Project 2 Report

Neer Patel (Miner2: npatel49, 90%)

***Abstract*—An Artificial Neural Network was developed to classify images into one of ten categories: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. The model was trained and tested while varying parameters such as its loss function, activation functions, optimizer, number of layers, nodes in each layer, and number of epochs. The model performed well and obtained a 90% accuracy when tested with the actual classifications for the images.**

***Index Terms*—ANN, Image Classification, Machine Learning, Categorical Data, Artificial Neural Network**

I. Introduction

The purpose of this project was to classify images into one of ten categories: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. These categories were represented as integers ranging from zero to nine, respectively. The data provided consisted of three files. The first file had a $50,000 \times 784$ table ("Xall.csv"), where each row represents an image and each column represents a pixel's value in the image; the second file had the correct category classification for each row, this was a $50,000 \times 1$ table ("Yall.csv"). The classification was to be done using an Artificial Neural Network (ANN) on the third file ("Xdeploy.csv"), after having trained the network.

II. Results and Discussion

A. Data Preprocessing

In order to properly feed the data into the ANN, the provided data files had to be preprocessed. The "Xall.csv" file consisted of rows that held string values for each of the pixels; in order to use this data in the ANN, the strings needed to be converted to integers and properly formatted as a $50,000 \times 784$ data frame. The processing for "Yall.csv" was more involved. Since there are ten possible categories, each row of the table was broken into an array of length ten; a one was placed in the index corresponding to the correct category, and all other indices had the value zero. For example, assume that a row in "Yall.csv" has the value 2, then this row would be converted into $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$. This was done to help improve the ANN's classifications. The last layer (output layer) of the ANN will consist of ten nodes, each node will correspond to an index, and the index of the maximum value will be chosen as the classification. Note that the largest value represents the best match, and the index represents the category, so the index of the max value will be the proper classification for the row. After preprocessing this data, the number of layers and how many nodes to place in each layer were determined.

B. Layers

Due to how the data was preprocessed, it is clear that the last layer will consist of ten nodes – one for each category – but the number of layers and the nodes in each are yet to be discussed. The number of nodes for each layer was determined using the provided data; since the images are represented with 784 pixels, it was decided that the first layer should have 784 nodes. This is because each node in the first layer acts as an input. To determine the number of nodes from the first layer to the last layer, assume that there are n layers. The idea was to break 784 into $n-1$ equal compartments; we have $n-1$ because the last layer must have ten nodes. To elaborate, let $x = \frac{784}{n-1}$, then the second layer would have $784 - x$ nodes, the third layer would have $784 - 2x$, the fourth would have $784 - 3x$, and so on.

Now that there is a method to determine the number of nodes in each layer, the number of layers can be discussed. Instead of an analytical approach to answering this, trial and error was used. Below is a table representing the number of layers and the corresponding accuracy observed for each layer; all other parameters were fixed, and only the number of layers was changed during these trials. The parameters were fixed as three epochs, LeakyReLU as the activation function for all layers except the last one, SoftMax as the activation function for the last layer, Adam as the optimizer, and Categorical Cross Entropy as the loss function.

Layers	Accuracy
2	0.76
4	0.75
6	0.68
8	0.74
10	**0.82**
11	0.79
12	0.8

As can be observed from the table, it was determined that ten layers would work the best. Therefore, ten layers were chosen for the ANN, and the number of nodes in each layer followed the convention described earlier. The next step was to figure out a good number of epochs.

C. Epochs

The number of layers and number of epochs are dependent on one another to determine the final accuracy. Even though ten layers was chosen, it could be possible that few layers and more epochs would produce a higher accuracy. In order to test this, epochs of 3, 4, 5, and 6 were chosen for both 10 layers and 6 layers. The table below displays the results of these trials, marking the accuracy for each epoch and layer combination.

Epochs	Accuracy	Layers
3	0.79	10
4	**0.80**	**10**
5	0.67	10
6	0.70	10
3	0.73	6
4	0.67	6
5	0.75	6
6	0.68	6

From the table, one can notice that more epochs does not always mean better accuracy – at least for this combination of parameters. Further, 6 layers still performed worse than 10, regardless of the influence of the number of epochs. This helped confirm the choice of ten layers, and established four to be a good number of epochs for future experiments.

D. Loss Function

The next parameter that was tuned was the choice of the loss function. The loss function is what determines how well the training is going. There are some loss functions that are provided within the TensorFlow package, and these were selected and experimented with. The other parameters were fixed, and the table below displays the results.

Loss Function	Accuracy
Categorical Cross Entropy	**0.82**
Categorical Hinge	0.1
Mean Squared Error	0.1
Mean Absolute Error	0.1
KLDivergence	0.1

As can be observed, there is some weird behavior that is causing the other loss functions to perform significantly worse. A possible reason for this could be the way that the data was preprocessed; the format for the output and "Ytrain" may not be compatible with the other loss functions. Another explanation might be that the choice of the activation function in the layers or the optimizer are causing the low accuracy for the other loss functions. The assumption was made that the loss function behaves independently from the other parameters, so the weird behavior with the other loss functions was likely due to how the data was preprocessed. Due to the significantly worse performance of the other loss functions, Categorical Cross Entropy was chosen – it best fits how the data was preprocessed.

E. Optimizer

Then, the best optimizer for the ANN was selected; TensorFlow has a large list of optimizers to choose from. Through experimentation, the optimizer that has the highest accuracy with the parameters selected so far was chosen. Below is a table listing the optimizers and their accuracies. Not all of the parameters were kept the same for this, it was discovered that these optimizers behaved differently as the number of epochs changed.

Optimizer	Accuracy	Epochs	Observations
Adam	0.82	4	
Adadelta	0.82	10	Better with more Epochs
Adagrad	**0.85**	**4**	**Better with more Epochs**
Nadam	0.75	2	Worse with more Epochs
RMSprop	0.29	4	
SGD	0.1	4	

From the table, it can be observed that some of the optimizers performed very poorly – RMSprop and SGD specifically. This may be due to the image classification problem not being suited for these optimizers or how the data was preprocessed. On the other hand, there were optimizers that performed extremely well, and continued to perform better with more epochs. Out of these, Adagrad performed the best, and with fewer epochs, too. Therefore, Adagrad was selected as the optimizer for the ANN.

F. Activation Function

After selecting the optimizer, the activation function was experimented with. The default activation function in the other trials had been LeakyReLU for all of the layers except the last one, the last layer used SoftMax as its activation function. This structure indicated that the last layer may require a different activation function than the previous layers, so the experiments were done accordingly. The table below displays the results from the trials with different activation functions from the TensorFlow library.

Activation	Accuracy	Last Layer
relu	0.85	softmax
sigmoid	0.1	softmax
softmax	0.1	All Layers
softplus	**0.86**	**All Layers**
softsign	0.75	softplus
tanh	0.78	softplus
selu	0.84	softplus
elu	0.85	softplus
exponential	0.1	softplus
relu	0.85	softplus

As can be observed, using SoftPlus for all of the layers performed the best. Some experiments were ran where other functions were used as the last layer, but it was quickly observed that they produced very low accuracies – the two best activation functions for the last layer were SoftPlus and SoftMax. Out of these, SoftPlus produced slightly higher accuracies as the last layer for many of the other functions. This difference was trivial, as can be observed by ReLU having the same accuracy for SoftMax and SoftPlus as its last layer's activation function. The reason SoftPlus was favored is because SoftMax has an axis parameter that it focuses on; with how the data was preprocessed, all of the axes should be considered equally. Therefore, SoftPlus was chosen as the activation function for all of the layers. This was later changed when all of the parameters were considered together.

G. All Together

The previous experiments indicated that the parameters were not entirely independent of one another. Although it is difficult to figure out exactly how and how much each influences the other, there are some observations that stand out. The main being the number of epochs. As the optimizer and activation functions change, it is clear that the number of epochs to obtain the best accuracy also changes. Thus, some more trials were ran with varying epochs; note that the epochs are larger this time because the Adagrad optimizer was observed to behave better over more epochs. The table below depicts the results from the trials.

Loss	Optimizer	Layers	Epochs	Activation	Accuracy
Categorical Cross Entropy	Adagrad	10	15	softplus	0.871
Categorical Cross Entropy	**Adagrad**	**10**	**15**	**relu + softplus**	**0.876**
Categorical Cross Entropy	Adagrad	10	20	relu + softplus	0.874
Categorical Cross Entropy	Adagrad	10	15	elu + softplus	0.874

One thing to note is that the activation layers influenced the time it took for each epoch to run; ReLU was the fastest and ELU was the slowest – this is why only ReLU was tried with 20 epochs. All of them produced very similar accuracies, and it was confirmed that increasing the epochs for this optimizer resulted in better accuracy. However, there was a limit, since 20 epochs did not improve the result compared to 15 epochs. Considering the runtime, accuracy, and Miner2 results for each of the "predict.csv" files these parameters generated, ReLU + SoftPlus was chosen as the activation layers with 15 epochs.

III. Conclusion

The ANN achieved a 90% accuracy for the "predict.csv" file that it generated when compared with the actual classifications on Miner2. After fine-tuning many of the parameters, the ANN was able to classify many of the images correctly. However, there are many areas that the model may still be improved. The loss function is extremely important when it comes to training the network; instead of defaulting to Categorical Cross Entropy, a custom loss function may be developed and used. One that fits how the data was preprocessed and better handles the image classification problem at hand. Another area of improvement would be to use more data. With a larger data pool, more samples can be used for training while still having a large enough testing data to work with. Lastly, instead of an ANN, a Convolutional Neural Network, Recurrent Neural Network, or other model might be better equipped to handle the image classification problem.