Digit Recognition Model using Neural Networks

A comparison between

Sigmoid and SoftMax as non-linear

activation functions

Software Development Project

Ashwattha Phatak

Second Year Semester II

Vishwakarma Institute of Technology

Electronics and Telecommunications Department

Guide: Rajshree Akolkar

Abstract

A study of the comparison between the efficiency of Sigmoid and Softmax functions when used in the last layer of the Neural Network with all the other variables being similar is what is the aim of this project. Both functions are widely used in prediction models which sometimes have overlapping results with respect to expected outcomes. However, both functions have some ideal applications when comparing their working within the outcomes of their predictions. This can also be an easy introduction to newer programmers and non-mathematicians to get an idea of how the subtle differences between two mathematical functions can affect their prediction models.

About Neural Networks

Neural Networks, Deep Neural Networks and Convolutional Neural Networks have emerged as a means to tackle complex problems such as image classification and speech recognition. This has been made possible due to the huge, accurate and organized databases that have emerged through the years like the CIFAR-10 and CIFAR-100 for images and the MNIST and EMNIST databases for handwritten digits. The enormous speed of calculations of modern computers has also encouraged more computer engineers to learn and play with Neural Networks.

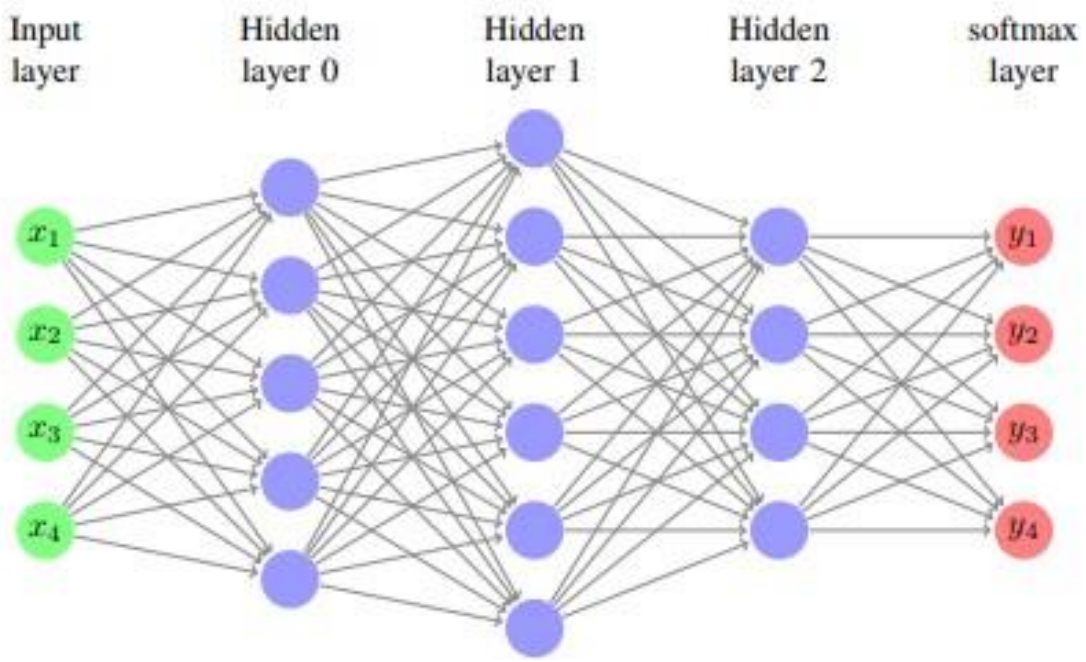


Fig 1.1: A typical Neural Network

A typical CNN consists of multiple layers, each one of which can be either convolutional, pooling, or normalization with the last one to be a non-linear activation function, usually the softmax or the sigmoid function. The hidden “dense” layers of the Neural Network can also contain a non-linear activation function, which in this case is ReLu.

The convolution layer is one of the basic types of layers of a CNN. This layer receives N "nodes” as input and extract M "nodes” as output, where generally it is such that (M<N). The basic operation is multiplication-addition in order to filter the feature map.

An activation layer applies an activation function, tanh, σ or ReLU on each input from the previous stage.

Here in this project, we use something called “normalization” as the initial step to reduce the complexity of calculations for our activation functions in the neural network.

Activation functions:

The ReLu function:

This is what the ReLu activation function, which is used in the hidden(dense) layers looks like:

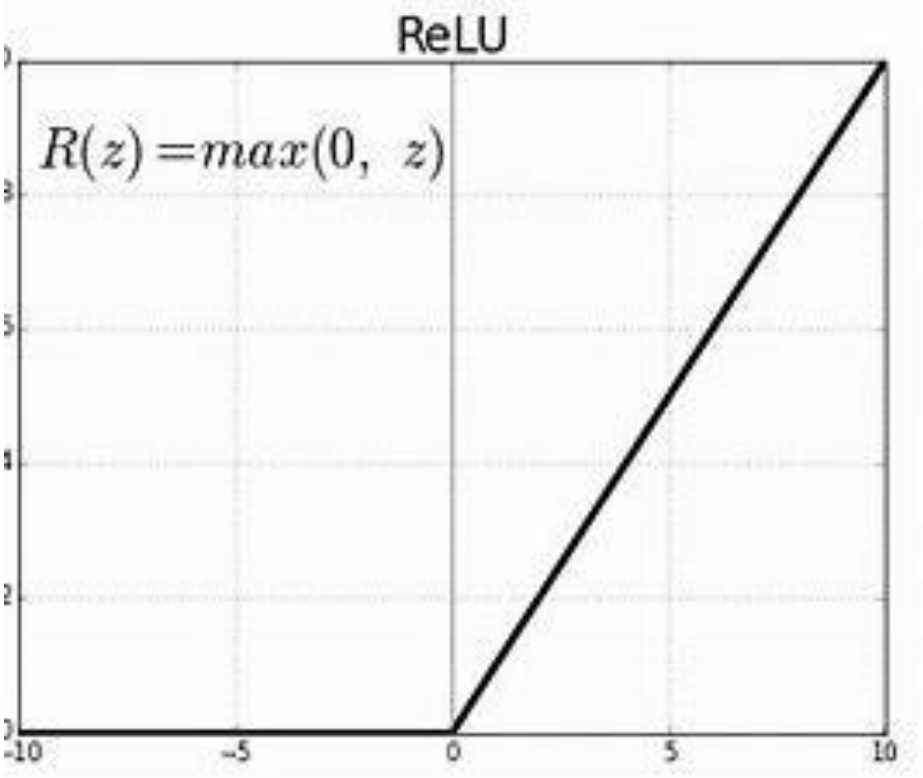


Figure 2.1: ReLu Function

We won't concern ourselves with what this means or how this function exactly works, as this is a topic of conversation for advanced mathematicians and not for new engineers.

The focus of study is on the Sigmoid and Softmax functions.

The Sigmoid Function:

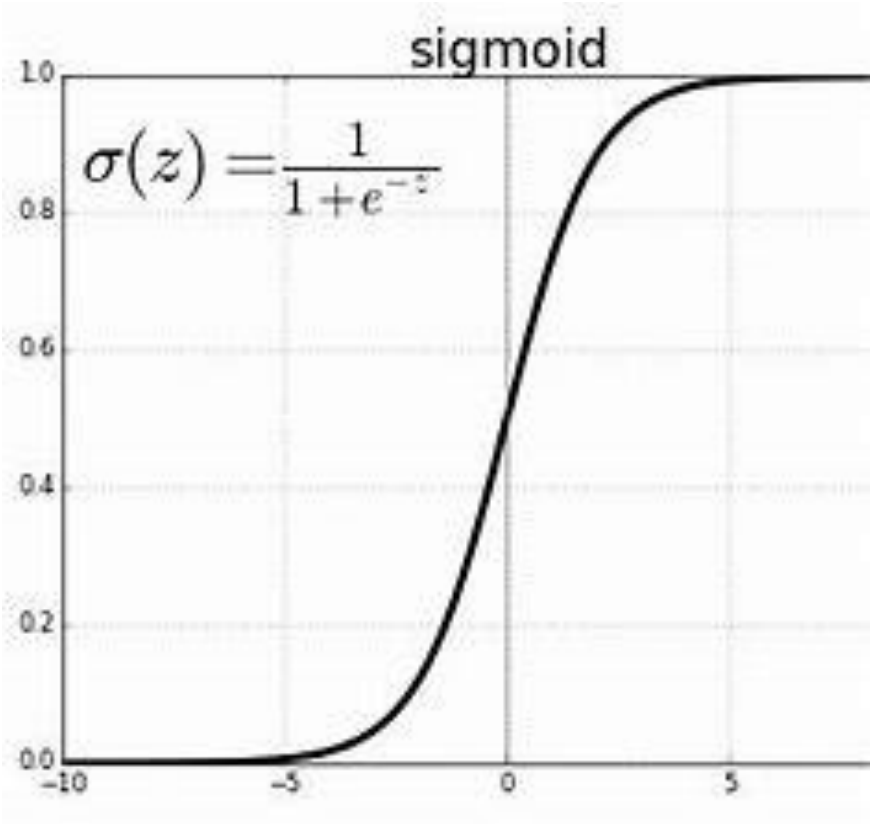


Figure 2.2: The Sigmoid Function

A sigmoid function is any mathematical function which gives a S-shaped curve when plotted on a graph. A common example of a sigmoid function is the logit function which is part of the basics of learning logistical regression in Data Science.

Sigmoid functions normally show a return value y in the range of 0 to 1. Another common range is from –1 to +1.

For this study, the range that is considered by out model is 0 to 1, where the output is the probability of the next node being the best fit.

The Sigmoid function is generally used for binary classifications models implemented on a certain dataset. It indicates the probability of each individual prediction independently; it does not take into account the probability of other predicted outcomes.

Hence, the sum of probabilities of the predictions may or may not go over 1.

The SoftMax function:

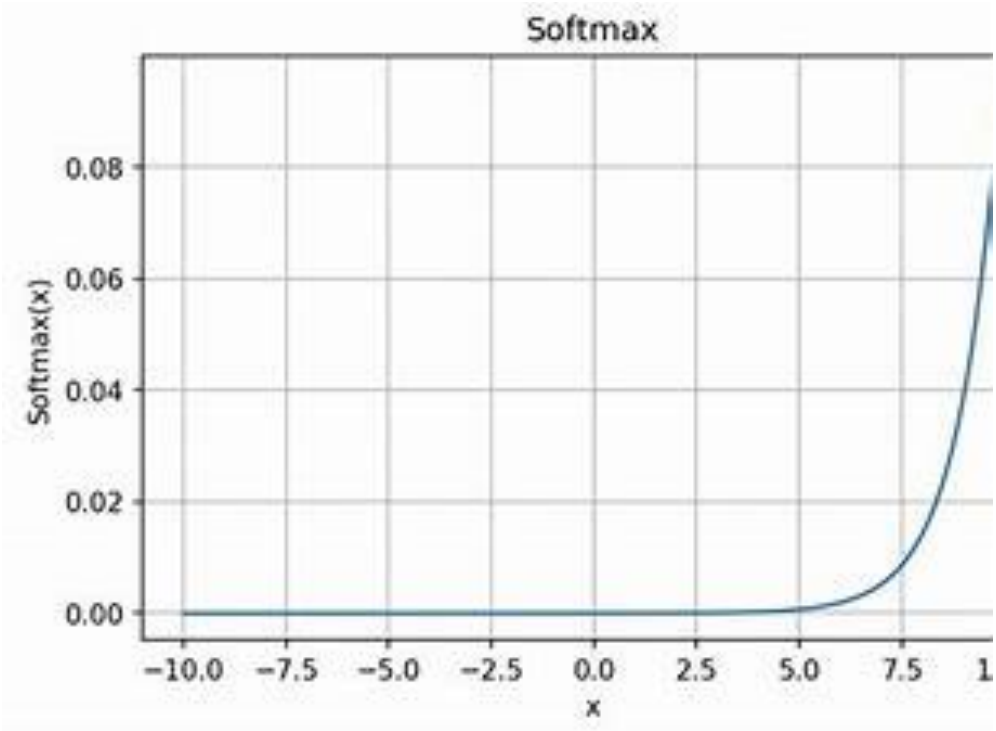


Figure 2.3: The Softmax Function

The softmax function is a generalization of the logistic function to multiple dimensions. It is used in multinomial logistic regression and is often used as the last activation function of a neural network.

After applying softmax, each component will be in the interval and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities.

SoftMax is generally is used for multi-classification problems, which makes it very popular in digit and image recognition models. The softmax function may be called "dependent", as it takes into account

the probabilities of all the predictions, so that the sum of the probabilities is always 1

The subtle difference between a dependent and independent prediction is easy to miss on paper.

However, this is very important when you have a defined expected output, as we will see further.

The Dataset:

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.

This dataset contains 60,000 training(x_train) and 10,000(x_test) testing images of handwritten digits from 0 to 9 of a size of 28x28 pixels.

Every image is having a black background on which there is a unique handwritten digit in white. The measurement value used for every pixel is the grayscale number of that pixel.

The grayscale value ranges from 0 to 255, with 0 being the darkest black and 255 being the brightest white possible.

Normalization of the image:

As mentioned, every pixel has a grayscale value between 0 and 255. However, we need the values to be between 0 and 1 for our Neural Network to give a probability between 0 and 1.

For this, we "normalize" the pixel values by dividing the grayscale values of each pixel by 255.

This step may or may not give better results, but it makes the load on our machine a bit lesser.

```
x_train = x_train/ 255.0,

x_test = x_test/ 255.0
```

Now, the value of every pixel in every training image is between 0 and 1.

Flattening:

Right now, all the image information in (x_train) is in the form of a [28][28], two-dimensional array. We do the flattening process to convert into a one-dimensional array.

This is because in a neural network, we need a one-dimensional dataset for mapping it using our activation function.

Now, the 28x28 array is converted into a single one-dimensional array of 764 array elements or *Nodes.*

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape = (28,28)),
```

This step is very important for mapping our Neural Network

The Hidden Layers:

As mentioned before, we use the ReLu non-linear activation function for mapping between the hidden layers.

There are a total of 5 hidden layers, each one successively downsized from the last layer, and each one using ReLu.

```
tf.keras.layers.Dense(392,activation ='relu'),
    tf.keras.layers.Dense(256,activation ='relu'),
    tf.keras.layers.Dense(128,activation ='relu'),
    tf.keras.layers.Dense(64,activation ='relu'),
    tf.keras.layers.Dense(32,activation ='relu'),
```

So, the length of each layer goes on decreasing from 764 nodes in the flattening layer, to 392>256>128>64>32 in the dense layer, and finally into just the 10 nodes in the last layer.

The last 10 nodes are nothing but the numbers from 0 to 9, and the last layer is where either one of the sigmoid or the softmax activation function is present.

Observations:

While both models gave very impressive results, there was still a small difference between the two models on paper, and a noticeable difference while experimentation.

Sigmoid:

Epochs	Layers	Accuracy
20	4	0.9908
20	5	0.9922
20	6	0.9939
50	6	0.9991

Table 1.1: Sigmoid Observations

Softmax:

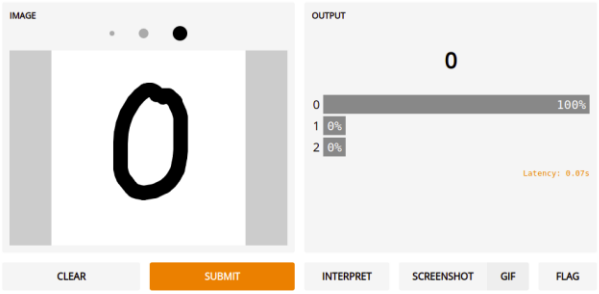
Epochs	Layers	Accuracy
20	4	0.9912
20	5	0.9924
20	6	0.9938
50	6	0.9996

Table 1.2: SoftMax Observations

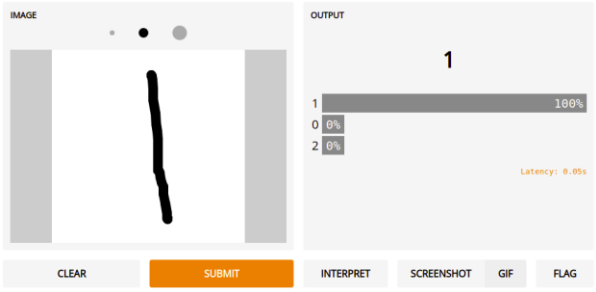
As you can observe in tables 1.1 and 1.2, there is not a significant difference in accuracy with both models giving very high accuracy within the 50-epoch range. This makes both models really good in terms of structure and prediction accuracy. However, some problems were encountered while experimentations.

Sigmoid:

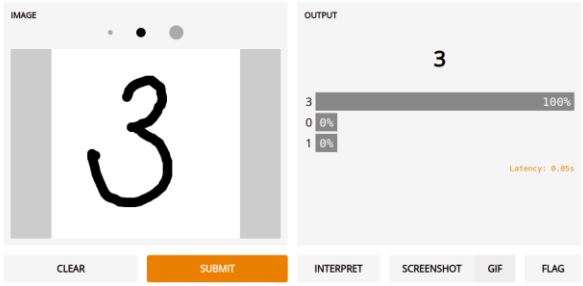
Correct predictions without anomalies:



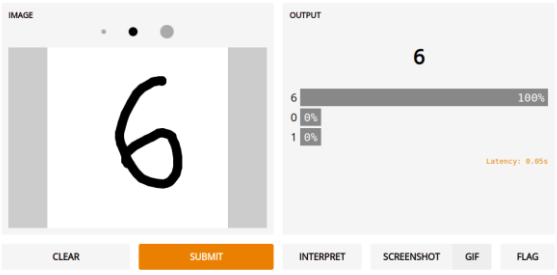
Prediction 1.1



Prediction 1.2

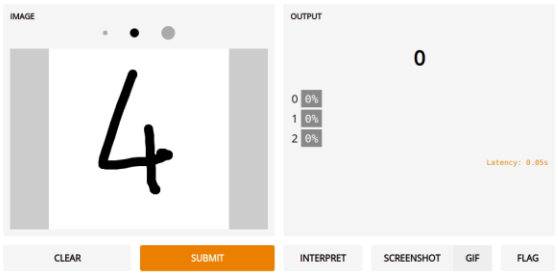


Prediction 1.3

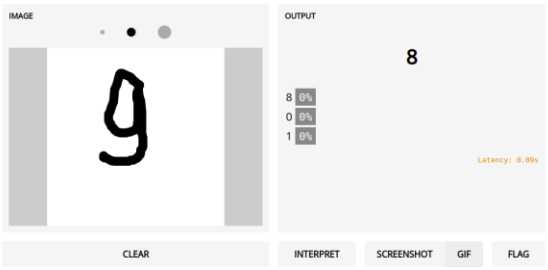


Prediction 1.4

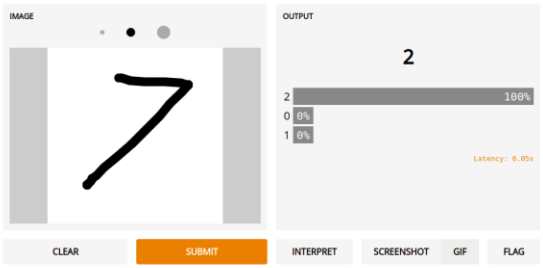
Anomalies:



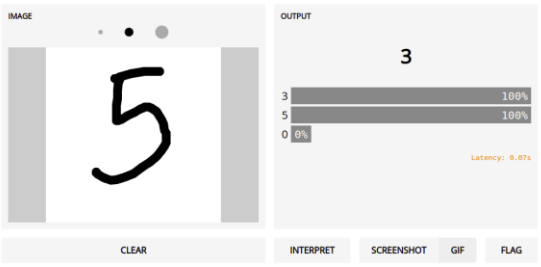
Anomaly 1.1



Anomaly 1.2



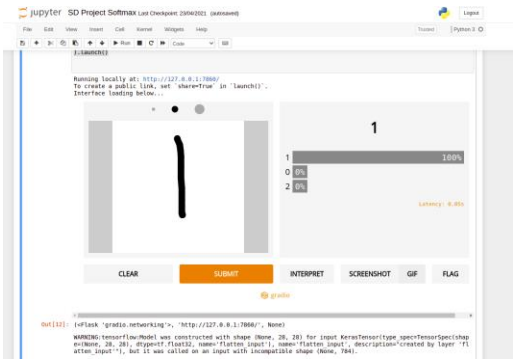
Anomaly 1.3



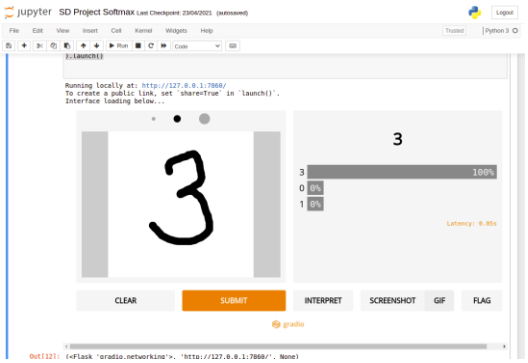
Anomaly 1.4

Softmax:

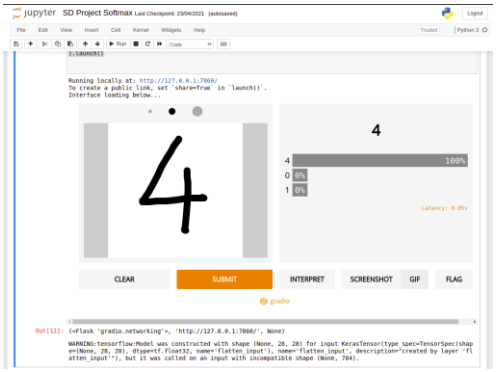
Correct predictions without anomalies:



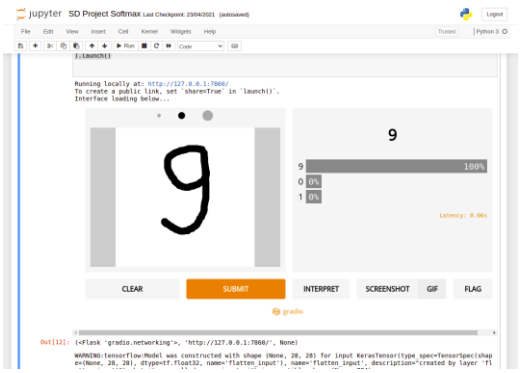
Prediction 2.1



Prediction 2.2

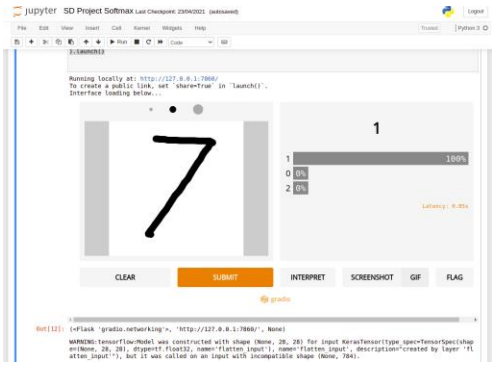


Prediction 2.3



Prediction 2.4

Anomalies:



Prediction 2.1

Conclusions:

Even though both the prediction models offer excellent accuracy with negligible difference on paper, the actual results vary enough to show a substantial difference between the two activation functions.

The nature of the false predictions however, is different.

As the Sigmoid function is an independent prediction model, it sometimes predicts two different numbers with the same probability like with Anomaly 1.4 (this happens especially in the case of the numbers 0,9 and 7). This may be useful is some cases, but only gives a faulty or inaccurate prediction in this model.

The SoftMax function on the other hand is a dependent prediction model, so it will always offer a unique and singular prediction. However, in case of an inaccurate prediction, it will not give overlapping probabilities; the prediction can be completely ruled out (as in Anomaly 2.1) as opposed to the sigmoid prediction (errors occur especially with the numbers 7, 5,6 9 but with much lesser frequency), where one of the two predictions may be accurate.

In conclusion, the best model for this project with regards to consistency of predictions and long-term error correction is the one which employs the SoftMax Activation Function.

References:

1. Kouretas, I., & Paliouras, V. (2019). Simplified Hardware Implementation of the Softmax Activation Function. 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST). doi:10.1109/mocast.2019.8741677
2. M. Y. W. Teow, "Understanding convolutional neural networks using a minimal model for handwritten digit recognition," 2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems (I2CACIS), 2017, pp. 167-172, doi: 10.1109/I2CACIS.2017.8239052.
3. C. L. Tan and A. Jantan, "Digit Recognition Using Neural Networks", MJCS, vol. 17, no. 2, pp. 40–54, Dec. 2004.
4. C. Tsai, Y. Chih, W. H. Wong and C. Lee, "A Hardware-Efficient Sigmoid Function with Adjustable Precision for a Neural Network System," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 62, no. 11, pp. 1073-1077, Nov. 2015, doi: 10.1109/TCSII.2015.2456531.
5. M. M. Abu Ghosh and A. Y. Maghari, "A Comparative Study on Handwriting Digit Recognition Using Neural Networks," 2017 International Conference on Promising Electronic Technologies (ICPET), 2017, pp. 77-81, doi: 10.1109/ICPET.2017.20.