NEURAL NETWORK FOR FASHION MNIST DATASET

By

MUHAMMAD ALI

Submitted to

The University of Liverpool

MSc Artificial Intelligence

CSCK506: Deep Learning

Word Count: 520

14/02/2022

NEURAL NETWORK FOR FASHION MNIST DATASET

Submitted to

The University of Liverpool

Word Count: 520

14/02/2022

TABLE OF CONTENTS

LIST OF FIGURES

Page

Chapter 1. INTRODUCTION

This report details the development of a software which implements a convolutional neural network (CNN) for classification of images in Fashion MNIST dataset (training set of 60,000 images and test set of 10,000 images) provided by Kaggle platform. Primary objective is to provide a comparative analysis when different activation functions and parameters are employed.

Chapter 2. SOFTWARE TOPOLOGY & RESULTS

This section will discuss the software in detail followed by discussion of results. Google's Colaboratory notebook (Jupyter notebook running in cloud) is used to write and execute the software. The notebook can be accessed with the following link:

https://colab.research.google.com/drive/1NBwm9eARucO1wnrvUymM4rG7YePxGqcB?usp=sharing

In **Part one,** the dataframes are converted in to numpy arrays of float32 type in order to conform to keras and tensorflow requirements. x_train/x_test contains all the rows and columns except the label column, whereas y_train/y_test contains all the rows and first column only. After scaling pixel data from 0-255 to 0-1, training data is further split in to actual training data and validation data as seen in figure 1.

```
[12] x_train,x_validate,y_train,y_validate = train_test_split(x_train, y_train, test_size = 0.2, random_state = 648)
```

Figure 1: Training data split.

Part two involves creating a two hidden layer fully connected neural network which utlises Stochastic gradient descent (SGD) optimiser, sigmoid activation function and cross entropy error function. Both (Anon., 2020) and (Kadam, et al., 2020) argue that CNN are excellent performers in image classification segment due to superior accuracy. This report utilises CNN, achieving optimum performance at a learning rate of 0.2 after trialing values between 0.15 to 0.45 (Ali, 2016) (Chauhan, et al., 2018). Figure 2 shows the result of

using sigmoid activation function on training and test data (see notebook for detailed data).

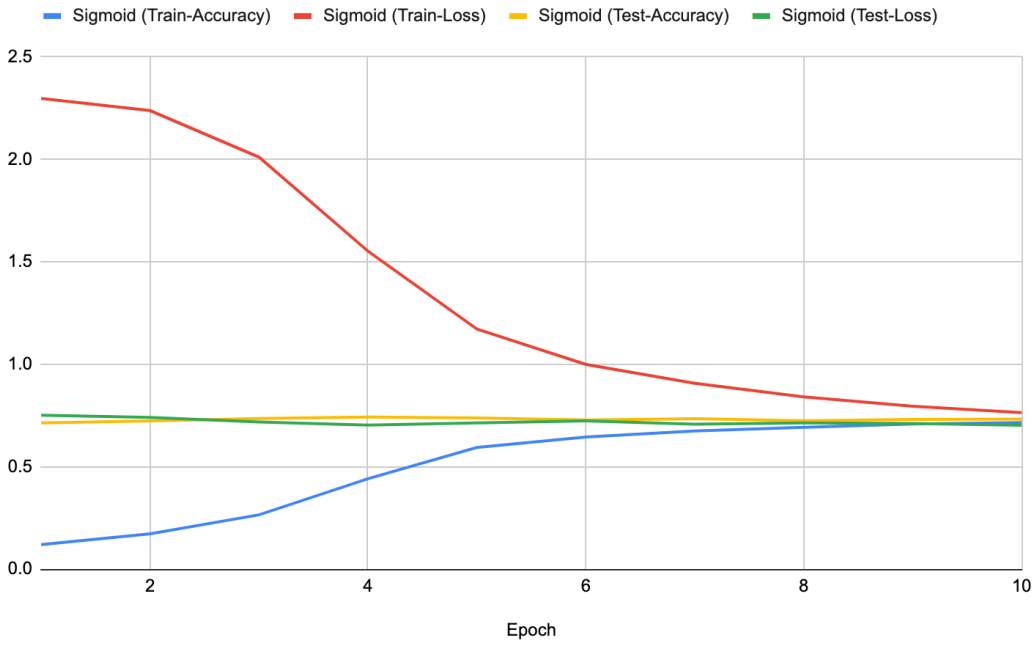

Figure 2: Accuracy and loss across training and test data.

Part three involves swapping sigmoid function with ReLU and introducing dropout regularisation.

```
relu_model_test = relu_cnn_model.fit(
    x_test,
    y_test,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    verbose=1,
    validation_data=(x_validate,y_validate),
)
```

```
Epoch 1/10
10/10 [==============================] - 4s 429ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
Epoch 2/10
10/10 [==============================] - 4s 429ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
Epoch 3/10
10/10 [==============================] - 4s 421ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
Epoch 4/10
10/10 [==============================] - 4s 423ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
Epoch 5/10
10/10 [==============================] - 4s 427ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
Epoch 6/10
10/10 [==============================] - 4s 423ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
Epoch 7/10
10/10 [==============================] - 4s 420ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
Epoch 8/10
10/10 [==============================] - 4s 421ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
Epoch 9/10
10/10 [==============================] - 4s 420ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
Epoch 10/10
10/10 [==============================] - 4s 422ms/step - loss: 11.0572 - accuracy: 0.1000 - val_loss: 11.0491 - val_accuracy: 0.1007
```

Figure 3: Performance with ReLU function on test data.

Figure 3 shows the results achieved by model on each epoch whereas figure 4 shows the result of ReLU with dropout regularization.

```
dropout_model_test = dropout_cnn_model.fit(
    x_test,
    y_test,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    verbose=1,
    validation_data=(x_validate,y_validate),
)
```

```
Epoch 1/10
10/10 [==============================] - 5s 472ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
Epoch 2/10
10/10 [==============================] - 5s 466ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
Epoch 3/10
10/10 [==============================] - 5s 471ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
Epoch 4/10
10/10 [==============================] - 5s 468ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
Epoch 5/10
10/10 [==============================] - 5s 470ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
Epoch 6/10
10/10 [==============================] - 5s 473ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
Epoch 7/10
10/10 [==============================] - 5s 469ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
Epoch 8/10
10/10 [==============================] - 5s 473ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
Epoch 9/10
10/10 [==============================] - 5s 469ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
Epoch 10/10
10/10 [==============================] - 5s 469ms/step - loss: 5.3031 - accuracy: 0.1000 - val_loss: 5.3272 - val_accuracy: 0.0995
```

Figure 4: Performance with dropout.

Part four introduces softmax activation function in the final layer of the CNN; figure 5 and 6 shows the result.

```
relu_softmax_model_test = relu_softmax_cnn_model.fit(
    x_test,
    y_test,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    verbose=1,
    validation_data=(x_validate,y_validate),
)
```

```
Epoch 1/10
10/10 [==============================] - 5s 481ms/step - loss: 0.3988 - accuracy: 0.8547 - val_loss: 0.3779 - val_accuracy: 0.8637
Epoch 2/10
10/10 [==============================] - 5s 476ms/step - loss: 0.3949 - accuracy: 0.8533 - val_loss: 0.3829 - val_accuracy: 0.8627
Epoch 3/10
10/10 [==============================] - 5s 474ms/step - loss: 0.3908 - accuracy: 0.8544 - val_loss: 0.3808 - val_accuracy: 0.8648
Epoch 4/10
10/10 [==============================] - 5s 473ms/step - loss: 0.3714 - accuracy: 0.8662 - val_loss: 0.3924 - val_accuracy: 0.8540
Epoch 5/10
10/10 [==============================] - 5s 479ms/step - loss: 0.3690 - accuracy: 0.8702 - val_loss: 0.3743 - val_accuracy: 0.8656
Epoch 6/10
10/10 [==============================] - 5s 482ms/step - loss: 0.3671 - accuracy: 0.8671 - val_loss: 0.4036 - val_accuracy: 0.8526
Epoch 7/10
10/10 [==============================] - 5s 482ms/step - loss: 0.3704 - accuracy: 0.8695 - val_loss: 0.3893 - val_accuracy: 0.8562
Epoch 8/10
10/10 [==============================] - 5s 476ms/step - loss: 0.3685 - accuracy: 0.8658 - val_loss: 0.3704 - val_accuracy: 0.8655
Epoch 9/10
10/10 [==============================] - 5s 478ms/step - loss: 0.3623 - accuracy: 0.8700 - val_loss: 0.3697 - val_accuracy: 0.8673
Epoch 10/10
10/10 [==============================] - 5s 476ms/step - loss: 0.3582 - accuracy: 0.8718 - val_loss: 0.3816 - val_accuracy: 0.8602
```

Figure 5: Performance of ReLU + Softmax.

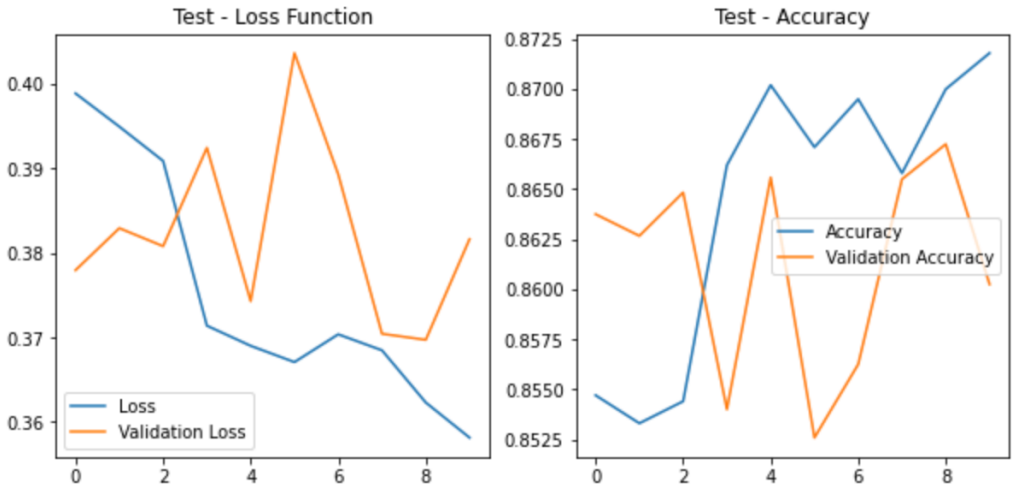

Figure 6: Accuracy and loss comparison on test data.

In term of results, ReLU + Softmax combination achieves the highest performance followed by sigmoid function and then ReLU function; the dying ReLU problem (always outputting same value) (Mastromichalakis, 2021) leads to network getting stuck in local minimum as seen in figure 3 and 4. Dropout (ignoring randomly selected neurons) (Mastromichalakis, 2021) halves the loss function as seen in figure 4 in comparison with figure 3. Sigmoid function is suited towards binary classification where labels are class values (Anon., 2020), although its often avoided in hidden layers due to problem of vanishing gradient (Chauhan, et al., 2018).

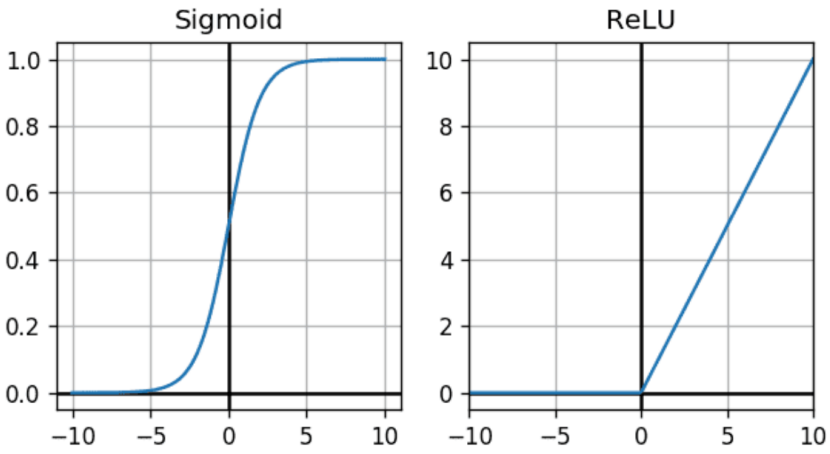

Figure 7: Differences between functions (Thakur, 2019).

Chapter 3. CONCLUSIONS

The author argues that selection of activation function (AF) is closely tied with domain of problem and significantly affects performance; Softmax excels in multiclass classification (domain of this report) through output normalisation in final layer (Mastromichalakis, 2021), hence combination of ReLU and Softmax yeilds optimum performance by addressing individual shortcomings.

REFERENCES

Ali, M., 2016. *High Performance Computing for High Fidelity Numerical Analysis,* Liverpool: https://github.com/EccentricEngineer/Stock-Market-Prediction-Software/blob/main/final-year-report_Muhammad_Ali.pdf.

Kadam, S. S., Adamuthe, A. C. & Patel, A. B., 2020. CNN Model for Image Classification on MNIST and Fashion-MNIST Dataset. *Journal of Scinetific Research,* 64(2).

Anon., 2020. *Classification of Garments from Fashion MNIST Dataset Using CNN LeNet-5 Architecture.* Aswan, 2020 International Conference on Innovative Trends in Communication and Computer Engineering.

Chauhan, R., Ghanshala, K. K. & Joshi, R. C., 2018. *Convolutional Neural Network (CNN) for Image Detection and Recognition.* s.l., 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC).

Mastromichalakis, S., 2021. SigmoReLU: An Improvement Activation Function by Combining Sigmoid and ReLU. *Preprints*, 09 June.

Thakur, D., 2019. *ML Basics #4: Replace Negatives with Zeros!.* [Online]
Available at: https://dhruvs.space/posts/ml-basics-issue-4/
[Accessed 14 February 2022].