Activation Functions

What is an activation function?

An activation function, in the context of neural networks, is a mathematical function applied to the output of a neuron. The purpose of an activation function is to introduce non-linearity into the model, enabling the network to learn and represent complex patterns in the data. Without non-linearity, a neural network would behave like a linear regression model, regardless of the number of layers it contains.

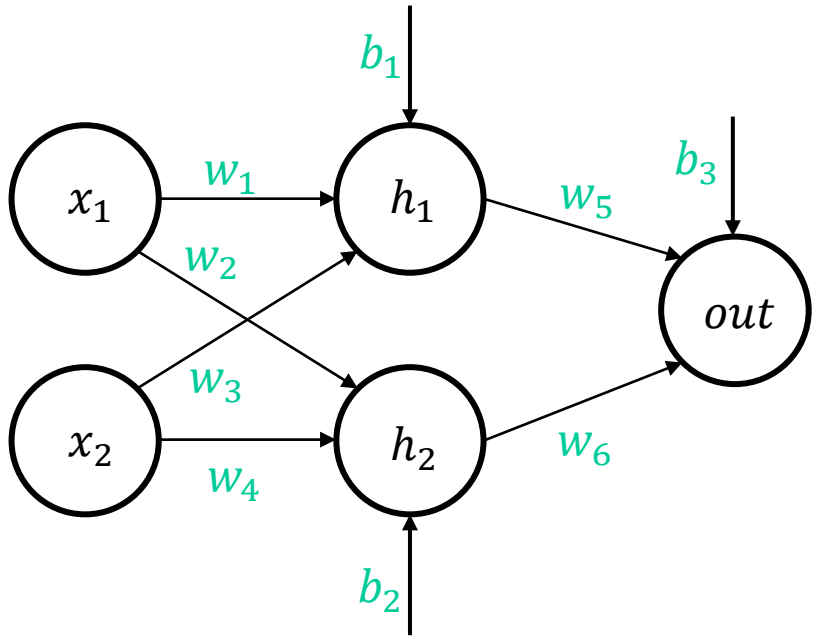
The idea of an activation function is inspired by the analysis of how neurons operate in the human brain. A neuron becomes active beyond a certain threshold, commonly referred to as the **activation potential**. Additionally, the activation function typically maps the output to a limited range in most cases. Sigmoid, hyperbolic tangent, ReLU, and ELU are the most popular activation functions.

Why do we need non-linear activation functions?

A neural network without a non-linear activation function is essentially equivalent to a linear regression model, regardless of the number of hidden layers.

The non-linear activation function applies a non-linear transformation to the input, enabling the network to learn and perform more complex tasks.

We will now examine an example to illustrate the importance of a non-linear activation function in a neural network using the following network.



Why do we need non-linear activation functions?

Based on the neural network, we can conclude the following statements:

$$out = w_5.h_1 + w_6.h_2 + b_3$$
$$h_1 = w_1.x_1 + w_3.x_2 + b_1$$
$$h_2 = w_2.x_1 + w_4.x_2 + b_2$$

Therefore, we can rewrite out as the following:

$$out = w_1'.x_1 + w_2'.x_2 + b'$$

Where:

$$w_1' = w_1.w_5 + w_2.w_6$$
$$w_2' = w_3.w_5 + w_4.w_6$$
$$b' = w_5.b_1 + w_6.b_2 + b_3$$

Based on the calculations, we see that the result is a linear function. It is discernible that even if we have several other hidden layers, only the final weights and bias will change, but the equation will continue to be of linear form.

Why do we need non-linear activation functions?

If we pass this output to a linear activation function, we will have a linear output. Let us assume the linear activation function we are using is the following:

$$y = a.x + b$$

Where the terms a and b are arbitrary constants. Now, we pass the result of the network to this activation function.

$$y = a.out + b$$

We substitute out with its value:

$$y = a.(w_1'.x_1 + w_2'.x_2 + b') + b$$

We rewrite the equation:

$$y = w_1''.x_1 + w_2''.x_2 + b''$$

Where:

$$w_1'' = a.w_1'$$
$$w_2'' = a.w_2'$$
$$b'' = a.b' + b$$

Why do we need non-linear activation functions?

Based on the calculations in this example, we observe that after passing the output of the network through a linear activation function, the result remains in linear form.
We can conclude that regardless of the number of hidden layers in the network and the arbitrary values used in our linear activation function, the result will maintain the same form. This is because the composition of two linear functions is itself a linear function.

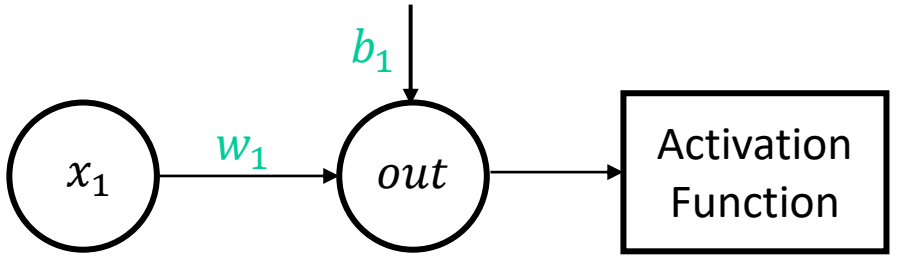
Now, what is the problem with the results always being in linear form? Networks cannot learn effectively with only a linear function attached to them. Most of the time, networks require non-linearity to better fit the given data; therefore, we need to employ a non-linear activation function.

How does a non-linear activation function introduce non-linearity?

In the previous example, we demonstrated that using a linear activation function results in a linear output, even with multiple hidden layers.

Now, we will examine how a non-linear activation function can yield a non-linear output for the network. In this example, we will use the hyperbolic tangent (tanh) as our activation function.

Let us consider the following network:



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

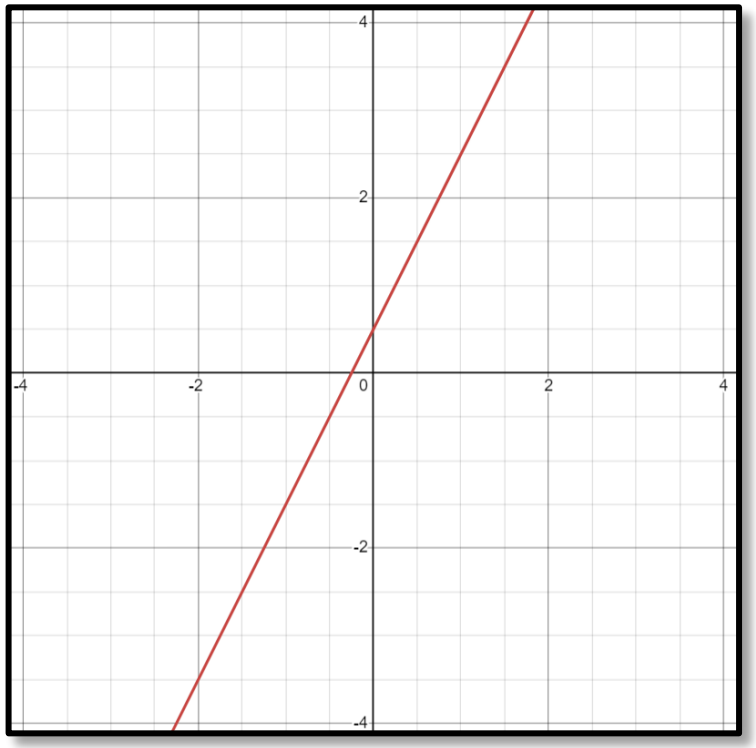
Where x is the input and e is a mathematical constant approximately equal to 2.71828.

How does a non-linear activation function introduce non-linearity?

Based on the neural network, we can conclude the following:

$$out = x_1.w_1 + b_1$$

Now, let us take a look at the graph for this function, assuming $w_1 = 2$ and $b_1 = 0.5$.
As we can see, this produces a linear graph. Next, we pass this output to the activation function to see how it transforms the graph.

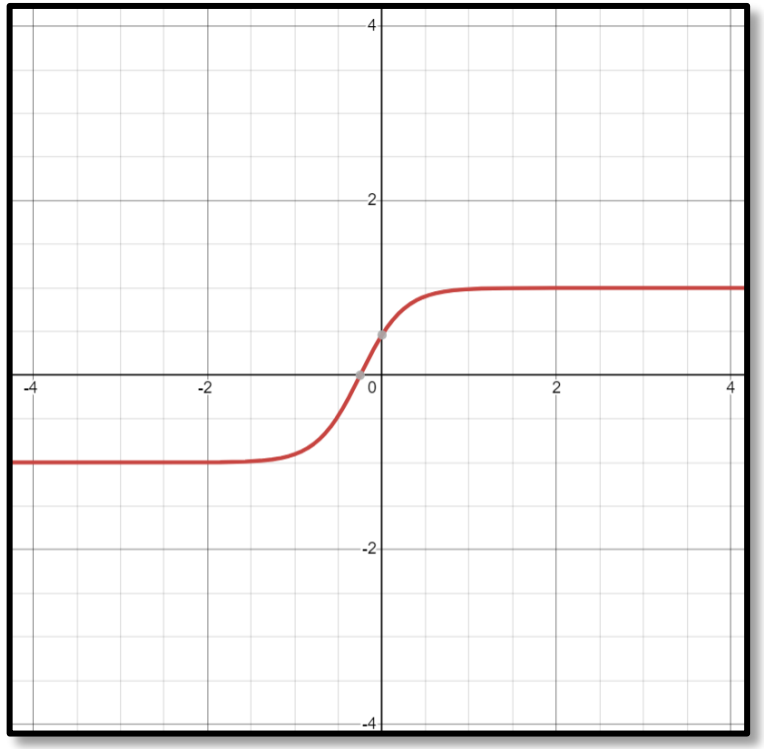


How does a non-linear activation function introduce non-linearity?

Now, we pass the result of out, which is $2x_1 + 0.5$ to our non-linear activation function, hyperbolic tangent, and take a look at the graph.

$$y = \tanh(2x_1 + 0.5)$$

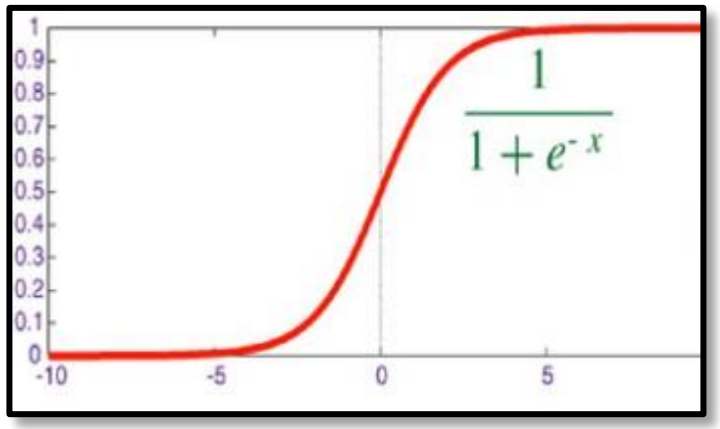
As we can see, after applying the hyperbolic tangent, the graph is no longer linear. This demonstrates how using a non-linear activation function introduces non-linearity into a neural network.



How would different activation functions evolve the output of a network?

1. Sigmoid

So far, we have examined the hyperbolic tangent as an activation function and its resulting graph. Now, we will take a look at other popular activation functions, such as Sigmoid, ReLU and ELU.
First, let us consider the Sigmoid function as our activation function. The main reason the Sigmoid function is widely used is that its output value always lies between 0 and 1. Therefore, it is a suitable choice for models where we need to predict probabilities as outputs.



Here is the equation for the Sigmoid function:

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

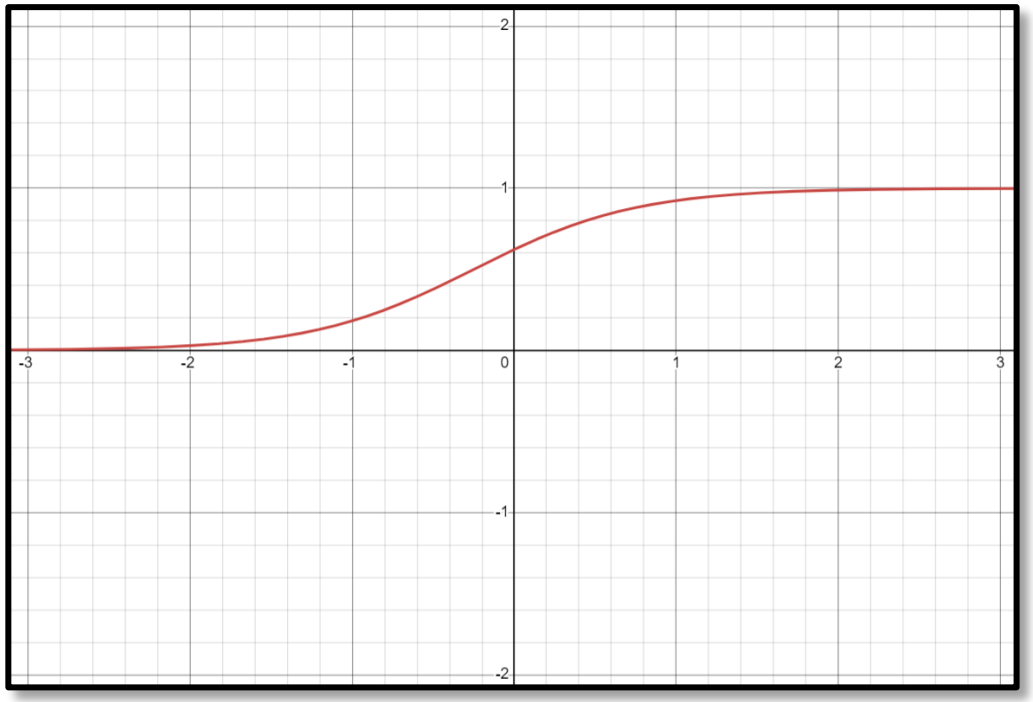
How would different activation functions evolve the output of a network?

1. Sigmoid

We pass the output of a network to this activation function to see the result. We will assume the output of our network is the same as the previous example, $2x_1 + 0.5$. We can conclude the following:

$$S = \frac{1}{1 + e^{-(2x_1+0.5)}}$$

Now, we will take a look at the result graph. As we can see, the graph is non-linear and the output of Sigmoid is between 0 and 1.

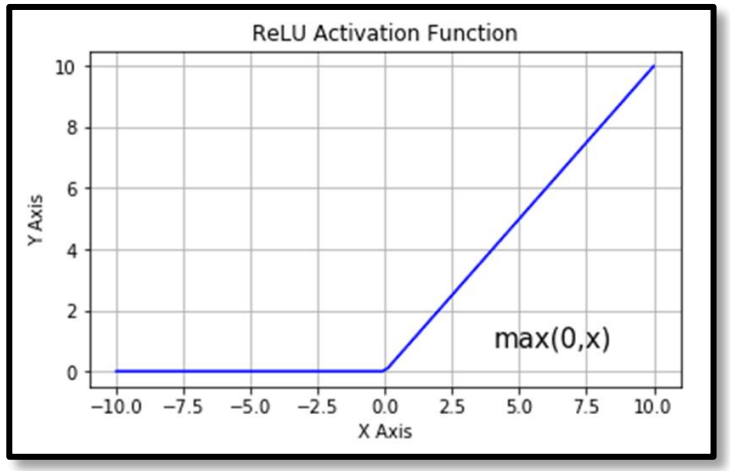


How would different activation functions evolve the output of a network?

2. ReLU

Now, we will take a look at ReLU, another non-linear activation function. The ReLU activation function works by applying a simple mathematical operation to the input value; if the input is greater than or equal to 0, the output is equal to the input value itself; and if the input value is less than 0, the output is zero.

ReLU activation function is important in deep learning models because it helps in overcoming the vanishing gradient problem, which can hinder the learning process in deep neural networks.



Here is the equation for the Sigmoid function:

$$f(x) = \max(0, x)$$

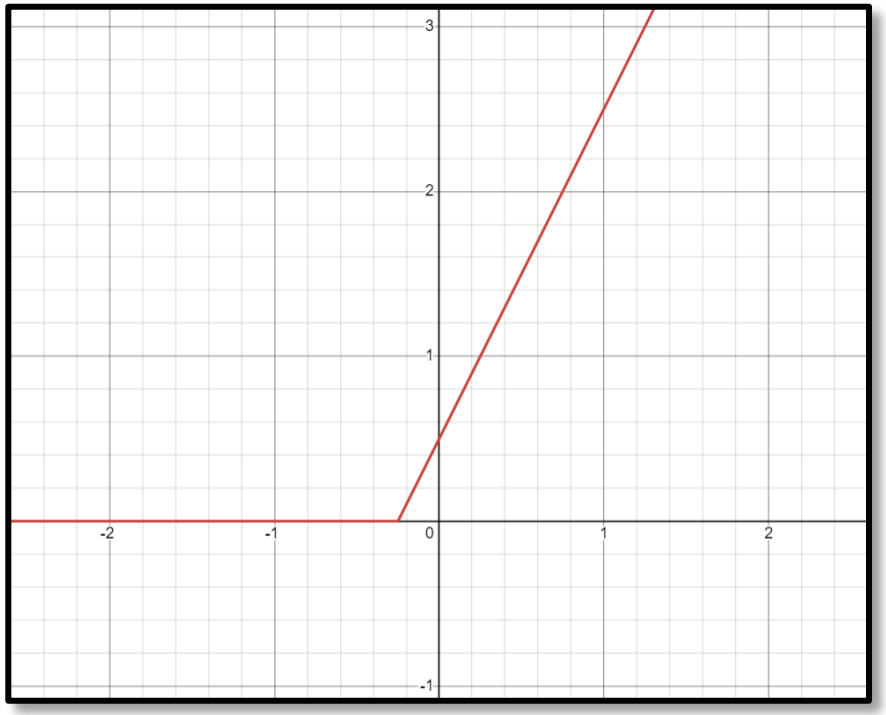
How would different activation functions evolve the output of a network?

2. ReLU

We pass the output of our network, $2x_1 + 0.5$, to this activation function to see the result. We can conclude the following:

$$f = \max(0, 2x_1 + 0.5)$$

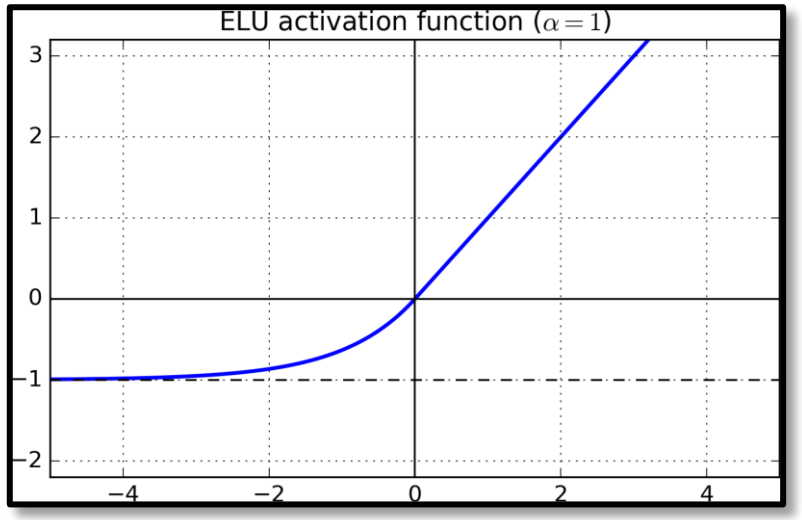
Now, we will take a look at the result graph. As we can see, the graph is non-linear and the output is zero when the input is negative.



How would different activation functions evolve the output of a network?

3. ELU

Now, let us examine the Exponential Linear Unit (ELU) as another non-linear activation function.
In contrast to ReLU, ELUs have negative values, which allows them to push the mean of activations closer to zero. Having the mean of activations closer to zero also leads to faster learning and convergence.
Let us take a look at the ELU graph, for $\alpha = 1$.



Here is the equation for the ELU function for $\alpha > 0$:

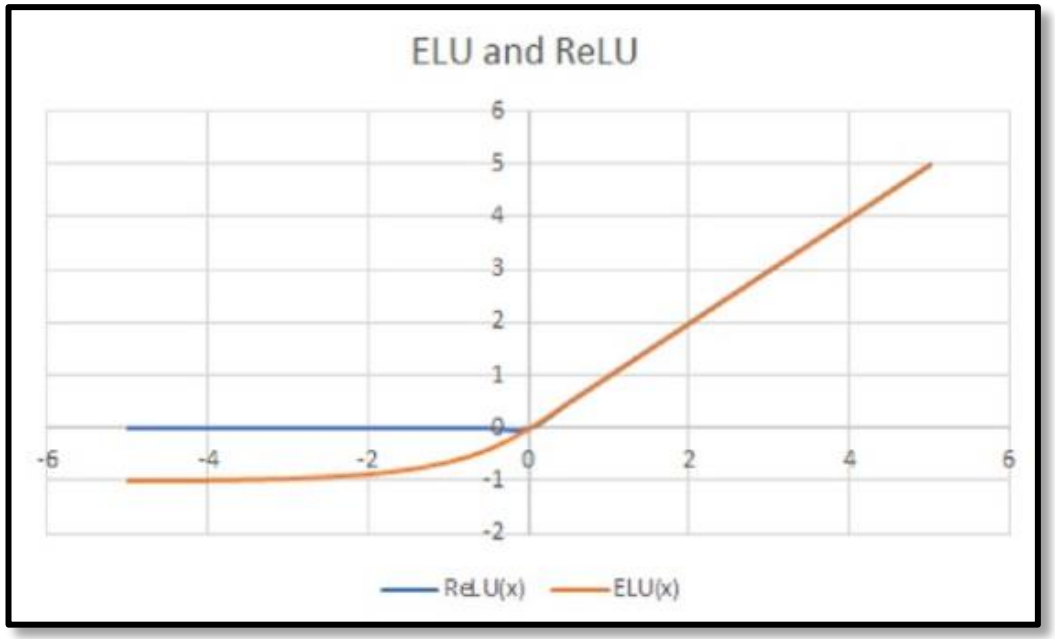
$$f(x) = \begin{cases} x & x > 0 \\ \alpha e^x - 1 & o.w. \end{cases}$$

How would different activation functions evolve the output of a network?

3. ELU

Now that we have discussed ReLU and ELU, let us examine a graph that illustrates the differences between the two activation functions in a more distinctive way.

As we can see, ReLU does not allow negative output values, whereas the ELU function permits negative values. However, when the input is greater than 0, the two graphs coincide.



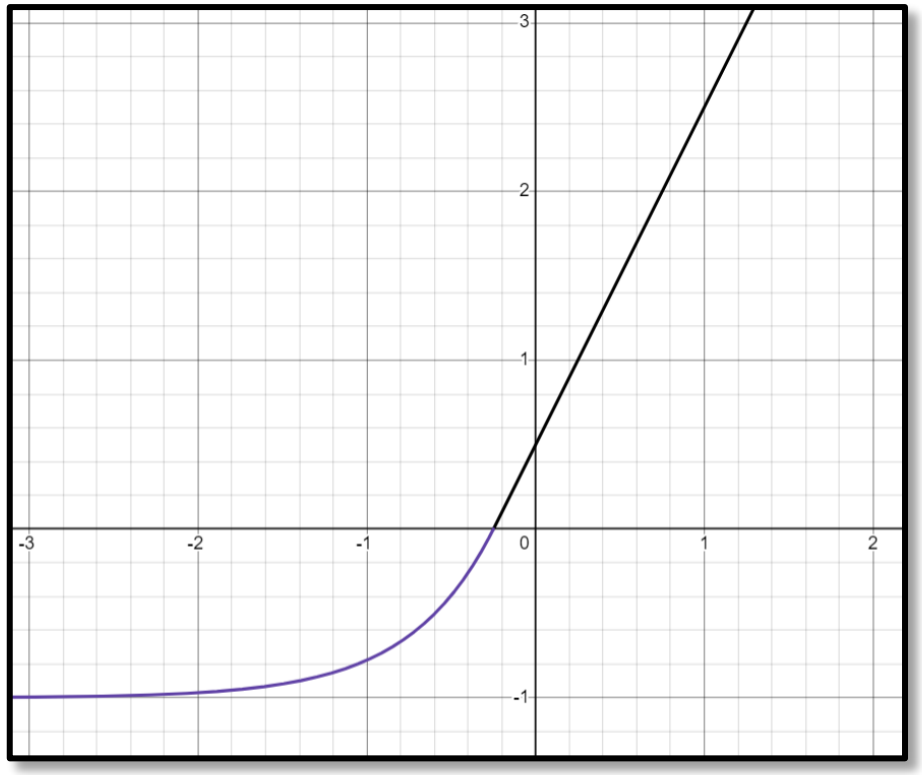
How would different activation functions evolve the output of a network?

3. ELU

Now, we pass the output of our network, $2x_1 + 0.5$, to ELU activation function and see the result. We assume $\alpha = 1$, and we can conclude the following:

$$f = \begin{cases} 2x_1 + 0.5 & 2x_1 + 0.5 > 0 \\ e^{2x_1+0.5} - 1 & o.w. \end{cases}$$

Now, we will take a look at the result graph. As we can see, the graph is non-linear and the output is zero when the input is negative.



Evaluating activation functions in TensorFlow

Now that we know the math behind different activation functions, let us see the codes to use these activation functions.
We first import the necessary modules.

```
import tensorflow as tf
import numpy as np
sess = tf.Session()
```

Next, we want to see the result of hyperbolic tangent for a given input.

```
E = tf.nn.tanh([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(E))
```

Here is the result of the code:

```
[ 1.         0.9640276  0.7615942  0.46211717 0.         -0.46211717
 -0.7615942  -0.9640276  -1.        ]
```

Evaluating activation functions in TensorFlow

Now, we take the same steps, using the Sigmoid function.

```
J = tf.nn.sigmoid([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(J))
```

Here is the result of the code:

```
[9.9995458e-01 8.8079703e-01 7.3105860e-01 6.2245935e-01 5.0000000e-01
 3.7754068e-01 2.6894143e-01 1.1920292e-01 4.5397872e-05]
```

Lastly, we want to see the result of ReLU for a given input.

```
A = tf.nn.relu([-2,1,-3,13])
print(sess.run(A))
```

Here is the result of the code:

```
[ 0  1  0 13]
```