Fall COMP 551 Assignment 3

October 2022
Group 3: Steve Wen, Chloe Si, Navid Hassan Zadeh

Abstract

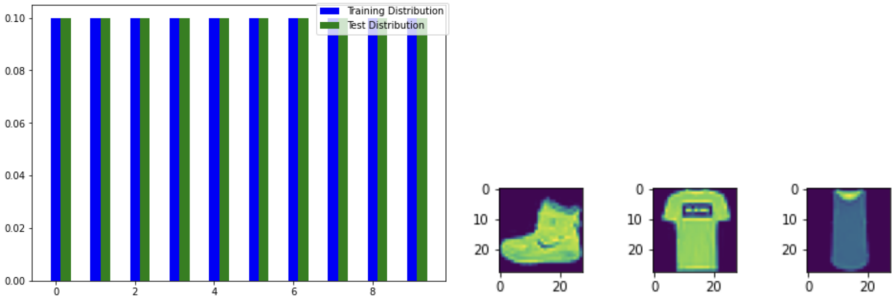
In this project we implemented a Multilayer Perceptron along with the Adam optimizer and optimized its hyperparameters and evaluated its performance on the fasion-MNIST dataset. We also explored the effects of data normalization and regularization on the model's performance. Our implmented MLP achieved 86.7% accuracy. However the optimized CNN outperformed our implementation of MLP with an accuracy of 90.3%. Finally, we investigated further into how changing the architecture of MLP by careful optimizations and increasing the number of neurons improves the performance. With these changes, we reached an accuracy of 89.6%.

Introduction

Multilayer perceptron is a popular model to apply on complex data and perform classification or regression tasks. One of the common uses of this method is image classification which has a variety of applications in modern day. In image classification it is often more effective to use multiple layers to better extract input features of data. In this project we implemented a multilayer perceptron from scratch and used the Fasion-MNIST database as benchmark to test our performance. The dataset is widely used in Computer Vision related tasks [Ko et al., 2022, Oyedotun et al., 2022]. We also investigated the performance of the model under different optimization parameters and applied the L2 regularization. We found that the model performs with higher accuracy when we adopted more layers in its structure. However, adding more layers resulted in a longer running time because of the hyperparameter optimization. Moreover, we investigated how we can modify the architecture to increase the performance of our MLP model. Also, we used the Keras library and created a convolutional network (CNN) to compare its performance to our implementation of MLP using the same database.

Dataset

We used the Fashion MNIST dataset[Xiao et al., 2017] in this project to evaluate the performance of our models. This dataset consists of seventy thousand 28×28 pixel images of 10 types of clothing items, such as dress, sandal, shirt, sneaker, ankle boot and others. These images are presented in the form of an array where each entry is a number ranging from 0 to 255 that determines the color of a pixel on a white to black spectrum. Based on our analysis in figure 1 (a), the ten categories are distributed evenly across the training and the testing dataset. Each category consists 10% of the whole dataset.



(a) Distribution of the 10 classes across training and testing data

(b) An example of three images in the dataset

Figure 1

Method

In this project, we developed a neural network with different number of hidden layers and activation functions. We implemented this model with 0, 1 and 2 hidden layers, where each hidden layer has 128 neurons. This was done to observe the changes in performance with each additional layer. Moreover, we used several activation functions such as ReLU, Tanh and Leaky-ReLU for determining the effect of activation functions. Also, different set hyperparameters are adopted for different tasks and architectures. For example, the scale of the unnormalized data is larger when compared to others, so we scale down the learning rate to make sure the training process could be continued. Additionally, we applied the L2 (Ridge) regularization on the model with 2 hidden layers and investigated the effects of changing the regularization coefficient on performance. Because the gradient calculation is error-prone, We also performed gradients perturbation tests on layers with parameters to ensure our gradients are correctly computed. Mini-batch gradient descent is deployed in which the size of each batch is 32. The network is initialized with random numbers sampled from 0 to 0.05. It effectively makes the gradients more stable and prevents it from exploding.

Results

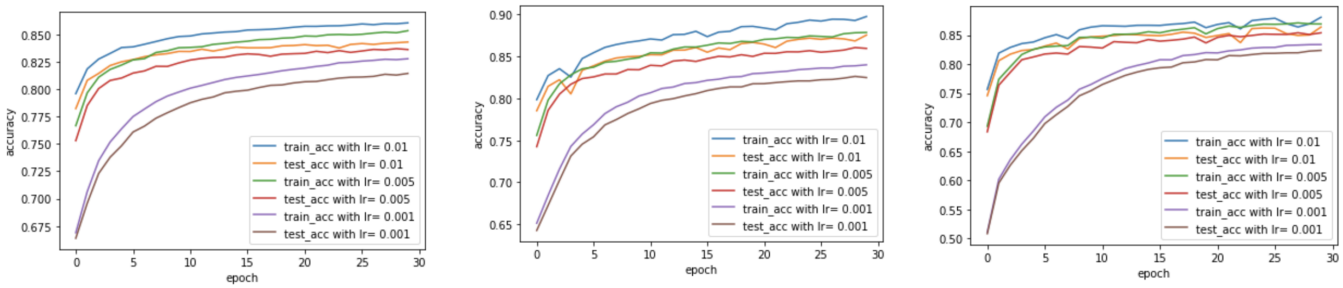
The results of the performance of our models are listed in Table 1.

Model	optimized lr/reg coef	Training Accuracy	Validation Accuracy	Testing Accuracy
MLP (0 hidden layers) +ReLU	0.01	85.9%	85.4%	83.6%
MLP (1 hidden layers) +ReLU	0.01	89.2%	87.7%	86.3%
MLP (2 hidden layers) +ReLU	0.01	88.3%	87.1%	86.1%
MLP (2 hidden layers) +Tanh	0.005	85.2%	84.9%	83.0%
MLP (2 hidden layers) +LeackyReLU	0.005	87.3%	86.6%	85.1%
MLP (2 hidden layers) + ReLU +Unnormalized	0.0005	87.3%	85.5%	84.4%
MLP (2 hidden layers) + ReLU +Adam	**0.0001**	**89.0%**	88.4%	**86.7%**
MLP (2 hidden layers) +ReLU & Regularization	lr: 0.01 reg coef: 0.0001	88.2%	87.2%	86.0%

Table 1: Accuracy Results with different variations of MLP and Hyperparameters

Figure 2 shows the changes in accuracy with respect to changes in the hyperparameters for (a) zero hidden layers, (b) one hidden layer, and (c) two hidden layers.
Regarding the number of hidden layers, we see that going from zero to one hidden layer improves the performance of MLP. This result is expected because the MLP without hidden layers is equivalent to Softmax regression which isn't sufficient and expressive enough for the given image data input. However, after going from one hidden layer to two hidden layers, we see a drop in accuracy. Since we didn't see signs of overfitting in the plots, we hypothesize that this may be because 2 layers model is trained slower than 1 layer model, and with our epsilon, the model is not identified as converged with epoch size 30. It's also possible that 1 layer is enough for expressiveness and adding more layers only lead to more complexity and confused the model prediction.

In addition to ReLU, we implemented two other activation functions to investigate how changing acti-

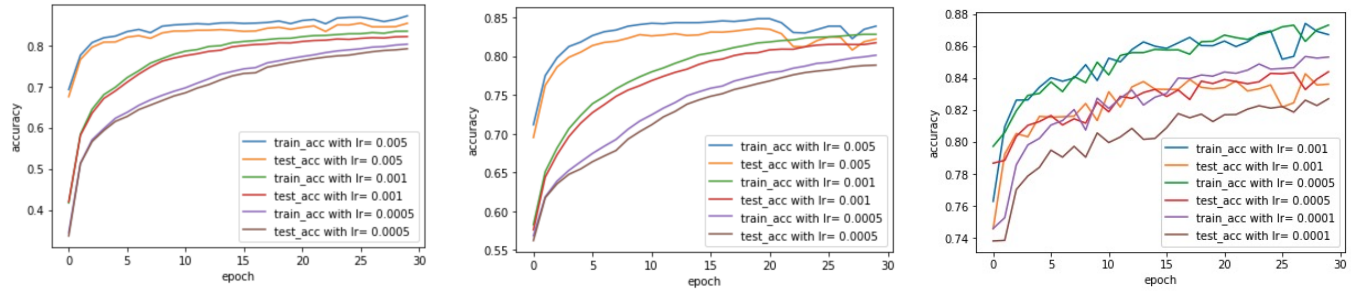


(a) Accuracy of model with 0 hidden layers as learning rate and epoch number change.

(b) Accuracy of model with 1 hidden layer as learning rate and epoch number change.

(c) Accuracy of model with 2 hidden layers as learning rate and epoch number change.

Figure 2: MLP's performance with different number of hidden layers and hyper paramaters



(a) Accuracy of model with 2 hidden layers and LeakyReLU activation function as learning rate and epoch number change.

(b) Accuracy of model with 2 hidden layers and Tanh activation function as learning rate and epoch number change.

(c) Accuracy of model with un-normalized data and 2 hidden layers and ReLU activation function as learning rate and epoch number change.

Figure 3: Different activation functions and un-normalized data

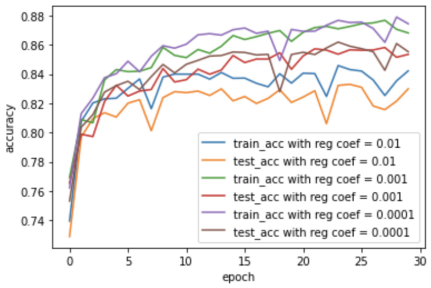
vation function affects the performance of the models. Figure 3 (a) and (b) compares the accuracy of MLP (2 hidden layers) with LeakyReLU and Tanh activation functions, respectively. We see from the results in Table 1 that in general, the performance doesn't change significantly but leaky relu and tanh activation function both give slight lower accuracy compared with ReLu.

Unnormalized Data

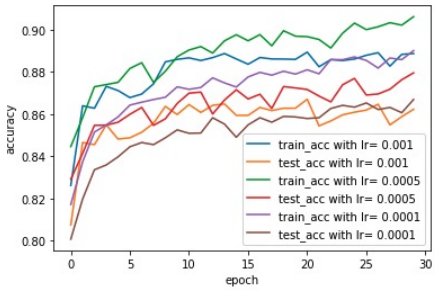
Figure 3 (c) shows the performance of MLP with 2 hidden layers applied to un-normalized data. Comparing this with the model with normalized data, we see that unnormalization results in an decrease of about 2% in accuracy and the plot fluctuates more. This is as expected. With normalization, we can reduce the effects of inconsistent data and improve overall data quality. Also, we scaled down the learning rate since un-normalized data has larger scales.

Regularization

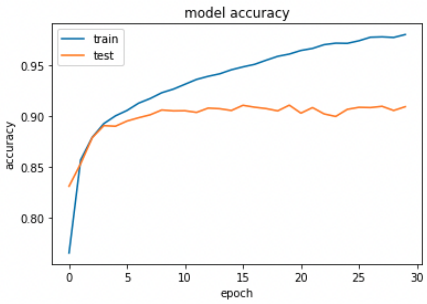
Figure 4 (a) shows how the regularization coefficients affect the performance of a model under a given learning rate. In this case, we considered the regularization coefficients as hyperparameter instead. We applied L2 regularization with the best learning rate from 2 layer model with relu activation function (0.01) and optimized the regularization coefficients to obtain the best accuracy. We saw from data preprocessing that the training data and testing data are almost from the same distribution, so regularization will shift the already overlapped distribution away, increase the variance and reduce the accuracy. We have lower regularization coefficients corresponding to higher accuracy as expected.
Besides using regularization to optimize our model, we also implemented Adam [Kingma and Ba, 2014]



(a) Accuracy of model (2 hidden layers) with ReLU activation function as the regularization coefficient and epoch number change.



(b) Accuracy of model (2 hidden layers) with ReLU activation function and Adam optimizer



(c) Convolutional neural network with 2 Channels size of 32 and 64, learning rate 0.005, stride 1

Figure 4

optimizer which takes momentum into consideration. The optimizer will regulate the gradients to make the training more stable.

Performance of model with CNN

We constructed CNN using Keras. We used two convolutional layers with kernel size 3 and two max pooling of size 2 after each convolutional layer. Then we passed it through two dense layers with 128 units with ReLU activation function, and finally a Softmax activation function in the end. To optimize our results, we ran experiments with different channel sizes and stride numbers for the convolutional layers as well as the learning rate. The best accuracy we arrived at was 90.8% which was obtained with a learning rate of 0.005 and stride number 1 and channel sizes 32 and 64 for the first and second convolutional layers, respectively. Please refer to Table 2 for further detail.

Best MLP architecture

To find the best MLP architecture, we first examined how the number of neurons in each layer affects the accuracy. We used both 2 layers and 1 layer models with ReLu activation function and lr=0.01. Table 3 below shows the results run on Mini-Batch Gradient Optimizer and we can see that with more units in each layer, we have overall better performances. This is as expected since more neurons mean more "stored information". Due to the limitation in computing resources, experiments on large scale cannot carry out. Further study would focus on experimenting with architectures with more layers to fully extract the power of deep learning and training the model with more epochs. Moreover, various optimizers (e.g. RMSProp) could also be deployed to see their effects.

From the experiments from Table 1, we can also see that Adam optimizer performed the best. Thus, we also implemented a 1 layer and a 2 layers model with Adam optimizer, ReLu activation function and lr=0.0005. The training accuracy, validation accuracy and testing accuracy are (96.7%, 89.8%, 89.1%) and (90.0%, 87.2%, 86.8%) respectively.

The current best model is 1 layer MLP with the number of neurons=512, ReLu activation function, Adam optimizer and lr=0.0005. We further examined the effects of regularization and the number of trained epochs on this model. With regularization coefficients=0.000001 and trained epochs=30 as usual, we obtained the highest validation accuracy=90.3% and corresponding testing accuracy = **89.6%**.

Lastly, as we discussed before maybe 30 epochs for 2 layer model is not enough, so we also trained the above best model with 2 layer MLP and the number of epochs=50. It turns out that the validation accuracy is only 88.7%, so the best MLP architecture we found is a 1 layer MLP with 512 neurons, ReLu activation function, Adam optimizer, regularization coefficient=0.000001, learning rate=0.0005

and trained on 30 epochs.

Conv1 # chnls. Conv2 # chnls.	Model Details	Train Acc.	Valid. Acc.	Test Acc.
Conv1:32, Conv2:32	lr:0.001, Stride:1	88.3%	87.6%	86.6%
Conv1:32, Conv2:32	lr:0.001, Stride:2	83.3%	83.5%	82.4%
Conv1:32, Conv2:32	lr:0.005, Stride:1	92.0%	89.3%	89.3%
Conv1:32, Conv2:32	lr:0.005, Stride:2	88.1%	87.3%	85.7%
Conv1:32, Conv2:64	lr:0.001, Stride:1	89.6%	89.3%	88.1%
Conv1:32, Conv2:64	lr:0.001, Stride:2	84.7%	84.5%	83.2%
Conv1:32 Conv2:64	**lr:0.005, Stride:1**	**93.4%**	**90.9%**	**90.3%**
Conv1:32, Conv2:64	lr:0.005, Stride:2	89.0%	87.2%	86.7%

Table 2

# of units in each layer	# of layers	Train accuracy	Valid. accuracy	Test accuracy
32	2	86.3%	85.9%	84.4%
	1	88.6%	87.6%	86.1%
64	2	86.9%	86.5%	84.7%
	1	89.1%	88.2%	86.3%
128	2	88.1%	87.2%	86.0%
	1	86.4%	85.8%	84.3%
256	2	88.3%	86.8%	86.1%
	1	90.0%	88.1%	87.1%
512	2	90.1%	88.5%	87.4%
	1	90.1%	88.8%	87.3%

Table 3: Experiments run on SGD optimizer

Discussion and Conclusion

We successfully implemented MLP and applied hyperparameter tuning and evaluated its performance using the fasion-MNIST dataset as benchmark. Moreover, we experimented with different number of layers and activation functions for the MLP. We found that ReLU and LeakyReLU perform almost equally well and better than Tanh activation function. The highest accuracy achieved was 86.7%, which was obtained using 2 hidden layers, ReLU activation function and Adam optimization. Moreover, we compared the use of normalized and unnormalized data for MLP and concluded that normalization increases the accuracy by 2%. This result is expected since normalization reduces data inconsistencies. We also applied regularization on the MLP, but we observed a slight drop in accuracy. This was because the training and testing data were taken from the same distribution and regularization ended up increasing the variance as a result of the overlapping distribution. We also created a CNN model with optimized architecture and hyperparameters and we arrived at the accuracy of 90.3%. This outperformed our previous MLP model by about 3%. However, we developed a more powerful architecture and modified our MLP to improve its performance and we reached 89.6% accuracy by the help of higher number of neurons in each layer and with carefully tunned hyperparameters. Due to limited computing resources, we are not able to carry out large experiments to train the model. Further study would focus on large-scale experiments such us using more epochs and more layers to fully examine this deep learning concept. We also find that with larger learning rate, our model trained very fast at the begining but becomes unstable in latter epochs. Thus, we may also examine Adagrad to see if adaptively changing learning rate could help improve model performance.

Statement of Contributions

Navid Hassen Zadeh: Report Writing, CNN
Chloe Si: Model Construction, Report Writing
Steve Wen: Model Construction, Report Writing

Bibliography

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

M. Ko, U. K. Panchal, H. Andrade-Loarca, and A. Mendez-Vazquez. Coshnet: A hybird complex valued neural network using shearlets. *arXiv preprint arXiv:2208.06882*, 2022.

O. K. Oyedotun, K. Papadopoulos, and D. Aouada. A new perspective for understanding generalization gap of deep neural networks trained with large batch sizes. *Applied Intelligence*, pages 1–17, 2022.

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