Activation Functions and Swish

Kailash Nadkar
nadkar.k@husky.neu.edu
INFO 7390, Summer 2019, Northeastern University

Abstract- At the heart of every deep network lies a linear transformation followed by an activation function f(*x*). Activation functions determine the output of a deep learning model, its accuracy, and the computational efficiency of training a model—which can make or break a largescale neural network. Activation functions also have a major effect on the neural network's ability to converge and the convergence speed. In this work, I have implemented activation function swish and swish-beta and compared them with most common activation functions like sigmoid, tanH, ReLU, and Leaky ReLU. For a valid comparison I have created a Convolutional Neural Network (CNN) model for all activation function with same learn rate, steps, epochs etc. The aim of this work is to compare and validate the Activation function based on their convergence speed, loss and accuracy.

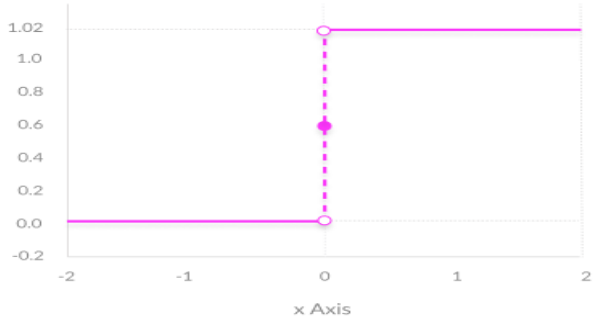
1. Introduction

1.1 Background

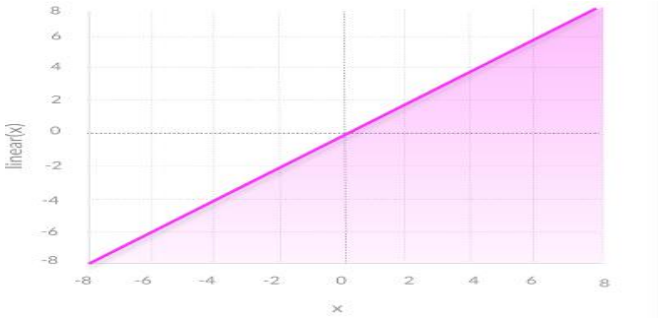
Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network and determines whether it should be activated ("fired") or not, based on whether each neuron's input is relevant for the model's prediction. Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1. An additional aspect of activation functions is that they must be computationally efficient because they are calculated across thousands or even millions of neurons for each data sample. Modern neural networks use a technique called backpropagation to train the model, which places an increased computational strain on the activation function, and its derivative function. The need for speed has led to the development of new functions such as Swish and Swish-Beta.

The Activation function can be of 3 type:

A. Binary Step Function: A binary step function is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.



B. Linear Activation Function: It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input.



C. Non-Linear Activation Functions: They allow backpropagation because they have a derivative function which is related to the inputs.

In our work we are Using following Non-Linear Activation Function:

1. Sigmoid :

1.2 Dataset

Intel Image Classification:

Data is acquired from Kaggle dataset of Intel Image Classification Competition. Data contains around 25k images of size 150x150 distributed under 6 categories. {'buildings', 'forest', 'glacier', 'mountain', 'sea', 'street'}.

There are around 14k images in Train, 3k in Test and 7k in Prediction. In this project I am using only 2 categories {'buildings', 'forest'}.

2. Algorithm and code source

The Entire project is run on TensorFlow GPU environment and, also on google colab. For Image classification I am using Convolutional Neural Network (CNN). CNN model is implemented using TensorFlow Estimator. An Estimator is a TensorFlow class for performing high-level model training, evaluation, and inference for our model. The models saved in local directory and can be used later for training, evaluation, and inference even after kernel reset or reconnecting your server.

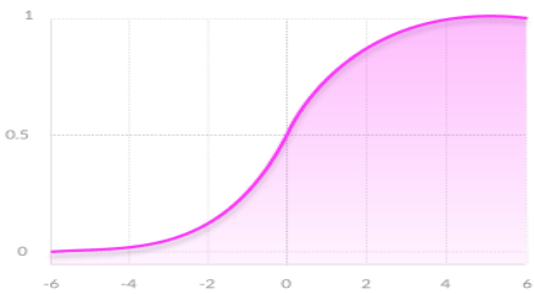
The CNN model consists following layers:

1. Input Layer (input_layer)
2. Convolutional Layer (conv1)
3. Pooling Layer (pool1)
4. Normalization Layer (norm1)
5. Convolutional Layer (conv2)
6. Pooling Layer (pool2)
7. Flatten Layer (pool2_flat)
8. Fully Connected/ Dense Layer (dense)
9. Logits/Output Layer (logits)

To calculate the loss 'Sparse SoftMax Cross Entropy' function and to optimize the output the 'Adam Optimizer' is used.

In this work I am comparing 6 different activation functions:

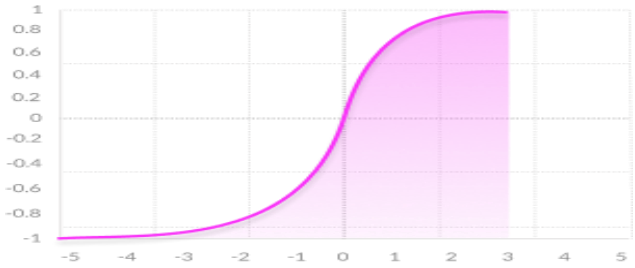
1. Sigmoid:



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

- **Advantage**: Smooth gradient, preventing "jumps" in output values.
- Output values bound between 0 and 1, normalizing the output of each neuron.
- Clear predictions—For X above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.
- **Disadvantage:** Vanishing gradient—for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further or being too slow to reach an accurate prediction.
- Outputs not zero centered**.**
- Computationally expensive

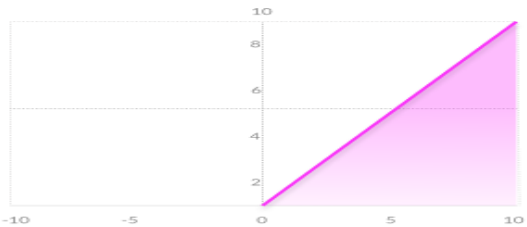
2. TanH / Hyperbolic Tangent:



$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

- **Advantages:** Zero centered—making it easier to model inputs that have strongly negative, neutral, and strongly positive values.
- Otherwise like the Sigmoid function.
- **Disadvantages:** Like the Sigmoid function

3. ReLU (Rectified Linear Unit):

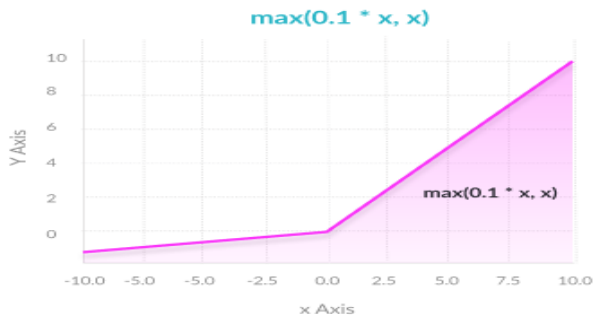


$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$

- **Advantages:** Computationally efficient—allows the network to converge very quickly
- Non-linear—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation
- **Disadvantages**: The Dying ReLU problem—when inputs approach zero, or are negative, the

gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

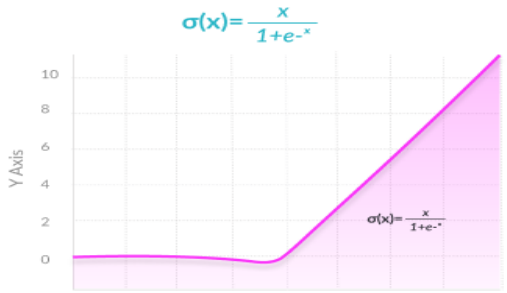
4. Leaky ReLU:



$$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

- **Advantages:** Prevents dying ReLU problem—this variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values
- Otherwise like ReLU
- **Disadvantages**: Results not consistent—leaky ReLU does not provide consistent predictions for negative input values.

5. Swish:



Swish is a new, self-gated activation function discovered by researchers at Google.

$$f(x) = x \cdot \sigma(x)$$

Where, `σ(x)= sigmoid function`

6. Swish Beta:

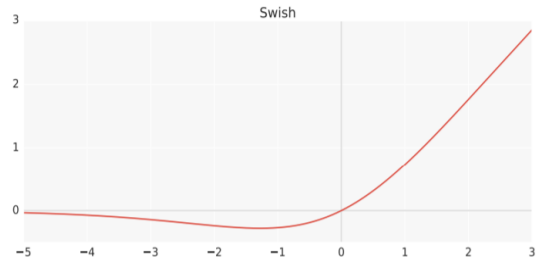


Figure 1: The Swish activation function.

$$f(x) = x * \text{sigmoid}(\beta * x)$$

Where, β is a constant or trainable parameter

3. Observation:

The leaky ReLU seems to face NAN Loss error when using ‘losses.sparse_softmax_cross_entropy’ with ‘Gradient Descent Optimizer’ and it can be avoided using ‘Adam Optimizer’ which can add small epsilon value to the prediction to prevent this divergence.

Surprisingly tanH function produced better result than most of the common function we used in Neural Networks.

The Leaky ReLU is one of the functions tested here, which is computationally more demanding giving more memory error compared to other functions

4. Conclusion:

Here we are implemented various activation functions used in Neural network, like Sigmoid, tanH, ReLU, Leaky ReLU and, also implemented new Activation functions like Swish and Swish-Beta

Comparison Table:

I avoided to round of values to get accurate idea of each activation function's performance since, most of them produce results very close to each other.

Activation Function	Accuracy	Loss	Execution Time	Number of Steps
Sigmoid	0.9593853	0.13093688	12.22 minutes	1000
tanH	0.9791438	0.101240434	12.32 minutes	1000
ReLU	0.94291985	0.22290945	12.83	1000

Activation Function	Accuracy	Loss	Execution Time	Number of Steps
			minutes	
Leaky ReLU	0.96597147	0.18424875	14.67 minutes	1000
Swish	0.9473106	0.14577803	12.93 minutes	1000
Swish Beta	0.96377605	0.12295562	12.92 minutes	1000

- All the activation functions are run on the same CNN model
- The Leaky ReLU perform significantly better but took more time to do so compared to other functions.
- Swish perform slightly better than ReLU, but Swish-Beta performance was noticeably better.
- Even though leaky ReLU performs very slightly better than Swish-Beta, its loss is higher than Swish-Beta and took more time to achieve so.
- The tanH and Swish Beta produce significantly better results

5. Future Scope:

- Tuning our CNN model adding more layers to it
- Testing different dataset and comparing activation functions results across them
- Adding more activation functions for comparisons
- Evaluating why particular activation function perform better while others work poorly and establishing guidelines that can be used by other data scientist when choosing activation function

6. Acknowledgment:

We want to thank Prof. Nick Brown for his constant support and guidance during this project.

7. References:

[1] https://arxiv.org/abs/1710.05941v1
[2] https://arxiv.org/abs/1710.05941v2
[3]https://www.kaggle.com/puneet6060/intel-image-classification
[4]https://www.tensorflow.org/tutorials/estimators/cnn
[5] https://www.kaggle.com/vincee/intel-image-classification-cnn-keras
[6] https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/
[7]https://www.tensorflow.org/api_docs/python/tf/nn
[8]https://colab.research.google.com/notebooks/welcome.ipynb.