Mini Project 3 Report

Classification of Image Data

COMP 551: Applied Machine Learning

Maxime Buteau	260868661
Charles Couture	260923463
Eric Pelletier	260895863

McGill University
April 7th, 2022

Abstract

A key element in machine learning is the ability to predict outcomes with the highest accuracy possible. In this project, we investigated the performance of a machine learning model —*Multilayer Perceptron*— on a benchmark dataset —*Fashion-MNIST*—. During the experiments, we evaluated the effects that difference number of hidden layers as well as difference activation functions had on our model's accuracy. In addition, we compared our model's accuracy to a *Convolutional Neural Network's* accuracy. In the end, we found that the *Multilayer Perceptron* model reported a better accuracy with 2 hidden layers compared to 1 or 0 hidden layers, and had the best accuracy using the relu activation function. Overall, the *Convolutional Neural Network* reported better accuracy than the *Multilayer Perceptron*, which was to be expected for image analysis.

1 Introduction

During this project, we implemented a *Multilayer Perceptron* from scratch and used it to classify different types of images. We also implemented a *Convulotional Neural Network* with the use of python libraries such as *TensorFlow* and *PyTorch* and compared the results with our *Multilayere Perceptron.* The dataset analyzed during this process was the *Fashion-MNIST dataset* which consisted of a training set of 60,000 examples and a test set of 10,000 examples, where each example is a 28x28 gray scale image and associates with a label from 10 classes. It is often the first dataset researchers try when testing and some say that "If it doesn't work on MNIST, it won't work at all" [Dan. B, 2022].

The *Multilayer Perceptron* algorithm is simply a feed forward artificial neural network which is characterized by several layers composed of nodes that are connected as a directed graph between the input and output layers. There also can be different layers between those two called *hidden layers* which can affect the training phase of the algorithm, which is done by back propagation. This back propagation uses a mini-batch stochastic gradient descent to optimize the values of weights and biases in the neural networ. On the other hand, the *Convulotional Neural Network* algorithm is another neural network that is specialized in processing data which are grid-like, such as images. The layers in this model are arranged in such a way that they can detect simpler pattern first, such as lines or curves, and more complex patterns further along, such as faces or objects.

By testing different number of hidden layers and different activation functions in the *Multilayer Perceptron*, we evaluated the effects that this value has on the training data accuracy and test data accuracy. Furthermore, we used *Convolutional Neural Network* to compare the resulting accuracies and determine which algorithm is the best for certain situations.

Overall, we found that different activation functions can significantly affect the results provided by the *Multilayer Perceptron.* We also observed that having a certain number of hidden layers can provide better accuracy up to a certain number. After that, the model starts to overfit the training data. However, the *Convulotional Neural Network* seems to offer even better results when testing image datasets such as *Fashion-MNIST.* This is due to CNN's ability to capture higher-level details in images such as shapes and lines.

2 Datasets

The dataset used in this project was the *Fashion MNIST* dataset. It consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 pixel grayscale image representing one of 10 pieces of clothing. Each pixel has a value between 1 and 255 and these pixels were used as inputs to our Mulilayer Perceptron. To adapt this dataset to our neural network, we first divided each pixel by 255 to normalize their values. Then, we flattened the 2D array representing our image into a 1D array so that each pixel can be easily passed as input to the MLP. Finally, we transformed the class values into a one-hot-encoded array of length 10 where every element has a value of 0 except the one at the index of the correct class which is set to 1. This makes it easier to compare the array of 10 probabilities that we get as output from our MLP with what the actual value should be.

3 Results

3.1 Comparison of the MLP's Accuracy with Various Number of Hidden Layers

For this experiment, we created 3 multilayer perceptron model with different number of hidden layers. The first model did not contain any hidden layers, the second model contained 1 hidden layer with 128 units and the third model contained 2 hidden layers with 128 units each. Each model used the ReLu activation function and performed 3 iterations of the stochastic gradient descent. The accuracy of each model after each iteration of the gradient descent as well as the accuracy reported by the model on the test set can be viewed in table 1.

MODEL	1 ITERATION	2 ITERATIONS	3 ITERATIONS	TS ACCURACY
MLP w/ 0 hidden layers	65.43%	73.30%	76.05%	75.54%
MLP w/ 1 hidden layer	81.64%	83.83%	84.93%	83.93%
MLP w/ 2 hidden layers	82.66%	84.42%	84.01%	83.25%

Table 1: MLP Model Accuracy with Various Number of Hidden Layers.

From this table, we can deduce that the MLP with 1 hidden layers performs slightly better than the model with 2 hidden layers after the 3rd iteration of the gradient descent and on the test set. However, the MLP with 2 hidden layers performs better than the model with 1 hidden layer after the first iteration. Both model with hidden layers perform better than the model with 0 hidden layers. These results are slightly different from what we expected as we predicted that the model with 2 hidden layers would perform better than the other 2 models.

3.2 Comparison of the MLP's Accuracy with Activation Functions

In the next experiment, we created 2 new MLP model's with 2 hidden layers containing 128 units each and different activation functions. The first model created used the Leaky-ReLu activation function and the second model created used the tanh activation function. Also, both of these model's performed 3 iterations of the stochastic gradient descent. We then compared these 2 new model's to the 2 hidden layer MLP model created in the previous experiment that uses the ReLu activation function. The accuracy of each model after each iteration of the gradient descent as well as the accuracy reported by the model on the test set can be viewed in table 2.

MODEL	1 ITERATION	2 ITERATIONS	3 ITERATIONS	TS ACCURACY
MLP w/ ReLu	82.66%	84.42%	84.01%	83.25%
MLP w/ Leaky-ReLu	82.36%	84.28%	85.24%	83.96%
MLP w/ tanh	49.74%	51.38%	41.32%	41.12%

Table 2: MLP Model Accuracy with Various Activation Functions.

From table 2, we notice that the models with the ReLu and Leaky-ReLu activation functions perform very similarly with the latter having a slight edge on the other. Both of these models perform significantly better than the model with the tanh activation function. This can be due to the fact that the tanh activation function is not able to handle the vanishing gradient problem like the other 2 activation functions are capable of doing. This problem causes the networks to not be able to backpropagate the gradient information to the input layers of the model correctly. Therefore, we can deduce that the ReLu and Leaky-ReLu activation functions are better than the tanh activation function for multilayer perceptrons.

3.3 MLP Accuracy with Drop Regularization

For this test, we implemented drop regularization into our MLP model with 2 hidden layers of 128 units each which utilized the ReLu activation function. Theoretically, this method should help our model's accuracy by reducing overfitting effects. We compared this model's accuracy to the same MLP model without dropout regularization. The accuracy of both model's can be displayed in table 3.

From table 3, we can deduce that the MLP utilizing dropout regularization performs slightly better than the MLP without dropout regularization.

MODEL	1 ITERATION	2 ITERATIONS	3 ITERATIONS	TS ACCURACY
MLP w/ Dropout	83.06%	84.61%	85.23%	84.64%
MLP w/o Dropout	82.66%	84.42%	84.01%	83.25%

Table 3: MLP Model Accuracy with Dropout Regularization.

3.4 MLP Accuracy on Unnormalized Images

In this experiment, we tried to feed the pixel values of the images into our MLP with Relu activation function but without performing any normalization first. This resulted in very poor results as can be seen from table 4

MODEL	1 ITERATION	2 ITERATIONS	3 ITERATIONS	TS ACCURACY
MLP w/ ReLu	9.22%	9.22%	9.22%	10.0%

Table 4: MLP Model Accuracy with Unnormalized Images.

3.5 CNN Model Accuracy

For this experiment, we used the *tensorflow* library to quickly train a convolutional neural network. As expected for images, we obtained slightly better results than with our implementation of MLP. This is likely because CNN is better at detecting higher-level features such as shapes and lines in images. The results can be seen in table 5.

MODEL	1 ITERATION	2 ITERATIONS	3 ITERATIONS	TS ACCURACY
CNN w/ ReLu	82.50%	88.36%	90.05%	88.41%

Table 5: Tensorflow CNN Model Accuracy.

3.6 Optimizing our MLP parameters

In this experiment, we tried to find the optimal parameters for our MLP implementation. We considered the learning rate, the number of hidden layers with 128 units each, the activation function used, and the number of epochs. Table 6 shows our top 3 best performing models and their parameters.

LEARNING RATE	HIDDEN LAYERS	ACTIVATION	EPOCHS	TS ACCURACY
0.01	2	ReLU	2	83.28%
0.01	1	Leaky ReLU	3	83.24%
0.1	0	Leaky ReLU	3	82.39%

Table 6: Best performing MLPs and their parameters

It is somewhat surprising to see that our MLP with 0 hidden layers could perform so well, which means only the softmax activation function was applied. This probably would not be the case for more complex and varying images where it is more important to find high-level features inside the image. It is worth noting that 28x28 images are very low resolution in today's standards.

3.7 Effect of the Number of Epochs

In this final experiment, we investigated how the number of epochs affected our accuracy results. This is an important consideration with neural networks, as a model that fits the training data too closely is very likely to be over fitting. Figure 1 shows our results with 1 to 10 epochs.

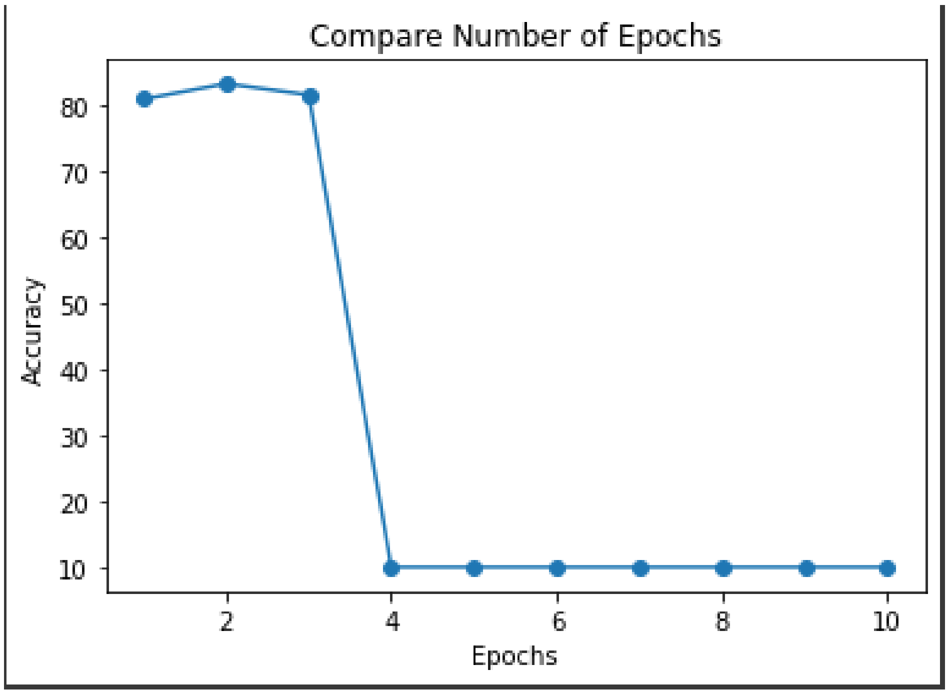

Figure 1: Test set accuracy for different epochs

We can see that after 4 epochs, our results become rather poor. While this is expected, it is rather drastic in our case and probably reflects some problem in our back propagation where we don't stop modifying our weights and biases once the error is under a certain threshold. Overall, the sweet spot seems to be between 2 and 3 epochs.

4 Discussion and Conclusion

In general, neural networks such as the Multilayer Perceptron or the Convolutional Neural Network seem to be great models to work with when it is not necessarily easy to extract concrete features from our dataset such as with images. An important consideration is the number of epochs used so as not to overfit the training data. In our case, this seemed to be around 2 or 3 epochs. This would likely be larger with higher resolution images. Both 1 and 2 hidden layers seemed to give good results, with the trade-off that more layers means a longer training time. Regarding activation functions, we found that both Relu and Leaky-Relu far outperformed Tanh.

For future tests, we could investigate how using batches larger than 1 affects our model when we apply mini-batch stochastic gradient descent during backpropagation. We could also stop modifying our weights once the error gets under a certain threshold so that we stop getting accuracies of around 10% after a certain number of epochs.

5 Statement of Contributions

We all contributed equally.

References

DanB, Shivam Bansal and Pavan Sanagapati. Fashion MNIST, 2022.
https://www.kaggle.com/datasets/zalando-research/fashionmnist.