MiniProject 3: Classification of Image Data

COMP 551, Winter 2022, McGill University

Manas Kale
261000867

Gauri Sharma
261026894

Avinash Bhat
260969537

Abstract

This mini-project is aimed at implementing a multilayer perceptron (MLP) from scratch to classify image data. The Fashion-MNIST dataset was used for this task. This dataset comprises of article images along with their labels and is released by Zalando research (Xiao et al., 2017). We developed multiple Multilayer Perceptron (MLP) models with Numpy with different number of hidden layers and hyperparameters and compared it with another model that we developed using Keras. We experimented with various activation functions and optimized the model using mini-batch gradient descent. Finally we tuned on our model with different learning rates, activation functions, momentum and dropout to and identified the best architecure for our MLP model. We were able to achieve a testing accuracy of **83.34%** with out best model, whereas CNN gave us an accuracy of **89%**.

1 Introduction

In this mini project, we implemented a multilayer perceptron model (MLP) (Ramchoun et al., 2016) from scratch using the Numpy library. We initially tried developing a generic framework that allows configuring any number of hidden layers in the network but due to multiple errors in the gradient calculations, we settled on deriving the gradient formulae on paper and translating it to code for 0,1 and 2 hidden layers. We implemented the ReLU, Leaky ReLU and tanh activation functions and used softmax for the final layer with 10 units since this is a multiclass classification problem with 10 classes. Softmax function assigns probabilities to each class whose sum is 1 and is straightforward to identify the class with maximum probability. We implemented the mini batch gradient descent algorithm for optimizing our model. We followed an iterative approach where we calculate the gradients for each data point in a mini batch and then average all the gradients that are calculated, and multiply the average with the learning rate to calculate the weight updates. For regularization, we implemented dropout with a customizable dropout probability as a hyperparameter. When training our network, we observed oscillations and drop in accuracy after an initial increase. We hypothesized this might be so, therefore, we also implemented momentum for stochastic gradient descent (Qian, 1999) for faster convergence.

2 Dataset

2.1 Fashion MNIST Dataset

Fashion-MNIST is a dataset of Zalando's article images (Xiao et al., 2017). It consists of a training set of 60,000 samples and a test set of 10,000 samples. Each sample is a 28x28 grayscale image. Each of them is associated with a label from 10 classes. Images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. The distribution is shown in **Figure 1**. We observed that both train and test datasets were very well balanced, and so the training set would give us a good approximation of the generalization error on the test set.

2.1.1 Data Processing

As we know, in this Fashion-MNIST dataset each pixel value has a range from 0 to 255 and is stored as a 2D array. To prevent our model from learning any unecessary scaling in brightness patterns, we scale this data to have a mean of 0 and also divide by the standard deviation (i.e., z-score standardization). Furthermore, we reshape it as a 1D array so that it can be fed to our MLP. This does destroy any spatial information contained in the data, which we hypothesize will lead to less than perfect accuracy.

2.1.2 Training, validation and testing sets

As indicated in the assignment, we only split this dataset into training and testing sets and do not have a validation set. We use the train/test split provided by the dataset's authors, giving us 60,000 samples to train and 10,000 samples to test with. We do not perform cross-validation and only use accuracy as our metric.

3 Results

We trained all the models for **50** epochs because we observed the accuracies flattened out around this number. We also used a learning rate of **0.0001** for these experiments which we settled on after some trial and error. For all our models, we used the absolute difference as the loss function.

Task 3.1

We developed three different models (i) MLP with zero hidden layers (ii) MLP with one hidden layer and 128 units (iii) MLP with two hidden layers and 128 units per layer. The second and the third models used ReLU activations and all of them had a softmax layer at the end. We did not use any momentum or dropout for this task.

(Figure 2a, 2b) Here we observe that the 1 hidden layer network (orange) performs the best - it reaches optimum training accuracy the fastest and stays there. For the 0 layer network (blue), we observe it reaches maximum accuracy slowly and that accuracy is much less than that of the 1 layer network. This is probably because a 0 layer network cannot learn complex relations in the data, and so will have a maximum accuracy ceiling compared to a more complex network. To our surprise the 2 layer network drops in testing accuracy after around 15 epochs. We believe this is because of the fact that it greatly starts overfitting the data since there is no dropout being used here. It could also be possible that the network overshoots a valley in the loss landscape, so we tried reducing the learning rate further but found that it makes the accuracy stagnate for too long and is not practical. We were also taught in class that a simpler model (1 layer) is always better than a complex one (2 layer), so we decided to proceed with the next tasks.

On the test set, we obtained **62.82%** accuracy for the 0-layer network, **79.5%** for the 1-layer network and **63.89%** for the 2-layer network. Note that we stopped training the 2-layer network before its accuracy metric started dropping.

Task 3.2

For this task we replaced the activation function on the 2 hidden layer network with *tanh* and *leaky-ReLU* functions. We trained both these networks for 50 epochs using the same loss function as before, without any momentum or dropout.

(Figure 3a, 3b) As we can clearly see, the *tanh* activation function performed far better than the other two and we obtained a testing accuracy of **76.79%** with this. This could be because tanh prevents the network from overfitting on the data beacuse it is symmetric around 0 whereas the other two aren't. ReLu and its variants generally turn negative gradients to 0 quickly, but this does not happen with tanh. As with most empirical results in machine learning tasks, we can only hypothesize why this is the case.

Task 3.3

For this task we added dropout regularization to the 2 hidden layer network with ReLu activation function. We used a dropout parameter with probability 0.5 since it is considered as the optimal value (Srivastava et al., 2014). We also took care to make sure dropout calculations were performed only when necessary, as this involved extra mask matrix multiplication and increased our training time.

(Figure 4a, 4b) We see adding dropout does not really improve our results. We tried changing this parameter for task 6 later on, but did not see any significant improvements. Our best testing accuracy for this task was **50.75%.**

Task 3.4

For this task our hypothesis was that the network will overfit and learn unnecessary patterns in the data. For example, some images of cloths might have been taken with a brighter or darker light. If they are not normalized, the network would learn that some cloths are associated with bright pixels. We observe that the accuracy indeed drops significantly within 8 epoches. This is probably because the network is overfitting to noise and so can't make predictions correctly.

(Figure 5a, 5b) As hypothesized, we obtained a low testing accuracy of **24.57%** for this unnormalized data.

Task 3.5

We hypothesized that a CNN would be the best type of architecture for this image classification task. Since we flatten the data in our MLP we lose any form of spatial information, something that is important for learning from images.

(Figure 6a & 6b) As hypothesized, we obtained a testing accuracy of **89%** with a CNN which was better than our best accuracy using MLP.

Task 3.6

To find the best network for this task, we first listed everything we learnt from the previous tasks:

- With task 3.2, we were able to measure the effect of tanh and Leaky-ReLU activation functions using the model with two hidden layers. We observed that tanh gave the best accuracy when compared with the others so we decided to use this activation function in our model.
- Since our loss was oscillating back and forth, we decided to implement a momentum term in our learning algorithm. Specifically, we tried different values for momentum from 0.5 to 0.9 and found 0.7 works the best for this dataset and loss function.
- We did not use any dropout since task 3.3 demonstrated it did not improve accuracy by a significant amount. We believe dropout may provide a benefit only when the model becomes much more complex than the 2-layer MLP it currently is.
- We used normalized images because as demonstrated by task 3.4, using unnormalized images leads to a decrease in testing accuracy.
- We considered both the 1 and 2 hidden layer networks for this task. We first thought the more complex network (2 layers) would perform better, but as demonstrated in task 3.1 we observed the 1 layer network may also achieve the same performance.

Taking all of the above factors in account, we came up with two potential network architectures for getting the best accuracy:

Parameter	**Architecture 1**	**Architecture 2**
Hidden layers	1	2
Units per layer	128	128
Learning rate	0.0001	0.0001
Dropout	No	No
Activation function	tanh	tanh
Data normalization	Yes	Yes
Momentum	0.7	0.7

Our hope was that the 1 hidden-layer network would perform better than the 2 layer network because a simpler network that gives the same results is always better than a more complex one. We also hypothesized that this will never be able to beat a CNN.

We see both the architectures performed very similarly during training. Architecture 1 achieved a training accuracy of **84.57%** (Figure 7a) and testing accuracy of **80.97%** whereas architecture 2 achieved a training accuracy of **86.63%** (Figure 7b) and testing accuracy of **83.34%**. Since there is only a small difference between their testing accuracies, we believe **Architecture 1** is the best architecture possible for this task as it's simpler.

Task 3.7

Please refer to the attached colab notebooks for code and plots. MLP implementation is included in the **mlp.ipynb**, whereas CNN and exploratory data analysis is included in **cnn.ipynb**.

4 Discussion and Conclusion

From our experiments, we see that multilayer perceptrons perform exceptionally well for the task of image recognition even while being one of the basic neural network architectures. We observe that deeper and complex models overfit fast (Task 3.1). CNN performed better than MLP by giving an accuracy of 89% in comparison to MLP's 83.34%. Dropout regularization does not provide any advantage (Task 3.3), in fact it only reduces the accuracy, which indicates to us that better hyperparameter tuning can help and the problem might not lie with overfitting. This hypothesis turned out right because we observe a better training accuracy of $\sim$**86%** for a two hidden layer MLP (Task 3.6).

The Fashion-MNIST dataset is a well cleaned dataset and is suitable for exploring neural network learning. However, in practise such data is hard to come by. For our future work, we plan to add noise signals to the training images for better noise robustness. To be specific, we might be able to explore concepts such as Gaussian smoothing that reduces the detailing in an image just enough so that our model is generalizable. However we believe that we are already close to the accuracy limit for a non-CNN network. Further improvements can only be made if the network incorporates a way of recognizing spatial information, which is only possible through a CNN.

We gained a newfound appreciation of libraries like PyTorch and Keras that abstract the matrix multiplication functionalities and use auto-differentiation to make it easier for us to develop complex architectures.

5 Statement of Contributions

Avinash: Worked on the report and contributed to the code.
Gauri: Data analysis, implemented CNN and worked on the report.
Manas: Implemented MLP, designed experiments and worked on the report.

References

Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151.

Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. 2016. Multilayer perceptron: Architecture optimization and training.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Appendix

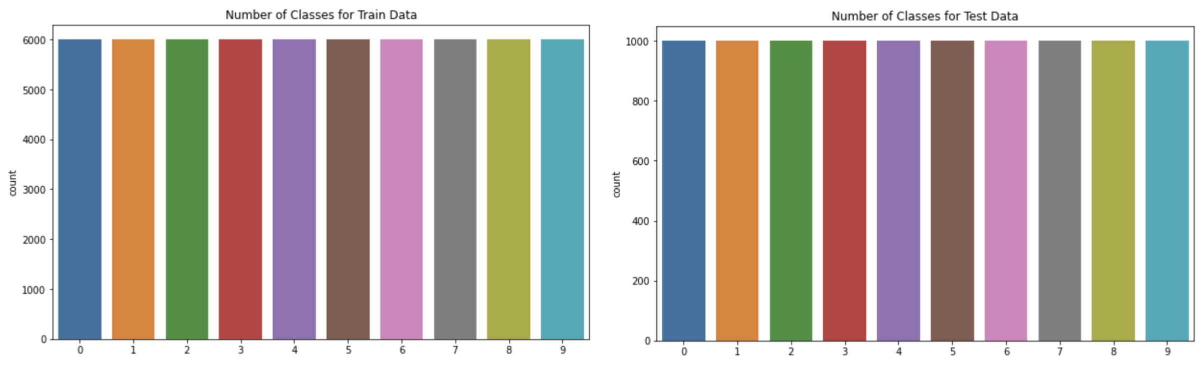
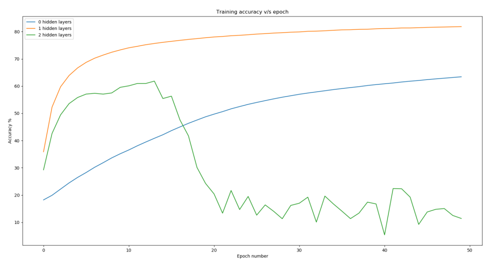
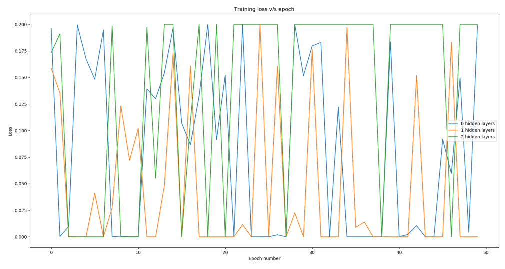


Figure 1: Class Distribution for Train and Test Dataset

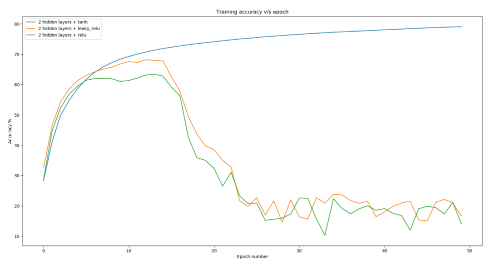


(a) Training accuracy v/s epoches for [0,1,2] hidden layers

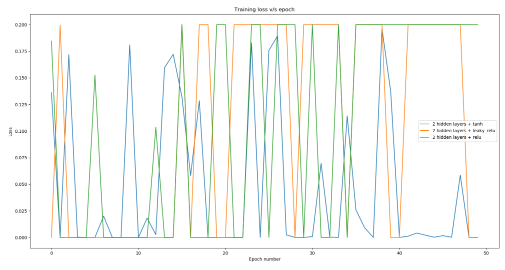


(b) Training loss v/s epoches for [0,1,2] hidden layers

Figure 2: Task 3.1

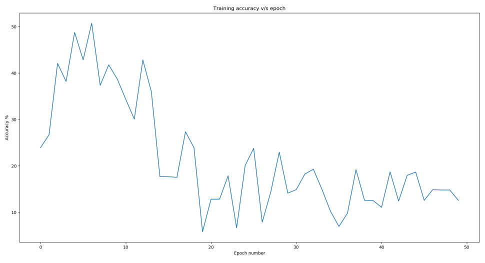


(a) Training accuracy v/s epoch for tahnh, leaky-ReLU, ReLu

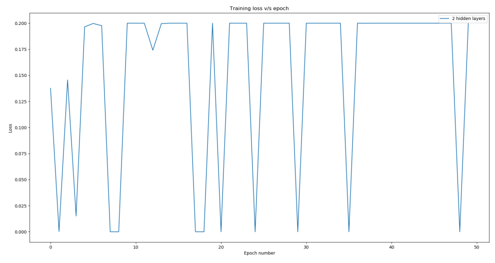


(b) Training loss v/s epoch for tahnh, leaky-ReLU and ReLu

Figure 3: Task 3.2

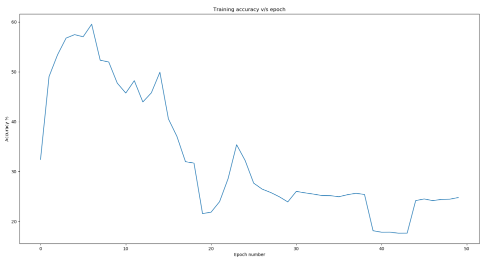


(a) Training accuracy v/s epoch for dropout

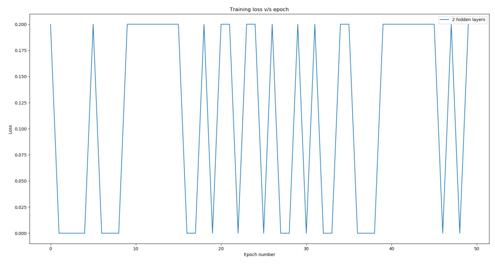


(b) Training loss v/s epoch for dropout

Figure 4: Task 3.3

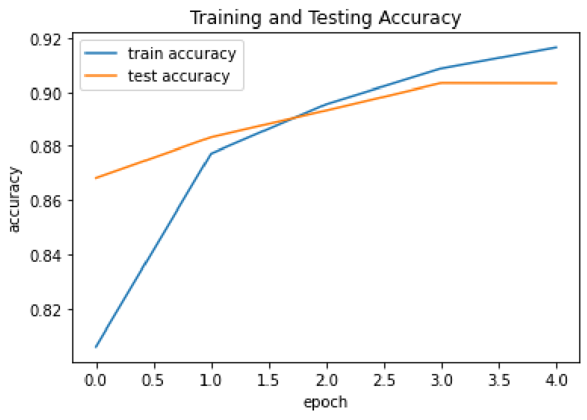


(a) Training accuracy v/s epochs for unnormalized data

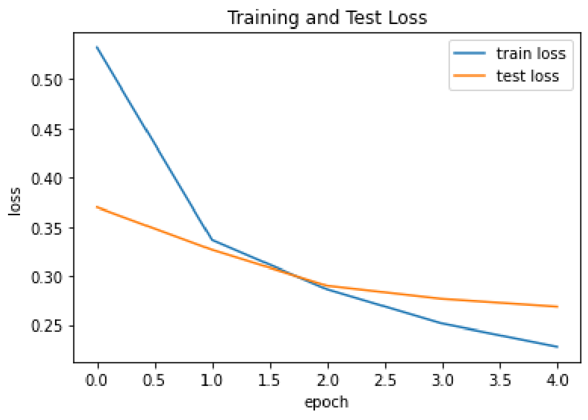


(b) Loss v/s epochs for unnormalized data

Figure 5: Task 3.4

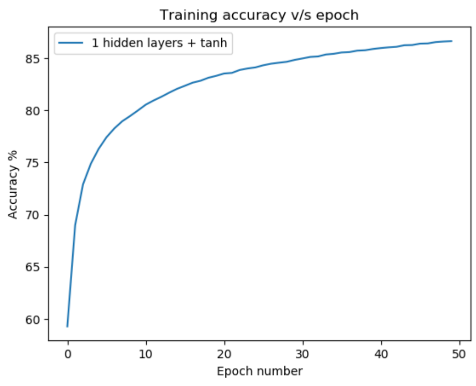


(a) Accuracy v/s Epochs for CNN

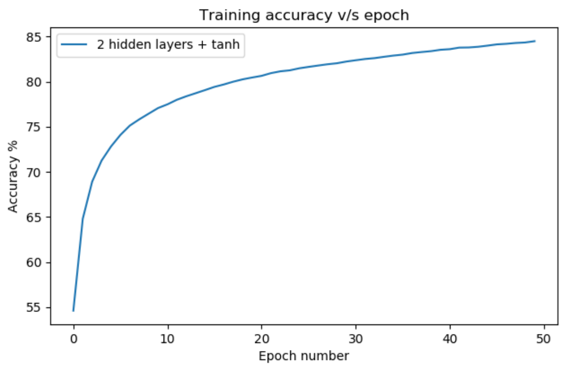


(b) Loss v/s Epochs for CNN

Figure 6: Task 3.5



(a) Training accuracy v/s epochs for Architecture 1



(b) Training accuracy v/s epochs for Architecture 2

Figure 7: Task 3.6