Fuzzy Logic & Neural Networks (CS-514)

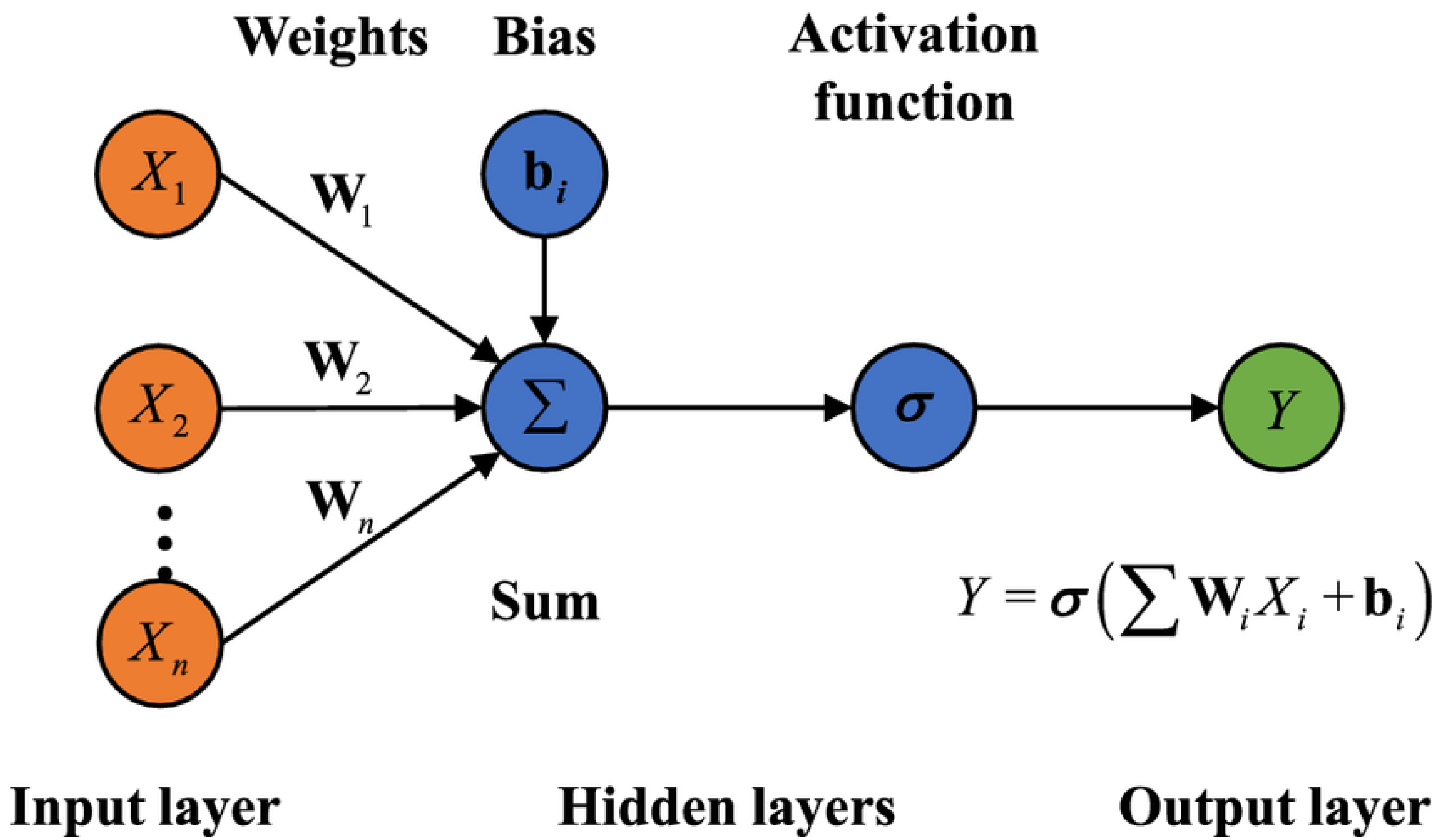
Dr. Sudeep Sharma

IIIT Surat

sudeep.sharma@iiitsurat.ac.in

Artificial Neuron Model

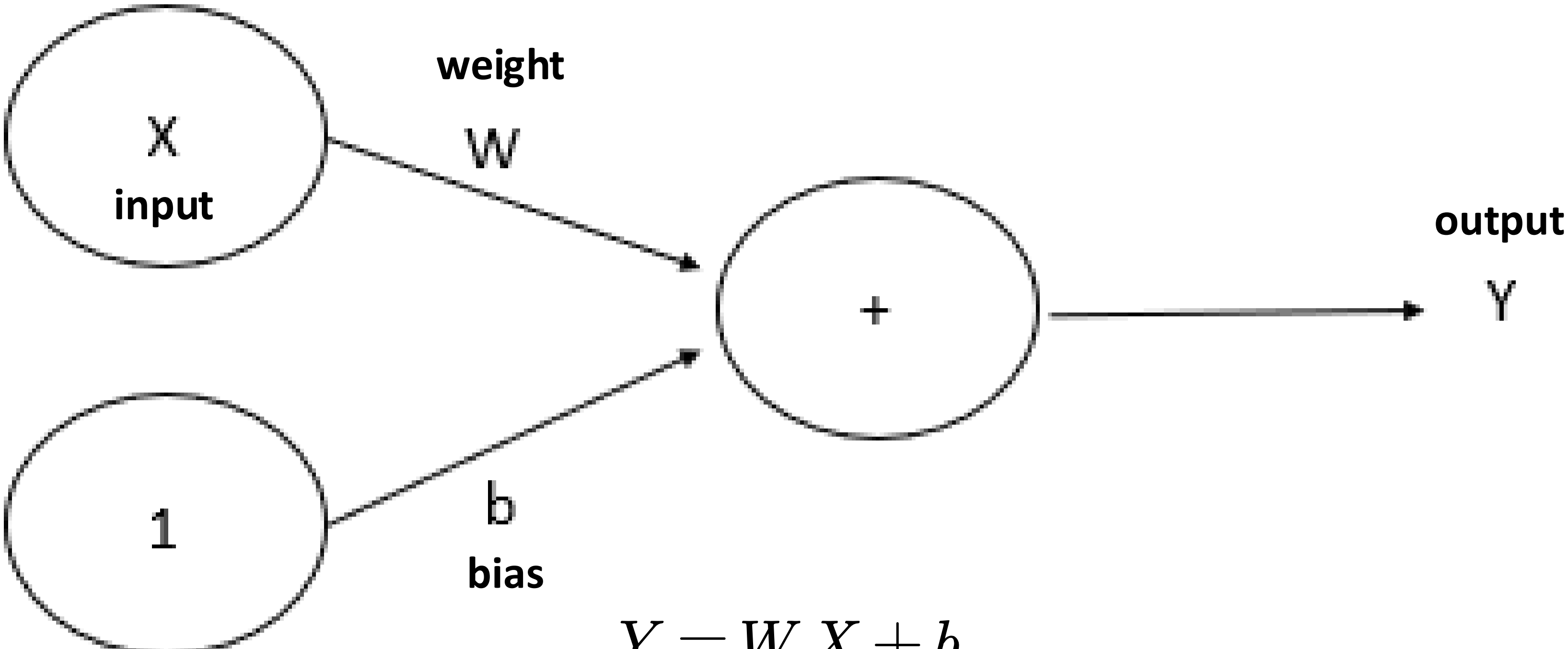
Model of an artificial neuron



Source: Google Images

Artificial Neuron Model

Simplest model of an artificial neuron



$$Y = W.X + b$$

$$y = m.x + c$$

Source: Google Images

Artificial Neuron Model

Implementing Simple Logic Circuits: AND Gate

2 - input AND gate

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Fig: Two Input AND Gate

Source: Google Images

Coding our First Neuron

A Single Neuron: Implementing the AND Gate:

```python
input_A = 1

input_B = 1

w_A = 0.5

w_B = 0.5

bias = -0.7

out = w_A*input_A + w_B*input_B + bias

if out<0:
  out = 0
else:
  out = 1

print(out)
```

Coding our First Neuron

A Single Neuron

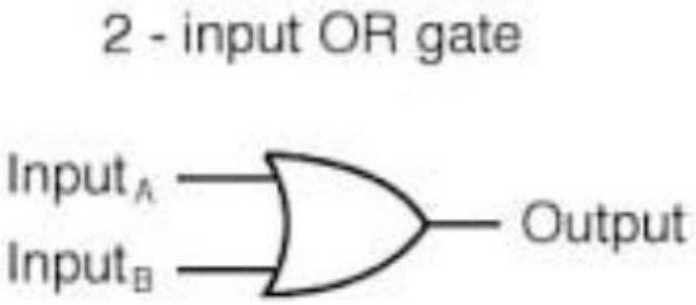
- **Implementing the AND Gate as a function:**

```
def AND_gate(input_A,input_B):

    w_A = 0.5

    w_B = 0.5

    bias = -0.7

    out = w_A*input_A + w_B*input_B + bias

    if out<0:
      return 0
    else:
      return 1
```

```
out = AND_gate(0,1)

print(out)
```

Artificial Neuron Model

Implementing Simple Logic Circuits: OR Gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Fig: Two Input OR Gate

Source: Google Images

Coding our First Neuron

A Single Neuron: Implementing the OR Gate:

```
input_A = 1

input_B = 1

w_A = 0.5

w_B = 0.5

bias = -0.4

out = w_A*input_A + w_B*input_B + bias

if out<0:
  out = 0
else:
  out = 1

print(out)
```

Coding our First Neuron

A Single Neuron

➢ **Implementing the OR Gate as a function:**

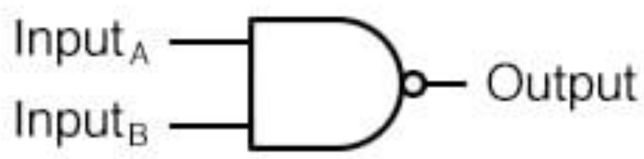
```
def OR_gate(input_A,input_B):

    w_A = 0.5

    w_B = 0.5

    bias = -0.4

    out = w_A*input_A + w_B*input_B + bias

    if out<0:
      return 0
    else:
      return 1
```

```
out = OR_gate(0,1)

print(out)
```

Artificial Neuron Model

Implementing Simple Logic Circuits: NAND Gate

2 - input NAND gate



A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Fig: Two Input NAND Gate

Source: Google Images

Coding our First Neuron

A Single Neuron: Implementing the NAND Gate:

```
input_A = 1

input_B = 1

w_A = -0.5

w_B = -0.5

bias = 0.7

out = w_A*input_A + w_B*input_B + bias

if out<0:
  out = 0
else:
  out = 1

print(out)
```

Coding our First Neuron

A Single Neuron

- Implementing the NAND Gate as a function:

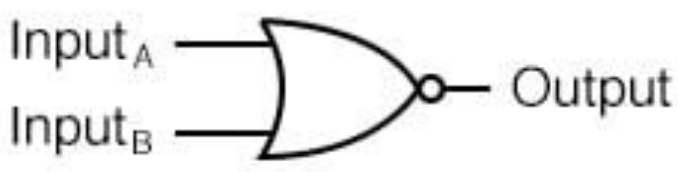
```
def NAND_gate(input_A,input_B):

    w_A = -0.5

    w_B = -0.5

    bias = 0.7

    out = w_A*input_A + w_B*input_B + bias

    if out<0:
      return 0
    else:
      return 1
```

```
out = NAND_gate(1,1)

print(out)
```

Artificial Neuron Model

Implementing Simple Logic Circuits: NOR Gate

NOR gate



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

Fig: Two Input NOR Gate

Source: Google Images

Coding our First Neuron

A Single Neuron: Implementing the NOR Gate:

```
input_A = 1

input_B = 1

w_A = -0.5

w_B = -0.5

bias = 0.4

out = w_A*input_A + w_B*input_B + bias

if out<0:
  out = 0
else:
  out = 1

print(out)
```

Coding our First Neuron

A Single Neuron

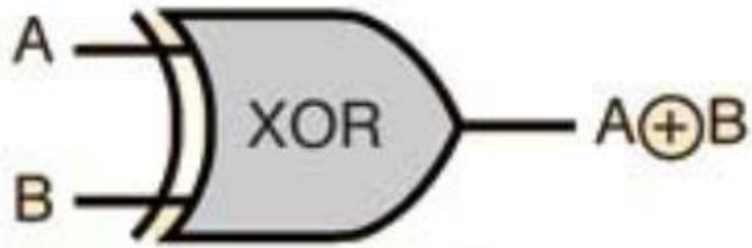
- Implementing the NOR Gate as a function:

```
def NOR_gate(input_A,input_B):

    w_A = -0.5

    w_B = -0.5

    bias = 0.4

    out = w_A*input_A + w_B*input_B + bias

    if out<0:
      return 0
    else:
      return 1
```

```
out = NOR_gate(0,1)

print(out)
```

Artificial Neuron Model

Implementing Simple Logic Circuits: XOR Gate



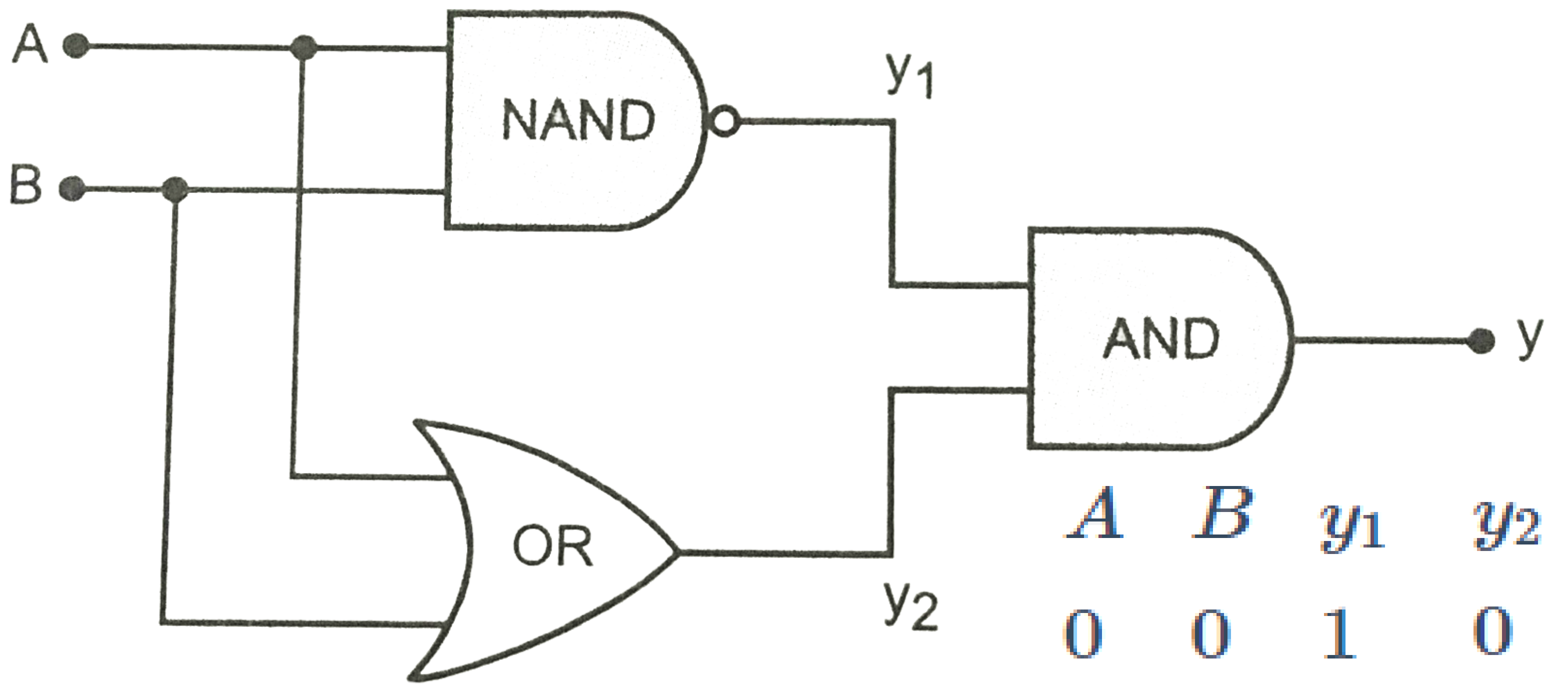
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Fig: Two Input XOR Gate

Source: Google Images

Artificial Neuron Model

Implementing Simple Logic Circuits: XOR Gate



A	B	y_1	y_2	y
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

Fig: Two Input XOR Gate using NAND, OR and AND gates

Source: Google Images

Coding our First Neuron

A Three Neuron Model

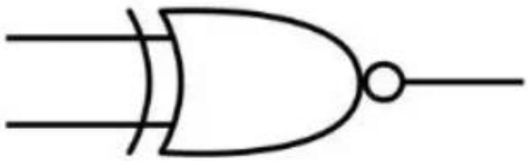
- Implementing the XOR Gate as a function:

```
def XOR_gate(input_A,input_B):

    y1 = NAND_gate(input_A,input_B)

    y2 = OR_gate(input_A,input_B)

    y = AND_gate(y1,y2)

    return y
```

```
out = XOR_gate(1,1)

print(out)
```

Artificial Neuron Model

Implementing Simple Logic Circuits: XNOR Gate



INPUT		OUTPUT
A	B	
0	0	1
1	0	0
0	1	0
1	1	1

Fig: Two Input XNOR Gate

Source: Google Images

Coding our First Neuron

Assignment: Implement the XNOR gate using simple gates.

Artificial Neuron Model

Implementing Simple Logic Circuits: XNOR Gate

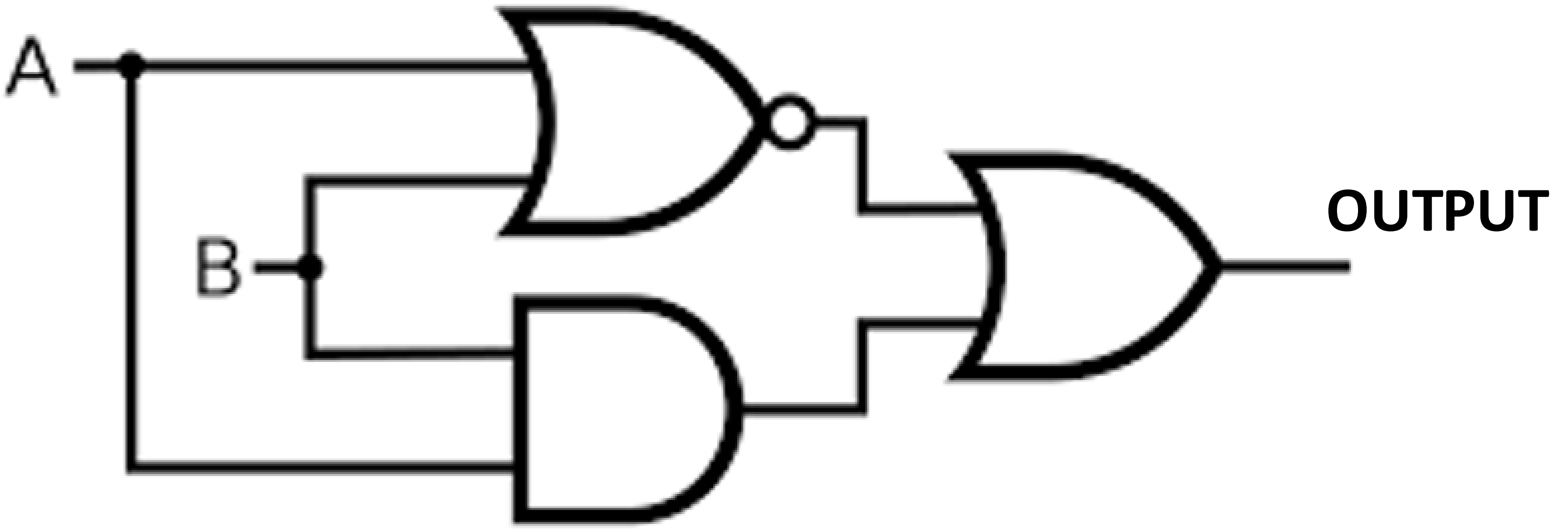


Fig: Two Input XNOR Gate using NOR, AND and OR gates

Source: Google Images

Activation Functions

Model of an artificial neuron

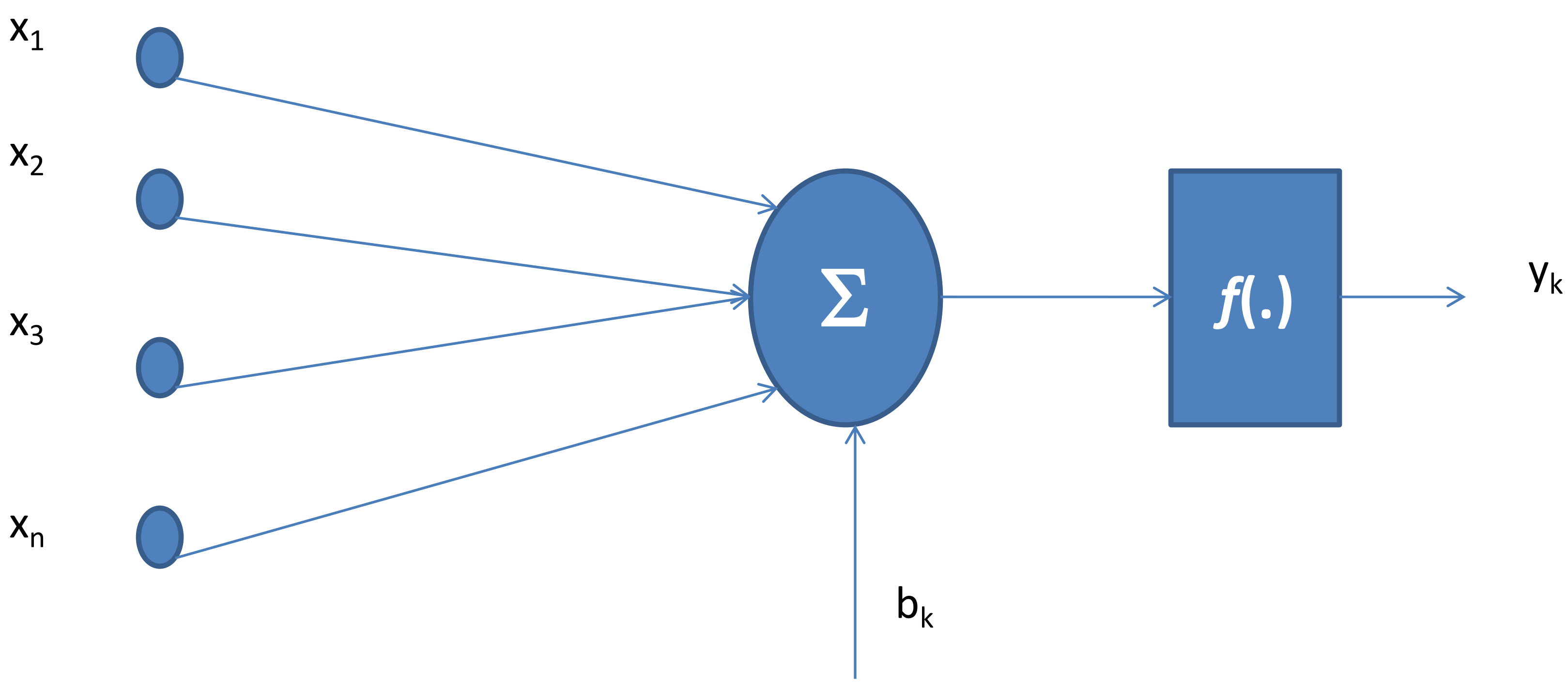


Fig: Typical Neuron Model having Activation Function

Source: Google Images

Activation Functions

About Activation Functions:

- The Activation Functions are applied to the output of a neuron.
- Activation Function modifies the neuron output.
- We use Activation Functions to introduce nonlinearity or desired mapping in the model.
- The neural networks use the activation functions in hidden layers and in the output layer.

Activation Functions

Why Use Activation Functions?

What is a nonlinear function?

A nonlinear function cannot be represented well by a straight line, such as a sine function:

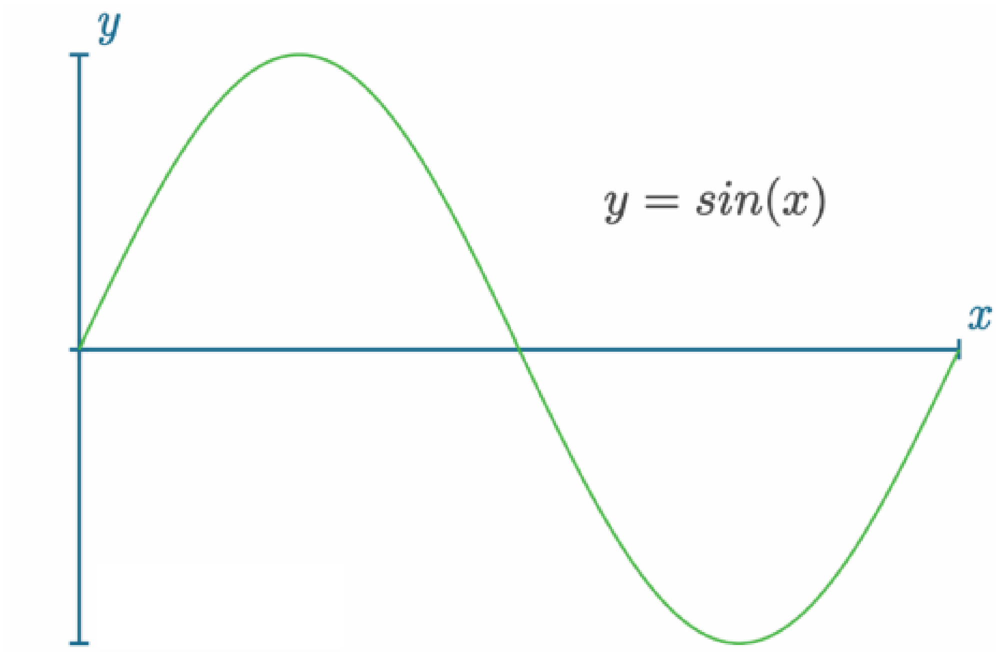


Fig: Graph of y=sin(x)

Source: Google Images

Activation Functions

The Linear Activation Function

This activation function is usually applied to the last layer's output in the case of a regression model.

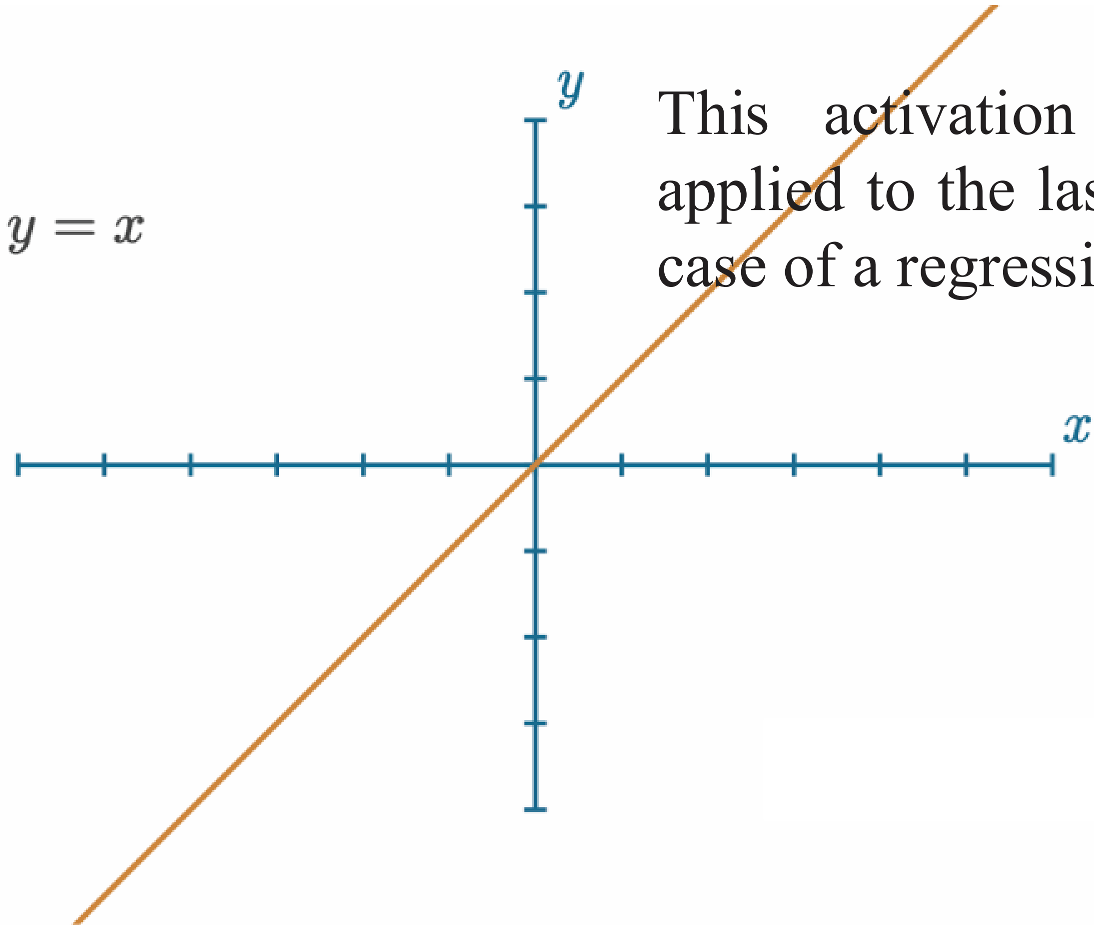


Fig: Linear function graph.

Source: Google Images

Activation Functions

The Step Activation Function:

A neuron “firing” or “not firing”

$$y = \begin{cases} 1 & x > 0 \\ 0 & x \leqslant 0 \end{cases}$$

This activation function has been used historically in hidden layers, but nowadays, it is rarely a choice.

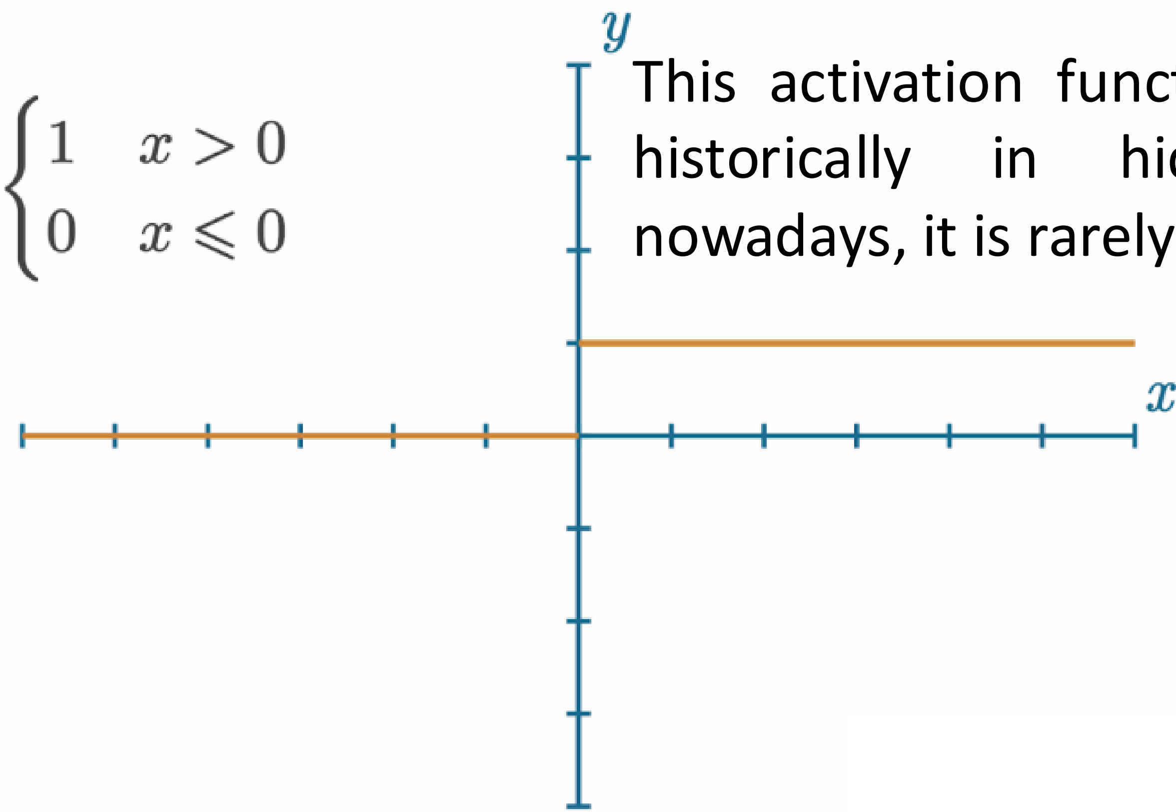


Fig: Step function graph.

Source: Google Images

Activation Functions

The Sigmoid Activation Function

$$y = \frac{1}{1 + e^{-x}}$$

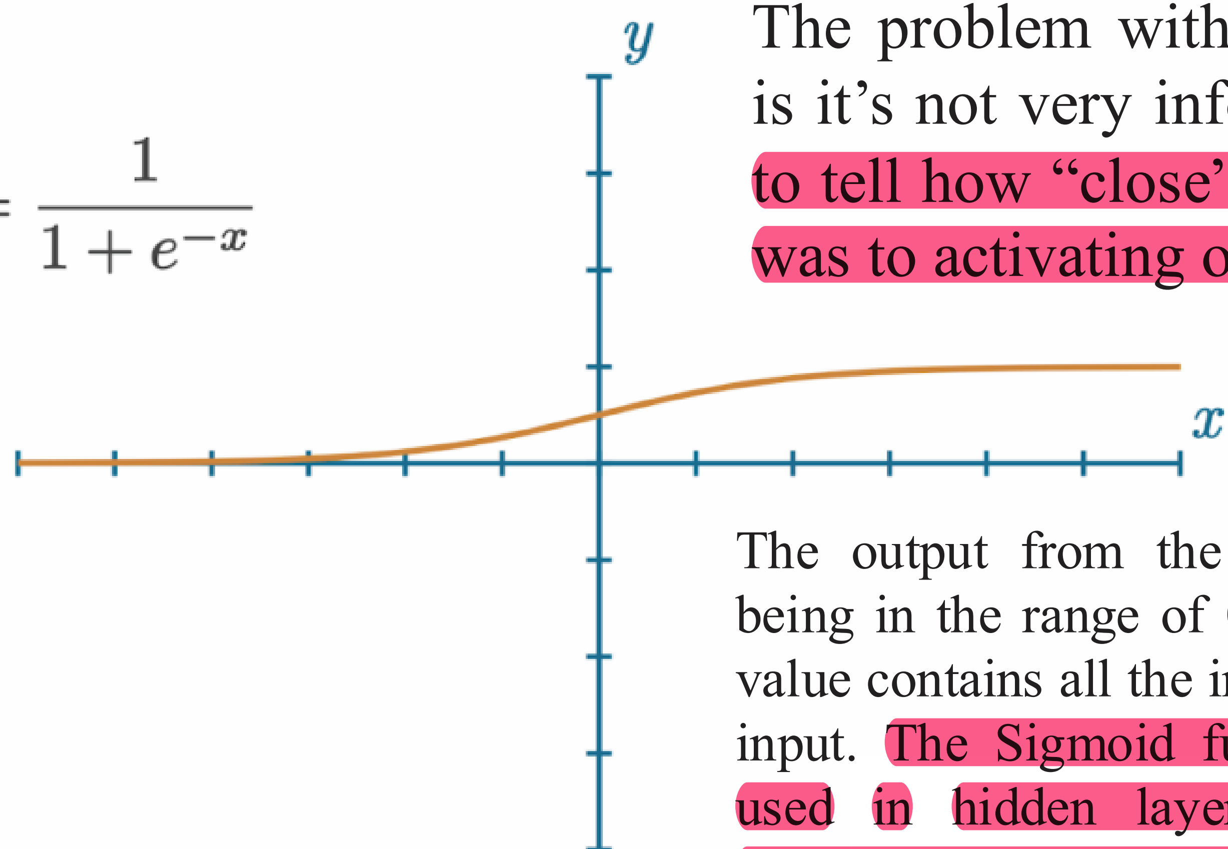


Fig: Sigmoid function graph.

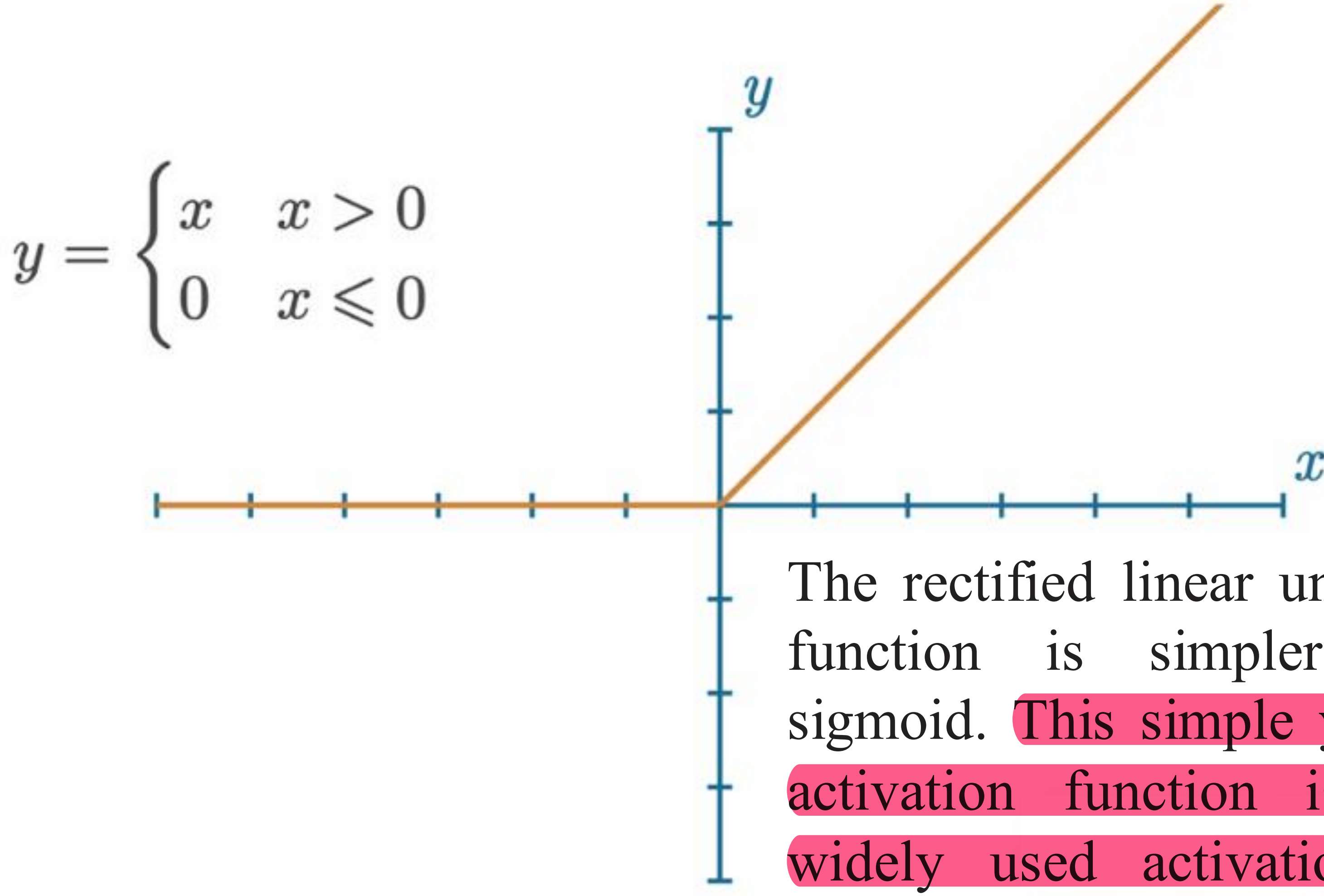
The problem with the step function is it's not very informative. It's hard to tell how "close" this step function was to activating or deactivating.

The output from the Sigmoid function, being in the range of 0 to 1, the returned value contains all the information from the input. The Sigmoid function, historically used in hidden layers, was eventually replaced by the Rectified Linear Units activation function or ReLU.

Source: Google Images

Activation Functions

The Rectified Linear Unit (ReLU) Activation Function



The rectified linear unit activation function is simpler than the sigmoid. This simple yet powerful activation function is the most widely used activation function, mainly due to speed and efficiency.

Fig: Graph of the ReLU activation function

Source: Google Images

Activation Functions

The Leaky Rectified Linear Activation Function

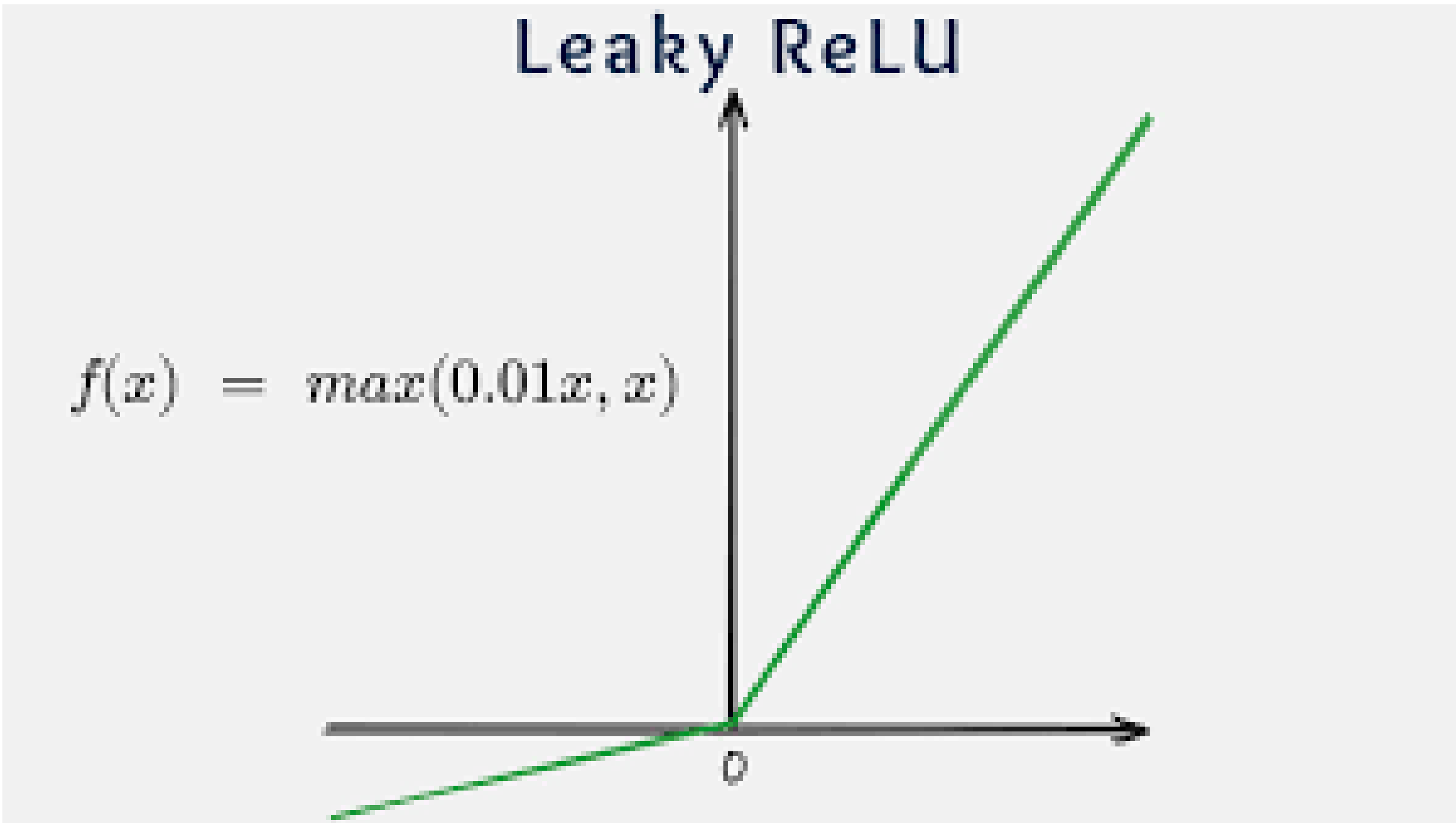


Fig: Graph of the Leaky ReLU activation function

Source: Google Images

Activation Functions

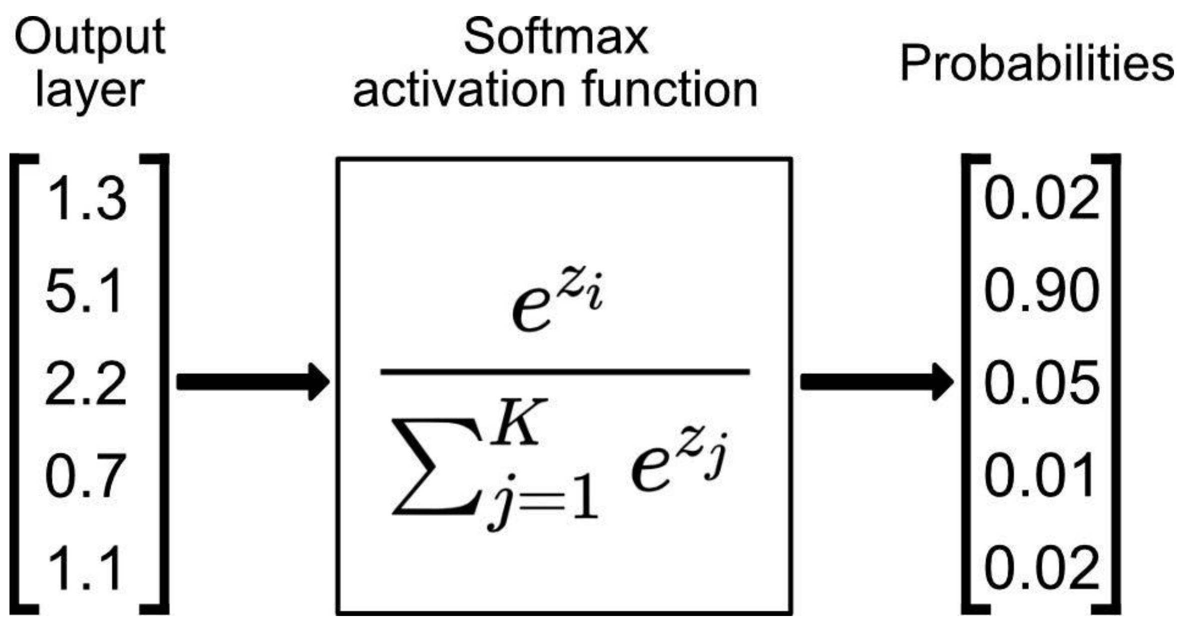
The SoftMax Activation Function

- For a classification problem, the SoftMax activation function is used.
- In case of classification, we want to see that for which class the input represents.
- SoftMax activation function represents confidence scores for each class and will add up to 1.
- The predicted class is associated with the output neuron that returned the largest confidence score.

Activation Functions

The Softmax Activation Function

$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



Source: Google Images

Activation Functions

ReLU Activation with a single Neuron

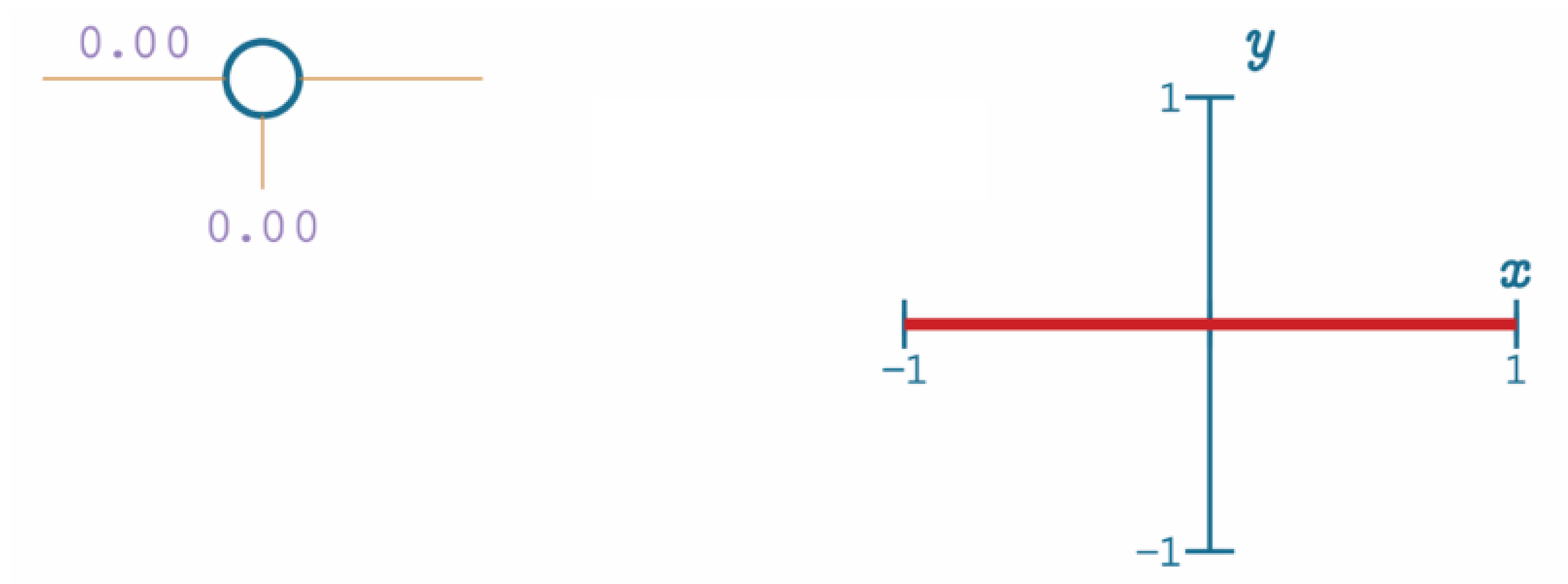


Fig: Single neuron with single input (zeroed weight) and ReLU activation function

Activation Functions

ReLU Activation with a single Neuron

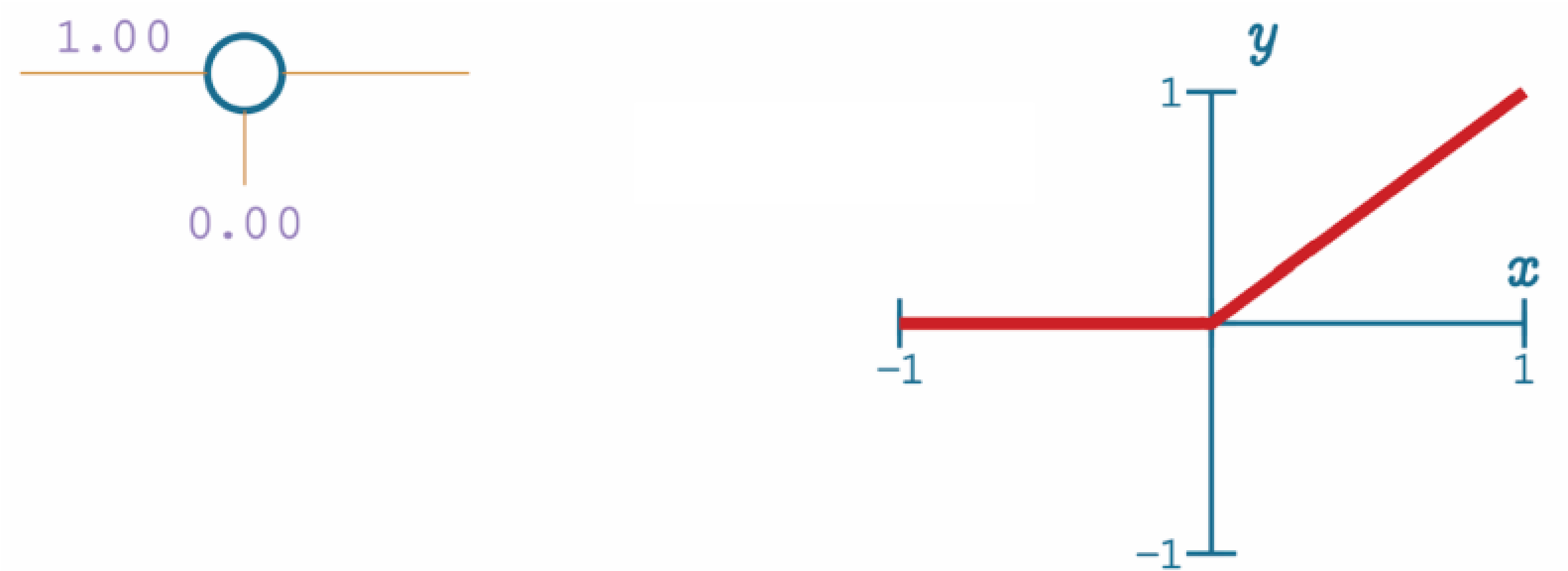


Fig: Single neuron with single input and ReLU activation function, weight set to 1.0.

Source: Google Images

Activation Functions

ReLU Activation with a single Neuron

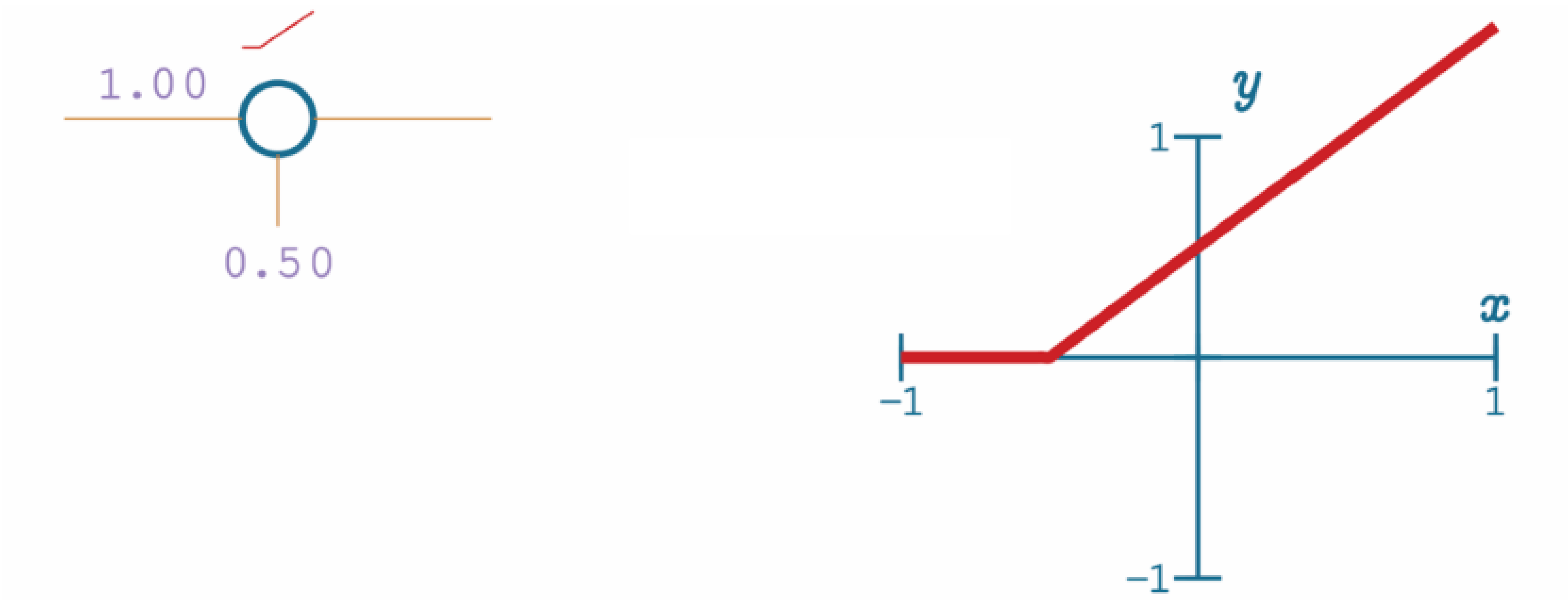


Fig: Single neuron with single input and ReLU activation function.

Source: Google Images

Activation Functions

ReLU Activation with a single Neuron

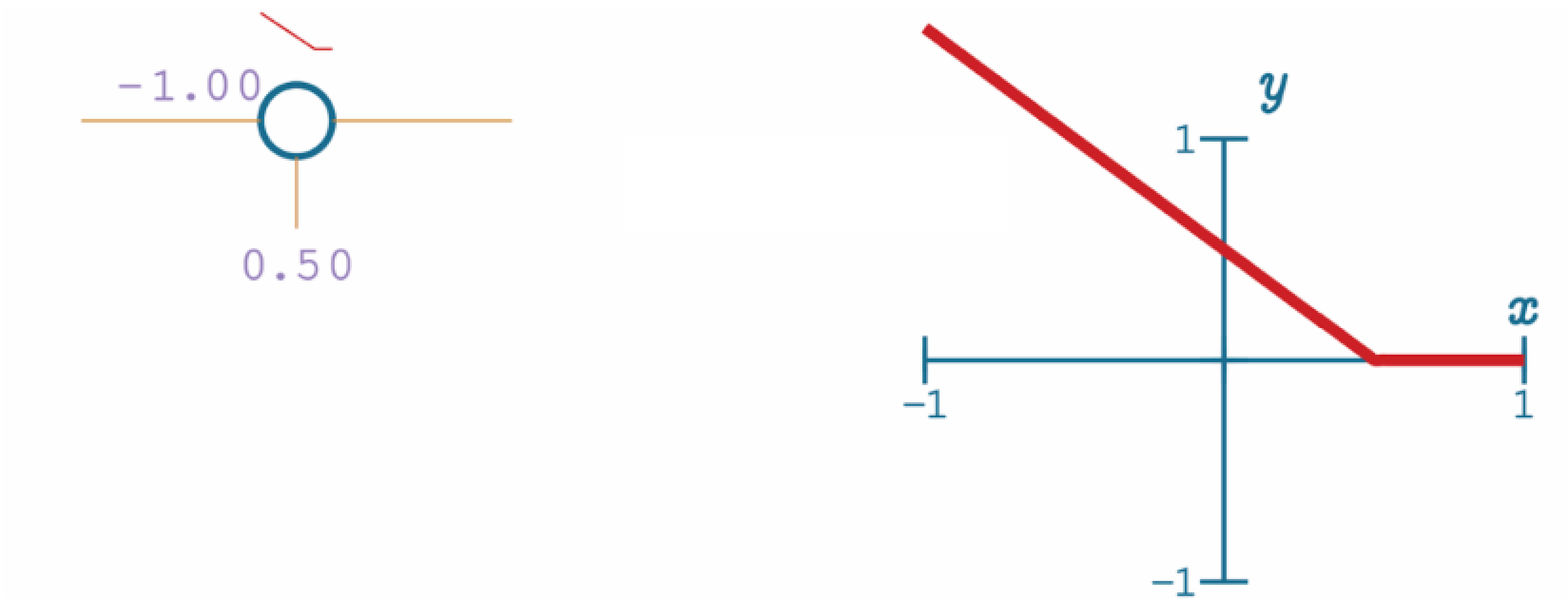


Fig: Single neuron with single input and ReLU activation function, bias applied.

Source: Google Images

Activation Functions

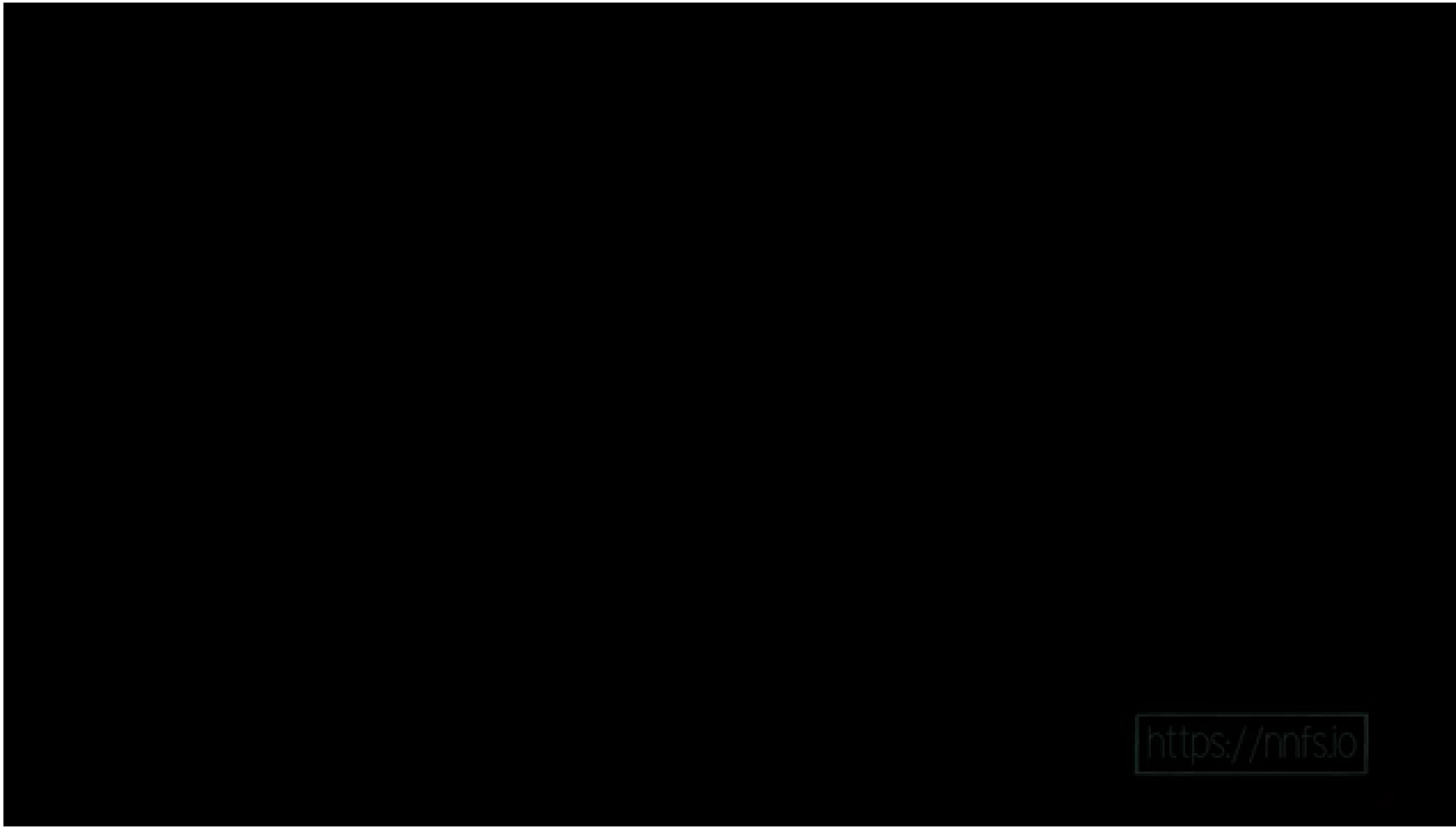
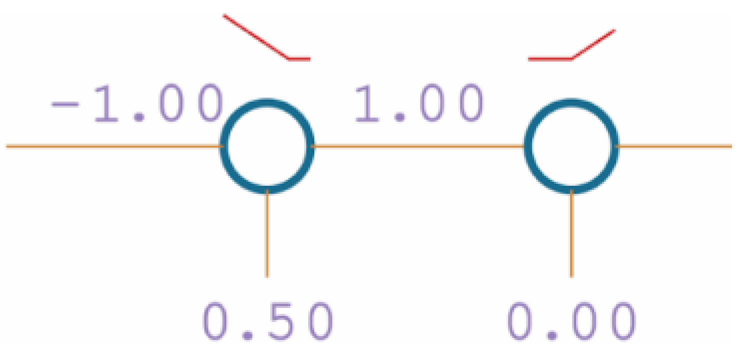


Fig: Single neuron with single input and ReLU activation function, bias applied.

Source: Google Images

Activation Functions

ReLU Activation in a Pair of Neurons



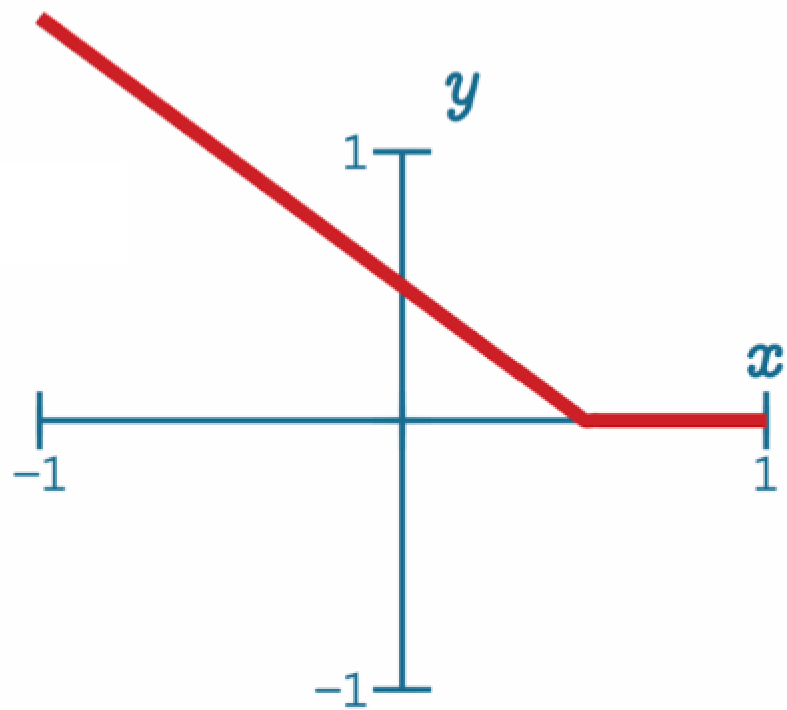
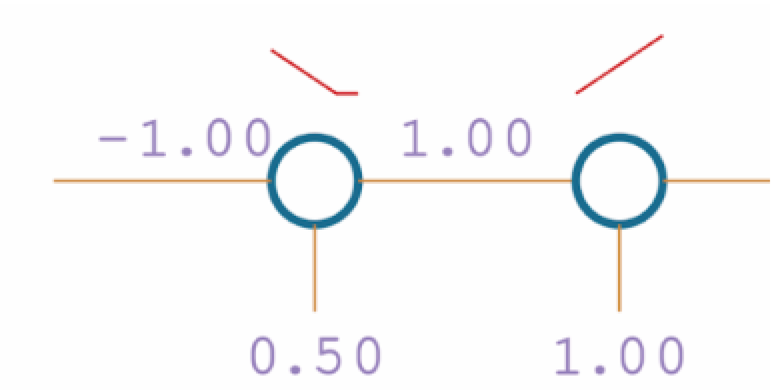


Fig: Pair of neurons with single inputs and ReLU activation functions.

Source: Google Images

Activation Functions

ReLU Activation in a Pair of Neurons



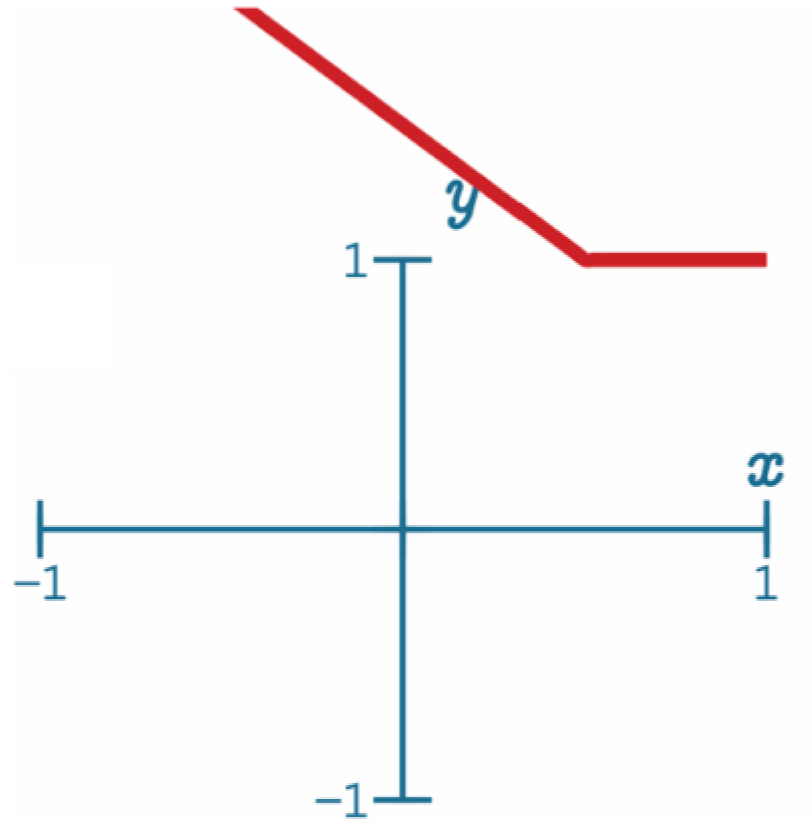


Fig: Pair of neurons with single inputs and ReLU activation functions, other bias applied.

Source: Google Images

Activation Functions

Why Use Activation Functions?

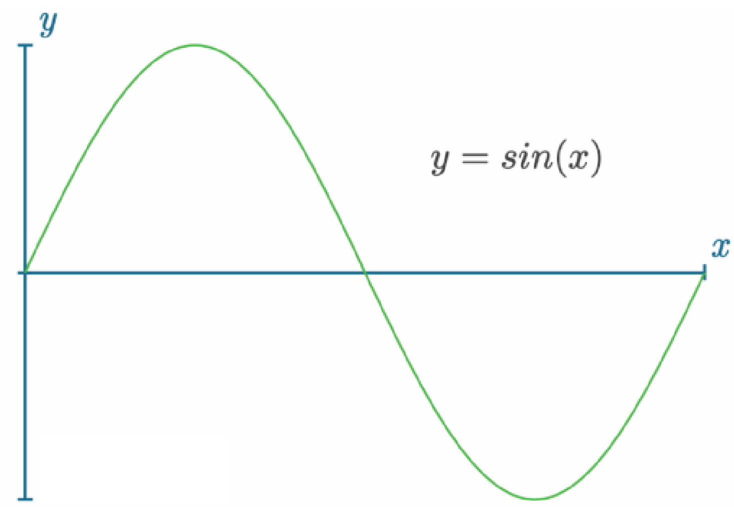


Fig : To fit sin(x) function

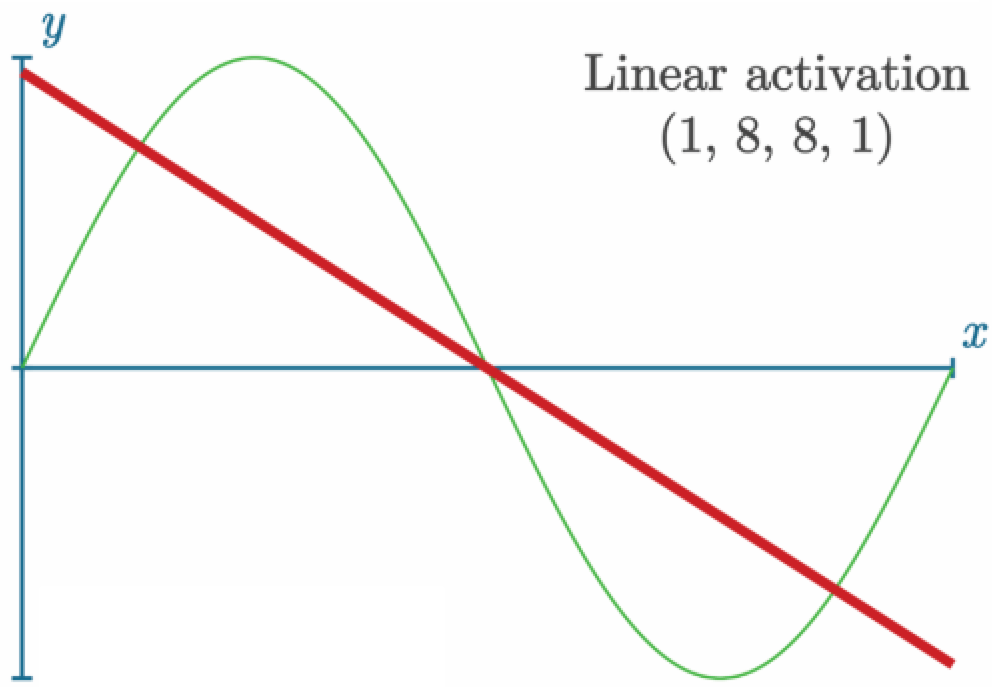
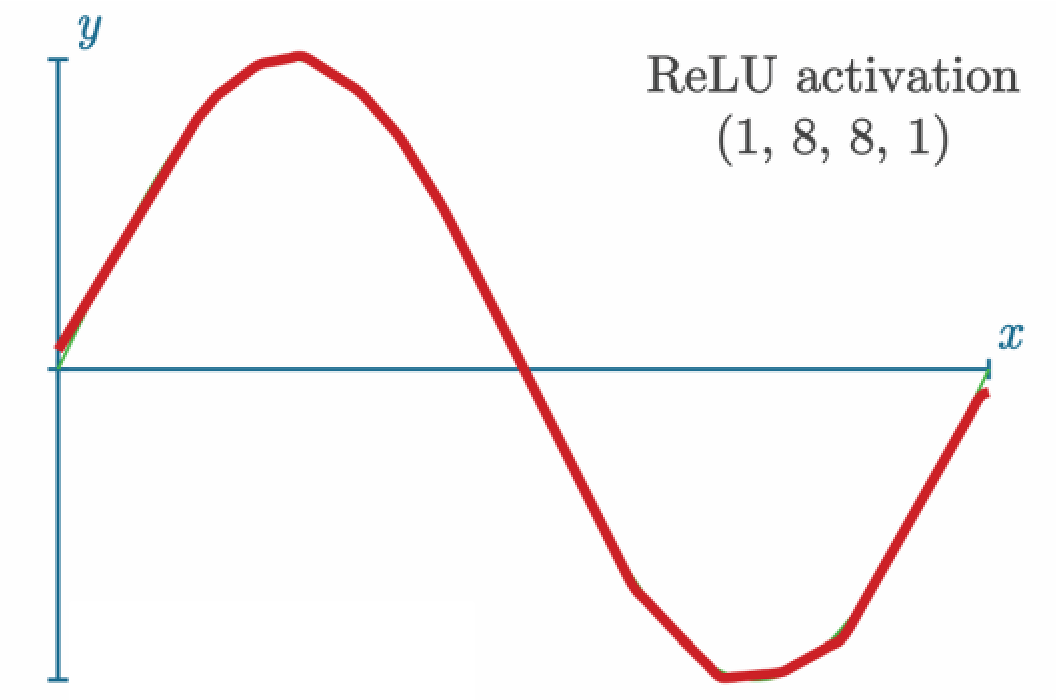


Fig : The simple mapping process to fit sin(x) function

Source: Google Images

Activation Functions

Using the ReLU Activation Function.



Source: Google Images

Activation Functions

ReLU Activation in the Hidden Layers

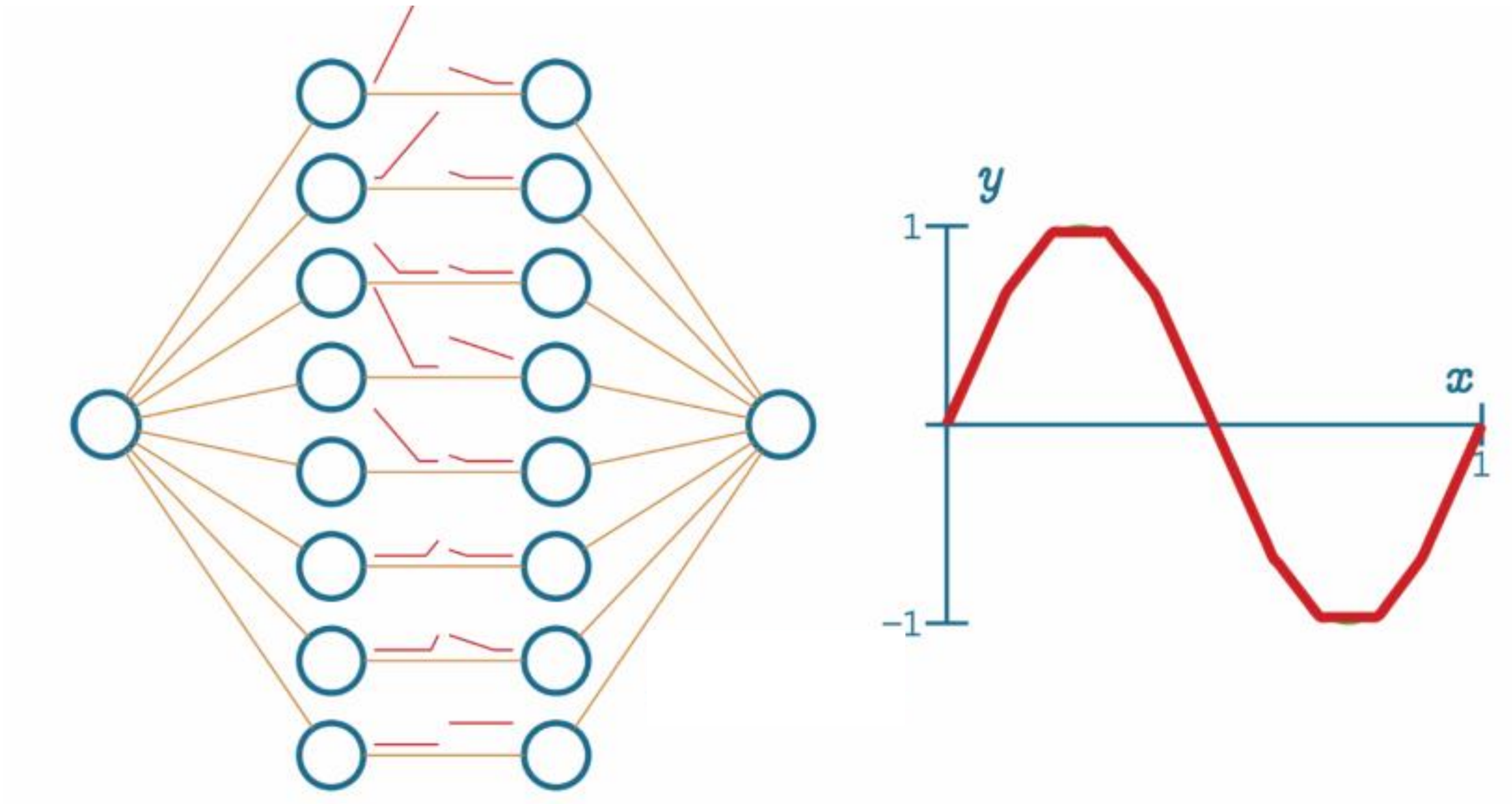


Fig : The simple mapping process to fit sin(x) function

Source: Google Images

Activation Functions

ReLU Activation in the Hidden Layers

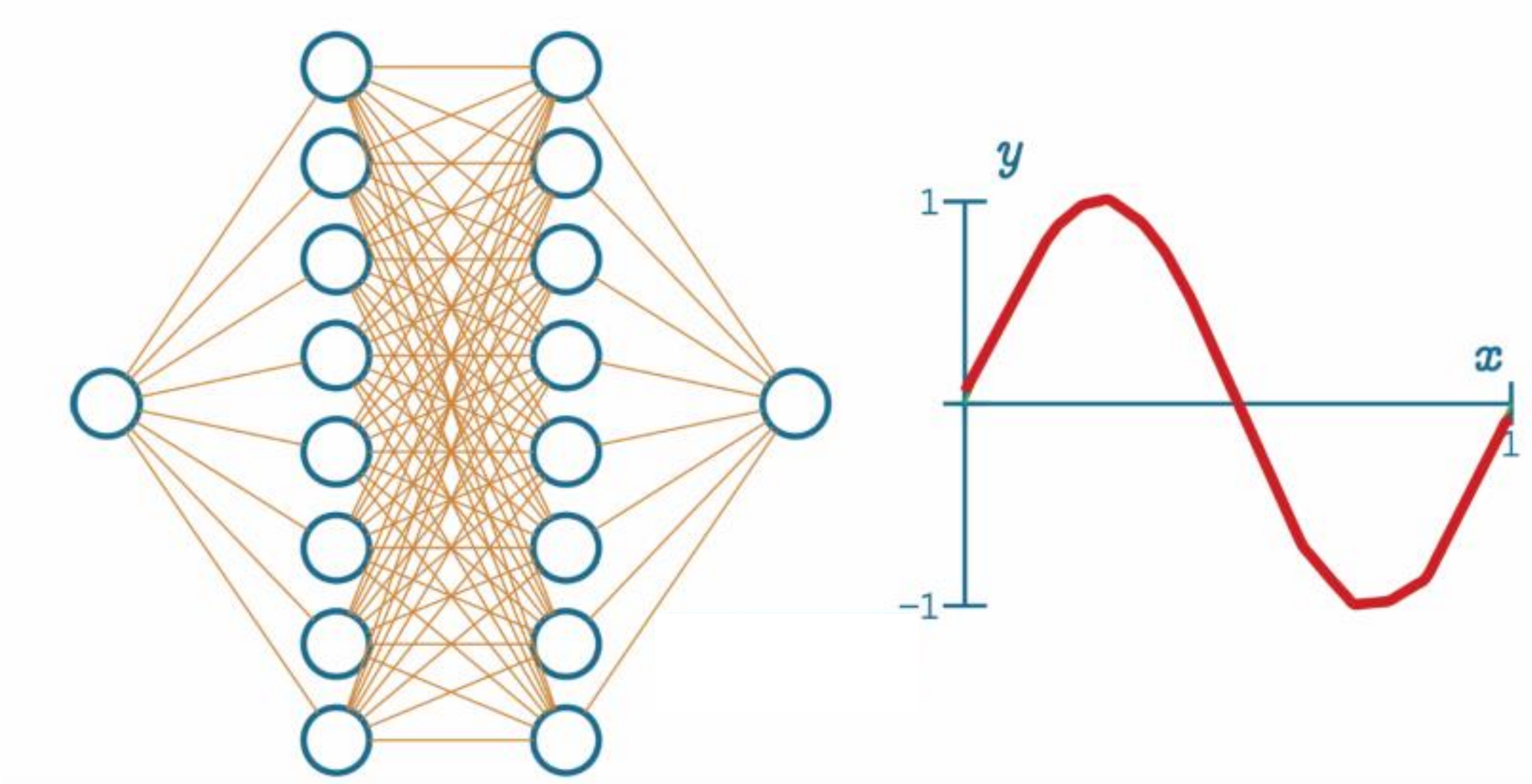


Fig : Example of fitment after fully-connecting the neurons and using an optimizer.

Source: Google Images

Activation Functions

ReLU Activation in the Hidden Layers

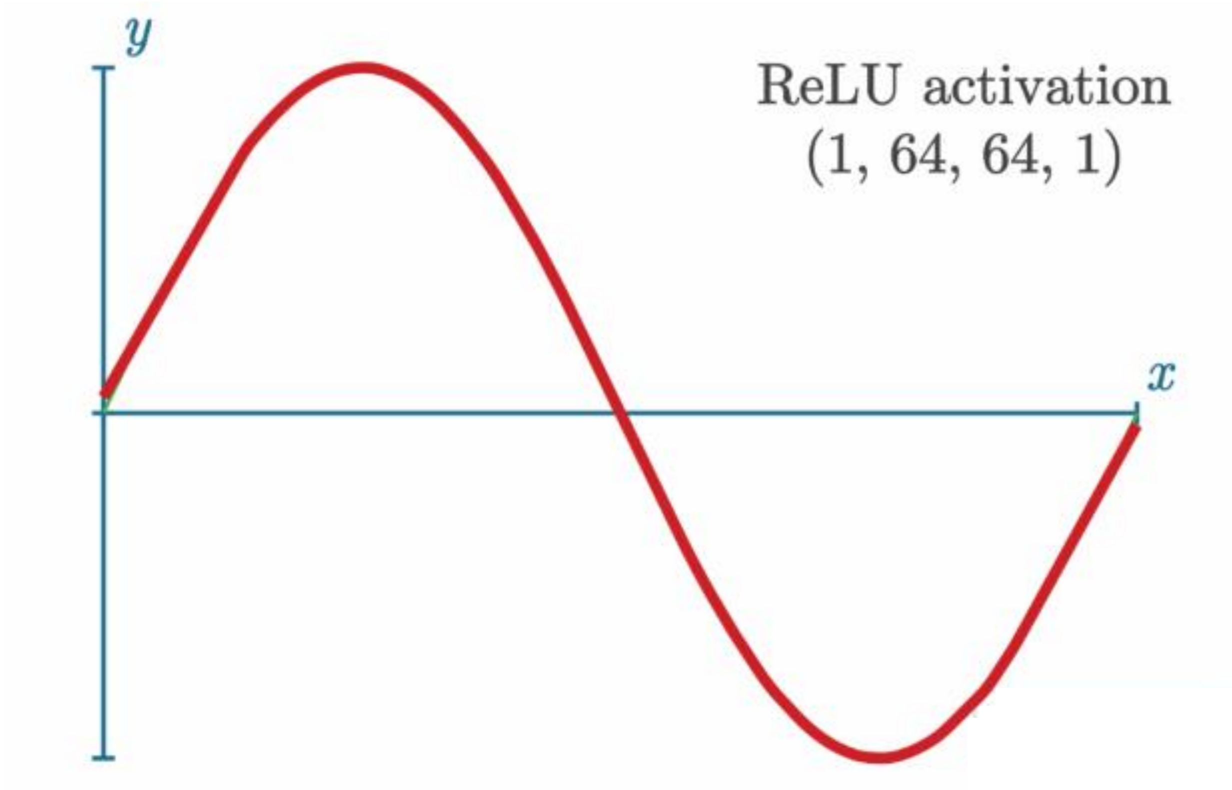


Fig : Fitment with 2 hidden layers of 64 neurons each, fully connected, with optimizer.