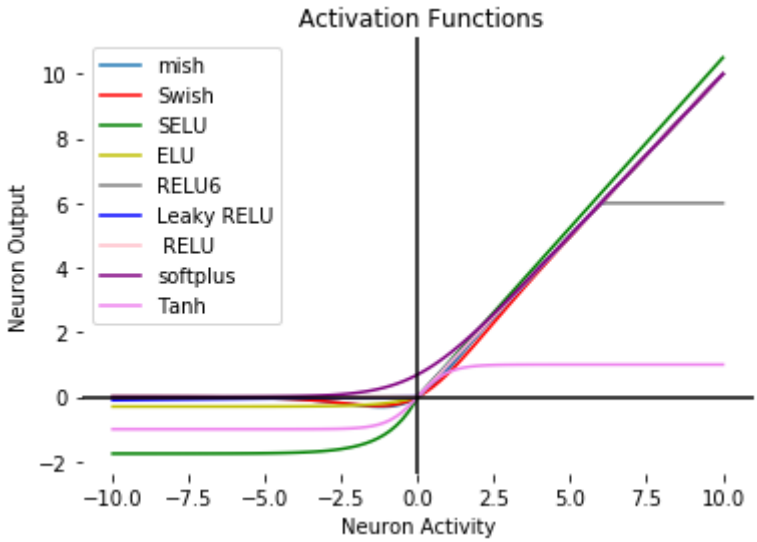
Different Activation Functions for Deep Neural Networks You Should Know

A quick snapshot of new and popular activation functions used in deep neural networks

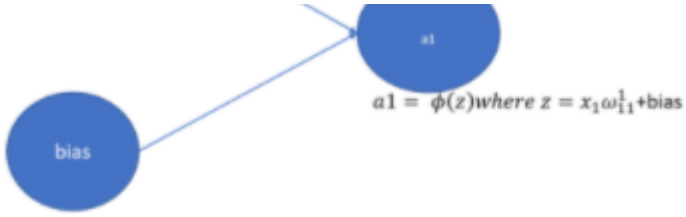
Renu Khandelwal
Mar 16 · 9 min read ★

Understand popular activation functions used in deep neural networks: Sigmoid, Softmax, tanh, ReLU, Softplus, PReLU, ReLU6, ELU, SELU, Swish, and Mish



Working of a Deep Neural Network

A deep neural network performs a linear transformation(z) using the node's input value(x), weight(w), and bias(b).



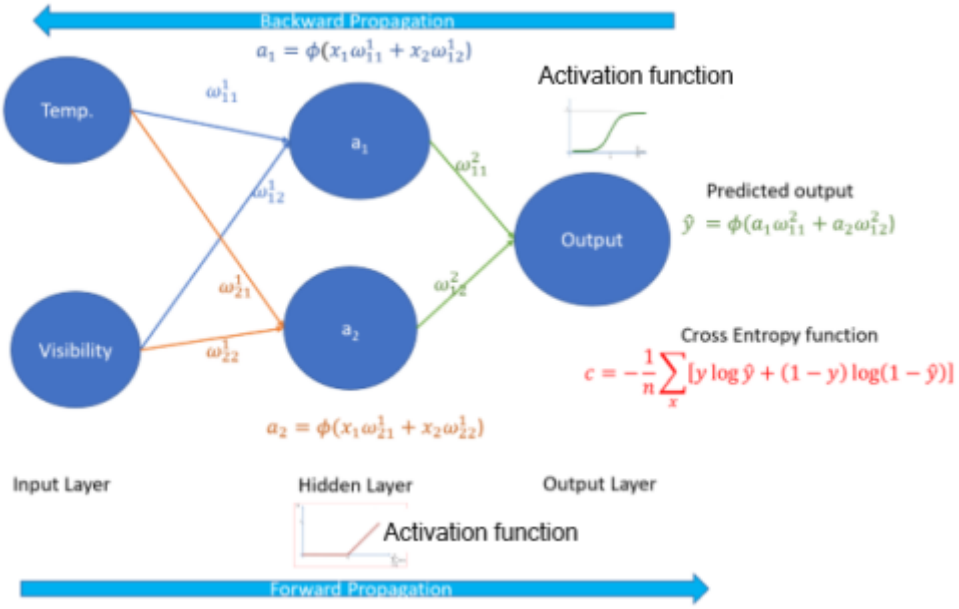
$$z = \sum(\omega x) + b$$

During forward-propagation, **an activation function transforms the node's linearity to non-linearity and decides if a neuron will be fired or not.**

The output of the neuron is based on the strength of the signal that the activation function outputs. **The activation function also normalizes the neuron's output** to a value between 0 and 1 or between -1 and 1, depending on the activation function used.

The output layer calculates the error function, which is the difference between the predicted value and the ground truth.

The error calculated at the final output layer is then backpropagated across the neural network and the gradients using the activation function to update the weights and the biases. This helps reduce the difference between the predicted value and the ground truth.



As a result, a deep neural network learns the input data's underlying features and map them to the appropriate output.

Role of the Activation function in a deep neural network

- **The activation function decides if a neuron will be fired or not. If the neuron is fired, then what will be the strength of the signal outputted by the neuron.**
- Activation functions determine the deep Learning model's accuracy and the computational efficiency for training a model.
- **Activation functions have to be computationally very efficient as it decides on the network's convergence speed** or if the network will ever converge.
- **The activation function performs a critical role in processing and passing the information across multiple layers of a deep neural network, thus impacting its performance**.

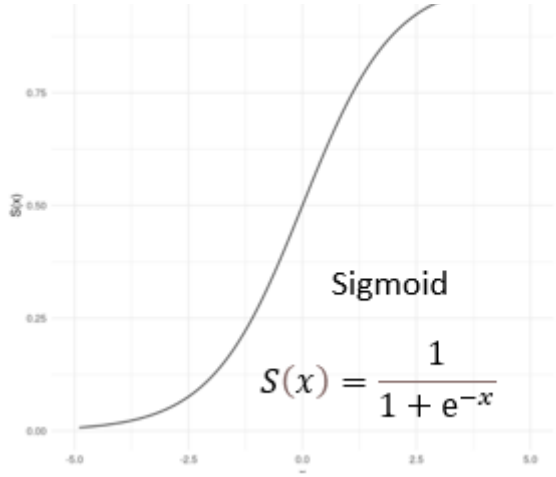
Features of the Activation function

- **Differentiable**: **Activation functions should be differentiable to allow the neural network to update the neuron's weights and biases during backpropagation** to reduce the error or loss of the neural network. Reduction in loss of the neural network will result in better predictions by the model.
- **Monotonic**: The activation function should be monotonic, either entirely non-increasing or entirely non-decreasing. **The monotonicity behavior of the activation function helps neural networks to converge and generate an accurate classifier.**
- **Approximates near origin**: Neural networks use gradient descend for faster convergence by initializing a neural network's weights and biases with values close to zero. **When activation functions approximate near the origin, the neural network learns efficiently due to strong gradients**.

Types of Activation functions

Sigmoid activation function or Logistics function

- Characterized by “S”-shaped curve or **sigmoid curve**
- Its value ranges between 0.0 and 1.0
- **used for binary classification as the model predicts probability between 0 and 1**



Sigmoid Activation Function

```
import math
def sigmoid(x):
     return  1 / ( 1 + math.exp( -x ))
```

If the inputs are negative, then the output of the sigmoid function will be smaller than 0.5. If the inputs are positive, then the output of the sigmoid function will be greater than 0.5.

- **Sigmoid suffers from vanishing gradient problem.**

Softmax Activation Function

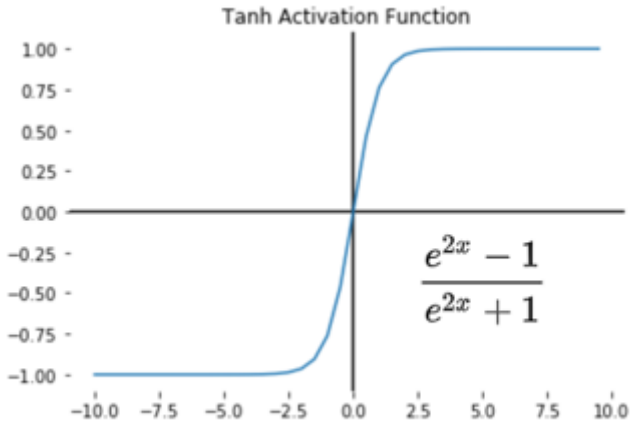
- **The Softmax function is also referred to as softargmax or normalized exponential function.**
- Used for multinomial logistics regression, hence **used in the output layer of a multi-class classification neural network.**
- **Softmax activation will output one value for each node in the output layer of the neural network. The target class will have the highest probability.**
- **The output of the Softmax function is in the range between 0 and 1. The output of all nodes adds up to 1 and forms a probability distribution.**
- Softmax classifier uses the cross-entropy loss.
- **Softmax normalization reduces the influence of outliers in the data without removing data points from the dataset.**

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

```
from numpy import exp
# softmax calculate the probabilities of a vector representing muit-class
def softmax(vector):
    e = exp(vector)
    return e / e.sum()
```

Tanh

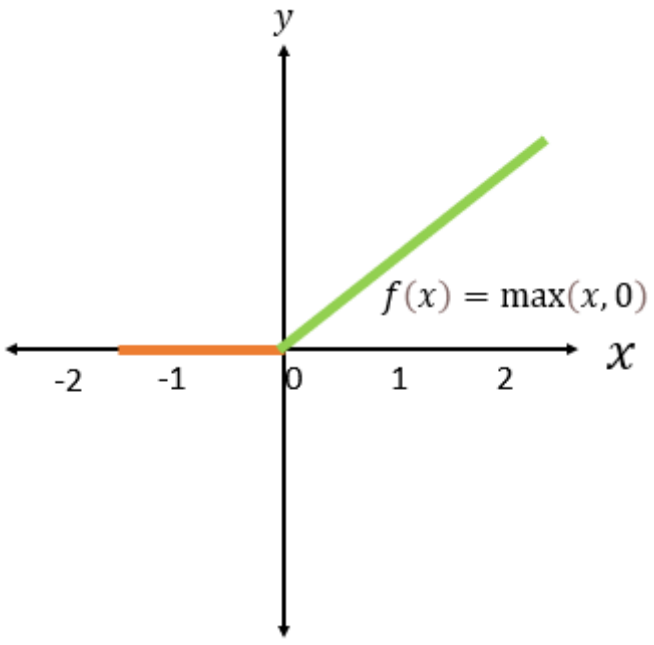
- **Tanh is a smoother, zero-centered function having a range between -1 to 1.**
- Unlike Sigmoid, Tanh's output is zero-centered.
- **Tanh's non-linearity is always preferred to the sigmoid nonlinearity.**
- The gradient is stronger for tanh than sigmoid.
- **Tanh also has the vanishing gradient problem.**




```
def tanh(x):
    return np.tanh(x)
```

Rectified Linear Unit(ReLU)

- **ReLU is an activation function that will output the input as it is when the value is positive; else, it will output 0.**
- **ReLU is non-linear around zero, but the slope is either 0 or 1** and has limited nonlinearity.
- **Deep networks with ReLUs are easily optimized than networks with sigmoid or tanh units as the gradient can flow across different layers when the input to the ReLU function is positive**.
- Used widely in hidden layers of deep neural networks for Computer vision, speech recognition **induces sparsity.**
- **Computations for ReLU are cheaper** due to simple computation without exponential function in activations, and sparsity also reduces the number of computations.
- When applying ReLU in the hidden layers, **if the value of the neuron 0 or a negative value, then the neuron will not be fired**, thus activate only a few neurons. Hence, **during the backpropagation process, these neuron weights and biases will not be updated, creating a dying ReLU problem**.
- **The Dying ReLU problem pushes neurons into states where they are never fired are become inactive for essentially all inputs causing a vanishing gradient issue.**
- ReLU ranges between 0 and infinity.

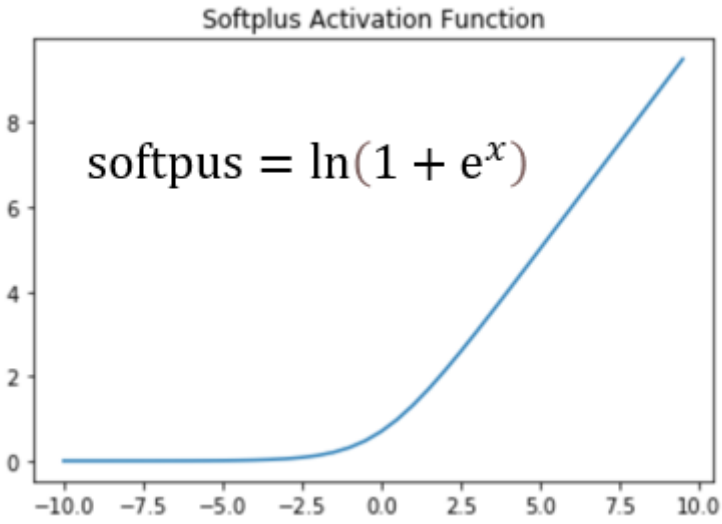


Rectified Linear Unit(ReLU)

```
def relu(x):
    if x<0:
        return 0
    else:
        return x
```

SoftPlus

- **SoftPlus is a smoother version of the rectifying non-linearity activation function** and can be used to constrain a machine's output always to be positive.
- SoftPlus function produces outputs in the range of $(0, +\infty)$
- **The difference between ReLu and softplus is near 0, where the softplus is enticingly smooth and differentiable.**
- ReLU has efficient computation, but **the computation for softplus function is more expensive as it has log and exp in its formulation.**

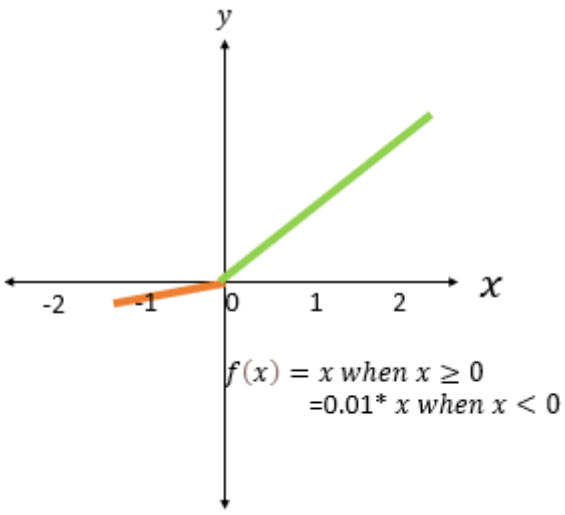


Softplus

```
def softplus(x):
    return math.log(1+ np.exp(x))
```

Leaky ReLU

- An improvement over ReLU by **introducing a small negative slope to solve the problem of dying ReLU.**
- **When the value of a neuron is negative, then we multiply the value with a small number like 0.01 so that the neuron is not deactivated, thus solving the dying ReLU problem**,
- **Leaky ReLU with a non-zero slope helps speed up training.**
- The range for Leaky ReLU is from -infinity to +infinity
- **Leaky ReLU has a negligible impact on accuracy compared with ReLU.**



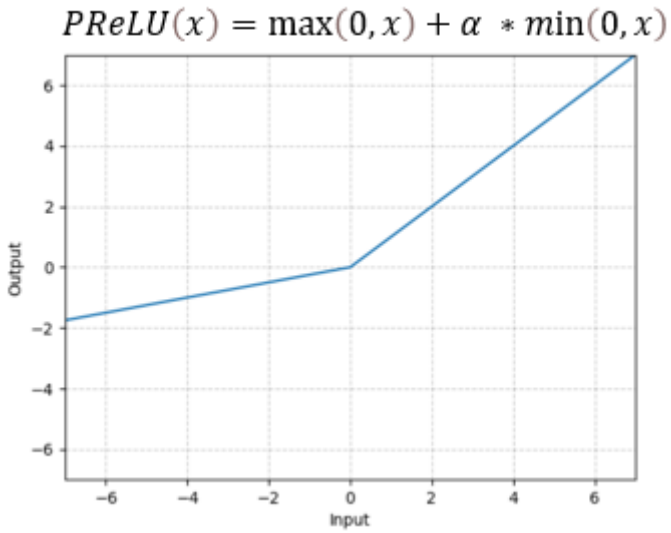
Leaky ReLU

```
def leaky_relu(x):
    if x<0:
        return x*0.01
    else:
        return x
```

Parameteric ReLU(PReLU)

- **PReLU generalizes the traditional ReLU with a small parameter value for the slope for negative values with nearly zero extra computational cost and little overfitting risk.**
- **PReLU solves the problem of vanishing gradient.**

- **When the parameter value is 0, the activation function acts like ReLu; when the parameter value is 0.01, it becomes Leaky ReLU.**
- PReLU introduces a very small number of extra parameters equal to the total number of channels.
- **f(x) = max(0, x) + α * min(0, x) where the coefficient is shared by all channels of one layeradding only a single extra parameter into each layer.**
- **The parameter value, the alpha, is not configured by the user; instead, it is learned during training and varies on different channels.**
- **PReLU solves the dying ReLU problem and also shows performance improvements.**



PReLU Activation Function

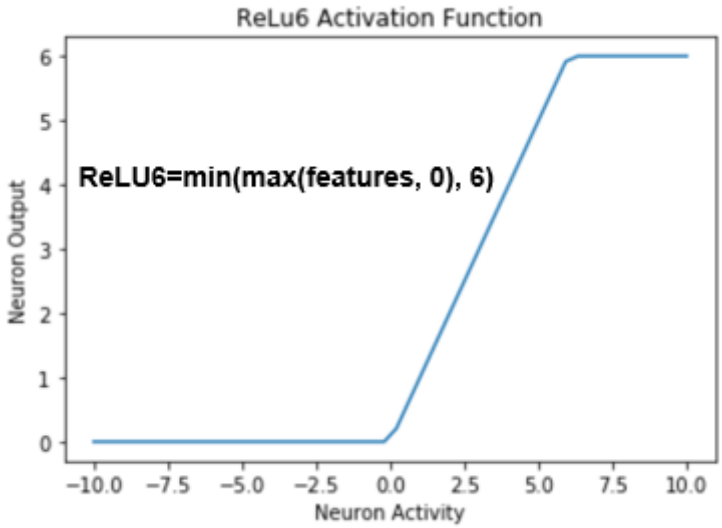
```
def prelu(x, alpha=0.2):
    max_x= max(0,x)
    min_x= min(0,x)
    return max_x + alpha * (min_x)
```

RELU6

- **ReLU6 is a modification over ReLU where we limit the activation to a maximum of 6.**
- **ReLU6 is used for low-precision computation required when the model is compressed or quantized.**

- The range is from 0 to 6, and nonlinearity only involves 8-bit integer arithmetic. **6 will take a max of 3 bits out of the 8 bits and leaving rest of 4/5 bits for the float values**

- Computes Rectified Linear 6: `min(max(features, 0), 6)`



ReLU6 Activation Function

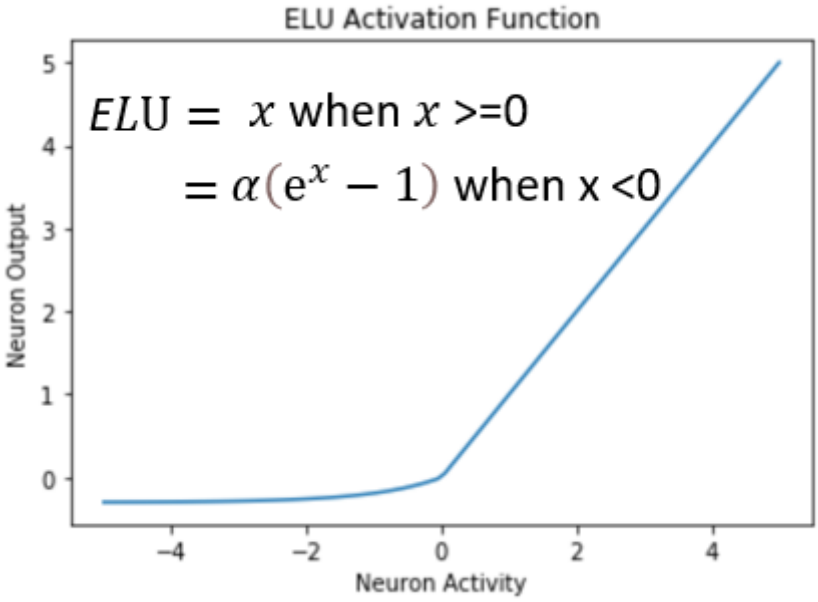
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plot
%matplotlib inline

x = np.linspace(-10, 10, 50)
output = tf.nn.relu6(x)

plot.xlabel('Neuron Activity')
plot.ylabel('Neuron Output')
plot.title('ReLu6 Activation Function')
plot.plot(x, output)
plot.show()
```

Exponential Linear Unit(ELU)

- **ELU is another variant of RELU that modifies the slope of the negative part of the function**. Both ReLU and ELU have the same identity function form for non-negative inputs.

- **ELU has an extra alpha(α) constant, which can values that can range between 0.1 to 0.3**

- ELU becomes smooth slowly until its output equal to -α whereas RELU sharply smoothes.
- **ELU eliminate the vanishing gradient issues, speed the training and the learning of the neural network**
- **ELU is more computationally expensive than the ReLU but more robust to noise**



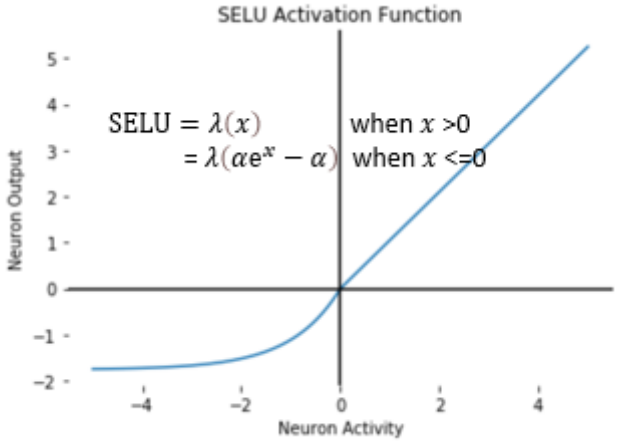
ELU Activation function

```
def elu(x, alpha):
    if x <0:
        return alpha*(np.exp(x) -1)
    else:
        return x
```

Scaled Exponential Activation Function(SELU)

- **Scaled Exponential Linear Units or SELU activation functions induce self-normalizing properties.**
- **The output of a SELU is normalized, internal normalization, where all the outputs are with a mean of zero and standard deviation of one and converges faster.**
- **SELU has two parameters α(alpha) and λ(lambda) and for standardized inputs, α=1.6733 and λ=1.0507**

- The hidden layers weight must be initialized with **LeCun normal initialization,** which will initialize the network's parameters as a normal distribution.
- **SELU induces self-normalizing properties like variance stabilization which in turn avoids exploding and vanishing gradients.**



SELU Activation Function

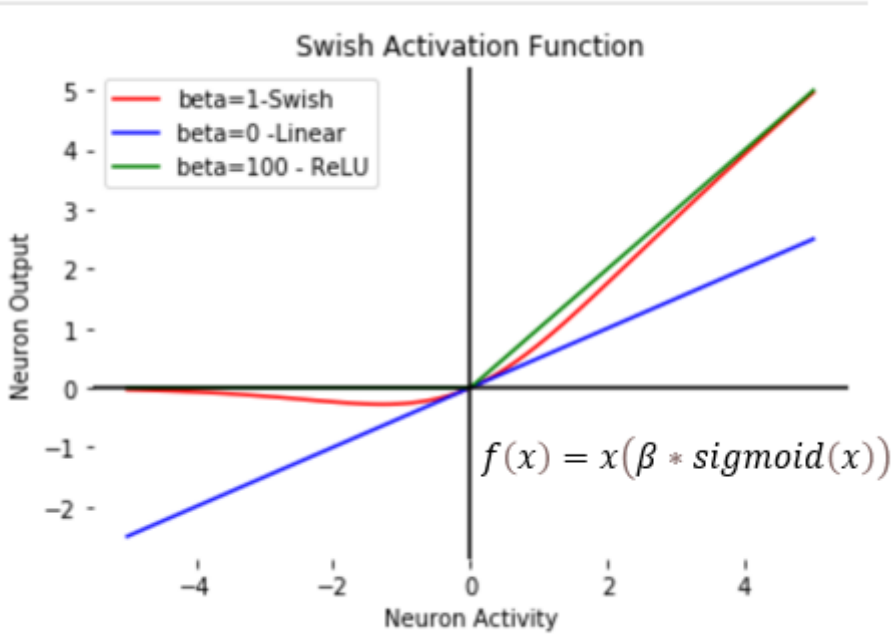
```
def selu(x, alpha=1.6733, lam=1.0507):
    if x>0:
        return lam*x
    else:
        return lam*((alpha*np.exp(x))- alpha)
```

Swish

- **Proposed by Google Brain in 2017. Swish is a gated version of the sigmoid activation function**
- **Swish is a smooth, non-monotonic function. unlike ReLU**
- **The non-monotonicity property of Swish distinguishes itself from most common activation functions. Non-monotonicity of Swish increases expressivity and improves gradient flow**.
- Swish performs better than ReLU on deeper models across many challenging datasets.
- Like ReLU, **Swish is unbounded above and bounded below**. Unbounded above is desirable in the activation function to avoid saturation, and bounded below helps

with a strong regularization effect.

- The simplicity of Swish and its similarity to ReLU gives better performance for deep neural networks by just replacing ReLU with Swish, which is just a simple one-line code change.



Swish Activation Function

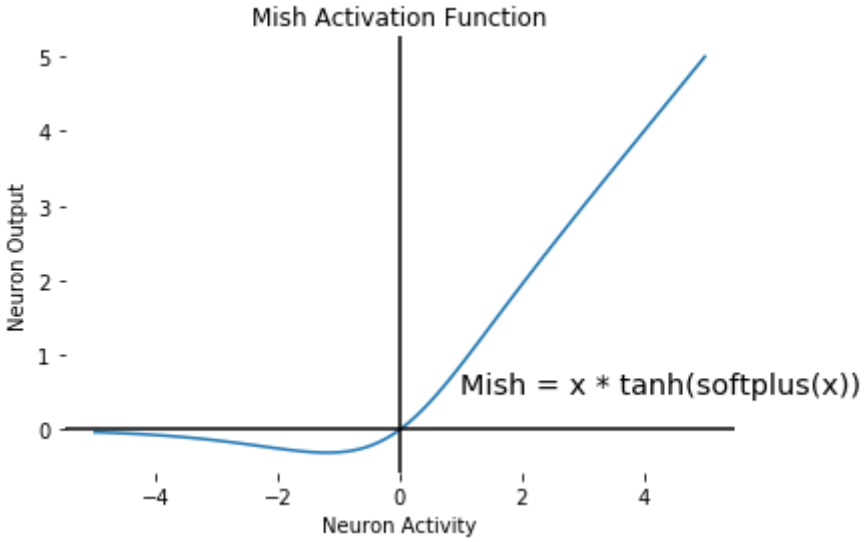
```
def swish(x, beta=1):
    return x * (sigmoid(beta*x))
```

Mish

- **Mish is a self-regularized non-monotonic activation function inspired by the self-gating property of Swish.**
- Mish is non-monotonic, has the ability to preserve small negative weights, and a smooth profile just like Swish, which gives consistent performance and improvement for deep neural networks.
- **Mish preserves a small amount of negative information, which eliminates the Dying ReLU phenomenon.**
- **Mish either matches or improves neural network architectures' performance compared to using activation functions like Swish, ReLU, and Leaky ReLU**

across different Computer Vision tasks.

- Mish is bounded below and unbounded above with a range of -0.31 to ∞**.** Being unbounded above avoids saturation, and being bounded below helps with a strong regularization effect.
- **Mish is much smoother and conditioned as compared to ReLU and Swish activation functions.**
- **Mish has a wider minima to improve generalization compared to that of ReLU and Swish.**
- **Mish obtained the lowest loss as compared to the networks equipped with ReLU and Swish.**




```
def mish(x):
    return x * tanh(softplus(x))
```

Conclusion:

Activation functions introduce nonlinearity to the linear transformed input in a neural network layer. They play a critical role in the performance and training time of neural networks, and thus the choice of the activation function is imperative for a neural network with better performance, less training time, and lower loss.

References:

https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Mish: A Self Regularized Non-Monotonic Activation Function

SWISH: A SELF-GATED ACTIVATION FUNCTION

Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Self-Normalizing Neural Networks

Activation Functions | Deep Neural Networks | Swish | Mish | Relu