Classification of Image Data using Multi-Layer Perceptron and Convolutional Neural Network

COMP 551 Mini Project 3

Chloe Mills, Shania Wan-Bok-Nale, Khoi Nguyen

March 2022

Abstract

In this project, we were tasked to implement Multi Layers Perceptron and Convolutional Neural Network classification models to classify the Fashion-MNIST dataset. The Fashion-MNIST dataset consists of over 70,000 example of 28x28 grayscale images over 10 classes. Our goal was to implement a Multi Layer Perceptron and a Convolutional Neural Network that correctly classifies the images. We implemented the models with different hyper-parameters to find the best performing models. In fact, we found the best performing multilayer perceptron to be a two hidden layer model which uses logistic activation function, trained with learning rate of 1.5 and mini-batch with batch size of 16. After 250 iterations, we achieved an accuracy of 82.34%. Our CNN with dropout model on the other hand, had a performance of 91.88% test accuracy. Overall, our CNN implementation performed the best.

Introduction

Our task was to create Multi Layer Perceptron Machine Learning models to run on the Fashion-MNIST to classify clothing items into the 10 classes. In 2021, it was estimated that 2.14 billion people worldwide would be using online platforms to shop. With this ever increasing online presence of the world population, the ability to classify images into categories seems to be desirable. The developers of this dataset argue that the original MNIST dataset is ”too easy” to classify and hence they came up with Fashion-MNIST, a more challenging dataset to train and test your machine learning models[1]. A model that passes with fashion-MNIST is hence more reliable than a model that passes with the traditional MNIST dataset. We trained our MLP models using a training set of 60,000 image instances from the Fashion-MNIST dataset. We decided on the score of our models based on the test accuracy. Each model was trained with different hyper-parameter settings to achieve the highest accuracy. After careful experimentation, we observed tests scores of 82.34% in our MLP implementation and 91.88% in our CNN implementation.

Dataset

Fashion-MNIST is a dataset developed by Zalando which consists of fashion and clothing items within 10 classes:T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. It was developed as an antecedent to the MNIST handwriting dataset to further challenge classification models. The dataset itself contains 70,000 instances: 60,000 of them for training and 10,000 of them for testing. 1 instance of the dataset consists of a 28x28 grayscale image and it’s classification target. The training set has a shape of (60000,1,28,28) and the test set has a shape of (10000,1,28,28).

When importing the dataset we normalized the values such that they are in the range of 0-1. We split the training dataset into 50,000 for training and 10,000 for validation. Figure 1 outlines the perfectly equal class distribution which ensures the model will not be biased towards a class.

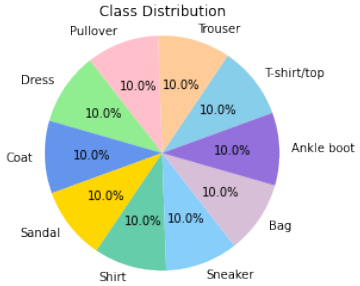

Figure 1: Equal class distribution in Fashion MNIST dataset

Results MLP

Our best multilayer perceptron model contains two hidden layers. Since the Fashion-MNIST dataset is a multi-class dataset, we implemented softmax regression as final layer (i.e. the output layer). Our steps are as follows: in the gradient function inside of the fit function, we took the dot product of the input data and the first hidden layer v (weight parameter) which represents the weights we attributed to the pixels. The input data has shape (50000, 784) and the weight parameter v has shape (784,128). We passed the resulting array through a ReLu activation function and took the dot product with our second hidden layer w. This resulted in an 2-dimensional array where each column represented one of the ten classes. Hence, we passed this into our softmax regression activation function which is our output layer. The output consists of the probability that an image is in each of the 10 classes. We calculated the loss by subtracting the training target from the output. From this point on we performed back propagation in order to improve the results of dv and dw according to the loss. dv and dw are the deriva-

tives of each hidden layer respectively. We did not uses biases in each layer since we did not want to negatively influence our accuracy as a result of a poor choice of bias.

We encountered overflow in our calculations, and in order to overcome it, we created a normalize function that takes a matrix as input and transforms the matrix to have normal distribution with a mean of 0 and standard deviation of 1. We normalized the parameter v and the start of each iteration of the gradient function as well as the dot product of the input data with v. Normalization helped most of the overflow errors.

After experimenting with different learning rate parameters in the range of 0.05 to 0.25 inclusive, using ReLu as our activation function we found 0.25 to produce the best results. We noticed that when the learning rate is too high, the gradient function may not converge. On the other hand, when the learning rate is too low, the gradient function function takes much longer to converge. This observation is to be expected because we multiply and subtract the current weight parameter by the learning rate, to obtain the updated weight parameter at each iteration of gradient descent. We also implemented a learning rate decay method that starts the gradient descent iteration at a high learning rate such as 0.4 and decreases the rate at each iteration by multiplying the current rate by a number between 0 and 1 until it reaches a certain threshold. After that threshold, the learning rate remains constant for the remaining number of iterations. Figure 2 outlines our results.

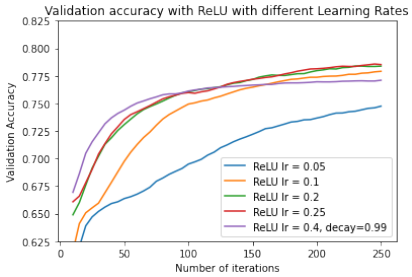

Figure 2: Validation accuracies of ReLU MLP with different learning rates

In order to experiment with the number of hidden layers in our multi-layer perceptron, we used the assumption that each hidden layer is produced by multiplying the previous layer with a weight parameter that we update using backpropagation. Hence, the MLP with 2 hidden layers has 2 weights, v and w respectively each with 128 units. It obtained a test accuracy of 77.98% after 250 iterations while using ReLu as the activation function and learning rate of 0.25. The MLP with no hidden layers, connects the input layer directly to the output layer(i.e. the softmax regression function). The input dataset is of size N x D and softmax requires an array of size N x C where C is the number of classes. Thus, we took the dot product of our input data transposed, and a matrix of 1's of size D x C in order to get the desired size to compute the output. The matrix of 1's gives equal weight to all pixels. Since our model must not have any hidden layers, we could not adjust these weights. The test accuracy of our MLP with no hidden layers was 10% since each class had equal probability of 0.1. This model is poor since it directly maps to input to the output and there are no parameters to train. This result was expected. For one hidden layer MLP implementation, we had one weight parameter w with 128 units. To our surprise, with learning rate set to 0.05 and activation function ReLu, the one hidden layer model reached test accuracy of 77.77% after 250 iterations. Since the no hidden layer model was much worse than the 2 hidden layer model, it was surprising that the 1 hidden layer model was close in test accuracy to the 2 layer model. Network depth affects the accuracy up to a certain extent but increasing the number of layers does not result in a linear climb of accuracy. Hence the accuracy increases when adding more hidden layers but this increase is non-linear.

We tested out different activation functions on our model and their impact on the validation accuracy in comparison to the ReLu activation function. The three activation functions we experimented with are Leaky-ReLu, tanh and logistic. Our results are summed up as follows: Leaky-ReLu has a learnable parameter y that we set to 0.1 but later updated to 0.25. When testing with smaller learning rates of 0.05, 0.2 and 0.25, leaky-ReLU with the parameter y=0.25 seems to perform the best, except with the highest learning rate instance of 0.25. This was expected since we likely encountered the "dying ReLU" problem where large negative weights become forever stuck at 0, a problem that leaky-ReLU avoids by allowing small negative weights instead of strictly forcing the values to be 0.

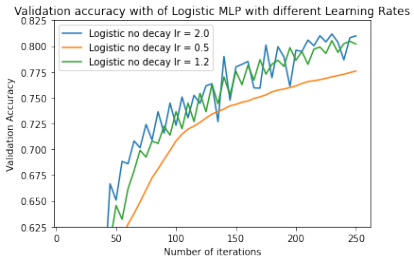

Figure 3: Validation accuracies of Logistic MLP with different learning rates

Logistic activation function requires a larger learning rate in order to perform well and we found that a learning rate between 0.5 and 2.0 produced good results with the best learning rate being 2.0 which yields a test accuracy of 80.39%. Figure 3 outlines these results. An unexpected result we found was that the logistic activation function outranks ReLu and Leaky-ReLu in terms of test accuracy. Firstly, it does not cause overflow because it limits the range of values to be between 0 and 1 and so it does not need the normalize function. Secondly, it does not diverge at high learning rates. It starts to become unstable when the number of iterations are high but when using decaying learning rate, it stabilizes. Hence, this helps to explain why logistic function produces a better result than ReLu and Leaky-ReLu.

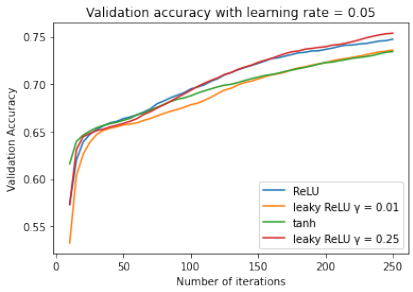

Figure 4: Validation accuracies of different MLP models with learning rate of 0.05

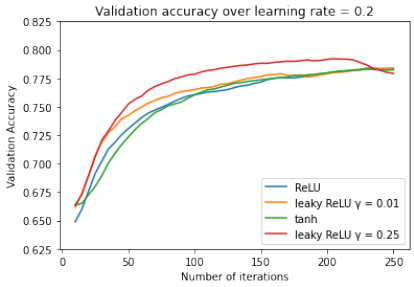

Figure 5: Validation accuracies of different MLP models with learning rate of 0.2

After careful observation, we found that the tanh activation function limits values between -1 and 1 similar to logistic which limits values to be between 0 and 1, hence it does not cause overflow. We also observed that tanh also performed better as we increase the learning rate. We then removed the normalization function between each layer and tested the tanh model with higher learning rate of 0.55. The result was the highest validation and test accuracy from any previous models using regular gradient descent at 81.13%. This confirmed our hypothesis that tanh needed higher learning rates with no normalization in-between layers to achieve better results. Furthermore, we observed that logistic and tanh both experience sharp drops and rises in accuracy on their path of convergence, reinforcing their similarity. The only difference we found is that tanh uses smaller learning rates compared to logistic as its accuracy is sensitive to its learning rate, being prone to divergence at high learning rates.

The reason why our model worked better with logistic and tanh activation functions, rather than ReLU or leaky-ReLU is because we encountered the exploding gradient problem. This is where values get exponentially bigger between layers and lead to overflow. Normally ReLU, works better than sigmoid activation functions since it fixes the vanishing gradient problem that sigmoid functions have, which typically causes underflow. However in our case, we encountered the opposite problem, which was fixed with sigmoid activation functions limiting the range of values, leading to higher test accuracies.

In order to perform regularization, we experimented with dropout on the MLP model with 2 hidden layers. We found that since the training and validation accuracy (and test accuracy) were so similar at each iteration, this method did not improve performance. In fact, since dropout decreases the variance by increasing bias, and our multi-layer perceptron validation accuracy is nearly equal to the test accuracy, implementing dropout simply increased the bias and decreased the accuracy in both test and train.

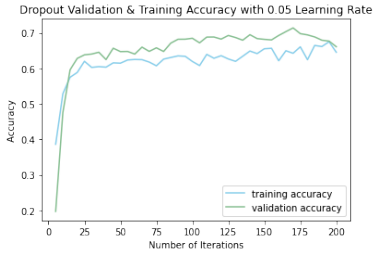

Figure 6: Dropout effect on MLP with 0.05 learning rate.

When importing the dataset using the library torchvision, and setting transform = transforms.Compose([transforms.ToTensor()]), this produced a normalized dataset with values between 0 and 1. To import the unnormalized form of the dataset, we used the keras library. While comparing the test accuracy of training the model on the unnormalized vs. normalized dataset, we noticed that their difference was smaller than expected. The normalized dataset produced a test accuracy of around 8% larger than the unnormalized dataset. A possible explanation as to why the difference is smaller than expected is because we normalized other matrices that represent the dot product of the unnormalized data and the weights of a certain hidden layer. Hence, the resulting matrix does not have values as large as expected of unnormalized data.

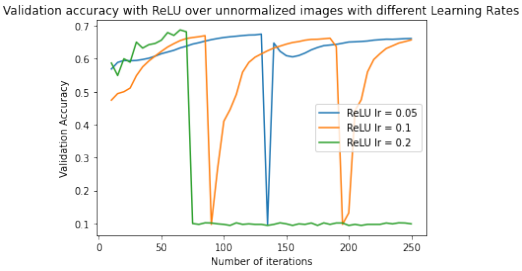

Figure 7: Validation accuracies of ReLU MLP with different learning rates over unnormalized training set

Figure 7 outlines the experimentation of testing different learning rates on the unnormalized dataset. Although the accuracies with learning rate of 0.05 and 0.1 respectively, drop at certain iterations, they climb back up to around 8% less than the normalized input.

Further experimentation included implementing stochastic gradient descent. Looping through 50000 samples with the maximum number of iterations set as low as 250, was costly since our model took a very long time to run. In order to overcome this obstacle, we used mini-

batches of 8, 16 and 32. This greatly increased how quickly the model ran and achieved high accuracy. The mini-batch of 16 produced the best results. We tried using decay on the learning rate while using stochastic gradient descent and the mini-batches but even without decay, the model did converge at a faster pace. Overall, using mini-batches increased our accuracy at each iteration, but only by a small amount. The cost was a longer training time compared to regular gradient descent.

We attempted to experiment using ADAM with stochastic gradient descent. Unfortunately, training the model using ADAM consistently gave a test accuracy result of 0.1 which implies that there was an overflow error. It was not until we increased the learning rate significantly while using batch size of 16 that we got a validation accuracy of around 0.47 before the model diverges. Since ADAM uses adaptive learning rates for each parameter it was odd that we had to set the learning rate to be very high in order for the model to converge, otherwise we found the model to instantly diverge.

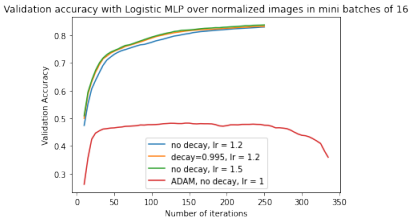

Figure 8: Validation Accuracy over different parameters using Logistic MLP in mini batches of 16

We wanted to compare the results of training the model with a smaller sample of the dataset to the 50000 samples we had used before. We tested the model with ReLu activation function using 1, 10, 100, 1000, and 10000 samples. At each sample size, we ensured the class distribution within the dataset remained equal with 1:10 ratio for each class (except for sample size=1). With one sample, the hidden layers were initialized with random values, therefore having a low validation accuracy of around 0.15 before converging to 0.1. This makes sense since we are fitting the model with exactly 1 class. Therefore it would predict this class every time, resulting in a probability of 1/10. With 10 samples, the accuracy went up to 0.51, which was to our surprise since we used a very small sample size. At 100 samples, the accuracy was 0.69 which is a much smaller increase but expected with diminishing returns. At 1000, the accuracy reached 0.76. Using 10000 samples, we got an accuracy of 0.79 which is very close to training with the 50000 samples. This is surprising since it begs the question if it is worth training with more samples if the accuracy is very similar while using 1/5th of the dataset.

In our gradient descent class, we printed out the training accuracy, validation accuracy and test accuracy at every 5th iteration of the while loop. Note that we did not give the model the test accuracy in order to improve, this was solely for our curiosity. To our surprise the three accuracy's were extremely similar. (1) A possible explanation could be the equal distribution of classes among the input data and the samples that belong to a class are very distinct in comparison to other classes. There were very few cases where the model over-fit to the training data which we can see from the similarity of the training accuracy and the test accuracy. The only cases where the test and training accuracies differ, was in the case of small samples of the dataset. For instance, with the 10 sample experiment, the training accuracy was considerably higher in comparison to the test accuracy. Perhaps, the number of iterations is set too low such that the model does not over-fit but we believe the explanation is outlined in (1).

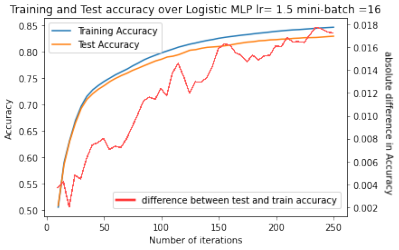

Figure 9: Training and Test accuracy over Logistic MLP lr=1.5, mini-batch=16

Results CNN

We were tasked to implement a Convolutional Neural Network model using existing libraries to be used on the same Fashion-MNIST dataset. We implemented our CNN using Tensorflow libraries like keras. We preprocessed the dataset by normalizing and reshaping the training and test sets which will be passed into the model. We then used one hot encoding to turn the target into boolean columns for each class. We then used SciKit Learn to split the training dataset into 80% training set and 20% validation set. We then constructed the CNN model using sequential() from Keras libraries. We were tasked to build 2 convolutional and 2 fully connected layers and we built the layers as follow: The first convolutional layer has 32 units- 3x3 filters, the second convolutional layer has 128 units - 3x3 filters. We also added max pooling layers of size 2x2 to each convolutional layers. We used activation layers using reLU activation function. We designed the two fully connected layers to be a dense layer of 128 units and the output layer of 10 units for the 10 classes respectively. We set the batch-size to 64 and the number of epochs to 10. Figure 10 and 11 highlights our training & validation accuracy as well as our training validation loss results. We then ran the CNN model on the test set and got a test loss of 0.279 and a test accuracy of 91.88%. We however think that our model is over-fitting. From the same figures 10 and 11, we observe that the validation accuracy stagnates after only around 3 epochs. Similarly, we observe that our validation loss also stagnates at around 3 epochs. This is clear signs of over-fitting.

To improve our CNN implementation we added dropouts to our CNN model to prevent over-fitting. Dropout

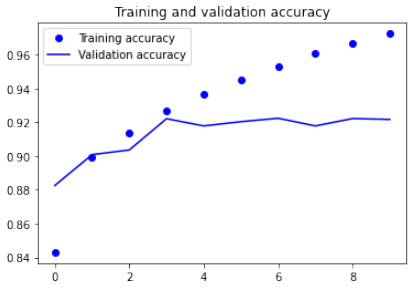

Figure 10: Training and Validation accuracy of our first CNN

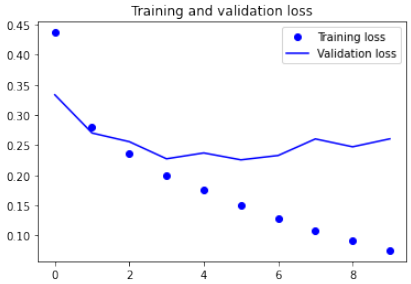

Figure 11: Training and Validation loss of our first CNN

layers were added after the first two max pooling with a rate of 0.25 and after the last activation layer with a rate of 0.30. Figure 14 and 15 in the appendix highlights our new accuracy and loss results over the training and validation sets. We see that the test and validation accuracies follow the same trend and thus we can conclude that this new model has a good fit.

We then ran the CNN with dropouts on the test set and got test loss of 0.22 and test accuracy of 91.61%. Lower test loss is a good sign but our accuracy did not improve from our first CNN implementation. However we can see that the loss and accuracy is consistent over our training and validation set. This implies that this CNN implementation is of a good fit and is thus, a more reliable model. Figure 14 and 15 highlights our results.

We further experimented with our new CNN model. We changed some hyper-parameters from our last implementation as follow. We increased the kernel sizes to 5x5, the pool size to 3x3, the batch size to 128 and decreased number of epochs to 5. We observed a higher test loss of 0.2777 and lower test accuracy of 89.89%.

We also varied the number of images being fed to train the model and saw how it would affect the test accuracy.At each sample size, we ensured the class distribution within the dataset remained equal with 1:10 ratio for each class. We compared our best performing CNN model with dropoouts to our own Logistic MLP implementation with learning rate of 1.5 which was our best MLP performing model. Figure 12 highlights the low test accuracies with little amount of trained images and the sharp increases in test accuracies as we increase the number of trained images.

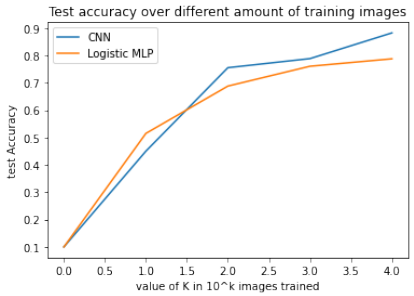

Figure 12: Test accuracy of MLP and CNN with different size of training set

Discussion and Conclusion

In this project, we implemented two machine learning models, namely Multi Layer Perceptrons and Convolutional Neural Network, for the purpose of classifying images of clothing items using the popular Fashion-MNIST dataset. We examine some points worth discussing.

We observed that our Multi Layer Perceptron implementation had low varying test scores when implementing the model with different activation functions, number of hidden layers and learning rates.We found the best performing multilayer perceptron was a two hidden layer model which uses logistic activation function, trained with learning rate of 1.5 and mini-batch with batch size of 16. After 250 iterations, we achieved an accuracy of 82.34%. That being said, we found our CNN model to outperform all our MLP implementations. Our CNN with 2 convolutional layers coupled with dropouts had a final test score of 91.88% test accuracy and test loss of 0.279, This was to be expected as CNN was built for image classification. As for future investigation, we would like to investigate more on how to improve our CNN implementation for better performance. We could do so by adding more convolutional layers and tweaking with he hyper parameters to achieve a model with a good fit and better test scores. We also want to experiment with dataset not part of Zalando which Fashion-MNIST originates from.

Statement of Contributions

All three members of the group contributed equally to this project, we even had a fun time together and it was a great learning curve for all three of us. Nguyen and Mills worked on the MLP models. Wan-Bok-Nale worked on the CNN model. We all contributed to the experiment. We all contributed to the write up of the report of our respective work.

Appendix and Citations

[1] Zalandoresearch. (n.d.). Zalandoresearch/fashion-mnist: A mnist-like fashion product database. benchmark. GitHub. Retrieved April 1, 2022, from https://github.com/zalandoresearch/fashion-mnist

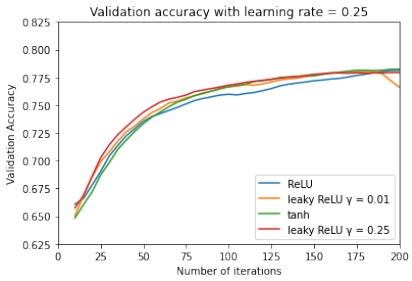

Figure 13: Validation accuracies of different MLP models with learning rate of 0.25

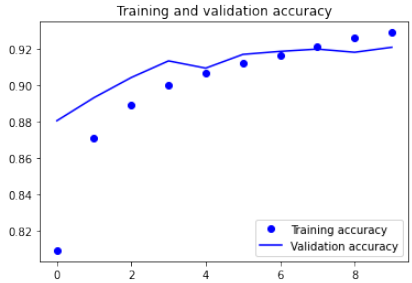

Figure 14: Training and Validation accuracy of our second CNN with dropouts

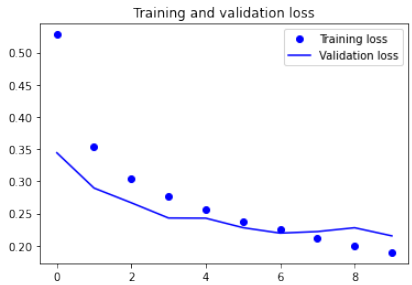

Figure 15: Training and Validation loss of our second CNN with dropouts