MiniProject 3: Classification of Image Data

COMP 551, Winter 2022, McGill University

Integrants:
Alicia Nguyen
Cristina Penadillo
Lei Zhao

April 4th 2022

Abstract

The objective of our project was to design a Multilayer Perceptron architecture to optimally classify images from the Fashion-MNIST dataset. Experimenting with depth, width, optimization hyperparameters and weight initialization we ultimately achieved an accuracy of 84%. This was accomplished using a 3 hidden layer network with ReLU, Leaky ReLU and tanh activation functions, *but also* with a much simpler 1 hidden layer Leaky ReLU network. We note that the capacity to model nonlinear functions as provided by a hidden layer was integral to our task, however, further enhancement of performance was only achieved by the structure-aware capacities offered by the convolution layers of a CNN.

Introduction

The study of image classification in machine learning is a broad field entailing a variety of network architectures. Multilayer Perceptrons(MLP) and Convolutional Neural Networks (CNN) have been demonstrated to be effective at this task (LeCun 1995). For training, we used the Fashion-MINST dataset, consisting of 60 000 training images and 10 000 testing images in 28x28 grayscale associated with 10 labeled classes. The data was vectorized, normalized, and further augmented by various image transformation techniques to acquire more data, with further variation and increase its information potential.

We sought an optimized model for grayscale image classification, experimenting with MLP architecture in the form of width, depth and activation function type. In addition, we explored optimization hyperparameters such as learning rate and epoque. Different weight initialization techniques were also deployed, including randomization with zero mean gaussian distribution as well as activation function specific initialization. The latter was found to be integral to avoiding activation outputs exploding or vanishing.

Different MLP constructions were also compared to a pre-constructed CNN, where CNN's demonstrated superiority. However, we note the decreased capacity of CNN's to perform on augmented data that includes certain kinds of translations, demonstrating the CNN architectures' equivariance rather than invariance capacity.

The maximal accuracy achieved, after experimenting with extensive combinations of network architectures, was 84%, with most architecture's accuracy stagnanting around this performance level. Given CNN's improved performance that broke this plateau, it is likely that its unique contribution of a convolution layer broke this performance ceiling.

Dataset

The dataset for this project is Fashion-MNIST. It is a dataset of Zalando's images. The images of this dataset belong to 10 different classes including shirts, bags, t-shirts, etc. Each image has 28*28 pixels with one grayscale channel. The dataset consists of a training set size of 60,000 and a test set size of 10,000. Figure 1.1 shows the 2-D T-SNE visualization of 6000 training data. The colour is defined by the data's class.

To use the data in MPL, data is vectorized from 28x28 to 784x1. We used data augmentation to get more training data sets, three transformations including flipping, rotation and shifting from the center were performed in the data to generate examples for our network. For flipping, we allow the images to be flipped randomly both vertically and horizontally. For rotation, we allow the maximum rotation angle of 90°. For shifting, we allow 20% of the total width/height shifting. 48,000 images are generated for training.

The value for each pixel for each image are integers within the range of [0,255], therefore, we did a data normalization so the value for each pixel is scaled to the range of [0,1]. The realization is done by first converting the integer to floats and then dividing the original value by 255.

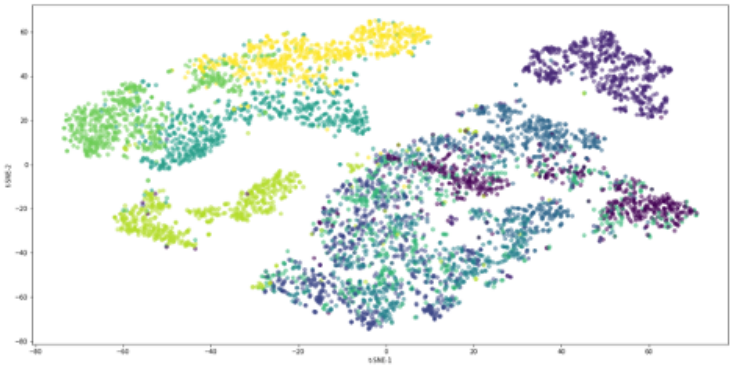

Figure 1.1 Fashion-MNIST 2-D T-SNE Visualization of 6000 Training Data

Results

3.1 Number of Hidden Layers

Initial results were unanticipated, where 1 additional layer (enabling non-linearity) showed an improvement from logistic regression, however, the addition of the second layer actually decreased accuracy. We expected that the addition of the capacity to model non-linear functions, combined with more learnable parameters should provide us with a better learning experience. Of course, this was resolved by tuning the hyperparameters for the layers with more models (see Appendix T3.1). As we added layers we increased the number of parameters that required learning and thus needed to increase the number of epoques just to properly train the model and ensure that gradient descent hadn't halted before finding a minimum. Adding non-linearity gave us a definitive performance boost over a logistic regression model, however, we were surprised to find that additional layers did not significantly improve accuracy, though did require substantially more training in terms of epoques.

The hyperparameter tuning showed that the accuracy of the model is better for medium values of learning rate (0.2) and a high number of iterations, best performing for 1 hidden layer and 128 hidden units.

Task 3.1 Best Accuracies, Hyperparameters with Accuracies, Untuned Hyperparameters						
	Varied HyperParameters			Non-varied Hyperparameters		
Number of Hidden Layers	Learning Rate	Epoques	Best Accuracy	Learning Rate	Epoques	Initial Accuracy
0	0.1	250	0.79	0.1	50	0.7044
1	0.2	400	0.83			0.7018
2	0.2	550	0.83			0.6159

3.2 Different Activation Functions

Activation functions are the backbone of our MLP's, and each has its own benefits and pitfalls. We initialized our weights to be activation specific for whichever layer was utilized to prevent activation outputs from exploding or vanishing: ReLu and Leaky Relu were initialized with He initialization that sampled numbers from a Gaussian distribution with a mean of 0 and standard deviation of sqrt(2/M_{L-1}), while tanh parameters were initialized with Xavier initialization that uniformly samples from a range of [-(1/sqrt(M_{L-1})), 1/sqrt(M_{L-1})]. In both cases, M_{L-1} was the number of units from the last layer.

Leaky ReLu solves the vanishing gradient problem (if properly initialized) and is computationally less expensive than the others. It also has the potential for creating dead neurons due to its derivative, however, which can be a desirable or undesirable addition of sparseness depending on the context. Leaky ReLu solves

this with a small constant, and our experiments showed that this layer achieved the best accuracy. Tanh also suffers from the vanishing gradient problem because of the multiplication involved in its derivative, however, our experiments were able to achieve its optimal accuracy with the lowest number of epoques however (see Appendix T3.2 for hyperparameter tuning). Generally, all activation functions achieved similar accuracy.

T3.2 Best Accuracy Achieved and Hyperparameter values for different activation functions			
Layers Type	Accuracy	Learning Rate	Epoques
ReLu	0.83	0.2	550
Tanh	0.81	0.3	250
Leaky ReLu	0.84	0.3	550

3.3 Dropout Regularization

Dropout is a variance reduction technique that helps simulate sparse activation at a given layer, and thus learn a sparse representation that aids MLP's with their enormous number of hyperparameters that can overfit to training data. With a light application of dropout layers [0.2, 0.2] we noticed that this regularization strategy was actually hurting the performance of our model. With some cursory research, we realized that adding dropout to the final layer before our output layer has the potential to cause harm, whereas introducing sparseness immediately before outputting a label was counterproductive. Continuing in our experiments we largely found that dropout provided little improvement to our accuracy, and upon looking at our *training* data accuracy, we could see it was relatively close to our unseen test accuracy. This indicated the model was unlikely to have to overfit to the training data, as we witnessed a fairly low variance, so an additional reduction did not improve our model, and could actually hurt our model if we were adding sparsity where we needed more or better information from training, or our accuracy improvement lay with finding a better local minimum.

In addition, models as small as this one with 128 hidden units and 2 hidden layers have a relatively low expressive power, and lower propensity to overfit compared to much deeper and denser models, so the capacity of dropouts to improve accuracy by introducing sparseness that helps the model generalize better by providing overfitting is limited, compared to models of much higher complexity and more propensity for high variance. We did note that with wider models beyond 2 hidden units layers, dropout improved accuracy. It is worth noting that despite, at times, dropping *half* of our hidden units, we did not substantially harm the accuracy, displaying how when utilized correctly regularization can be a powerful tool for building better generalizing models with sparser representations.

Dropouts Results for 2 Layer ReLu Network with 128 Hidden Units				
Hidden Unit Dropped Percentages per Layer	Learning Rate	Epoques	Accuracy with Dropout	Accuracy without dropout
[0.2, 0.2]	0.3	150	0.7269	0.7088
[0.2, 0.2]	0.3	180	0.7156	0.7875
[0.3, 0.3]	0.3	180	0.7308	0.7875
[0.2, 0.2]	0.3	200	0.7625	0.7728
[0.3, 0]	0.3	180	0.7627	0.7875
[0.7,0]	0.3	180	0.75	0.7875
[0.5, 0]	0.2	550	0.8163	0.8301
[0.7, 0]	0.2	550	0.8226	0.8301
[0.7, 0]	0.3	180	0.7517	0.7875

3.4 Unnormalized Data

As we expected, training the model with the unnormalized data resulted in poor performance. The accuracy of the model drops dramatically to around 0.1. MLP learns with certain learning rates, having pixels

(features) with different distributions may cause oscillation. Also, by making input data follow the same direction, the network converges faster. (Results in Appendix T3.4.)

3.5 CNN

A CNN model is created using an existing library with TensorFlow. The model in this project is constructed by 2 convolution layers with a pooling layer, followed by a fully connected layer with 128 units, and a softmax/relu layer. The categorical_crossentropy loss function is used for our multi-class classification problem. The traditional gradient descent optimizer with learning rates = 0.1 and momentum = 0.9 is used. The model was trained with 10 epochs with a batch size of 32.

Firstly, we set the activations in all of the layers to be Relu. The best result we got is a model accuracy of 23.7%.

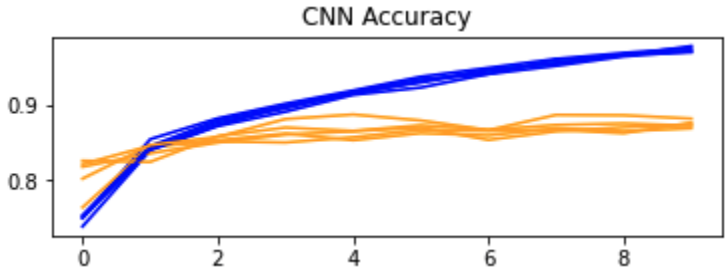

Figure 3.1 CNN Accuracy Training/Validation vs Epoch

Then, we used softmax for our output layer while keeping Relu as our activation in all of the other layers. We did get better performance with around 88.25% accuracy. (The comparison of the result can be found in T3.5 in the appendix). The training accuracy (blue line) and testing accuracy (orange line) vs numbers of epochs can be found in figure 3.1. The graph indicates overfitting after 2 epochs.

CNN performs a bit better than MLP. Firstly, CNN converges faster and is deeper than the MLP in this project. Furthermore, CNN is well known ‘go-to’ method when the inputs include images because of its ability to find spacial dependencies in data.

3.6 Best Model

Type of Hidden Layers	Number of Hidden Layers	Dropout Keep Percentages	Hidden Unit Number	Learning Rate	Epoques	Accuracy
Relu	3	[0.5, 0.5, 0]	200	0.3	300	0.6678
Relu	3	[0.5, 0.5, 0]	200	0.3	1000	0.8322
Relu	3	[0.9, 0.6, 0]	150	0.2	1300	0.8191
Relu	3	[0.8, 0.5, 0]	300	0.15	1000	0.8105
tanh	3	[0.8, 0.4, 0]	300	0.3	1000	0.6218
Leaky Relu	1	[0.9]	400	0.2	600	0.8407
LeakyRelu	3	[0.8, 0.4, 0]	200	0.2	1000	0.8146
LeakyRelu	3	[0.8, 0.4, 0]	200	0.3	1000	0.8274
LeakyRelu	4	[0.6, 0.3, 0.3, 0]	300	0.3	1000	0.7829
tanh, LeakyRelu, Relu	3	[0.8, 0.4, 0]	300	0.3	1000	0.8409

We found that a Neural Network with 3 hidden layers deploying one of each activation functions was a marginal improvement. After attempting to give the network more expressive capacity by increasing its available number of parameters to learn, we still found that the performance gain was only marginally superior to softmax regression. Increasing network depth and width provided some capacity gains but quickly plateaued, ultimately increasing the width beyond the original 128 hidden units did not improve our accuracy. In addition, we required dropout regularization, and more epoques to train our models, so with regards to

accuracy and efficiency, the 2 Layer Leaky ReLu in Task 3.2 or the wider version with 400 hidden units actually offered comparable accuracy for significantly less complexity.

3.7 CNN & MLP Training and Epoques

As our accuracy (see Appendix 3.7) begins to stagnate around a certain number of iterations, we can discern that our model is likely not overfitting, but in fact may have an overly large learning rate, overstepping a local minimum that would otherwise improve it. This could be ameliorated with an adaptive learning rate.

Discussions and Conclusions

In evaluating the effect of depth on this layer, we found that adding adaptive bases to allow the model to learn a nonlinear function (logistic regression) significantly improved prediction on this training set. Though we did not find that adding more depth ReLu layers made as substantial an improvement as that first leap of nonlinearity. This may also be due to the fact that we did not find an optimal local minimum, because our step size was too big, and future investigations would benefit from an adaptive step size.

Experimenting with different activation functions in Task3.2, we found that different activation functions achieved similar accuracy, though required vastly different hyperparameters. We successfully avoided the activation outputs from exploding or vanishing by initializing weight parameters based on the layer activation type. We saw that dropouts may be helpful for regularizing a model, and saw model performance increase in larger models with more parameters (depth and width), however, we did not see significant performance gains for our relatively small models. The image augmentation provided us with more data that helped reduce the variance of our model, and in turn, improve performance.

The maximal accuracy we achieved after extensive combinations of network architectures was 84%. As we witnessed a stagnation of accuracies at certain epoque points, it is possible that the step size remained too large to find a true local minimum, which in future experiments could be achieved by further experimentation and using an adaptive learning rate. However, given CNN's improved performance that broke this plateau, it is likely that it is its unique contribution of a convolution layer that ultimately breaks our performance ceiling from an MLP.

Statement of Contributions

Alicia Nguyen created the MLP and conducted the depth, width, dropout and optimal model tasks. Lei Zhao did the data loading, vectorization, normalization, and augmentation. Also did the CNN. And helped with report drafting. Cristina Penadillo, helped with the report.

Appendix

T3.1 Hyperparameter comparisons for different depth layers

0 Layer MLP			1 Layer MLP			2 Layer MLP ReLu		
Learn Rate	Epoques	Accuracy	Learn Rate	Epoques	Accuracy	Learn Rate	Epoques	Accuracy
0.1	50	0.7158	0.01	60	0.6602	0.01	100	0.3365
0.1	70	0.7176	0.1	250	0.7988	0.1	100	0.62
0.1	100	0.7424	0.2	60	0.7313	0.2	150	0.7088
0.1	150	0.7639	0.2	80	0.7474	0.2	450	0.8265
0.1	250	0.7881	0.2	90	0.7404	0.2	550	0.8301
0.2	40	0.6908	0.2	90	0.7586	0.3	150	0.7844
0.2	50	0.7023	0.2	350	0.8189	0.3	180	0.7875
0.2	60	0.4729	0.2	400	0.8325	0.3	200	0.7625
0.2	50	0.698	0.2	400	0.831	0.3	250	0.806
0.2	55	0.7105	0.2	500	0.835	0.3	350	0.829

T3.2 Hyperparameter comparisons for different activation functions (2 Layers)								
ReLu			tanh			Leaky Relu		
Learning Rate	Epoques	Accuracy	Learning Rate	Epoques	Accuracy	Learning Rate	Epoques	Accuracy
0.01	100	0.3365	0.01	100	0.2567	0.01	100	0.3994
0.1	100	0.62	0.1	100	0.6641	0.1	100	0.6195
0.2	150	0.7088	0.2	150	0.7588	0.2	150	0.7576
0.2	450	0.8265	0.3	180	0.7808	0.3	180	0.7885
0.2	550	0.8301	0.3	190	0.7648	0.3	200	0.763
0.3	150	0.7844	0.3	250	0.8117	0.3	250	0.8116
0.3	200	0.7625	0.3	350	0.7989	0.3	350	0.803
0.3	250	0.806	0.3	450	0.8143	0.3	450	0.8178
0.3	350	0.829	0.3	550	0.828	0.3	550	0.8422

T3.4 Hyperparameter comparisons for 2 Hidden Layers, Relu, with unnormalized images		
Learn Rate	Epoques	Accuracy
0.02	150	0.1
0.2	150	0.1
0.5	150	0.1

T3.5 CNN Performance		
Output Layer	Data	Accuracy
Relu	Original data	0.2375
Softmax	Original data	0.8745
Softmax	Augmentation	0.9105

Bibliography

Bhatt, Preeti P and Isha Patel (2018). “OPTICAL CHARACTER RECOGNITION USING DEEP LEARNING – A TECHNICAL REVIEW”. en. In: p. 13.

LeCun, Yann (1995). “Comparison Of Learning Algorithms For Handwritten Digit Recognition”. en. In: p. 10.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(11).

CS231n Convolutional Neural Networks for Visual Recognition Course Website (https://cs231n.github.io/neural-networks-2/#reg)

Weight Initialization Techniques in Neural Networks. Blog by Saurabh Yadav (https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78)

Weight Initialization for Deep Learning Neural Networks. In: Machine Learning Mastery website. Post by Jason Brownlee (https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/)

How to implement a neural network (4/5) - vectorization of operations. In website: https://peterroelants.github.io/posts/neural-network-implementation-part04/

Activation Functions. GitHub post in: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html