Introduction to Neural Networks

Dr. Mahmoud N Mahmoud
mnmahmoud@ncat.edu

North Carolina A & T State University

September 30, 2020

Talk Overview

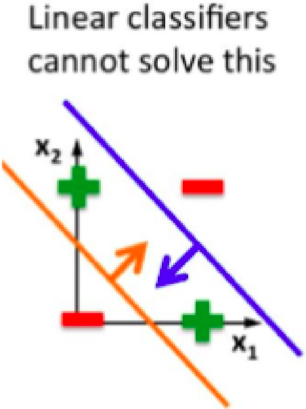
1. Introduction
2. FeedForward Neural Network
3. Activation Functions

Outline

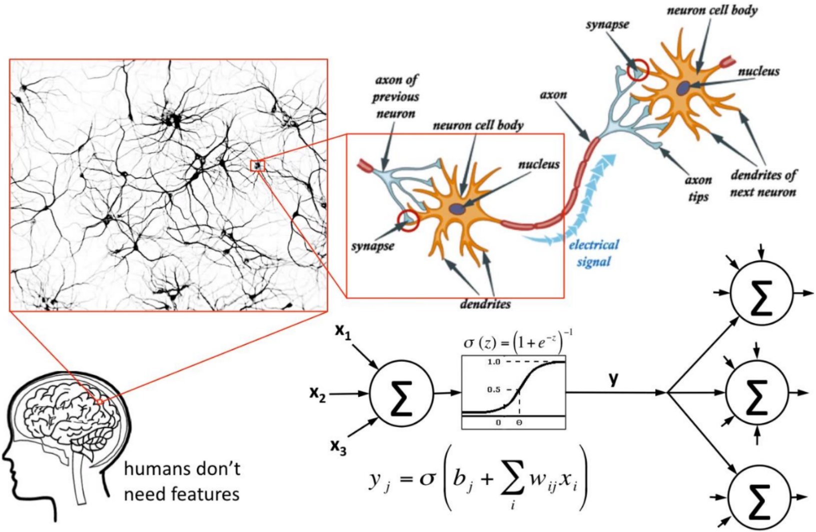
Introduction

- One of the oldest and one of the newest machine learning models.
- Goes back to 1940, when people started to build modles the imitate the human brain.
- Logistic regression (perceptron) is the core of neural networks started in 1950.
- However, scientists in that time showed that a single perceptron can not solve xor problem (died).
- Reborn in 1980, discovery of merging perceptrons together. But died due to the resources requirements
- Reborn in the last decade with the advancement of the computation resouces.

Xor Problem

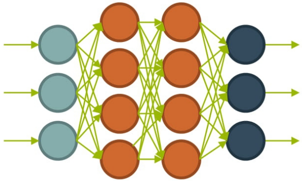


Neurons and the brain

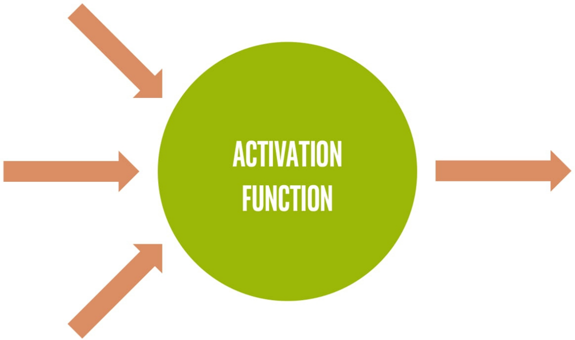


MOTIVATION FOR NEURAL NETS

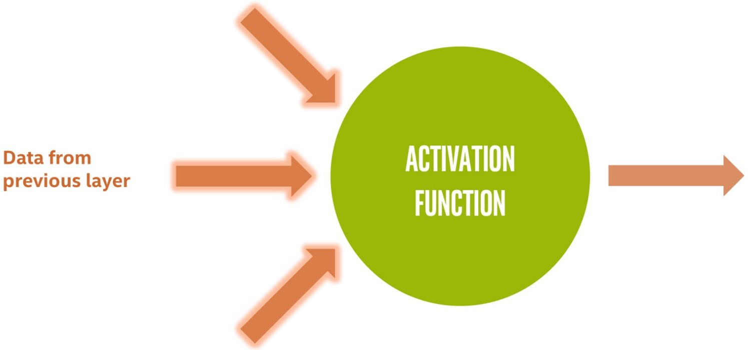
- Use biology as inspiration for mathematical model
- Get signals from previous neurons
- Generate signals (or not) according to inputs
- Pass signals on to next neurons
- By layering many neurons, can create complex model



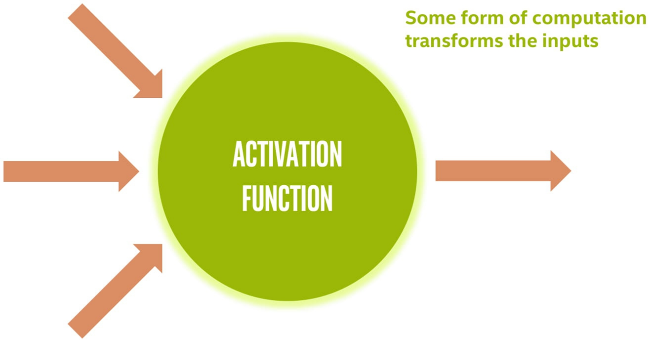
BASIC NEURON VISUALIZATION



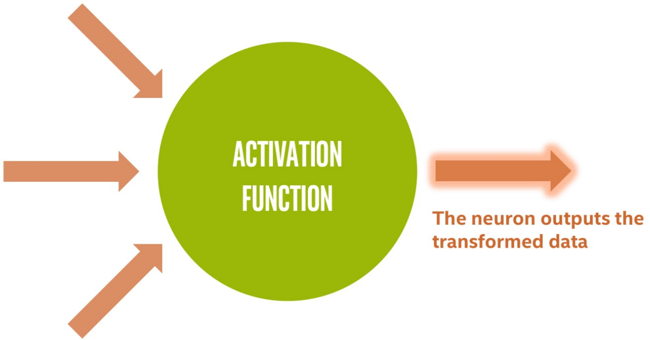
BASIC NEURON VISUALIZATION



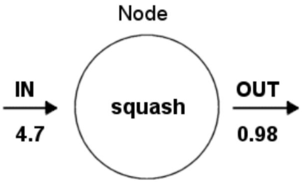
BASIC NEURON VISUALIZATION



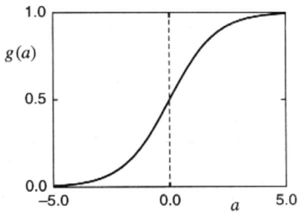
BASIC NEURON VISUALIZATION



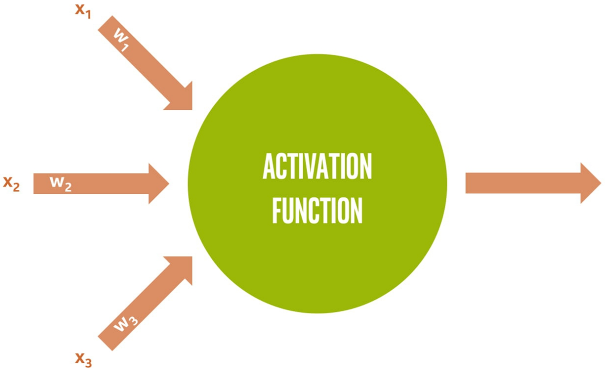
Structure of a node

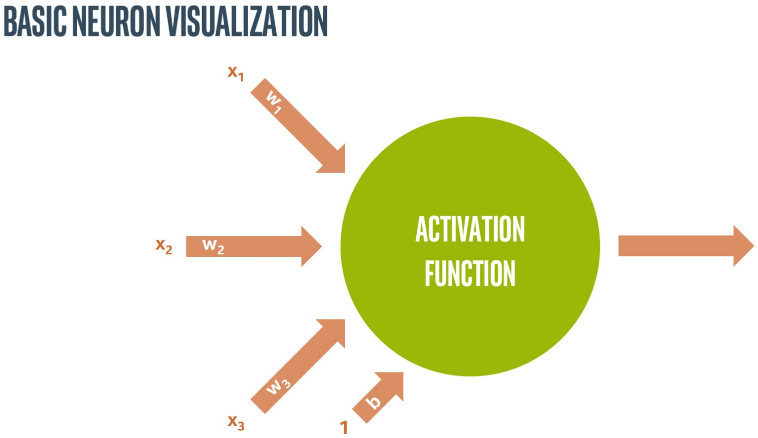


Squashing/Activation function limits node output:



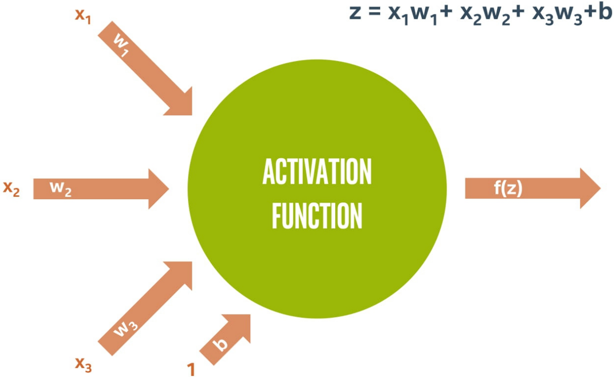
BASIC NEURON VISUALIZATION




BASIC NEURON VISUALIZATION
x1
w1
x2
w2
ACTIVATION FUNCTION
w3
x3
b
1

BASIC NEURON VISUALIZATION

$$z = x_1w_1 + x_2w_2 + x_3w_3 + b$$



IN VECTOR NOTATION

z = "net input"

b = "bias term"

f = activation function

a = output to next layer

$$z = b + \sum_{i=1}^{m} x_i w_i$$

$$z = b + x^T w$$

$$a = f(z)$$

RELATION TO LOGISTIC REGRESSION

When we choose: $f(z) = \frac{1}{1+e^{-z}}$

$$z = b + \sum_{i=1}^{m} x_i w_i = x_1 w_1 + x_2 w_2 + \cdots + x_m w_m + b$$

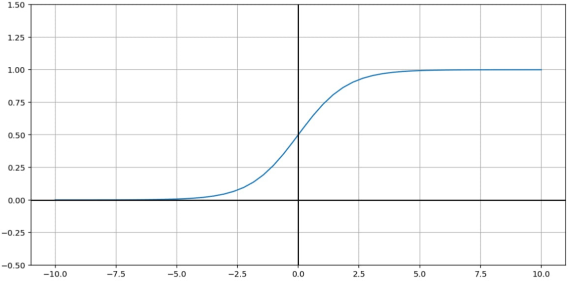
Then a neuron is simply a "unit" of logistic regression!

weights ⇔ coefficients **inputs ⇔ variables**

bias term ⇔ constant term

RELATION TO LOGISTIC REGRESSION

This is called the "sigmoid" function: $\sigma(z) = \dfrac{1}{1+e^{-z}}$



NICE PROPERTY OF SIGMOID FUNCTION

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

Quotient rule

$$\frac{d}{dx} \cdot \frac{f(x)}{g(x)} = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$$

$$\sigma'(z) = \frac{0-(-e^{-z})}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{\cancel{1+e^{-z}}}{(1+e^{-z})^{\cancel{2}}} - \frac{1}{(1+e^{-z})^2}$$

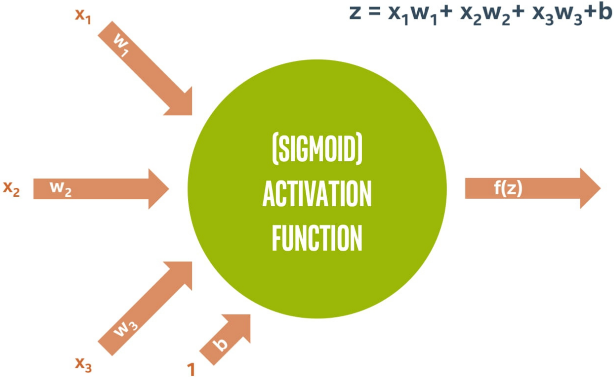
$$= \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

This will be helpful!

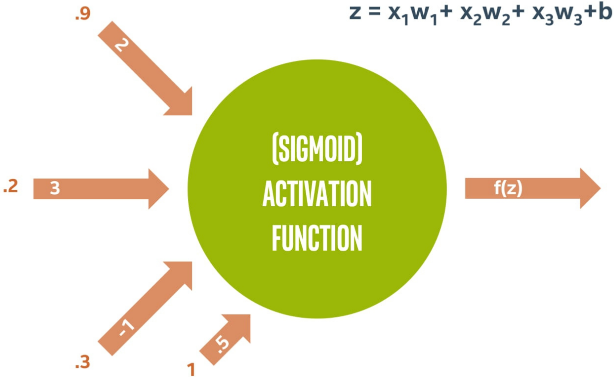
EXAMPLE NEURON COMPUTATION

$$z = x_1w_1 + x_2w_2 + x_3w_3 + b$$



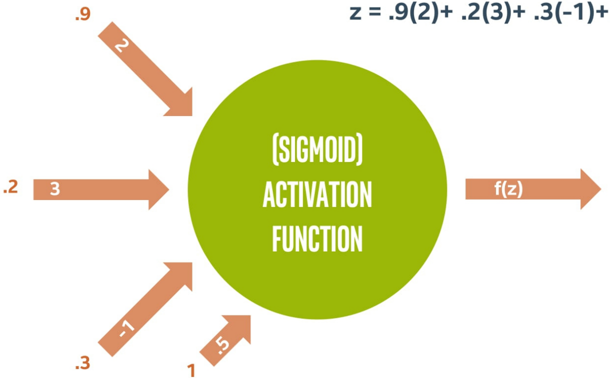
EXAMPLE NEURON COMPUTATION

$$z = x_1w_1 + x_2w_2 + x_3w_3 + b$$



EXAMPLE NEURON COMPUTATION

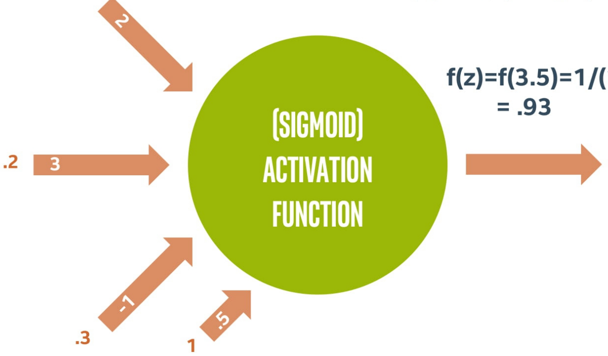
$$z = .9(2) + .2(3) + .3(-1) + .5 = 2.6$$



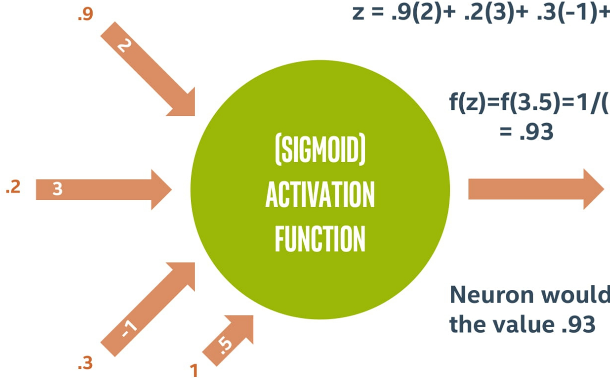
EXAMPLE NEURON COMPUTATION

$$z = .9(2) + .2(3) + .3(-1) + .5 = 2.6$$

$$f(z) = f(3.5) = 1/(1+\exp(-2.6)) = .93$$



EXAMPLE NEURON COMPUTATION



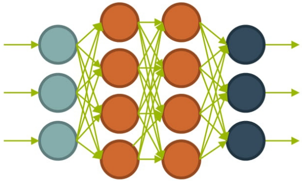
z = .9(2)+ .2(3)+ .3(-1)+.5 = 2.6

f(z)=f(3.5)=1/(1+exp(-2.6))
= .93

Neuron would output the value .93

WHY NEURAL NETS?

- Why not just use a single neuron? Why do we need a larger network?
- A single neuron (like logistic regression) only permits a linear decision boundary.
- Most real-world problems are considerably more complicated!

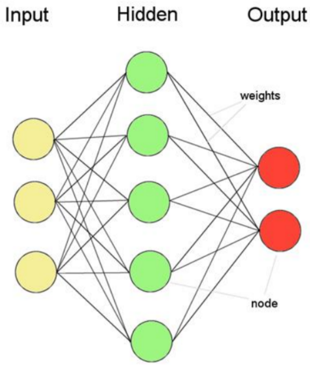


Outline

Artificial NNs

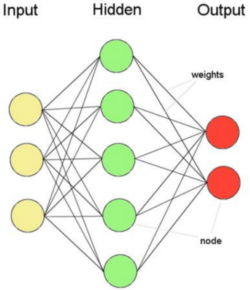
- ANNs incorporate the two fundamental components of biological neural nets.

1. Neurones (nodes)
2. Synapses (weights)

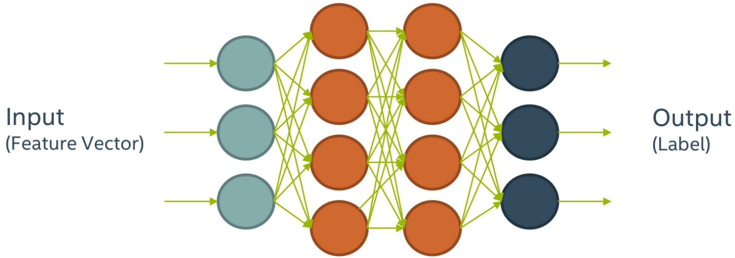


Types of Layers

1. The input layer
   - Introduces input values into the network.
   - No activation function or other processing.
2. The hidden layer(s)
   - Perform classification of features
   - Two hidden layers are sufficient to solve any problem
3. The output layer
   - Functionally just like the hidden layers
   - Outputs are passed on to the world outside the neural network.

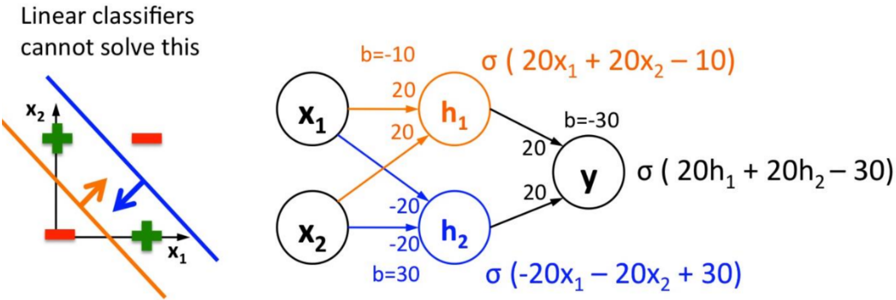


NEURAL NET STRUCTURE



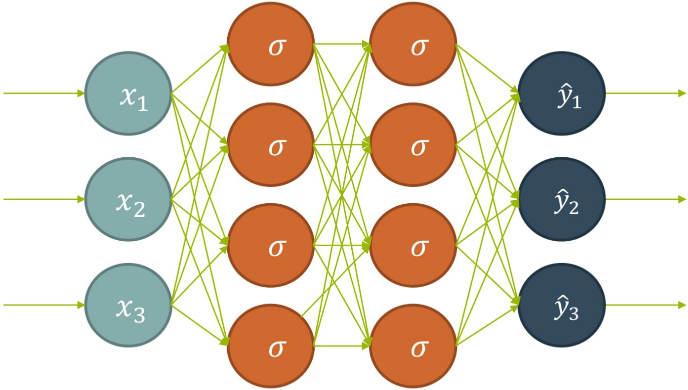
- Can think of it as a complicated computation engine
- We will "train it" using our training data
- Then (hopefully) it will give good answers on new data

Solving XOR with a Neural Network

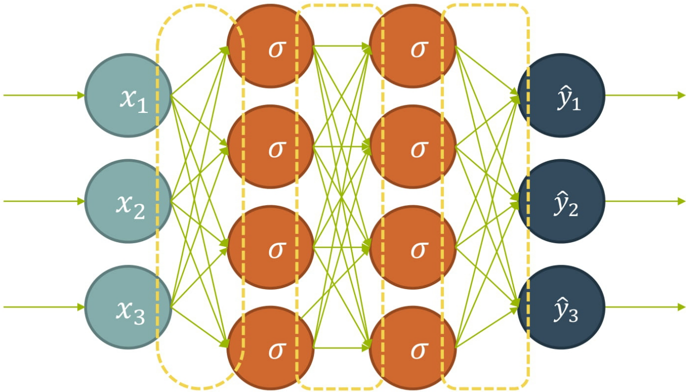


$\sigma(20*0 + 20*0 - 10) \approx 0$	$\sigma(-20*0 - 20*0 + 30) \approx 1$	$\sigma(20*0 + 20*1 - 30) \approx 0$
$\sigma(20*1 + 20*1 - 10) \approx 1$	$\sigma(-20*1 - 20*1 + 30) \approx 0$	$\sigma(20*1 + 20*0 - 30) \approx 0$
$\sigma(20*0 + 20*1 - 10) \approx 1$	$\sigma(-20*0 - 20*1 + 30) \approx 1$	$\sigma(20*1 + 20*1 - 30) \approx 1$
$\sigma(20*1 + 20*0 - 10) \approx 1$	$\sigma(-20*1 - 20*0 + 30) \approx 1$	$\sigma(20*1 + 20*1 - 30) \approx 1$

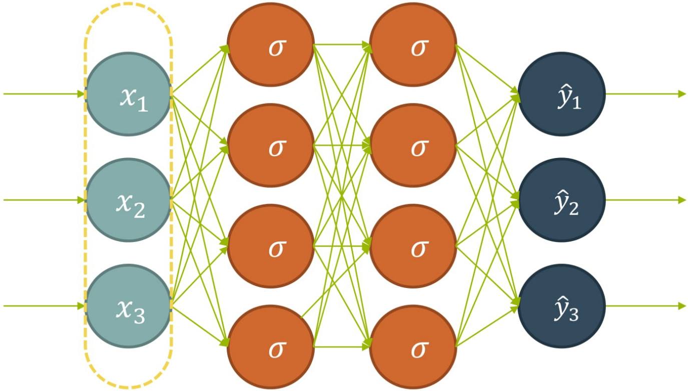
FEEDFORWARD NEURAL NETWORK



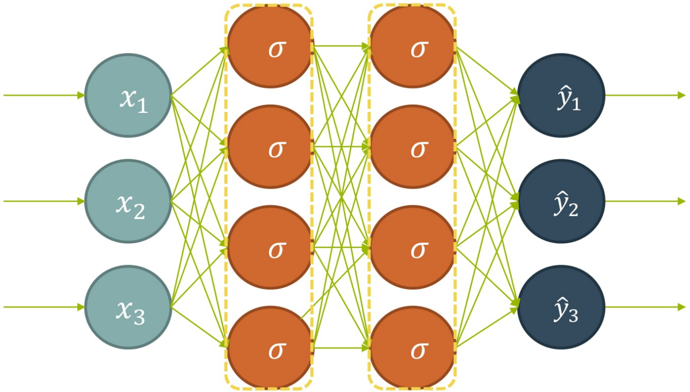
WEIGHTS



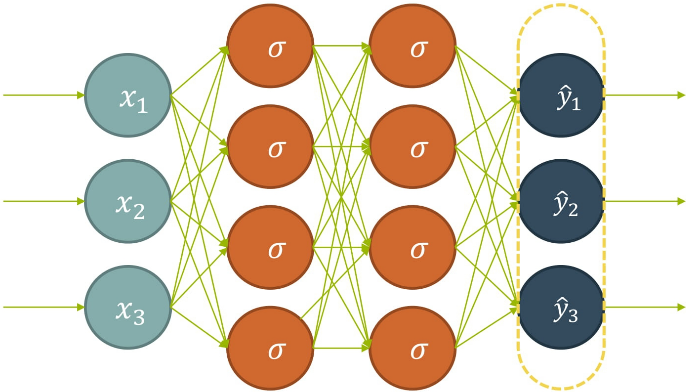
INPUT LAYER


x_1
x_2
x_3
σ
σ
σ
σ
σ
σ
σ
σ
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$

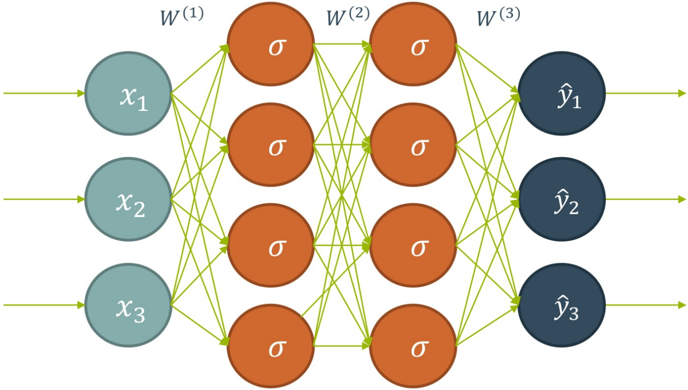
HIDDEN LAYERS



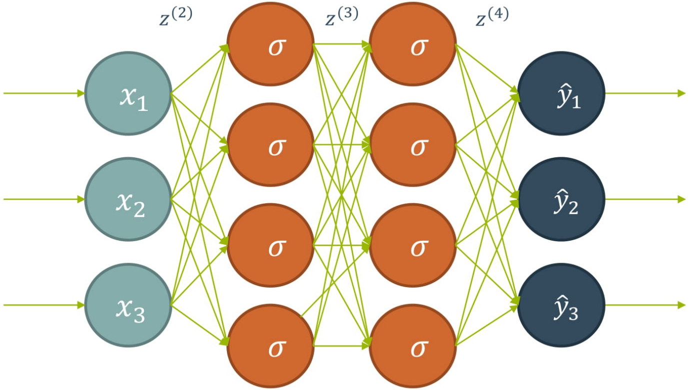
OUTPUT LAYER



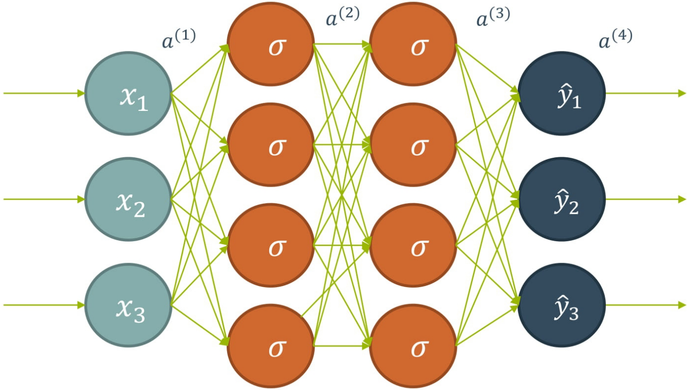
WEIGHTS (REPRESENTED BY MATRICES)



NET INPUT (SUM OF WEIGHTED INPUTS, BEFORE ACTIVATION FUNCTION)

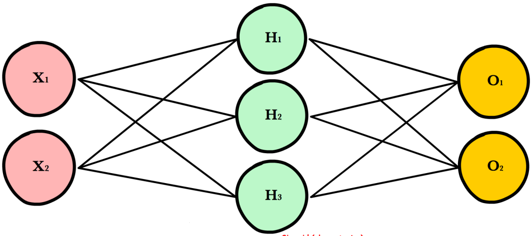


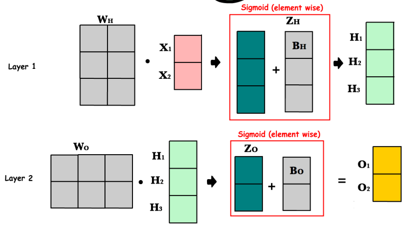
ACTIVATIONS (OUTPUT OF NEURONS TO NEXT LAYER)



Working with Matrices

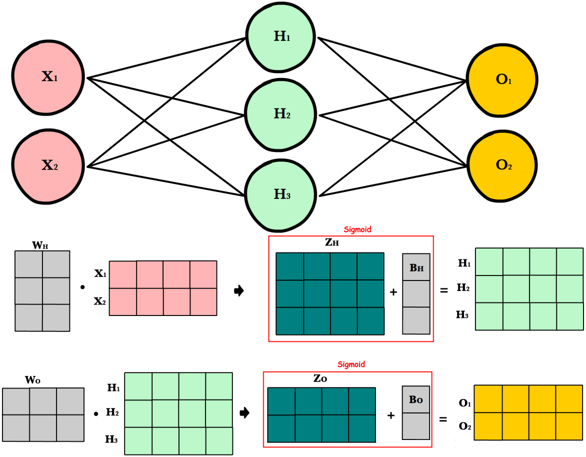
- Three features.
- One input at the time.





Working with Matrices

- Three features.
- Four inputs at the time.



Feed Forward Properties

- **Input dimension:** No of features $\times$ No of Samples in the Batch
- **Weight dimension at (just before) layer L:** No of neurons in layer L $\times$ No of neurons in layer (L-1)
- **Data dimension at layer L:** No of neurons in layer L $\times$ No of Samples in the Batch.

FeedForward Equations

1. $\mathbf{Z}^L = \mathbf{W}^L \cdot \mathbf{X} + \mathbf{b}^L \Rightarrow$ Matrices Operations
2. $\mathbf{A}^L = \sigma(\mathbf{Z}^L) \Rightarrow$ Element wise Operations

Outline

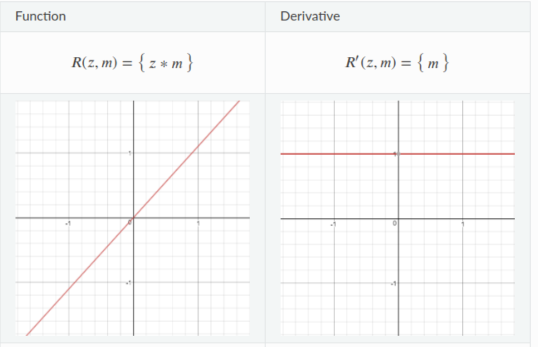
Activation functions

Activation functions typically have the following properties:

- **Non-linear:** To model complex relationships
- **Continuously Differentiable:** To improve our model with gradient descent.
- **Fixed Range Activation** functions typically squash the input data into a narrow range that makes training the model more stable and efficient.

Linear Activation Function

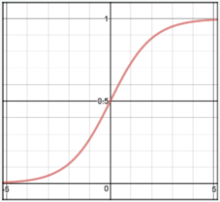
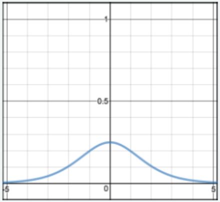
- The stacking of linear functions introduce nothing new. All layers of the neural network collapse into one.
- No one use it.
- It can blow up the activation.



Linear Activation Function

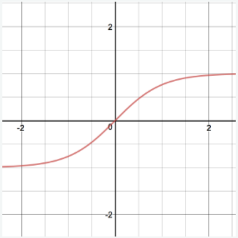
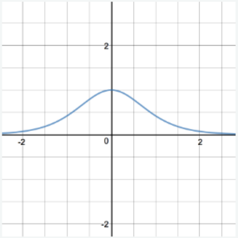
Sigmoid / Logistic

- The stacking is possible.
- Smooth gradient.
- Outputs not zero centered.
- Computationally expensive.

Function	Derivative
$S(z) = \frac{1}{1+e^{-z}}$	$S'(z) = S(z) \cdot (1 - S(z))$
	

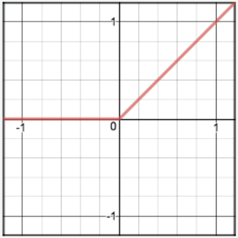
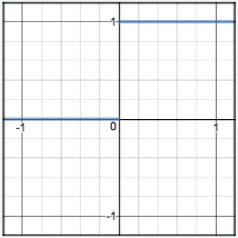
TanH / Hyperbolic Tangent

- The stacking is possible.
- Smooth gradient.
- Zero centered output
- Gradinet is steeper than the sigmoid.
- Computationally expensive.

Function	Derivative
$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$tanh'(z) = 1 - tanh(z)^2$
	

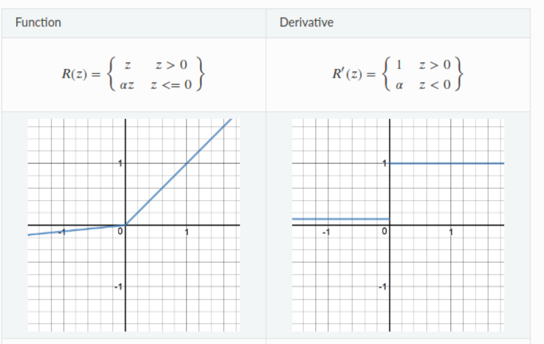
ReLU (Rectified Linear Unit)

- Computationally cheap.
- Suffer from the Dying ReLU problem when inputs approach zero, or are negative.
- Gradient not smooth.
- It can blow up the activation.

Function	Derivative
$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$	$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ 0 & z < 0 \end{Bmatrix}$
	

Leaky ReLU

- Computationally cheap.
- No Dying ReLU problem.
- Sometimes results are not consistent.
- Gradient not smooth.
- It can blow up the activation.



Leaky ReLU

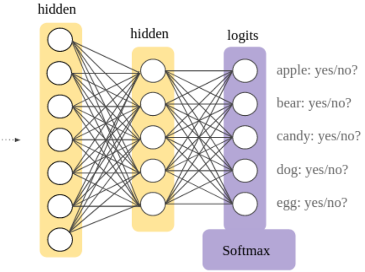
Multi-Class Neural Networks: Softmax Activation

Softmax extends binary logistic regression idea (probability adds upto 1) into a multi-class world.

- It helps training converge more quickly than it otherwise would.
- Usually have better performance against one vs. all classification.

$$p(y = j|\mathbf{z}) = \frac{e^{\mathbf{w}_j^T \mathbf{z} + b_j}}{\sum_{k \in K} e^{\mathbf{w}_k^T \mathbf{z} + b_k}}$$

- K is the number of classes. (j and k $\in K$)
- $\mathbf{z}$ is the input vector.
- $\mathbf{w}_{(.)}$ is the weight vector associated with each output neuron.



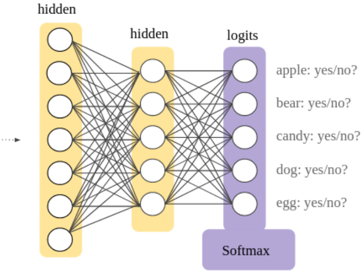
Multi-Class Neural Networks: Softmax Activation

Softmax extends binary logistic regression idea (probability adds upto 1) into a multi-class world.

- It helps training converge more quickly than it otherwise would.
- Usually have better performance against one vs. all classification.

$$p(y = j|\mathbf{z}) = \frac{e^{\mathbf{w}_j^T \mathbf{z} + b_j}}{\sum_{k \in K} e^{\mathbf{w}_k^T \mathbf{z} + b_k}}$$

- K is the number of classes. (j and k $\in K$)
- $\mathbf{z}$ is the input vector.
- $\mathbf{w}_{(.)}$ is the weight vector associated with each output neuron.



Why this ugly forumla?

Now we know how feedforward NNs do Computations.

Next, we will learn how to adjust the weights to learn from data.

References

Intel Nervana AI Academy
https://software.intel.com/content/www/us/en/develop/training

Questions