a single hidden-layer using a given number of neurons and a small number of training patterns applying to facial expression recognition applications from three large datasets GEMEP FERA 2011, the Cohn-Kanade facial expression and the facial expression recognition FER-2013 databases.

#### 2.1.1 Citation

M. Kayed, A. Anter and H. Mohamed," Classification of Garments from Fashion MNIST Dataset Using CNN LeNet-5 Architecture," 2020 International Conference on Innovative Trends in Communication and Computer Engineering (ITCE), 2020, pp. 238-243, doi: 10.1109/ITCE48509.2020.9047776.

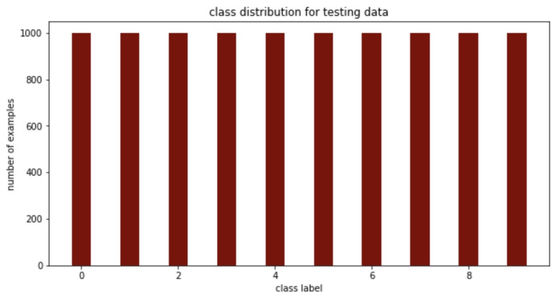
Hayet Boughrara et al. "Facial expression recognition based on a mlp neural network using construc- tive training algorithm." In: Multimedia Tools and Applications 75.2 (2016), pp. 709–731.

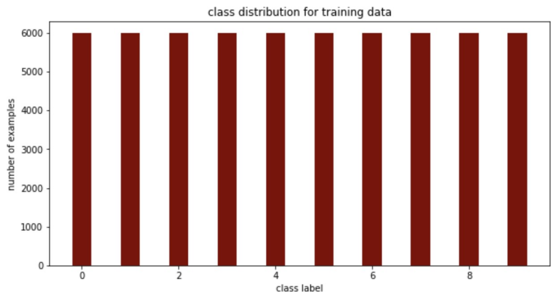
## 3 Datasets

Fashion-MNIST is a dataset consisting of a training set of 60,000 images and a test set of 10,000 images. The dataset was made to fit into a 28 by 28 pixel bounding box and anti-aliased gray-scale, this kind of image data is commonly used in training machine learning algorithms. The dataset is modified based on the original MNIST dataset, which contains a lot of handwritten digits and is always used as a benchmark to validate ML algorithms. However, the original data set is overused and too simple, so we need a more complex data set to train and test the model. We have labels from 10 classes that are'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', and 'Ankle boot'.

We loaded the training and testing samples from Tensorflow where each row of the input features denotes a data point of shape 28*28, and the labels are just the corresponding class label.

We found that the both the training and the testing sample are uniformly distributed with respect to the class, as shown by the bar plot below from which we observed that there are 6000 data points in each class for the training while there are 1000 for each in the testing sample. We believed that the equally allocated data ensured the randomness in later permutation step, improving the validity of our model.





In order to better adapt to our algorithm, the data was flattened and then normalized by using the formula below:

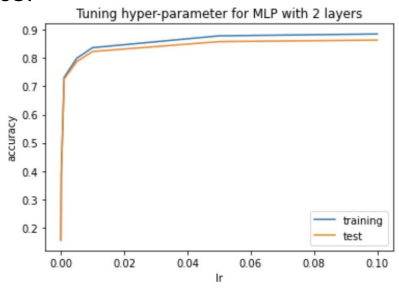
$$data = \frac{(data - mean(data, axis=0))}{255}$$

Therefore, the size of our total input features became 60000*784 and 10000*784 respectively.

## 4 Results

### 4.1 Tuning Hyper-parameter

Before comparing the accuracy, we first tuned the hyper-parameter by running experiments for learning rate = [1e-05, 0.0001, 0.001, 0.005, 0.01, 0.05, 0.1], for which the corresponding test accuracy was [0.1574, 0.3866, 0.7255, 0.7874, 0.823, 0.8579, 0.8632]. We plotted the training and test accuracy varying with different lr values and thus obtained the optimum lr = 0.1, for which the accuracy hit 0.863.



### 4.2 Varying MLP Depth

We created three different models which are (1) an MLP with no hidden layer, (2) an MLP with a single hidden layer having 128 units and ReLU activation, (3) an MLP with 2 hidden layers each having 128 units with ReLU activation. And through experiments using batch size = [32, 64, 128, 256], we found that using batch size of 256 gave the best result, because a greater proportion of samples were involved in the training step. However, we avoided using a very large batch size in order to improve the computation efficiency of this assignment. We also performed checking gradient using small perturbation and obtained a very small value for the softmax layer being 1e-14.

By monitoring the accuracy against the number of iterations, we concluded that the training and test accuracy started to increase very slowly after 1000 epochs for all of the three experiments, so we ran 1500 epochs for the 3 models. The training and test accuracy together with the variation of loss are shown in the following plots. For simplicity, we only plot the accuracy every 100 epochs, which means that 1500 epochs lead to 15 different results.

Figure 1: accuracy and loss for 2-layer MLP

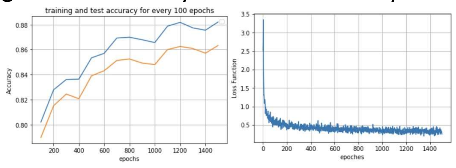


Figure 2: accuracy and loss for 1-layer MLP

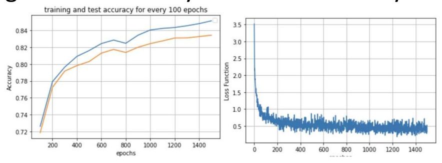
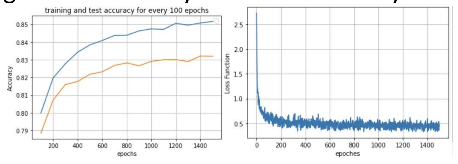
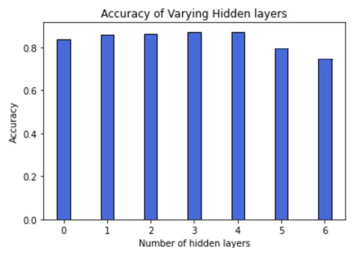


Figure 3: accuracy and loss for 0-layer MLP



From the plots we can conclude that 2-layer MLP gave the best result where the accuracy of 0-layer and 1 layer were almost the same. Overall, the three experiments all achieve quite good accuracy. Therefore, we see that more hidden layers fostered more accurate result. The reason for this could be that using only softmax failed to capture the complexity of the data. Hence, when hidden layers and non-linearities applied, the complex feature is resolved, contributing to a better model.

### 4.2.1 More on MLP Depth



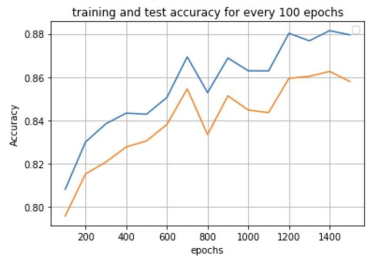
Although we already used the number of layers equal to 0, 1, and 2 to investigate the effect of depth and obtained the expected result, we were curious about the over-fitting problem with too many hidden layers. Therefore, we added the depth up to 6 and the following plot illustrates the result. Again, we used epoch of 1500 and all the layers are Relu layers.

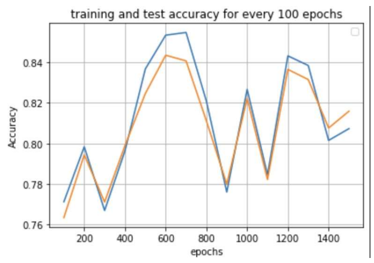
As we can see actually the model with 3-relu layers gave the best performance with the accuracy reaching 0.876. And the result was increasing slowly from 0-layer to 3 layers. However, the accuracy started to drop after 4 layers, and the worst model with 5 layers only gave around 0.7 accuracies, which proved that our assumption was correct.

## 4.3 Changing Activation functions

Using the same model with 2 layers, we ran experiments with 2 other types of activation functions: Leaky-relu and Hyperbolic tangent. We did not perform hyper-parameter tuning and used the same learning rate = 0.1 as above. It is worth mentioning that for Leaky-ReLU, initially we tried alpha = 0.5, but obtained extremely low accuracy = 0.1 and rising loss with epochs. We suspected this may be caused by too many negative values added when computing the minimum of (a, 0), leading to a vanishing gradient problem since the exponential of negative values approach 0. We then changed alpha to 0.3 which is the default in Tensorflow.

The two figures below plot the accuracy of the Leaky-relu and hyperbolic tangent respectively, relative to epochs. Our results showed that the tanh model performed the best, achieving a test accuracy of 0.867 whereas the Leaky-ReLU model gave a rather low-test accuracy of 0.817. Moreover, it can be observed that there were dramatic fluctuations in the training and test accuracy of the Leaky-relu model, suggesting that this model could be very unstable due to many negative values. In the contrast, the hyperbolic tangent function mapped all the input to the range [-1, 1], so there was less variation in the plot.



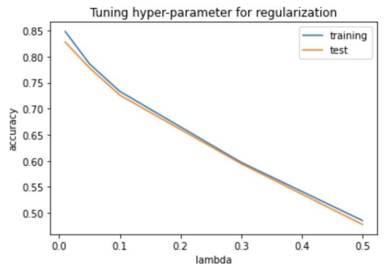


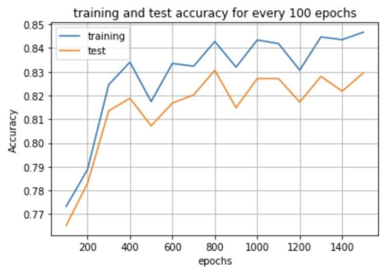
## 4.4 Adding L2 Regularization

In this experiment, we added L2 regularization to the cost of the MLP with 2 hidden layers using ReLU activations.

Before comparing the accuracy, we first tuned the hyper-parameter by running experiments for lambda = [0.5, 0.3, 0.1, 0.05, 0.01], for which the corresponding test accuracy was [0.478, 0.5946, 0.726, 0.7783, 0.8378]. We hence used lambda of 0.001 in this experiment.

We plotted the training and test accuracy varying with different lambda values and thus obtained the optimum lambda = 0.01. Using this lambda we achieved 0.834 accuracy, which was surprisingly lower than the model with no regularization added. One reason might be that regularization with the hyper-parameters and activation functions we have might be leading to over-fitting.

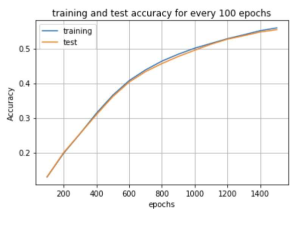




## 4.5 Training with Unnormalized images

This model used 2 hidden layers with 128 units and ReLU activations, but was trained using unnormalized images. Therefore, the range of each value in the unit vector is between 0 and 255.

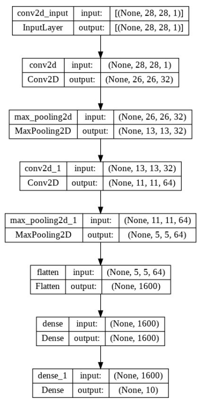
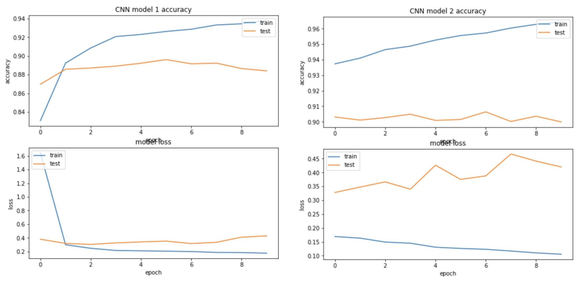
Initially, we used the optimum learning rate of 0.1 from the previous tuning, but nan values started to occur from the second epoch. We changed it to 1e-6 to address this overflow issue but got very low accuracy of 0.554. This is exactly as we expected since we now have values up to 255 in the input vector, resulting in the weight sum of the layer being far from 0 and taking away the non-linear impact of the activation function.



## 4.6 Training with ConvNet

We constructed our convolutional neural network this way with 2 convolutional and 2 fully connected layers with ReLu activation function as shown. We chose our filter size to be three-by-three and the number of filters to be 8.

In the first experiment, we achieved a higher accuracy than the normal MLP as shown in the plot of ”cnn model 1 accuracy”. In the second experiment, we added zero-padding and dropout techniques to our model, the achieved accuracy is shown in the plot” cnn model 2 accuracy”. Clearly, we can see that model 2 achieves better accuracy than model 1.


conv2d_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 26, 26, 32)
max_pooling2d | input: | (None, 26, 26, 32)
MaxPooling2D | output: | (None, 13, 13, 32)
conv2d_1 | input: | (None, 13, 13, 32)
Conv2D | output: | (None, 11, 11, 64)
max_pooling2d_1 | input: | (None, 11, 11, 64)
MaxPooling2D | output: | (None, 5, 5, 64)
flatten | input: | (None, 5, 5, 64)
Flatten | output: | (None, 1600)
dense | input: | (None, 1600)
Dense | output: | (None, 1600)
dense_1 | input: | (None, 1600)
Dense | output: | (None, 10)


Furthermore, we tested on CNN with different filter sizes and the result is shown in the table.

| num of filter | 1 | 4 | 8 | 12 |
|---|---|---|---|---|
| accuracy | 0.9335 | 0.9426 | 0.9493 | 0.9423 |

We can see that there is a small improvement as we increased the number of filters in accuracy. And the best number of filters is 8 which is the parameter we chose to construct our model.

## 4.7 Comparing best MLP with CNN

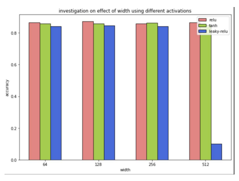
After running all the experiments above, we found that the MLP with the highest accuracy is 0.876, with 3 hidden layers and a learning rate of 0.1. While for 2-convolutional-layer and 2-fully-connected CNN model, the highest accuracy is 0.964 with 8 filters plus zero-padding and dropout regularization. We can see that CNN performs much better than the normal neural network.

## 4.8 Investigation on Effect of Width

We conducted experiments on 64, 128, 256 and 512 hidden units using activation function as ReLU, tanh and Leaky-ReLU respectively. We plotted the results together as following for better comparison.

The following plot and data show the results of our various experiments.

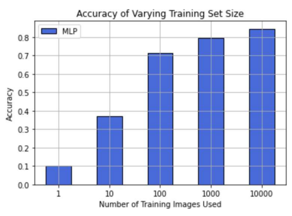
We achieve the best test accuracy of 0.872 using 2 hidden layers with 128 units each and a ReLU activation function while the worst occurred when using 512 units with Leaky-relu, resulting in a 0.1 accuracy. With the models using Tanh, the most accurate result is 0.862 using a hidden layer width of 256 units. Hence, we achieved best result using either 128 or 256 units, and very serious over-fitting occurred when using 512 with Leaky-relu.



## 4.9 Investigation on Effect of Training Size

By raining the MLP and ConvNet with $10^k$, where k is in 0, 1, 2, 3, 4 images, the following is the plot for the test accuracy for MLP. The test accuracies with 1, 10 and 100 training images are all extremely low below 0.7. This is expected as the model lacks data to make the correct decisions. As the number of samples in the training data increases, the accuracy approached the best result in our previous experiments. When 10 000 training images are used, a testing accuracy of 0.844 was achieved.

## 5 Discussion and Conclusion



As our results above, choosing the number of hidden layers and hidden units are the key factor for the success of the MLP model. If we use a large number of hidden units or hidden layers, over-fitting may occur. The most accurate MLP has 2 hidden layers of 128 units each and use ReLU activations without L2 regularization. The neural network has a learning rate of 0.1 and uses 1500 training epochs. Comparing the accuracies of 0, 1 or 2 hidden layers of an MLP with ReLU activation, 2 hidden layers are much more accurate. However, in this case, using a tanh activation is more or less the same as using ReLU. Furthermore, for the dataset, adding L2 regularization causes a decrease in accuracy. We think this dataset does not the regularization to penalize large weights. At last, CNN generally performs much better than MLP, with padding and regularization, we achieved better accuracy.

As we did our research for related work for neural networks, we noticed that a great amount of related work chooses to train the model with all batches per epoch, as compared to our implementation of only one random mini-batch per epoch. We think that this approach would achieve higher accuracy in a smaller number of epochs. We want to do some further experiments on this approach to see if our hypothesis is correct.

## 6 Statement of Contributions

Yusen Tang: Data Loading and Preprocessing and Running Experiments
Keyu Yao: CNN implementation, Hyperparameter Tuning and Write-up
Jaylene Zhang: MLP Implementation, Hyperparameter Tuning and Report Write-up