Classification using Python + Keras

1st Rene Jerez
Computer Engineering Master Degree
Universidade da Beira Interior
Covilhã, Portugal
rene.jerez@ubi.pt

***Abstract*—Neural networks, a fundamental concept in machine learning courses, are inspired by the human brain to tackle complex, non-linear tasks. They consist of layered neurons that process data through weighted connections and activation functions to capture patterns. Feed-forward networks like multi-layer perceptrons (MLPs) use input, hidden, and output layers to refine predictions. Convolutional Neural Networks (CNNs) excel in computer vision by identifying spatial hierarchies through convolution and pooling layers, enhancing accuracy with activation functions such as ReLU. Learning through backpropagation enables CNNs to adjust weights and minimize errors, making them invaluable in applications like image recognition and diagnostics.**

***Index Terms*—CNN, Neural Network, ReLU, MLP**

I. Introduction

Regarding Azhary and Ismail [1], Convolutional Neural Networks (CNNs) are a prevalent machine learning algorithm for image classification due to their capacity to capture spatial hierarchies through convolutional layers, making them well-suited to learning complex image patterns and features.
Their study specifically examines the impact of different activation functions on CNN performance, as activation functions play a critical role in overcoming challenges like the vanishing gradient problem, which can hinder deep networks' training and accuracy. By analyzing activation functions such as ReLU, Sigmoid, Tanh, and newer options like Swish and Leaky ReLU, the study aims to identify optimal functions that enhance CNN effectiveness, particularly in deep architectures like ResNet, which are prone to gradient issues due to their complexity.
In this report, I will similarly evaluate three activation functions to determine which yields the best accuracy, following the approach taken by the authors in their analysis.

II. Methodology

To carry out this classification task, we will use the MNIST dataset, a widely adopted benchmark in machine learning. The dataset consists of 60,000 training examples and 10,000 test examples, each representing a handwritten digit from 0 to 9. Each image is 28x28 pixels, with pixel values ranging from 0 (black) to 255 (white). The images are provided in CSV format, with each row containing 785 values: the first value corresponds to the label (digit) and the remaining 784 values represent pixel intensities.

Identify applicable funding agency here. If none, delete this.

Our approach involves developing a feedforward neural network using Python and Keras. This model will be structured to recognize and classify the digit images based on the pixel input values. The model will be constructed with a parameterized topology, allowing for flexible adjustment of the network structure by specifying the number of neurons in each layer. For instance, a topology parameter "53" will represent a network with three layers: an input layer with 5 neurons, a hidden layer with 3 neurons, and an output layer with 10 neurons to represent the 10 digit classes.
Following these setup instructions, the next step involves implementing three different activation functions to evaluate their impact on model accuracy. We will use activation functions such as ReLU, Sigmoid, and Leaky ReLU, assessing which function yields optimal performance. This comparative analysis will provide insight into activation function effectiveness within the feedforward neural network model and guide our selection for best accuracy in digit classification.

A. Data Source

To begin, the setup includes importing essential libraries, such as Keras and TensorFlow, which will support the construction and training of the neural network. Additionally, foundational Python libraries like NumPy and Pandas are imported for data handling, as well as Matplotlib for visualizing results. This setup ensures that all necessary tools are accessible for efficient data processing and model development.

Following the setup, the dataset used is the MNIST dataset, a well-known collection of handwritten digit images, which is loaded directly. The data is divided into training and test sets, with 60,000 images for training and 10,000 images for testing. Each image is represented as a 28x28 grayscale pixel matrix, flattened into 784 features per example. The data is then normalized by scaling pixel values to a range between 0 and 1, optimizing it for neural network training and improving model convergence during learning.

B. Model Implementation

The construction of this classification model follows the resource guidance provided in the project sheet, along with recommendations from relevant tutorials. The model is structured as a simple feedforward neural network, using layers

defined based on both project requirements and suggested configurations for effective image classification. The architecture includes an input layer of 784 neurons (one for each pixel in the flattened 28x28 image), a specified hidden layer configuration, and an output layer with 10 neurons representing the possible digit classes (0-9), employing a softmax activation for probability distribution across classes.

Following these recommendations, six variations of the model will be run to evaluate the effects of different activation functions on classification performance. These configurations involve constructing the model with Sigmoid, ReLU, and Leaky ReLU activation functions, each tested across two model topologies. For each activation function, one configuration includes a single hidden layer, while the other uses a deeper network with multiple hidden layers, allowing for a robust assessment of how these functions impact accuracy and convergence under varying complexities.

This approach was inspired by the work of Azhary and Ismail, who conducted a similar comparative analysis of activation functions for optimizing CNN performance in image classification [1].

C. Evaluation Metrics

The Sigmoid Large model achieved moderate performance with a test accuracy of 0.8733 and a test loss of 0.4822, outperforming the Sigmoid Small model, which had significantly lower accuracy (0.4318) and higher loss (1.7714). This suggests that larger networks are more effective in utilizing the Sigmoid activation for this classification task.

The ReLU and Leaky ReLU models generally performed better than Sigmoid, with the Large configurations for both achieving the highest test accuracies—0.9533 for ReLU and 0.9538 for Leaky ReLU—and the lowest test losses, around 0.159. The Small configurations of ReLU and Leaky ReLU, while lower in accuracy than their large counterparts, still performed better than the Sigmoid models.

Overall, the table indicates that larger model sizes improve accuracy and reduce loss across activation functions, and that ReLU-based activations outperform Sigmoid in this dataset.

Configuration	Train Acc	Test Acc	Train Loss	Test Loss
Sigmoid Large	0.866	0.873	0.505	0.482
Sigmoid Small	0.425	0.431	1.786	1.771
ReLU Large	0.955	0.953	0.157	0.159
ReLU Small	0.793	0.792	0.768	0.751
LeakyReLU Large	0.955	0.953	0.156	0.159
LeakyReLU Small	0.644	0.649	1.108	1.092

TABLE I
PERFORMANCE WITH DIFFERENT ACTIVATION FUNCTIONS AND CONFIGURATIONS.

In the folow chart the Sigmoid Large model shows moderate improvement over the Sigmoid Small model. By around epoch 5, Sigmoid Large begins to outperform Sigmoid Small, with lower test loss and higher accuracy. At the end of training, Sigmoid Large achieves a test accuracy of 0.8733 and a test loss of 0.4822, whereas Sigmoid Small stabilizes at a much

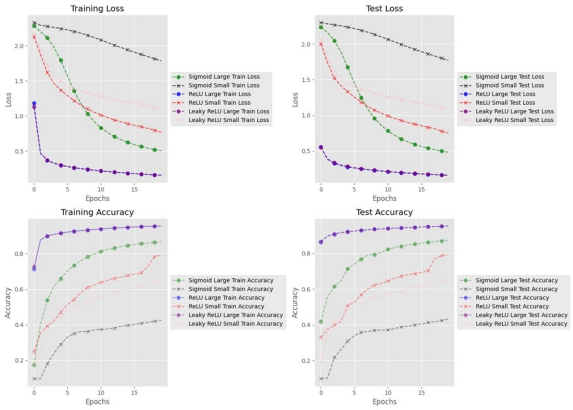

Fig. 1. Results.

lower accuracy of 0.4318 and a higher test loss of 1.7714. This pattern suggests that Sigmoid activation, while improving with larger network sizes, struggles to achieve competitive performance, particularly in smaller configurations.

In contrast, the ReLU models demonstrate faster convergence and higher overall performance. The ReLU Large model begins to pull ahead by epoch 3, achieving test loss values below 1.0 and test accuracy exceeding 0.8. By the final epoch, ReLU Large reaches a test accuracy of 0.9533 with a low test loss around 0.159, indicating effective generalization. Although ReLU Small does not reach the same level of accuracy, it still outperforms both Sigmoid models, with a final test accuracy of 0.7929 and test loss of 0.7514, highlighting ReLU's effectiveness in both large and small networks.

The Leaky ReLU models show similar strengths, closely tracking the performance of the ReLU models. The Leaky ReLU Large model surpasses 0.9 in test accuracy around epoch 5 and reaches a final test accuracy of 0.9538 with a test loss of 0.1598, performing on par with ReLU Large. The Leaky ReLU Small model achieves a test accuracy of 0.6492 and a test loss of 1.0923, outperforming the Sigmoid Small model and even slightly surpassing ReLU Small in the early epochs, suggesting an advantage in handling gradient issues in smaller networks.

III. CONCLUSION

In conclusion, these observations demonstrate that ReLU and Leaky ReLU activations deliver superior performance and faster convergence compared to Sigmoid, especially when paired with larger network configurations.

I made a script consider Leaky which offers a flexible approach to neural network design by allowing users to specify the model's architecture through a simple command-line parameter, enhancing experimentation and optimization for diverse tasks. By dynamically building layers based on user-defined topology, the code provides a streamlined way to test various configurations, empowering users to find the best structure for the MNIST classification challenge.

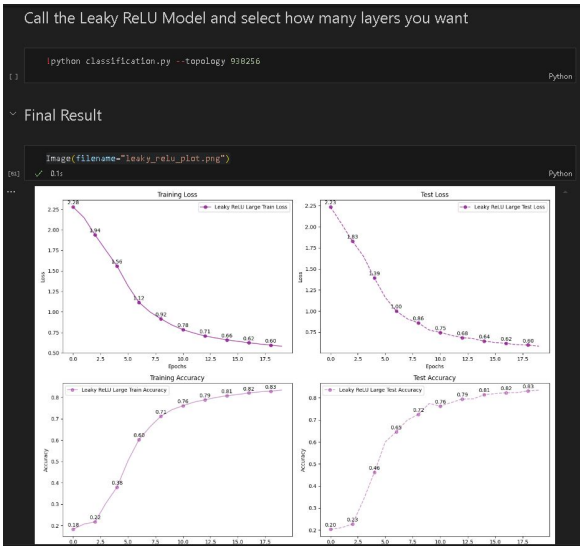

Fig. 2. Metrics.

REFERENCES

[1] M. Z. R. Azhary and A. R. Ismail, "A Comparative Performance of Different Convolutional Neural Network Activation Functions on Image Classification," *International Journal on Perceptive and Cognitive Computing (IJPCC)*, vol. 10, no. 2, pp. 118, Jul. 2024. doi: 10.31436/ijpcc.v10i2.490.