Deep Learning Using TensorFlow

Dr. Ash Pahwa

Lesson 3: Neural Networks + TensorFlow

Lesson 3.2: Neural Networks Math

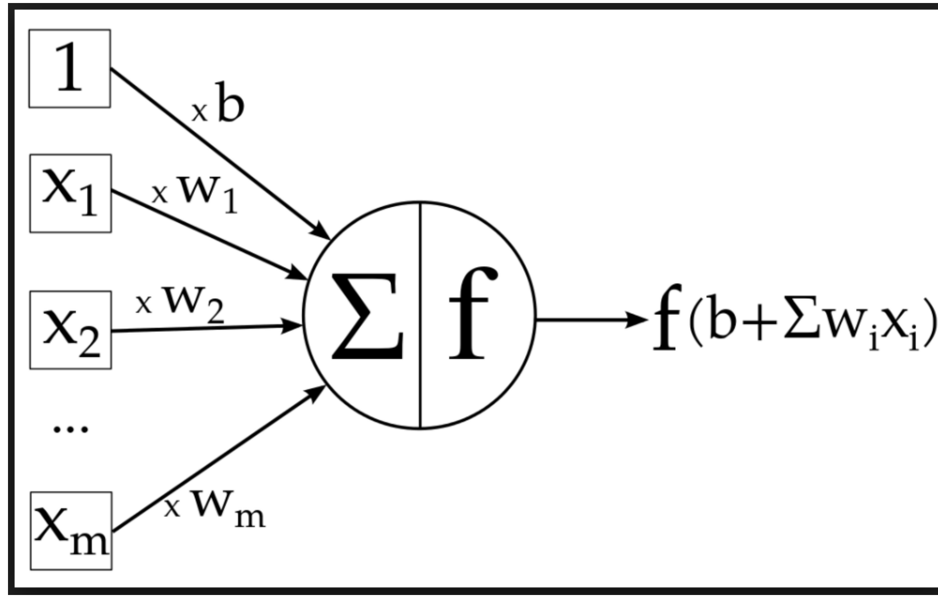
Outline

- Neuron Functions
- Activation Functions
  - Unit Step Function
  - Sigmoid Function
  - Rectified Linear Unit Function (ReLU)
- Feed Forward Fully Connected - Neural Network
- Computing the Layer Output Using TensorFlow
- Solution of XOR Problem
  - Logical XOR Gate
  - Hidden Layer Solution to XOR Problem

Neuron Function

Single Neuron

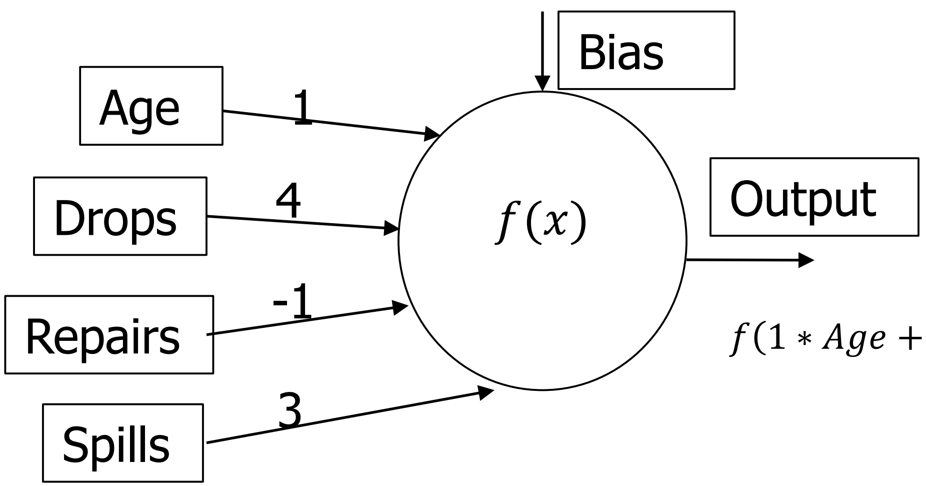
- $Inputs: x_1, x_2, \dots x_m$
- $Weights: w_1, w_2, \dots w_m$
- $Bias = b$
- $Activation\ Function = f(x)$
- $Output = y$



$$Activation\ Function$$
$$y = f(x_1 w_1 + x_2 w_2 + x_3 w_3 + \cdots + x_m w_m + \mathrm{b})$$

Example of a Single Neuron

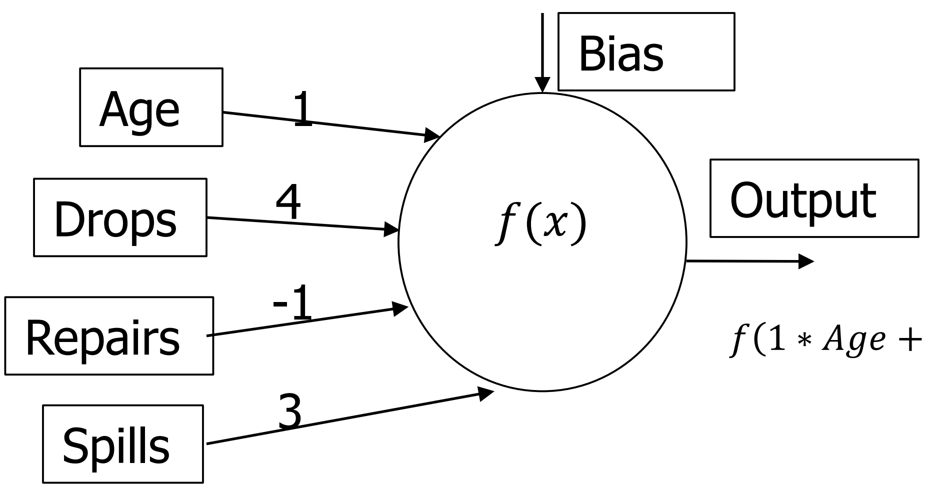
	Variables	Laptop Death	Weight
1	Age	Age on years	1
2	Drops	# of times laptop has been dropped	4
3	Repairs	# of times laptop has been repaired	-1
4	Spills	# of times liquid was spilled on laptop	3



$f(1 * Age + 4 * Drops - 1 * Repairs + 3 * Spills)$

Example of a Single Neuron

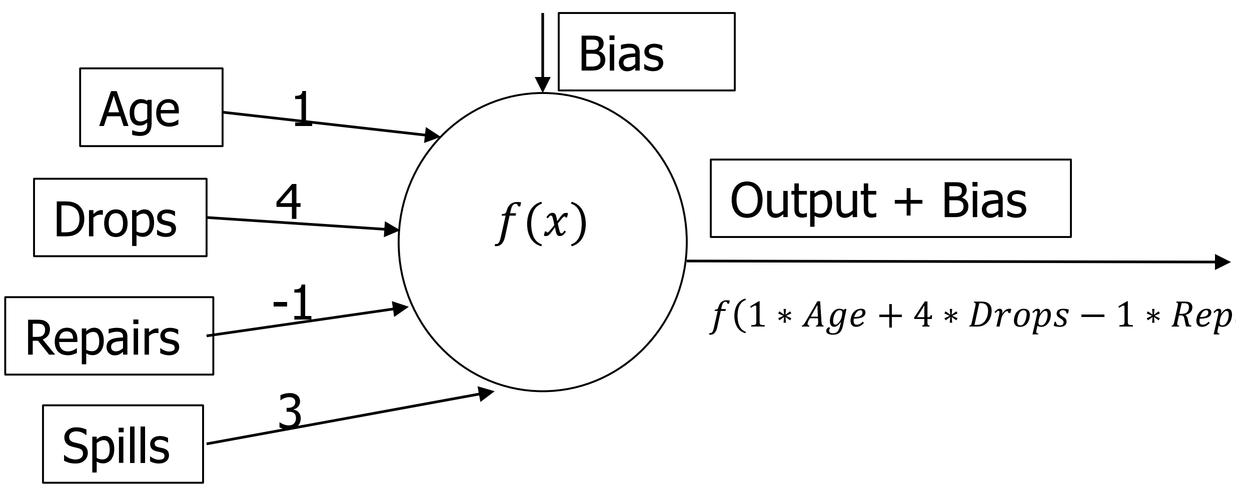
	A	B	C	D	E	F
1	**Training Data**					
2		**Age**	**Drops**	**Repairs**	**Spills**	**Output Function**
3	1	3	0	1	0	2
4	2	1	0	0	0	1
5	3	2	1	2	1	7
6	4	4	0	3	0	1
7						
8	Weight - Age	1				
9	Weight - Drops	4				
10	Weight - Repairs	-1				
11	Weight - Spills	3				
12						



$$f(1 * Age + 4 * Drops - 1 * Repairs + 3 * Spills)$$

Example of a Single Neuron

	A	B	C	D	E	F	G	H
1	**Training Data**							
2		**Age**	**Drops**	**Repairs**	**Spills**	**Output Function**	**Bias**	**Output+ Bias**
3	1	3	0	1	0	2	-2	0
4	2	1	0	0	0	1	-2	-1
5	3	2	1	2	1	7	-2	5
6	4	4	0	3	0	1	-2	-1
7								
8	Weight - A	1						
9	Weight - D	4						
10	Weight - R	-1						
11	Weight - S	3						

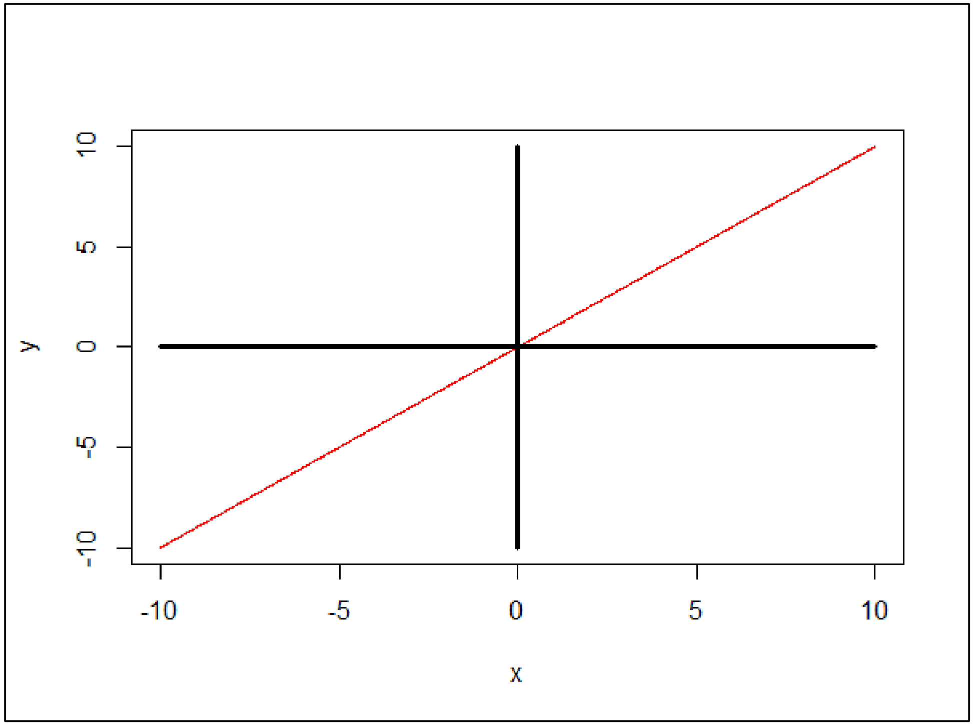


$$f(1 * Age + 4 * Drops - 1 * Repairs + 3 * Spills) + Bias$$

Activation Functions

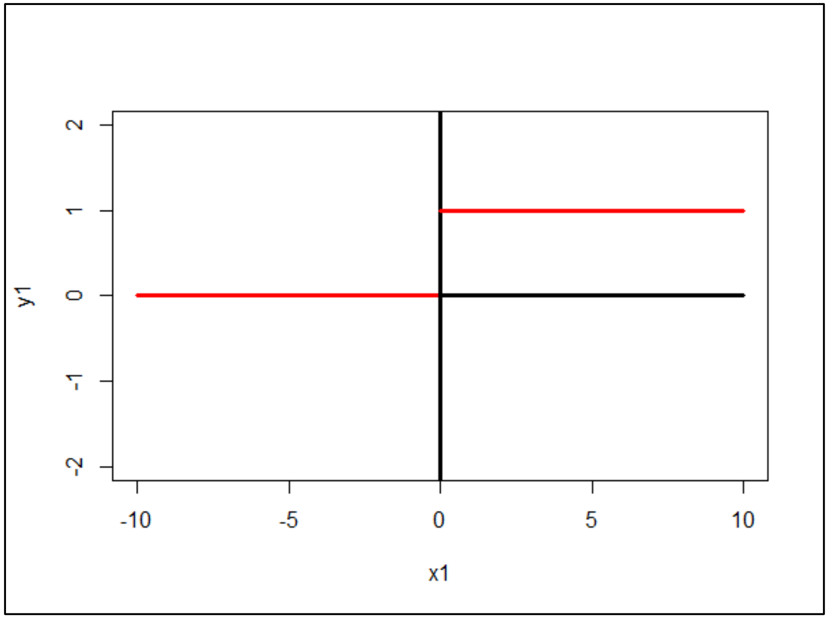
Linear Function

- $y = f(x) = x$



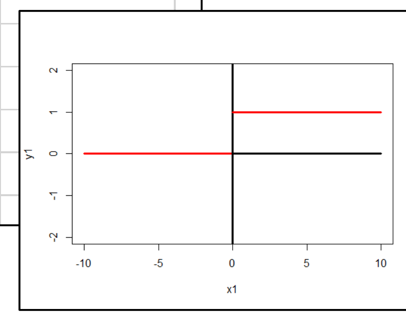
Unit Step Function

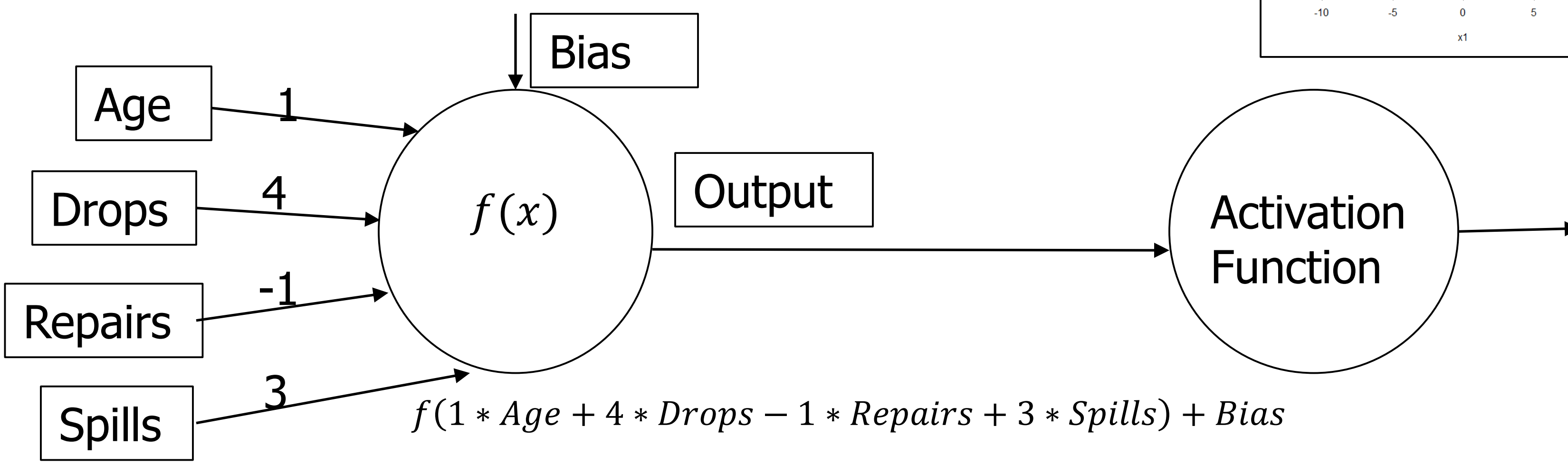
- $y = f(x) = 0 \; when \; x < 0$
- $y = f(x) = 1 \; when \; x > 0$



Neuron + Unit Step Activation Function

	A	B	C	D	E	F	G	H	I
1	**Training Data**								
2		**Age**	**Drops**	**Repairs**	**Spills**	**Output Function**	**Bias**	**Output + Bias**	Activation Function - Unit Step
3	1	3	0	1	0	2	-2	0	0
4	2	1	0	0	0	1	-2	-1	0
5	3	2	1	2	1	7	-2	5	1
6	4	4	0	3	0	1	-2	-1	0
7									
8	Weight - Age	1							
9	Weight - Drops	4							
10	Weight - Repairs	-1							
11	Weight - Spills	3							
12									

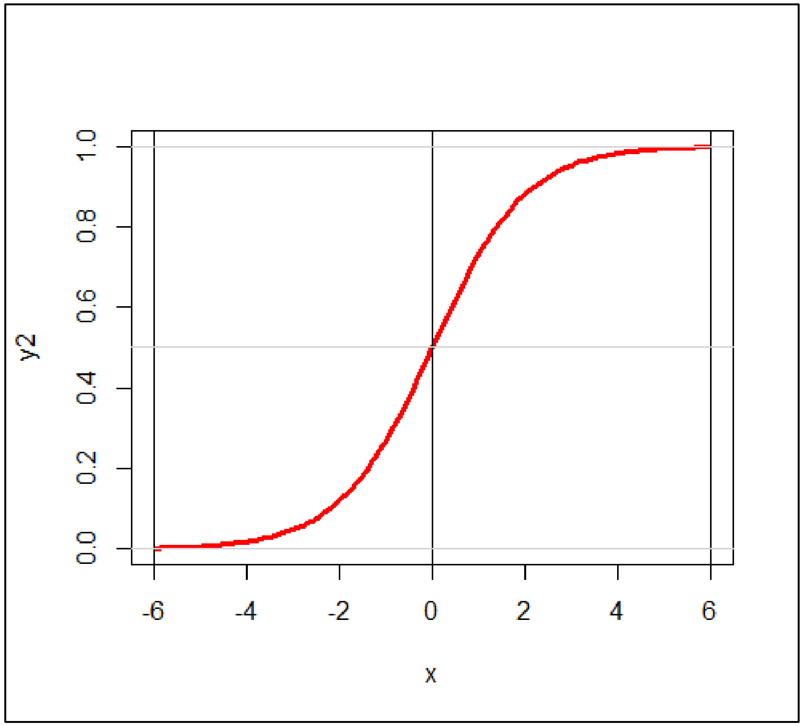




$$f(1 * Age + 4 * Drops - 1 * Repairs + 3 * Spills) + Bias$$

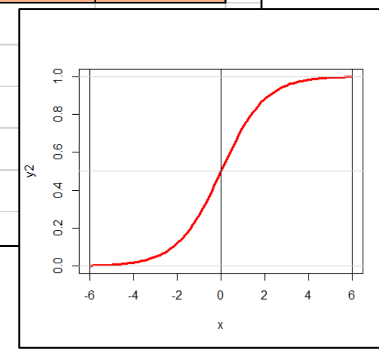
Sigmoid Function

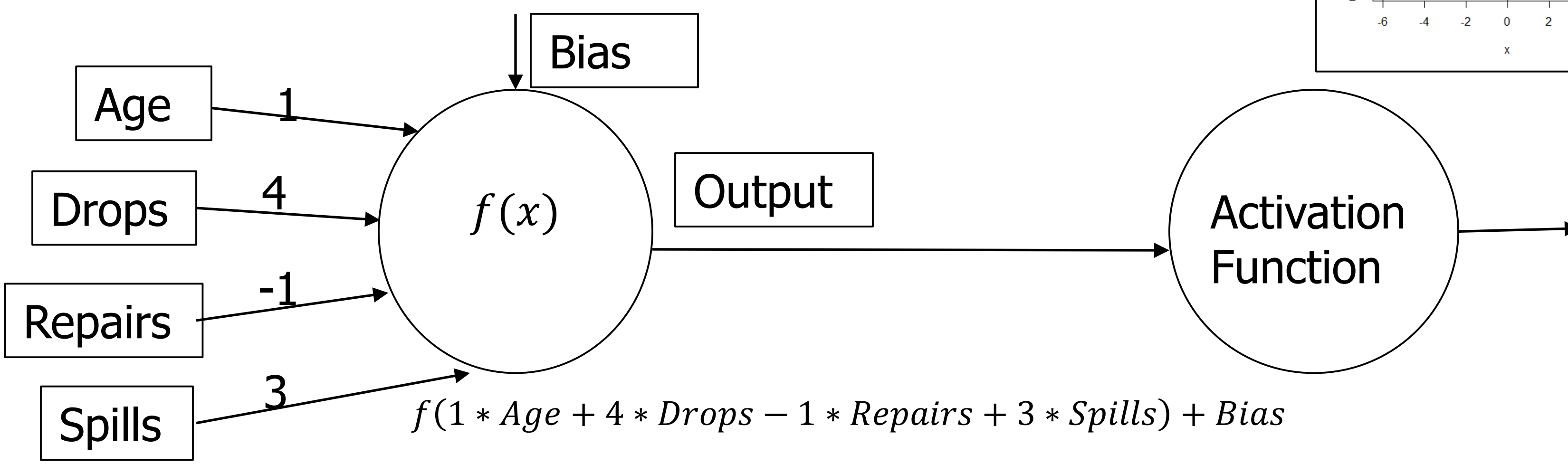
$$f(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}$$



Neuron + Sigmoid Activation Function

	A	B	C	D	E	F	G	H	I	J
1	**Training Data**									
2		**Age**	**Drops**	**Repairs**	**Spills**	**Output Function**	**Bias**	**Output+ Bias**	Activation Function - Sigmoid	If Activatio n > 0.5, 1, 0
3	1	3	0	1	0	2	-2	0	0.5	0
4	2	1	0	0	0	1	-2	-1	0.2689414	0
5	3	2	1	2	1	7	-2	5	0.9933071	1
6	4	4	0	3	0	1	-2	-1	0.2689414	0
7										
8	Weight - A	1								
9	Weight - D	4								
10	Weight - R	-1								
11	Weight - S	3								
12										

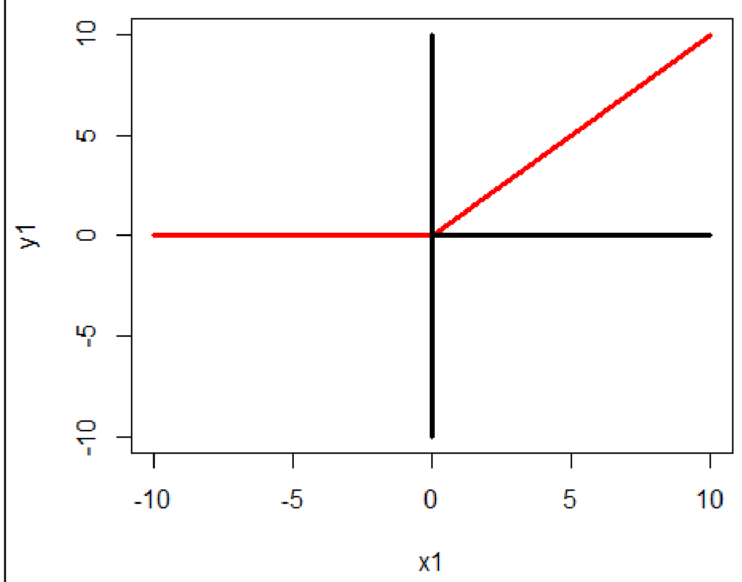




$$f(1 * Age + 4 * Drops - 1 * Repairs + 3 * Spills) + Bias$$

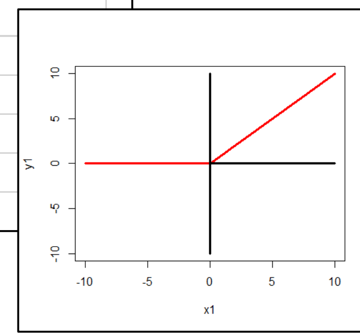
Rectified Linear Unit (ReLU) Function

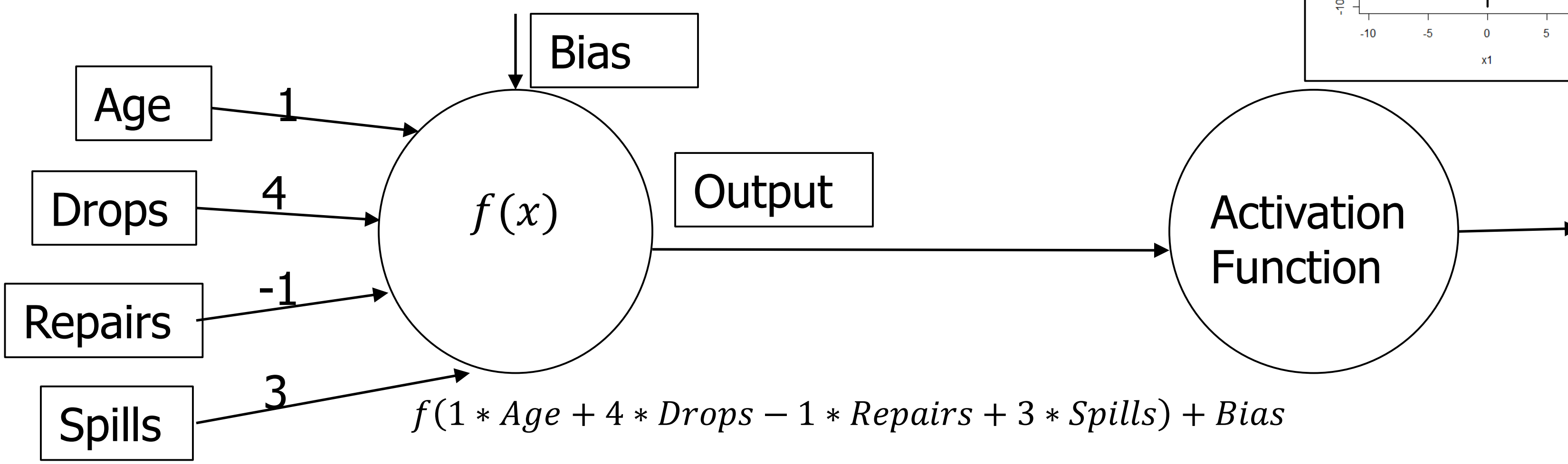
- $ReLU(x) = 0 \; when \; x < 0$
- $ReLU(x) = x \; when \; x \geq 0$
- ------------------------------
- $ReLU(a) = \max(0, a)$



Neuron + ReLU Activation Function

	A	B	C	D	E	F	G	H	I	J
1	Training Data									
2		Age	Drops	Repairs	Spills	Output Function	Bias	Output+ Bias	Activation Function - ReLU	If Activation > 1, 1, 0
3	1	3	0	1	0	2	-2	0	0	0
4	2	1	0	0	0	1	-2	-1	0	0
5	3	2	1	2	1	7	-2	5	5	1
6	4	4	0	3	0	1	-2	-1	0	0
7										
8	Weight - A	1								
9	Weight - D	4								
10	Weight - R	-1								
11	Weight - S	3								
12										





$$f(1 * Age + 4 * Drops - 1 * Repairs + 3 * Spills) + Bias$$

Which Activation Function is the Best?

- Activation function should have the following properties
  - It should be differential. It should not cause gradient to vanish
  - It should be simple and efficient

Problems in Using Sigmoid Function

- Computations are time consuming and complex
- It is slow in convergence

Most popular Activation Function

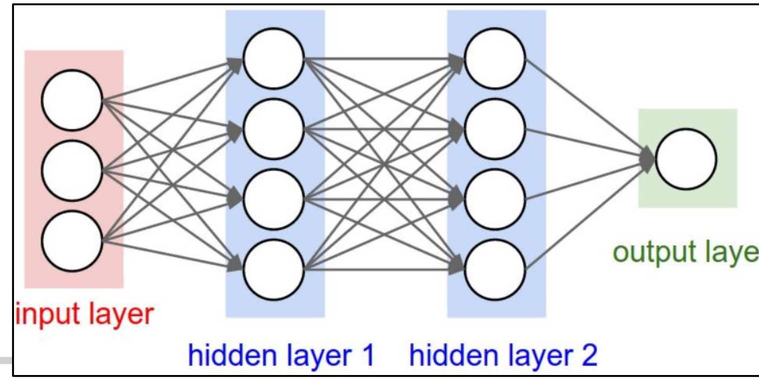
- ReLU – Rectified Linear Unit
- It is simple and efficient

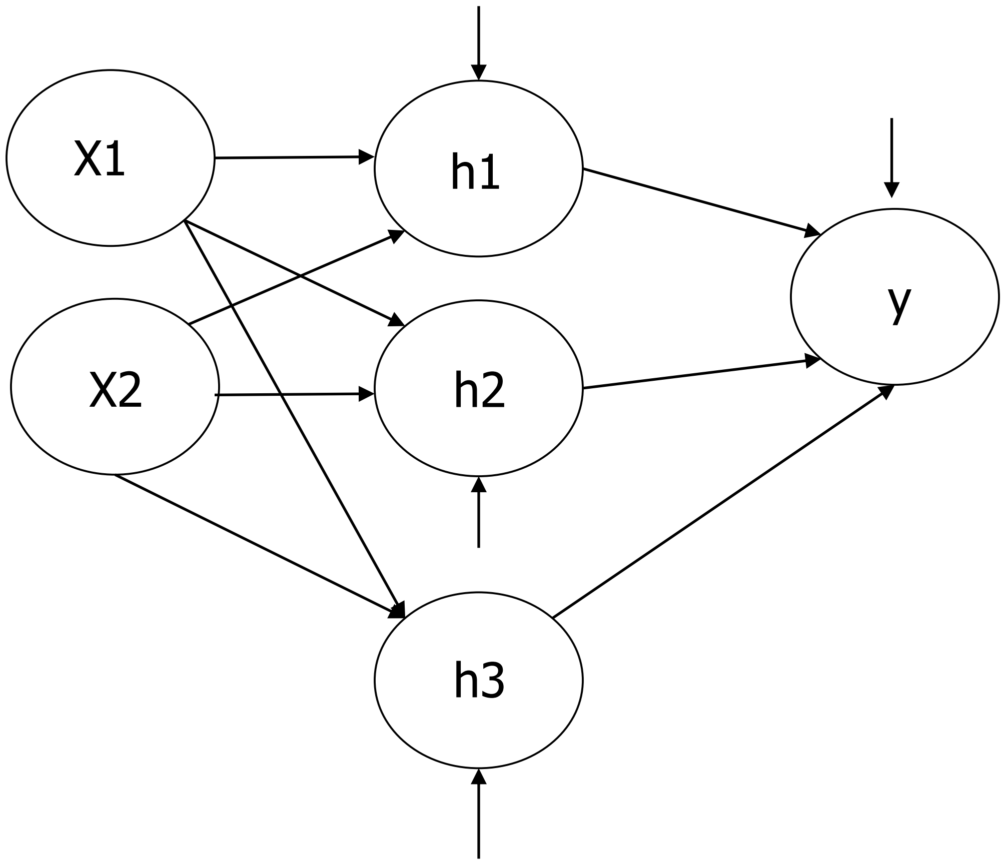
Neural Network and Regression

- Neural Network
  - with a single neuron and
  - with 'sigmoid' activation function
  - is same as
    - Linear Regression + Sigmoid Function

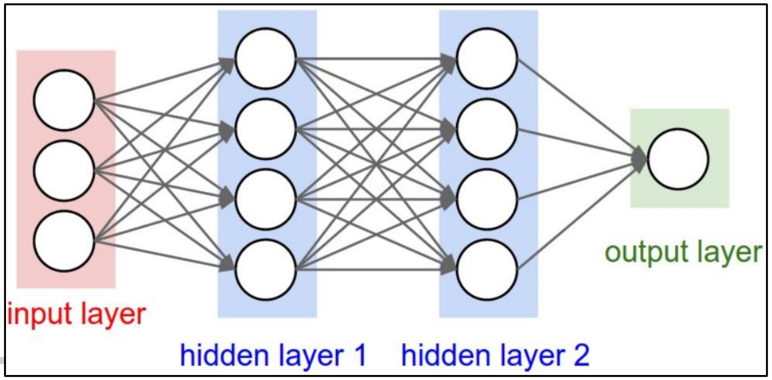
Feed Forward Fully Connected Neural Network

Feed Forward - Fully Connected Neural Network





Neural Network



- Any Mathematical Function can be computed using this set-up
  - Feed Forward - Fully Connected Neural Network
- Deep Learning
  - A Neural Network with more than one hidden layer

Algorithm of Computing Weights

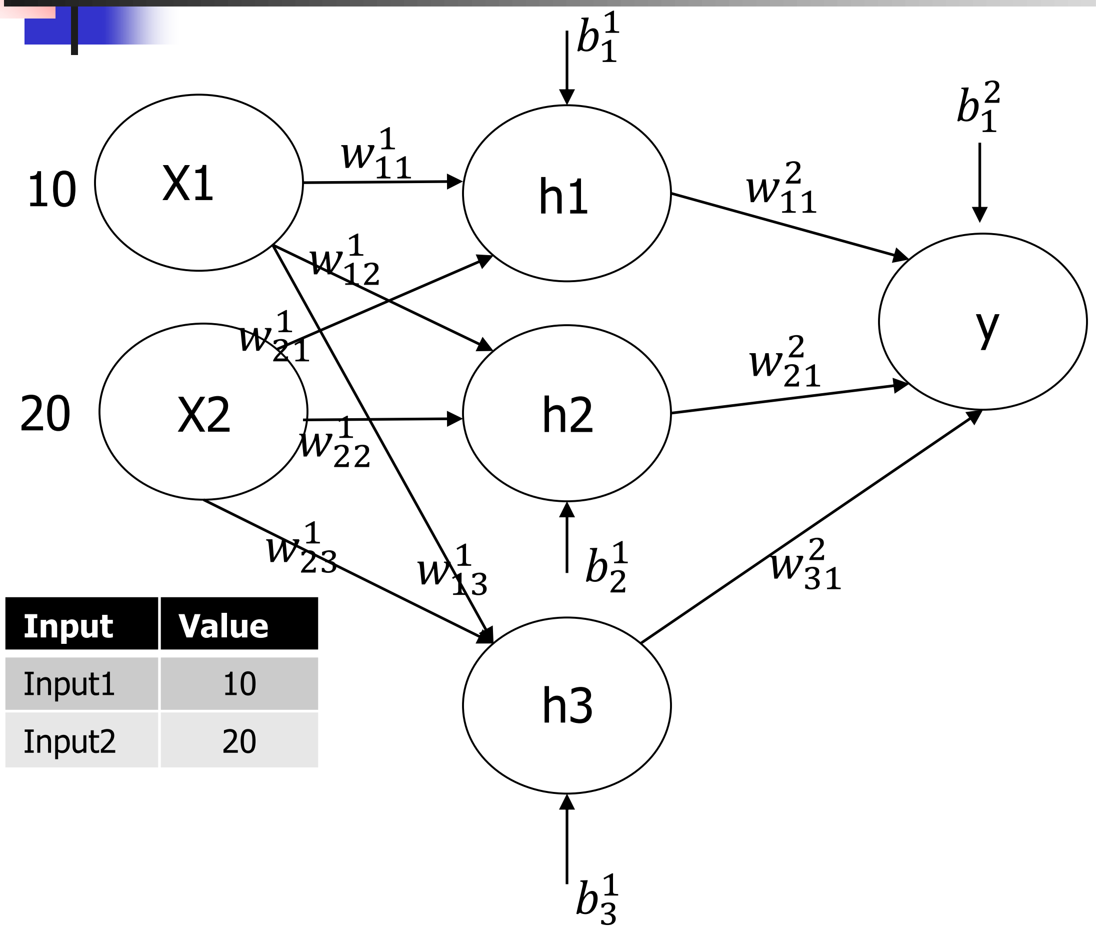
- Assign random values to all the weights
- Compute the output
- Compare the output with the observed output and compute the error
- Adjust the weights using back propagation algorithm till the error is minimized

Computing the Layer Output Using TensorFlow: Matrix Multiplication

Feed Forward - Fully Connected Neural Network

Layer 1	Value
w_{11}^1	1
w_{12}^1	2
w_{13}^1	3
w_{21}^1	4
w_{22}^1	5
w_{23}^1	6
b_1^1	7
b_2^1	8
b_3^1	9

Layer 2	Value
w_{11}^2	10
w_{21}^2	11
w_{31}^2	12
b_1^2	13

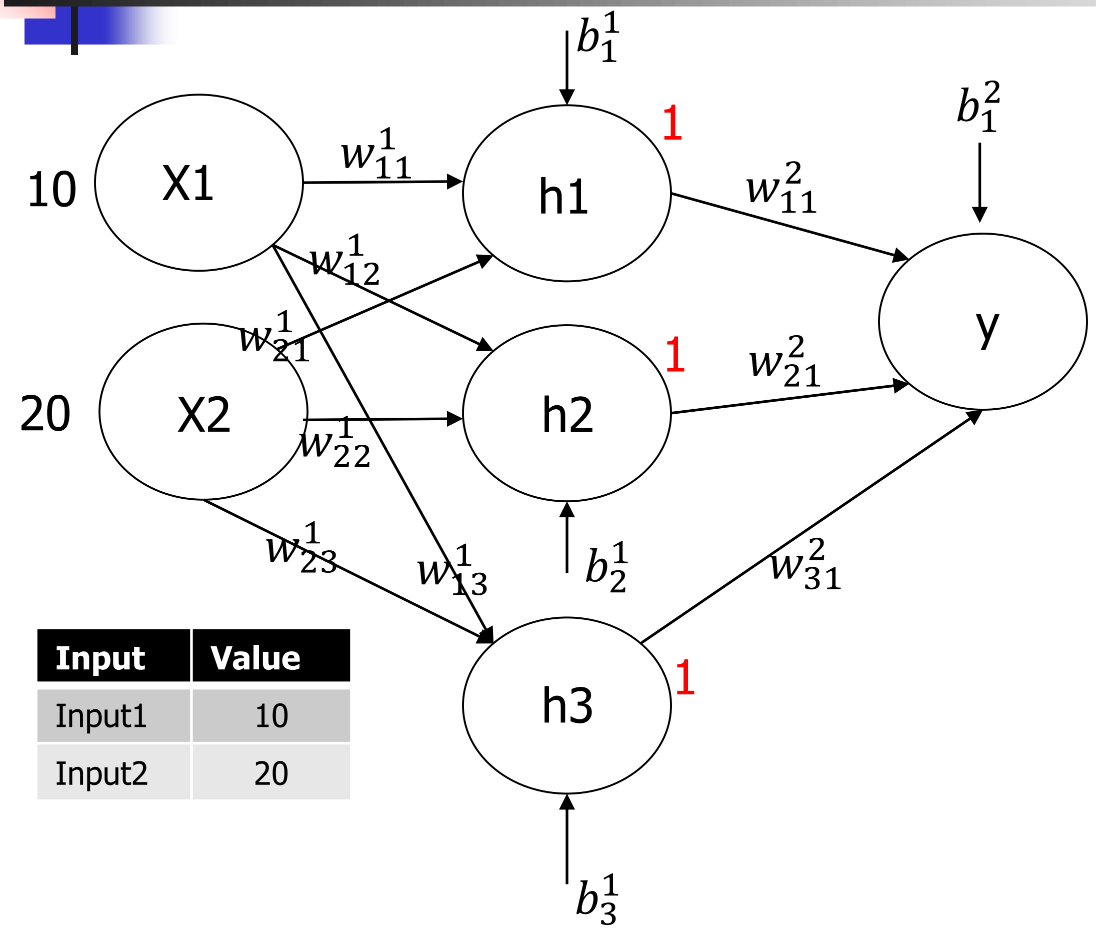


Input	Value
Input1	10
Input2	20

Feed Forward - Fully Connected Neural Network: Layer 1

Layer 1	Value
w_{11}^1	1
w_{12}^1	2
w_{13}^1	3
w_{21}^1	4
w_{22}^1	5
w_{23}^1	6
b_1^1	7
b_2^1	8
b_3^1	9

Layer 2	Value
w_{11}^2	10
w_{21}^2	11
w_{31}^2	12
b_1^2	13



Input	Value
Input1	10
Input2	20

$$h1 = (x1 * w_{11}^1 + x2 * w_{21}^1) + b_1^1$$
$$h1 = (10 * 1 + 20 * 4) + 7 = 97$$
$$outH1 = sigmoid(h1) = 1$$

$$h2 = (x1 * w_{12}^1 + x2 * w_{22}^1) + b_2^1$$
$$h2 = (10 * 2 + 20 * 5) + 8 = 128$$
$$outH2 = sigmoid(h2) = 1$$

$$h3 = (x1 * w_{13}^1 + x2 * w_{23}^1) + b_3^1$$
$$h3 = (10 * 3 + 20 * 6) + 9 = 159$$
$$outH3 = sigmoid(h3) = 1$$

Layer 1	Value
w_{11}^1	1
w_{12}^1	2
w_{13}^1	3
w_{21}^1	4
w_{22}^1	5
w_{23}^1	6
b_1^1	7
b_2^1	8
b_3^1	9

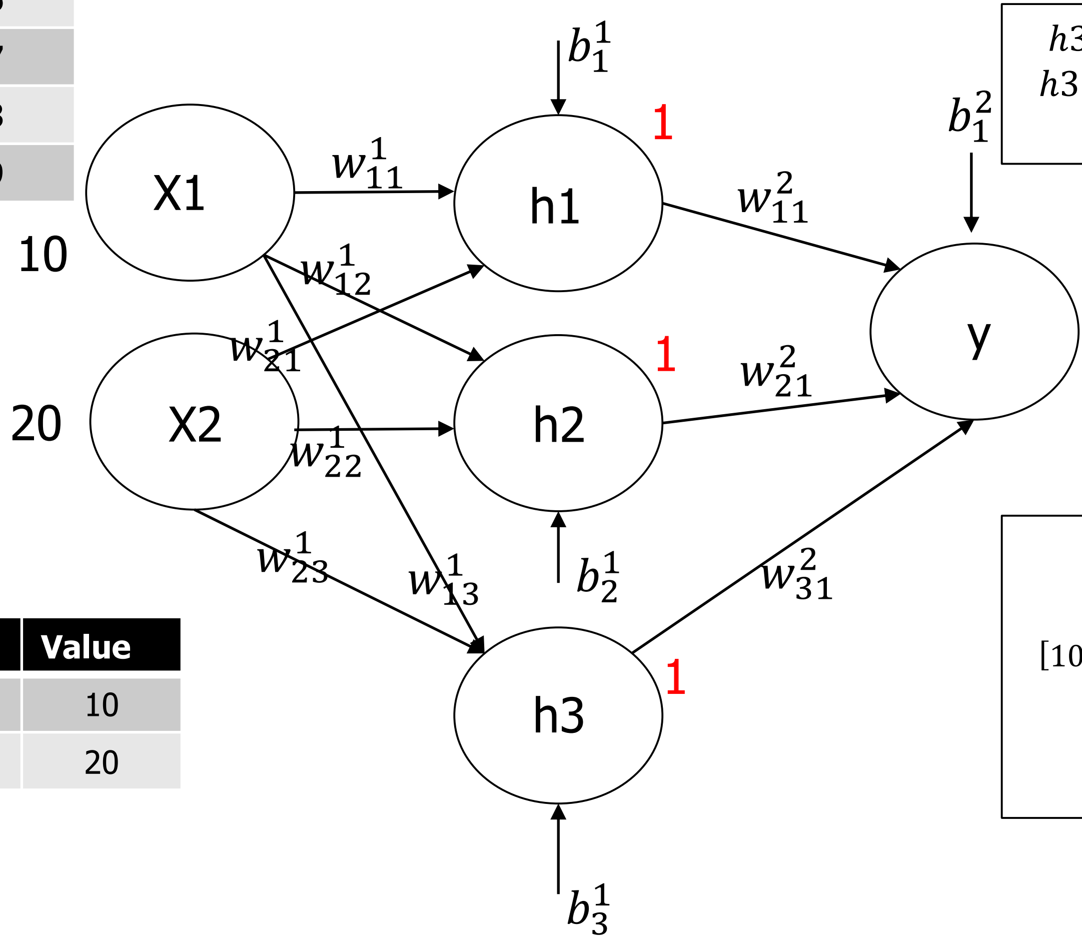
Layer 2	Value
w_{11}^2	10
w_{21}^2	11
w_{31}^2	12
b_1^2	13

Layer 1

$$h1 = (x1 * w_{11}^1 + x2 * w_{21}^1) + b_1^1$$
$$h1 = (10 * 1 + 20 * 4) + 7 = 97$$
$$outH1 = sigmoid(h1) = 1$$

$$h2 = (x1 * w_{12}^1 + x2 * w_{22}^1) + b_2^1$$
$$h2 = (10 * 2 + 20 * 5) + 8 = 128$$
$$outH2 = sigmoid(h2) = 1$$

$$h3 = (x1 * w_{13}^1 + x2 * w_{23}^1) + b_3^1$$
$$h3 = (10 * 3 + 20 * 6) + 9 = 159$$
$$outH3 = sigmoid(h3) = 1$$



Input	Value
Input1	10
Input2	20

$$h = X * W + b$$
$$outH = sigmoid(h)$$
$$\begin{bmatrix} 10 & 20 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 97 \\ 128 \\ 159 \end{bmatrix}$$
$$sigmoid \begin{bmatrix} 97 \\ 128 \\ 159 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

TensorFlow Code: Matrix Form: Layer 1

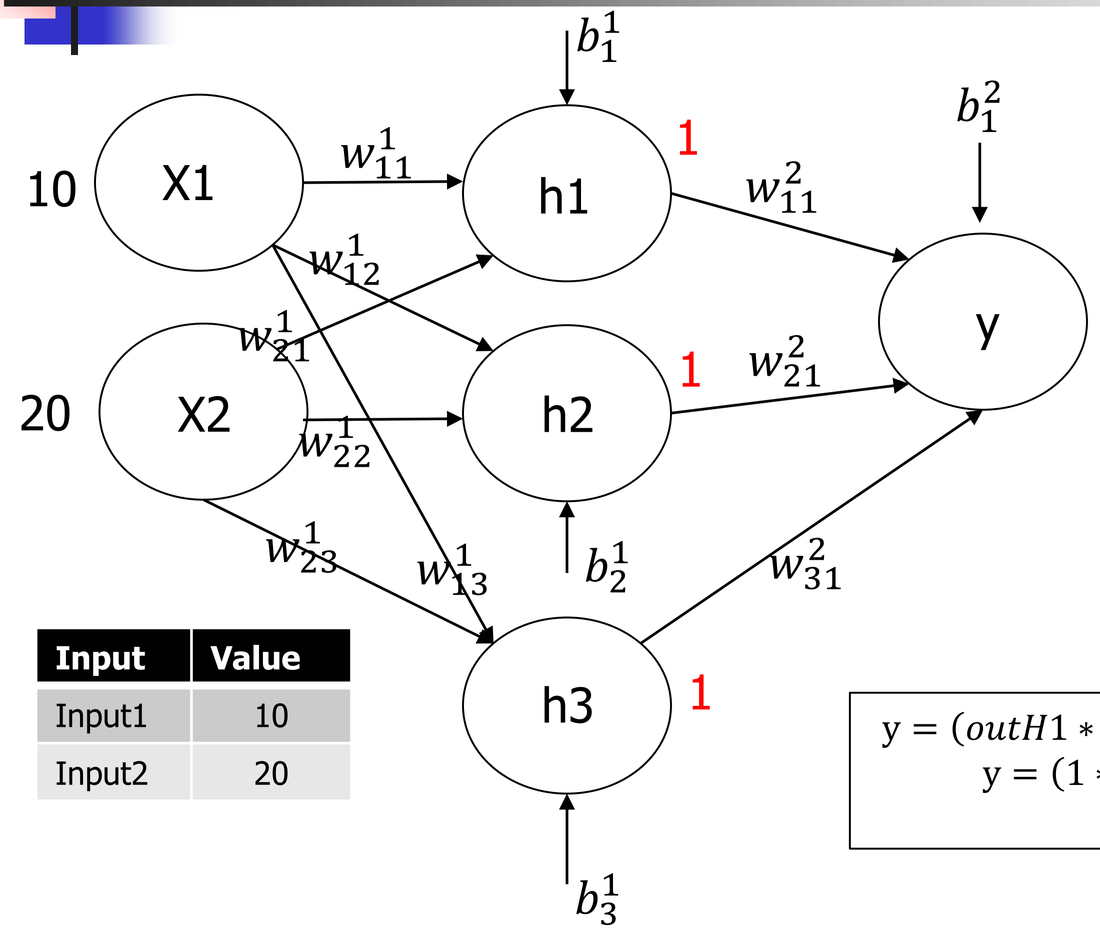
```
inputData = tf.constant([[10, 20]])
print(inputData.shape)
(1, 2)
print(sess.run(inputData))
[[10 20]]
###################################################
# Layer 1
#
W1 = tf.constant([[1, 2, 3],[4, 5, 6]])
print(W1.shape)
(2, 3)
print(sess.run(W1))
[[1 2 3]
 [4 5 6]]
###################################################
b1 = tf.constant([[7,8,9]])
print(b1.shape)
(1, 3)
print(sess.run(b1))
[[7 8 9]]
#####################################################
outputH1 = tf.matmul(inputData, W1) + b1
print(outputH1.shape)
(1, 3)
print(sess.run(outputH1))
[[ 97 128 159]]
outputH1_Activation = tf.sigmoid(tf.cast(outputH1, tf.float32))
print(outputH1_Activation.shape)
(1, 3)
print(sess.run(outputH1_Activation))
[[ 1.  1.  1.]]
```

$$h = X * W + b$$
$$outH = sigmoid(h)$$
$$\begin{bmatrix} 10 & 20 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 97 \\ 128 \\ 159 \end{bmatrix}$$
$$sigmoid \begin{bmatrix} 97 \\ 128 \\ 159 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Feed Forward - Fully Connected Neural Network: Layer 2

Layer 1	Value
w_{11}^1	1
w_{12}^1	2
w_{13}^1	3
w_{21}^1	4
w_{22}^1	5
w_{23}^1	6
b_1^1	7
b_2^1	8
b_3^1	9

Layer 2	Value
w_{11}^2	10
w_{21}^2	11
w_{31}^2	12
b_1^2	13



Input	Value
Input1	10
Input2	20

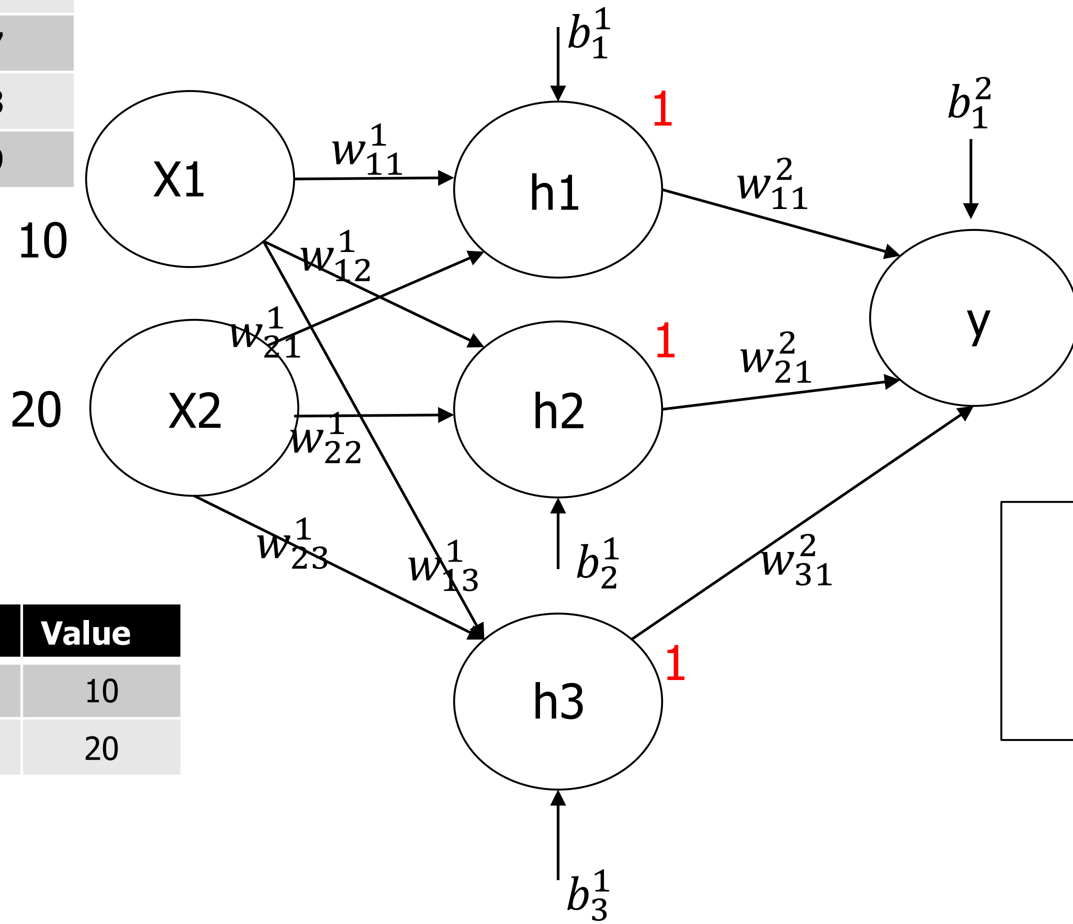
$$y = (outH1 * w_{11}^2 + \text{outH2} * w_{21}^2 + \text{outH3} * w_{31}^2) + b_1^2$$
$$y = (1 * 10 + 1 * 11 + 1 * 12) + 13 = 46$$
$$outY = sigmoid(y) = 1$$

Layer 1	Value
w_{11}^1	1
w_{12}^1	2
w_{13}^1	3
w_{21}^1	4
w_{22}^1	5
w_{23}^1	6
b_1^1	7
b_2^1	8
b_3^1	9

Layer 2	Value
w_{11}^2	10
w_{21}^2	11
w_{31}^2	12
b_1^2	13

$$\mathrm{y} = (outH1 * w_{11}^2 + \mathrm{outH2} * w_{21}^2 + \mathrm{outH3} * w_{31}^2) + b_1^2$$
$$\mathrm{y} = (1 * 10 + 1 * 11 + 1 * 12) + 13 = 46$$
$$outY = sigmoid(y) = 1$$

Layer 2



Input	Value
Input1	10
Input2	20

$$h = X * W + b$$
$$outH = sigmoid(h)$$
$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} + [13] = [46]$$
$$sigmoid[46] = [1]$$

TensorFlow Code: Matrix Form: Layer 2

```
#####################################################
# Layer 2
#
W2 = tf.cast(tf.constant([[10], [11], [12]]),tf.float32)
print(W2.shape)
(3, 1)
print(sess.run(W2))
[[ 10.]
 [ 11.]
 [ 12.]]
###################################################
b2 = tf.cast(tf.constant([[13]]),tf.float32)
print(b2.shape)
(1, 1)
print(sess.run(b2))
[[ 13.]]
#####################################################
outputH2 = tf.matmul(outputH1_Activation, W2) + b2
print(outputH2.shape)
(1, 1)
print(sess.run(outputH2))
[[ 46.]]
outputH2_Activation = tf.sigmoid(tf.cast(outputH2, tf.float32))
print(outputH2_Activation.shape)
(1, 1)
print(sess.run(outputH2_Activation))
[[ 1.]]
```

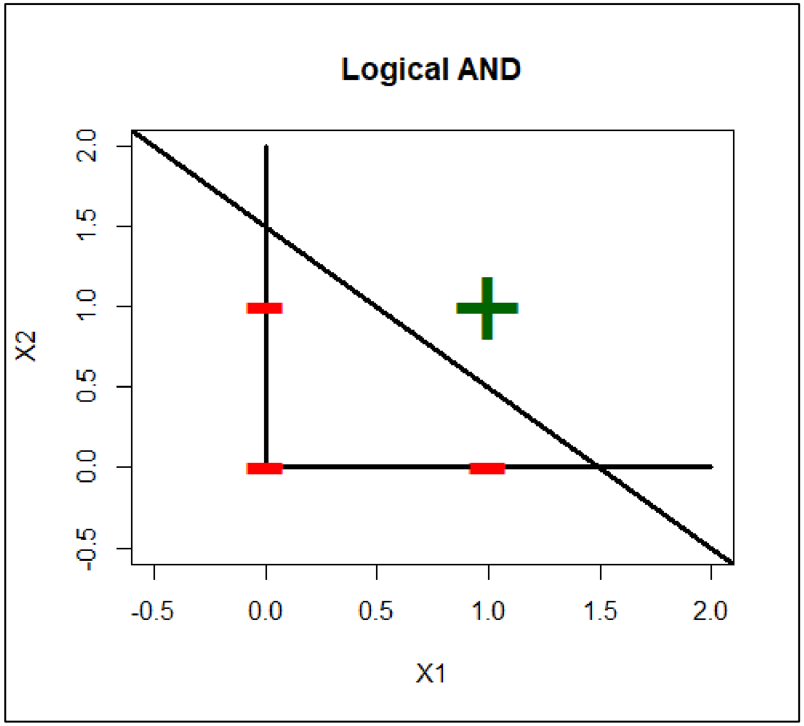
$$h = X * W + b$$
$$outH = sigmoid(h)$$
$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} + [13] = [46]$$
$$sigmoid[46] = [1]$$

Solution to XOR Problem

Logical AND Gate

X1	X2	X1 AND X2 Gate
0	0	0
1	0	0
0	1	0
1	1	1

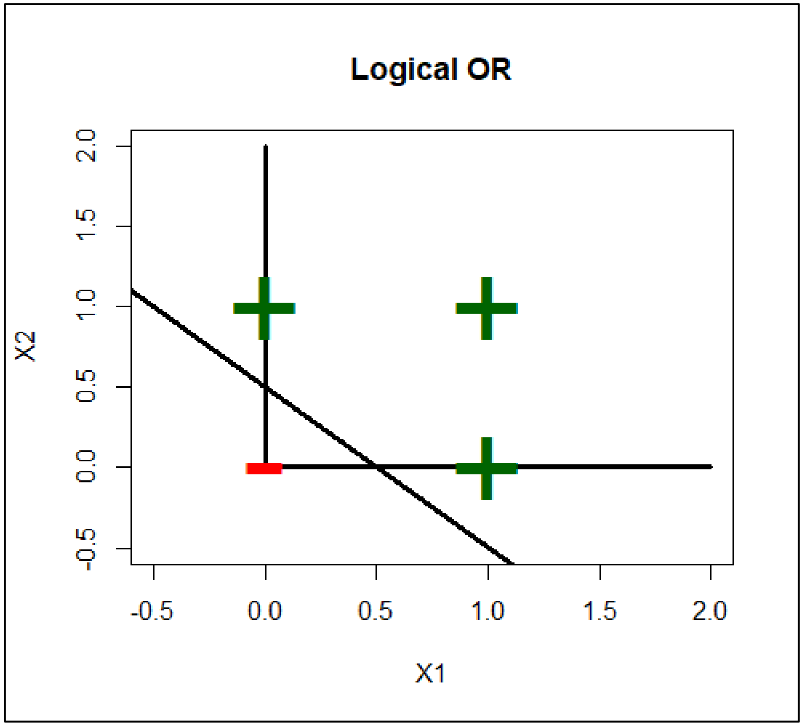
Linearly Separable Data



Logical OR Gate

X1	X2	X1 OR X2 Gate
0	0	0
1	0	1
0	1	1
1	1	1

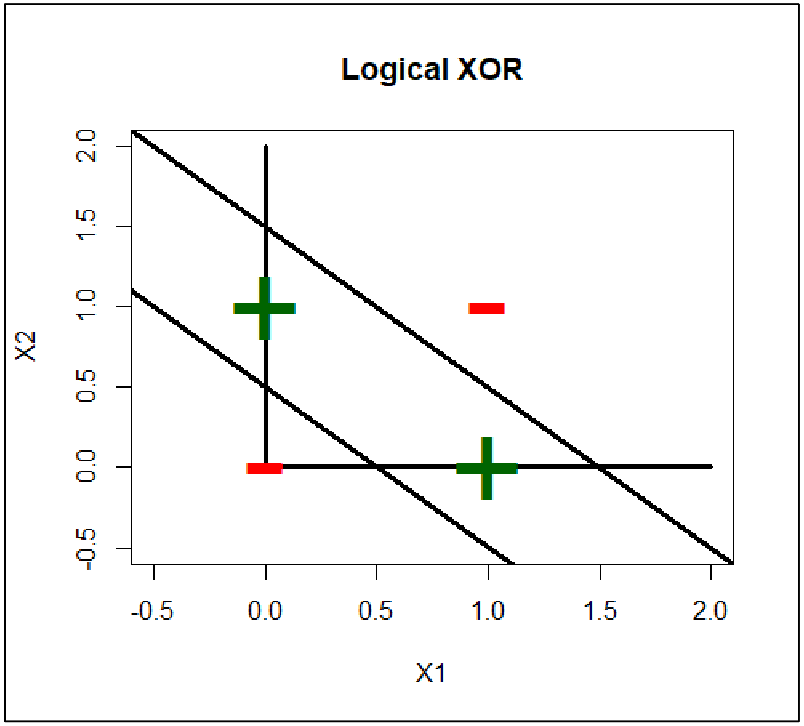
Linearly Separable Data



Logical XOR Gate

X1	X2	X1 XOR X2 Gate
0	0	0
1	0	1
0	1	1
1	1	0

NOT Linearly Separable Data



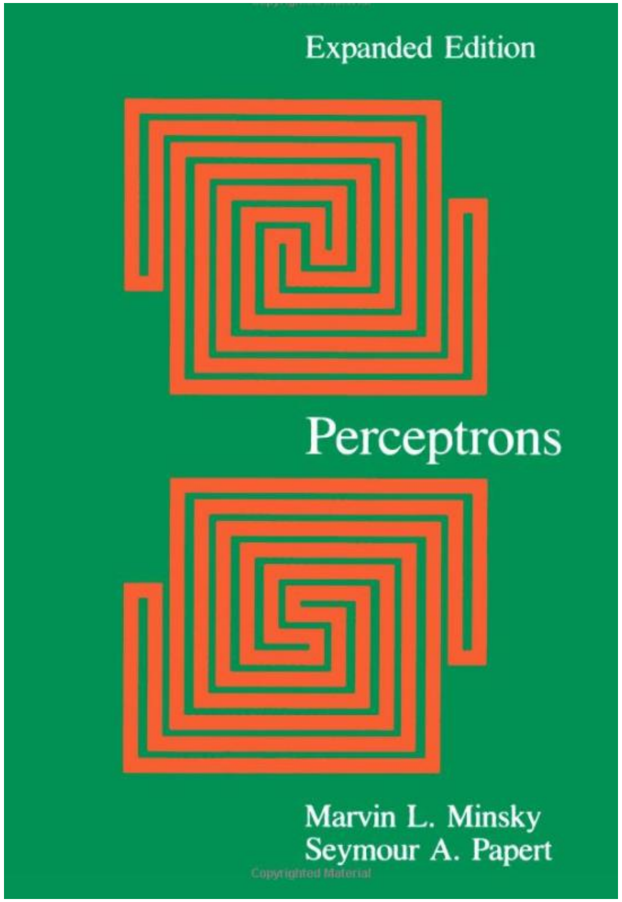
Perceptron

- Neural Network concept was criticized by Marvin Minsky (1969)
  - MIT
- Publicly challenged Rosenblatt that Perceptron can learn anything
- XOR pattern cannot be learned by Perceptron
  - However it can be learned by multi-layer neural network
  - At that time technology was not advanced enough to build a multi layer neural network

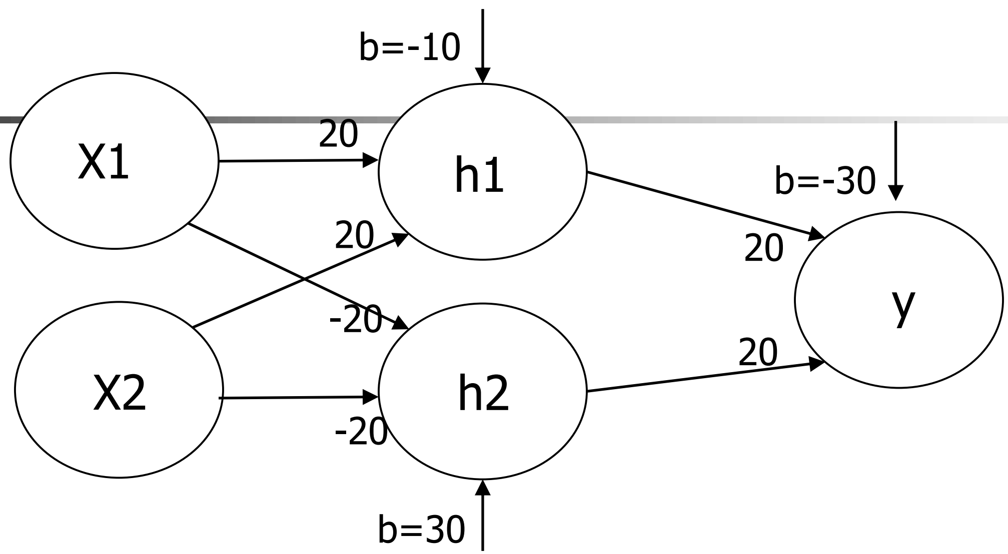
Limitations of Neural Networks Marvin Minsky (1969)

- Marvin Minsky and Seymour Papert publish their book Perceptrons, describing some of the limitations of perceptrons and neural networks.

- The interpretation the book shows that neural networks are fundamentally limited is seen as a hindrance for research into neural networks.



Neural Network to Solve XOR Problem



X1	X2	X1 XOR X2 Gate
0	0	0
1	0	1
0	1	1
1	1	0

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

x1	x2	$h1: \sigma(20 * x1 + 20 * x2 - 10)$	$h2: \sigma(-20 * x1 - 20 * x2 + 30)$	$y: \sigma(20 * h1 + 20 * h2 - 30)$
0	0	$\sigma(20 * 0 + 20 * 0 - 10) = \sigma(-10) = 0$	$\sigma(-20 * 0 - 20 * 0 + 30) = \sigma(30) = 1$	$\sigma(20 * 0 + 20 * 1 - 30) = \sigma(-10) = 0$
1	0	$\sigma(20 * 1 + 20 * 0 - 10) = \sigma(10) = 1$	$\sigma(-20 * 1 - 20 * 0 + 30) = \sigma(10) = 1$	$\sigma(20 * 1 + 20 * 1 - 30) = \sigma(10) = 1$
0	1	$\sigma(20 * 0 + 20 * 1 - 10) = \sigma(10) = 1$	$\sigma(-20 * 0 - 20 * 1 + 30) = \sigma(10) = 1$	$\sigma(20 * 1 + 20 * 1 - 30) = \sigma(10) = 1$
1	1	$\sigma(20 * 1 + 20 * 1 - 10) = \sigma(30) = 1$	$\sigma(-20 * 1 - 20 * 1 + 30) = \sigma(-10) = 0$	$\sigma(20 * 1 + 20 * 0 - 30) = \sigma(-10) = 0$

Summary

- Neuron Functions
- Activation Functions
  - Unit Step Function
  - Sigmoid Function
  - Rectified Linear Unit Function (ReLU)
- Feed Forward Fully Connected - Neural Network
- Computing the Layer Output Using TensorFlow
- Solution of XOR Problem
  - Logical XOR Gate
  - Hidden Layer Solution to XOR Problem