TensorFlow Playground(Activation Function)

BITI 3113 Neural Network Mini Project S1G1 Group C

KOO CHUN CHENG
B032020044 2-BITI S1G1 DE
Fakulti Teknologi Maklumat dan Komunikasi
Universiti Teknikal Malaysia Melaka
Gurun, Malaysia
b032020044@student.utem.edu.my

ABDALLA REDA MAAMOUN ELSAID
B031910450 2-BITI S1G1
Fakulti Teknologi Maklumat dan Komunikasi,
Universiti Teknikal Malaysia Melaka
Melaka, Malaysia
b031910450@student.utem.edu.my

MOHAMMAD TARIQ ALI BIN SALAMUDEEN ALI
B031910049 2-BITI S1G1
Fakulti Teknologi Maklumat dan Komunikasi
Universiti Teknikal Malaysia Melaka
Butterworth, Malaysia
b031910049@student.utem.edu.my

JEYA JASSVINE A/P JEYABALA SUNDRAM
B031910136 2-BITI S1G1
Fakulti Teknologi Maklumat dan Komunikasi,
Universiti Teknikal Malaysia Melaka
Ipoh, Malaysia
b031910136@student.utem.edu.my

***Abstract*—Activation functions are critical in Artificial Neural Networks because they aid in the learning and understanding of non-linear and complex mappings between inputs and outputs. The activation function is the most fundamental component of a convolutional neural network (CNN), and it is responsible for the network's nonlinear transformation capabilities. To acquire distinct nonlinear transformation capabilities, several activation functions make the original input compete with different linear or nonlinear mapping terms.**

Keywords—neural network, activation function, non-linear, mapping.

I. Introduction

In a neural network, an activation function specifies how the weighted sum of the input is turned into an output from a node or nodes in a layer. The activation function is sometimes known as a "transfer function." When an activation function's output range is restricted, it's referred to as a "squashing function." Many activation functions are nonlinear, which is known as "nonlinearity" in the layer or network architecture.

The activation function chosen has a significant impact on the neural network's capabilities and performance, and different activation functions may be utilised in different portions of the model. Although networks are designed to use the same activation function for all nodes in a layer, the activation function is applied within or after the internal processing of each node in the network.

II. Why There Is Need for Activation Function?

A) Learning complex pattern

An activation function is a function that is introduced to an artificial neural network to assist it in learning complex patterns from data. When compared to a neuron-based model seen in our brains, the activation function is responsible for determining what is to be fired to the next neuron at the end of the process.

B) Ability to add on non-linearity

Neural networks with a linear activation function are only effective one layer deep, regardless matter how sophisticated their architecture is, they cannot be linear. In most cases, the input to networks is a linear transformation (input * weight), however the real world and problems are not. We utilise a nonlinear mapping technique called activation function to make the input data nonlinear. A decision-making function that determines the presence of a specific brain characteristic is known as an activation function. It is mapped between 0 and 1, with 0 indicating the lack of the characteristic and 1 indicating its presence. Unfortunately, because the activation values can only accept 0 or 1, small changes in the weights cannot be reflected in the activation values. Nonlinear functions must therefore be continuous and differentiable over this range. A neural network must be able to accept any input from -infinity to +infinity, but it must also be able to map it to an output that is between 0 and 1 in some situations, necessitating the use of an activation function. Because the goal of activation functions in a neural network is to construct a nonlinear decision boundary using nonlinear combinations of weight and inputs, they must be nonlinear.

III. Step by Step Learning and Analysis

For this research project, we are investigating the activation functions which are Linear, Sigmoid, TanH, and ReLU. The fixed parameters that we use are as follows :

- Learning rate = 0.03
  Learning rate is a configurable hyperparameter. The learning rate range is from 0.00001 to 10. It controls how fast the model adapts to the problem. Larger learning rates result in rapid changes and need lesser training epochs.
- Regularization = none
- Regularization rate = 0
  This is used to tune the function by adding an additional penalty term in the error function but in our case, the regularization rate is 0.

- Problem type = Classification
  Problem type can be divided into 2 which is classification and regression. Here we have chosen classification. Classification problems will have outputs that are categorical.

- Ratio of training to test data = 30%
  The dataset is always divided into 2 which is training and testing data. Here the ratio is divided into 30:70.

- Noise = 5
  Noise is distortion in data.

- Batch size = 10
  Batch size is the number of training data points used in one epoch. It usually ranges from 10 to 100.

- Features
  Features which are x1 and x2 maintained as a fixed parameter setting.

Here it is represented in two colors which are blue and orange . Blue is positive and orange shows negative values. In the output layer the dots are classified in blue and orange in an almost circular shape. The intensity of the colors also shows how confident the prediction is but in this diagram the color intensity is not as much.

IV. How Analysis Coneducted

A) In Term of Ability to Solve Non-linear problem

With the fixed setting, try each activation function to solve the classification problem of each dataset. If fixed setting cannot solve the problem, adjust the setting other than the features that is fixed to X_1 and X_2 to solve the problem.

B) In Term of Comparison

When both activation functions can solve the problem with fixed setting, compare them in term of test loss and training loss at certain epoch (with certain dataset classification problem and same hidden layer setting).

V. Content

There are 4 activation function provided in TensorFlow Playground, which are Linear, Sigmoid, TanH, and ReLU.

A) Linear

Linear activation function is a straight-line function, the activation is proportional to input. The following figure shows its graph.

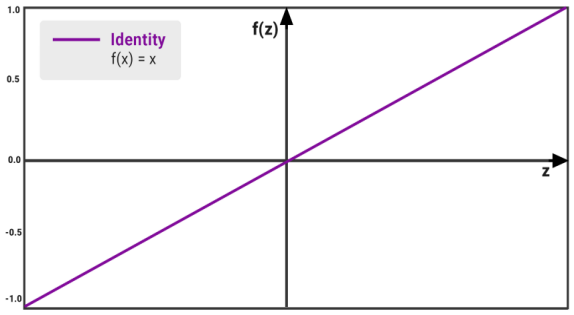

Figure 1: graph of Linear activation function

The equation used to represent linear activation function is:

$$f(x_i) = kx_i$$

Where x_i is the input of the activation function f on the i^{th} channel, and k is a fixed constant, $f(x_i)$ is the corresponding output of the input. So, the derivative with respect to x_i is:

$$f'(x_i) = k$$

If all the layers' activation functions are linear, then the final activation function of the last layer is nothing more than a linear function of the first layer's input, implying that the rest of the network is equivalent to a single layer with linear activation function. Of course, the outputs can be anything on the range [-∞, +∞]. The derivative is k, note k is a constant, so the gradient is a constant, and has no relationship with input, so people cannot decrease the error by gradient.

1) Linear activation Function in Playground (tensorflow)

Playground (Tensorflow) is used to study the effect of linear activation function with constant values for other parameters:

- Learning Rate = 0.03
- Regularization = none
- Regularization rate = 0
- Problem type = Classification
- Ratio of training to test data = 0.3
- Noise = 5
- Batch size = 10

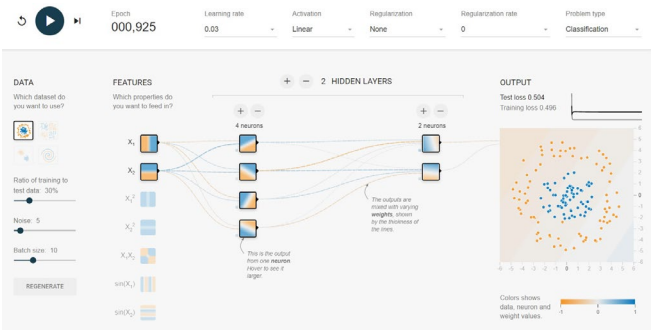

Figure 2: Neural network using playground.tensorflow using Linear activation function.

From the playground (Tensorflow) we were able to see that the linear activation function was not able to handle the complexity as the figure shown below:

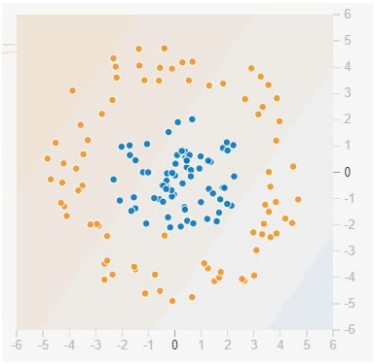

Figure 3: Output of neural network using Linear activation function on circle dataset.

From the image above we can conclude that linear activation is not suitable to handle complex datasets, hence we will see that we cannot derive at the best decision boundary which separates the orange and blue points even after a greater number of epochs and no improvement in train and test loss.

As other datasets with the same complexity, we conclude that linear activation is not suitable as well and as the figures shown below:

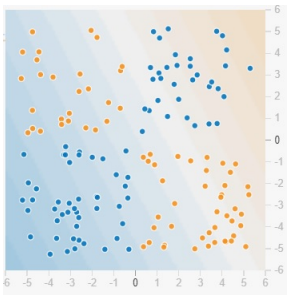

Figure 4: Output of neural network using Linear activation function on exclusive or dataset.

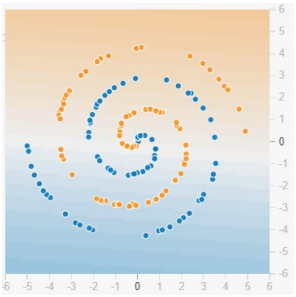

Figure 5: Output of neural network using Linear activation function on spiral dataset.

We can see in the images above that the dots are crossed despite the number of Epoch, which means that linear classifier has no effect.

2) Method to improve performance.

We can improve the performance of the model that uses the linear activation function by several methods:

i. Changing the activation function from linear to sigmoid

Changing the activation function from linear to sigmoid brought non-linearity to the network, making it powerful enough to capture the relationship within the data.

ii. Using simpler dataset

By using a simpler dataset, boundary will be easier to create, hence a better result.

B) Sigmoid

The Sigmoid Function curve looks like a S-shape. The following figure shows the graph of sigmoid activation function.

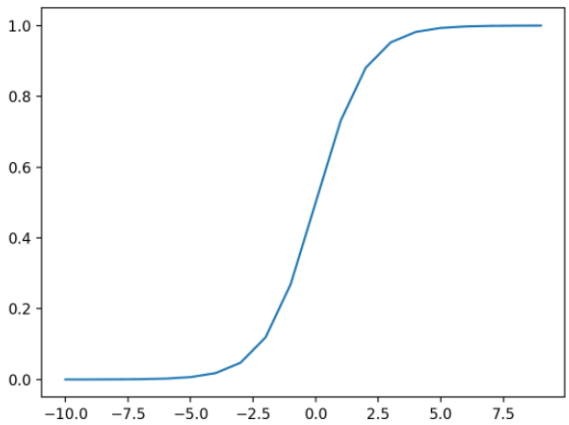

Figure 6: graph of sigmoid activation function

The equation used in sigmoid activation function is:

$$S(x) = \frac{1}{1 + e^{-x}}$$

- S(x)=Activation function
- e=Euler's number(2.71828)

We utilize the sigmoid function since it exists between two points (0 to 1). As a result, it is particularly useful in models where the probability must be predicted as an output. Because the likelihood of anything only occurs between 0 and 1, sigmoid is the best option. It is possible to differentiate the function. That is, the slope of the sigmoid curve may be found at any two places. Although the function is monotonic, the derivative is not.

The logistic sigmoid function has the potential to cause a neural network to become stuck during training.

Non-smooth functions, known as hard Sigmoid, are occasionally employed instead of smooth functions in artificial neural networks for efficiency. Sigmoid functions are employed as waveshaper transfer functions in audio signal processing to simulate the sound of analog circuitry clipping. The Hill equation and the Hill–Langmuir equation are sigmoid functions in biochemistry and pharmacology.

1) Sigmoid activation Function in Playground (Tensorflow)

Playground(Tensorflow) is used to study the effect of sigmoid activation function with constant values for other parameters:

- Learning Rate = 0.03
- Regularization = none
- Regularization rate = 0
- Problem type = Classification
- Ratio of training to test data = 0.3
- Noise = 5
- Batch size = 10

The following figure shows neural network using playground.tensorflow using Sigmoid activation function.

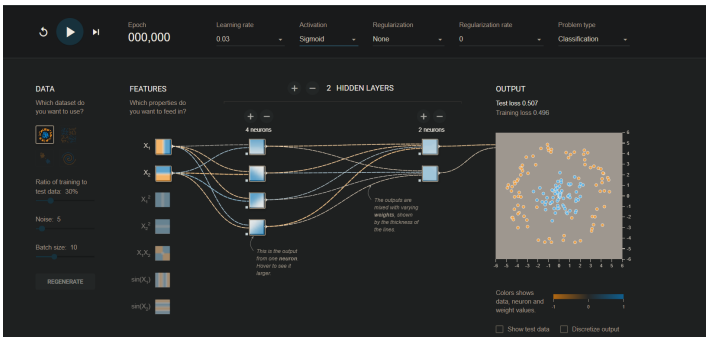

Figure 7: Neural network using playground.tensorflow using Sigmoid activation function.

From the playground(tensorflow) we able to see that the Sigmoid activation function able to find the shape of data. Resulted in a triangular shape which as the figure shown below:

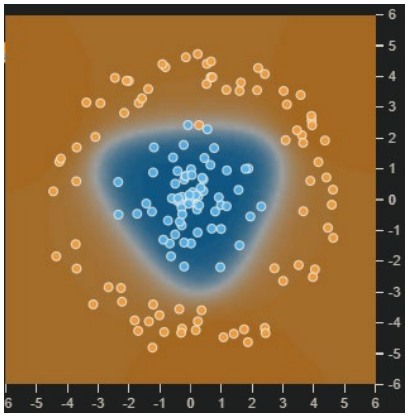

Figure 8: Output of neural network using sigmoid activation function on circle dataset

From the image above we can see that the Sigmoid function has barely clarify between blue and orange data in the dataset. Even though it has found the pattern of the dataset, but it not does not same with the original shape of the dataset which is circular shape for blue dataset. This indicates the sigmoid function only can barely cluster the data . This is because:

- Sigmoid saturate and kill gradients: The output of sigmoid saturates (i.e., the curve becomes parallel to x-axis) for a large positive or large negative number. Thus, the gradient at these regions is almost zero. During backpropagation, this local gradient is multiplied with the gradient of this gates' output. Thus, if the local gradient is very small, it will kill the gradient and the network will not learn. This problem of vanishing gradient is solved by ReLU.
- Not zero-centered: Sigmoid outputs are not zero-centered, which is undesirable because it can indirectly introduce undesirable zig-zagging dynamics in the gradient updates for the weights.

2) Method to improve performance.

We can improve the performance of model which uses the sigmoid activation function by several methods:

i. Increasing the neurons of the hidden layers

Increasing number of neurons enable the neural network model to generalize the data .Not only that, after increasing the neuron the number of epochs needed for the train and test data loss to stable also will decrease.

ii. Increasing the number of features

Increasing the number of features which is the input can lead to faster generalization .This is

because the higher the feature the higher the features extraction can be done by the neural network model. This also can result in lower train and test loss.

iii. Changing the activation Function

Use ReLU instead of sigmoid. ReLU does not saturated compared to sigmoid. The gradient does not "vanishing" or "diminishing." When the input is greater than zero, ReLU has a gradient of one. Otherwise, it has a gradient of zero. In the backprop equations, multiplying a number of ReLU derivatives together has the good property of being either 1 or 0. The gradient either stays the same or becomes exactly 0 as it goes to the lower levels.

C) TanH

The TanH Function curve looks like a S-shape. It is very similar to the Sigmoid curve. The following figures are the TanH graph and the comparison between the TanH and Sigmoid graph.

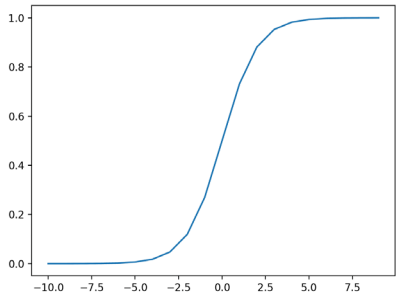

Figure 9: TanH graph

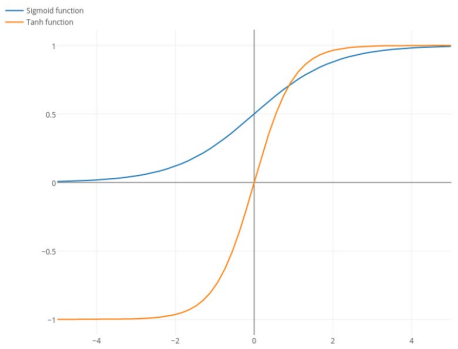

Figure ?: Comparison of the TanH and Sigmoid graph

The TanH curve is much steeper but has the same shape. The reason why we compare both these graphs is because they are extremely similar to each other. The only difference is the range scale whereby TanH starts at -1 to 1 and Sigmoid starts at 0 to 1. In this figure, the orange line represents TanH and the blue represents Sigmoid.

- The Tanh activation function is calculated as follows:

tanh(x) = (e^x – e^-x) / (e^x + e^-x)

Whereby, e is the mathematical constant.

The function takes any real value as input and outputs values in the range -1 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

One of the advantages of TanH is that the negative inputs will be mapped strongly negative, and the zero inputs will be mapped near the zero in the graph. This TanH graph is differentiable at every point and its derivative comes out to be as follows : $1\text{-}\tanh^2(x)$. It is also monotonic (which means it is either entirely non increasing or non-decreasing) but the derivatives are not monotonic. TanH and Sigmoid activation functions are used in feed-forward nets.

The differentiable value contains TanH hence, it can make the backward propagation faster. Despite lower chances of failure compared to the Sigmoid function, this function will still have some flaws such as vanishing gradient effect.

TanH and Sigmoid graphs are extremely similar to each other. The gradient of the TanH is steeper. When it comes to big data, TanH may be preferred more because the derivatives are larger. In other words, cost functions can be minimized. TanH is also more centered around the zero values compared to Sigmoid hence why it is more preferred.

1) TanH activation Function in Playground (Tensorflow)

Playground (Tensorflow) is used to study the effect of TanH activation function with constant values, the parameters:

- Learning Rate = 0.03
- Regularization = none
- Regularization rate = 0
- Problem type = Classification
- Ratio of training to test data = 30%
- Noise = 5
- Batch size = 10

The following figure shows the output value of the TanH function with the fixed parameters used.

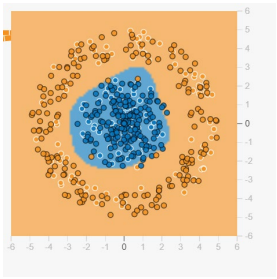

Figure 10: Output value of the TanH function with fixed parameters

From the result obtained above, the TanH activation function provided a result that is almost close to the desired output, but the output Is still not achieved. It is unable to very accurately classify and the reason to why this happens is because of the vanishing gradient problem that TanH activation function undergoes.

The vanishing gradient problem occurs when derivatives get smaller as we travel in backward propagation, which means the model will travel from output layer instead. The derivatives get smaller as we go backwards. This issue only occurs in Sigmoid function and TanH activation function because the derivatives are between the range of 0 to 0.25 and also 0 to 1. Therefore, when the model is supposed to update its weights, there is only a very small change in the values which is pretty much similar to the previous value. This is what causes the vanishing gradient problem.

To avoid it is when we use the ReLU activation function as it has gradient 0 for negatives and zero input and 1 for positive input.

D) ReLU

ReLU, rectifier or Rectified Linear Unit. In the context of artificial neural networks, ReLU is an activation function defined as the positive part of its argument.

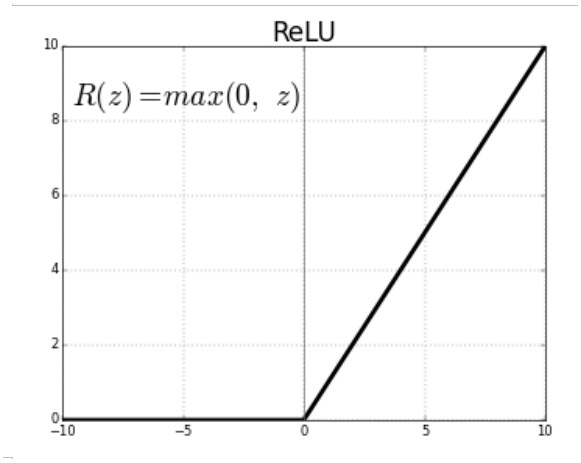

Figure 11: Plot of ReLU

Visualise how the ReLU how the ReLU solving a Exclusive OR Problem. The following figure shows that the ReLU react to the positivie value.

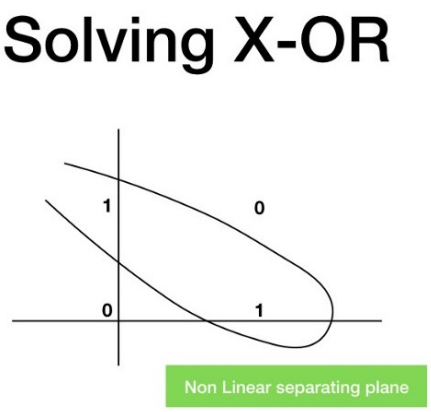

Figure 12: ReLU solving X-OR

From the figure above, ReLU is surrounding the positive while near to it, the result is positive part of its argument as mention before, this make it can surround the positive value that needed for classification task.

1.) Tensorflow Playground

- *Features Used: With the normal features X_1 and X_2 feed in as the most basic and simplest input. The following figures show features X_1 and X_2.*

Figure 13: Features X_1 and X_2

- *Dataset Used:* This circle dataset can be used as the non-linear problem. The following figure shows the dataset for this experiment.

Figure 14: Circle Dataset used for ReLU

To let ReLU classify the dataset successfully it need at least one hidden layer with 3 neurons. The following figure shows the minimum requirement for ReLU to classify the dataset succesfully.

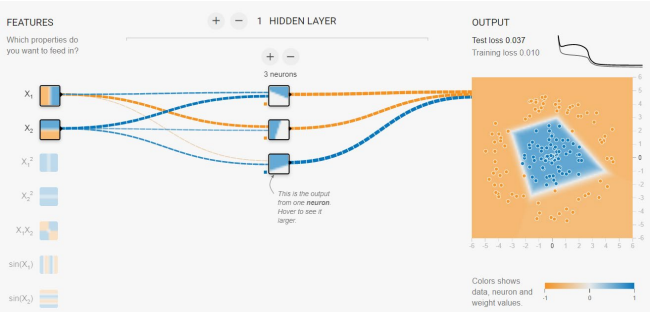

Figure 15: Minimum Requirement for ReLU to success

And then the layer with at least 3 neurons must be the first layer. With this requirement, no matter how many hidden layer or neuron are added, the dataset will be classified successfully, just the time taken increases as the setting increases and the shape of the result will be different. For instance, the following figure shows any setting with the minimum requirement.

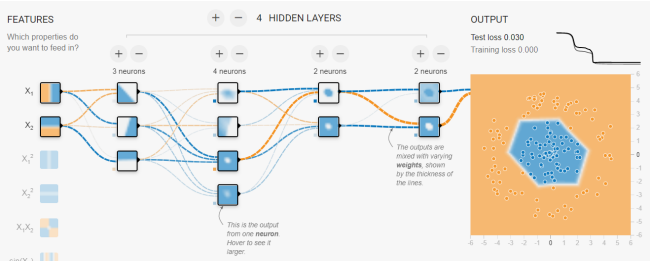

Figure 16: Any hidden layer and neurons with minimum requirement

The previous statement prove that ReLU can solve circle dataset classification problem, now look at another type of dataset, Exclusive OR, which is the most popular example people used to explain non-linearity. The following figures show XOR dataset and the result.

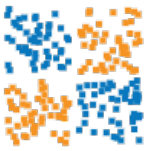

Figure 17: Exclusive OR Dataset

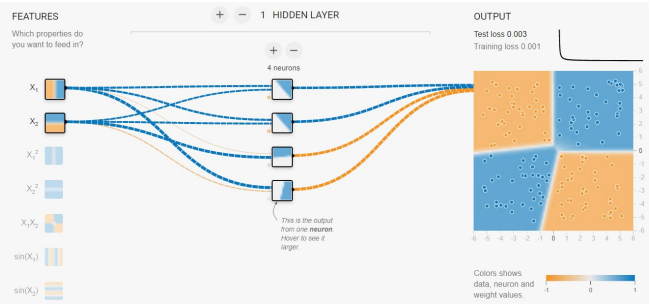

Figure 18: Exclusive OR Result

2.) All datasets with fixed setting

A certain level of finding on ReLU activation is done, now get into the analysis conduct where whether ReLU can solve all the 4 types of datasets provided which are Circle, Exclusive OR, Gaussian, and Spiral for the classification task with the fixed setting in this mini project and with the following hidden layer set up:

- Hidden Layer =1 layer with 4 neurons

The following figures shows the result of each dataset with using ReLU.

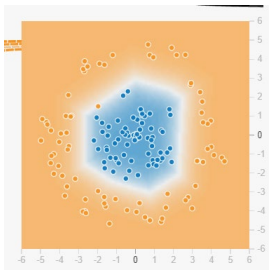

Figure 19.1: Circle Dataset with ReLU

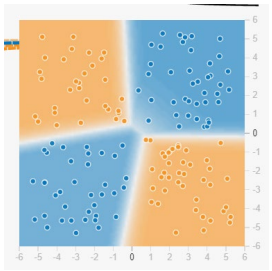

Figure 19.2: Exclusive OR Dataset with ReLU

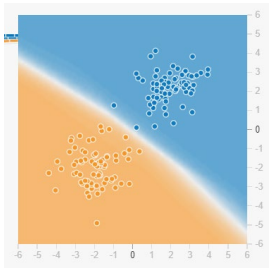

Figure 19.3: Gaussian Dataset with ReLU

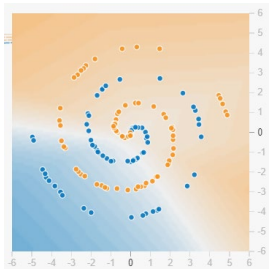

Figure 19.4: Spiral Dataset with ReLU

3.) ReLU with spiral dataset

From the previous set of figures showing the results of each dataset, only spiral dataset classification cannot be solved with fixed setting. But does ReLU able to solve the

complex spiral dataset? The answer is yes. The following figure shows the spiral dataset classification solved by the ReLU.

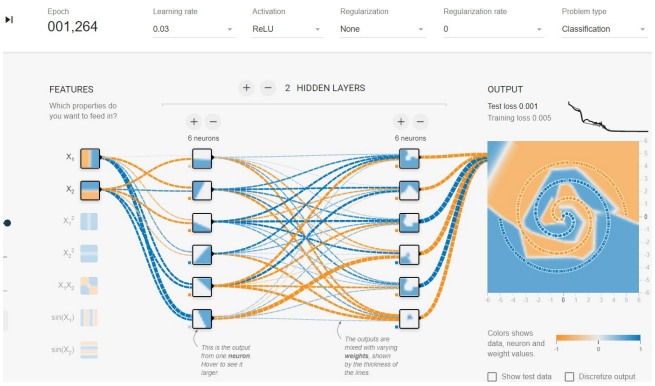

Figure 20: Spiral Dataset with ReLU (other setting)

To obtain the result above, some setting from the fixed setting is change:

- 2 hidden layers
- Each hidden layer with at least 6 neurons
- Ratio of training to test data = 90%
- Noise = 0

Now prove that ReLU can solve all the non-linear dataset classification task with main the features being feed in that is X_1 and X_2. From observation, increase hidden layer and neuron only increase ability of ReLU to surround and classify the dataset with no significant negative result such as the Sigmoid does.

E) Comparison

In term to compare the performance between the activation functions available is Tensorflow Playground, the fixed setting is applied using the circle dataset, and the result is observed. With the following set up:

- Data = Circle
- Hidden Layer =1 layer with 4 neurons

The following figures shows the result from each activation functions when complete, and its test loss and training loss at epoch=100.

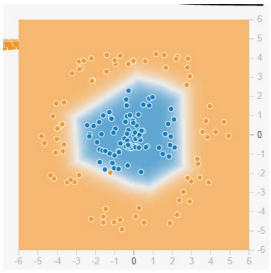

Figure 21.1: ReLU Fixed Setting Result

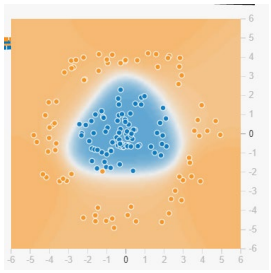

Figure 21.2: TanH Fixed Setting Result

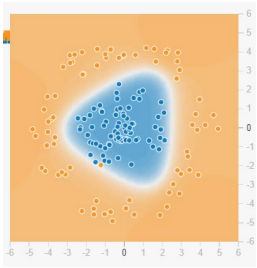

Figure 21.3: Sigmoid Fixed Setting Result

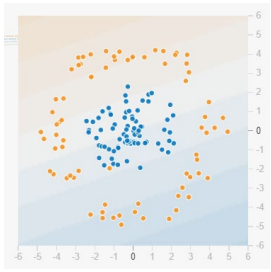

Figure 21.4: Linear Fixed Setting Result

Test loss and Training loss for each Activation function at epoch=100 :-

- ReLU Test loss =0.020
  Training loss =0.029
- TanH Test loss =0.072
  Training loss =0.074
- Sigmoid Test loss =0.472
  Training loss =0.462
- Linear Cannot solve

All activation function can solve to the problem except the Linear activation, so Linear is at last of the ranking for the conclusion. For other 3 activation function we can rank them based on the test loss and training lost, the lower have higher rank, so ReLU > TanH > Sigmoid. ReLU is the have the best performance so no doubt it is the first. For TanH and Sigmoid, TanH not only have lower value of test loss and training loss, but it does also have no problem while increasing the number of hidden layers compare to Sigmoid.

VI. Conclusion

From the result and analysis from V.E) *Comparison*, a conclusion that ReLU > Tanh > Sigmoid > Linear is made for this mini project. This also prove that ReLU activation function is the most used activation function in the world right now.

References

[1] Z² Little (2020, May 18). Activation Functions (Linear/Non-linear) in Deep Learning. Medium. https://xzz201920.medium.com/activation-functions-linear-non-linear-in-deep-learning-relu-sigmoid-softmax-swish-leaky-relu-a6333be712ea

[2] SAGAR SHARMA (2017, September 06). Activation Functions in Neural Networks. Towards Data Science. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[3] Sakshi Tiwari, (2020, October 08). Activation functions in Neural Networks. Geeks for geeks. https://www.geeksforgeeks.org/activation-functions-neural-networks/

[4] (Kyle), K. A. (2019, April 12). *Best Place to Learn Neural Network: Interactive Tensorflow Playground*. Medium. https://kyleake.medium.com/technical-demo-visualize-neural-network-with-tensorflow-playground-9f6a1d8eb57a

[5] Carter, D. S. and S. (n.d.). *Tensorflow - Neural Network Playground*. A Neural Network Playground. https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=NaN&regularizationRate=0&noise=0&networkShape=4,2&seed=0.25715&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=true

[6] Khan, M. R. (2019, January 25). *Deep Learning with TensorFlow Playground*. Medium. https://medium.datadriveninvestor.com/deep-learning-with-tensorflow-playground-e6b194ee8fac

[7] Wood T. Sigmoid Function. DeepAI. Published September 27, 2020. Accessed June 23, 2021. https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function

[8] Nahua Kang. Multi-Layer Neural Networks with Sigmoid Function—Deep Learning for Rookies (2). Medium. Published June 27, 2017. Accessed June 23, 2021. https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f#:~:text=Sigmoid%20function%2C%20unlike%20step%20function,into%20our%20neural%20network%20model.&text=This%20non%2Dlinear%20activation%20function,decision%20boundary%2C%20such%20as%20XOR

[9] SAGAR SHARMA (Sep 6, 2017·5). Activation Functions in Neural Networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[10] Wikipedia. Rectifier (neural networks) https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[11] Lalit Pal (Apr 15, 2019). Understanding Basics of Deep Learning by solving XOR problem. https://medium.com/analytics-vidhya/understanding-basics-of-deep-learning-by-solving-xor-problem-cb3ff6a18a06