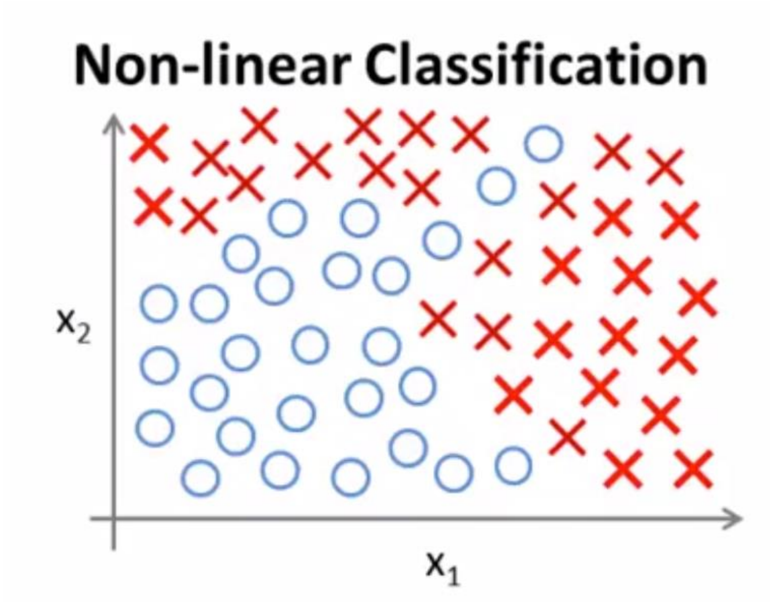
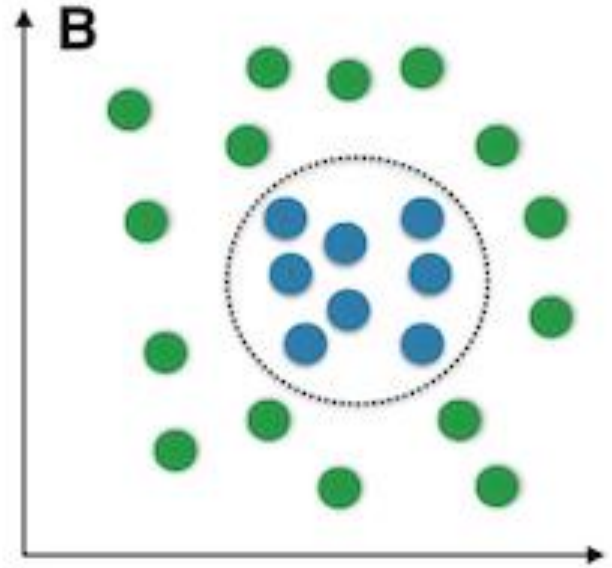
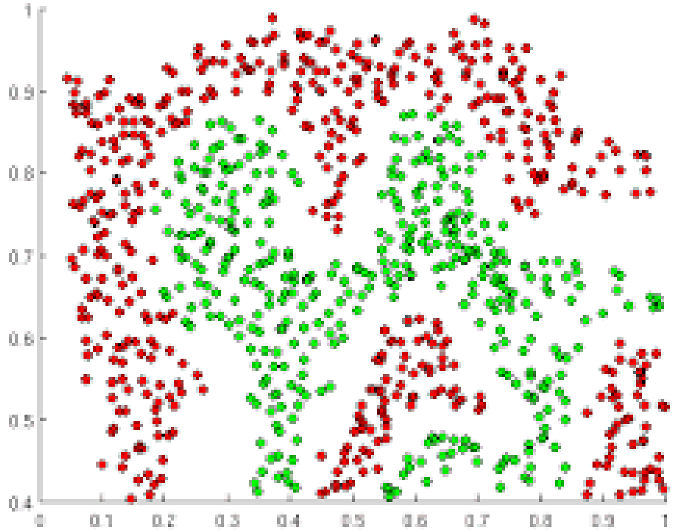
Classification Model

Nonlinear classification

- Decision Boundary is complex hyperplane

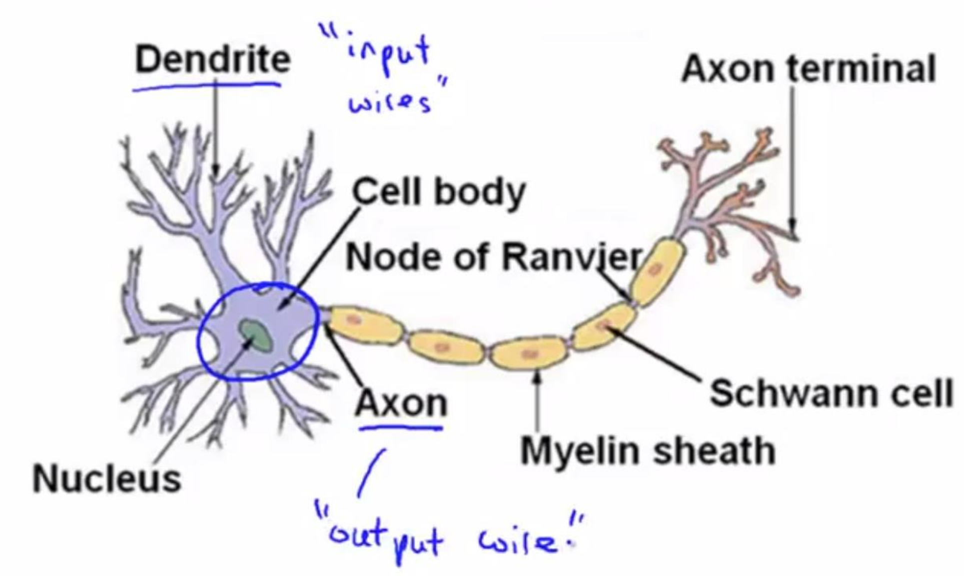






Neural network model

Neural network: history



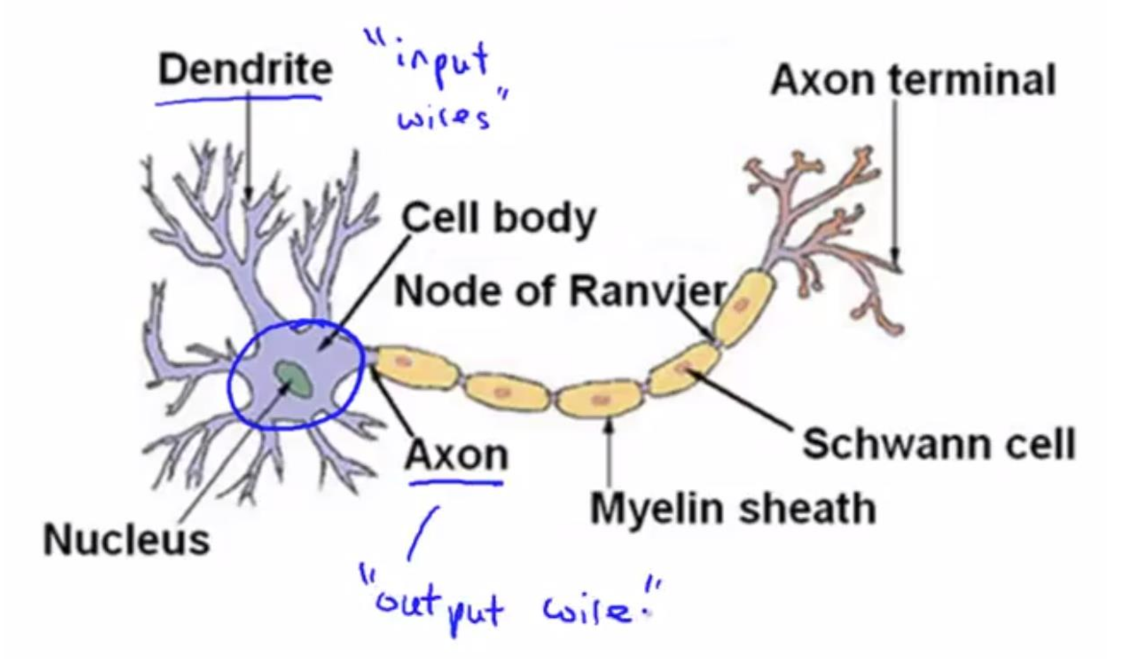
Origins: Algorithms that try to mimic the brain.

Was very widely used in 80s and early 90s; popularity diminished in late 90s.

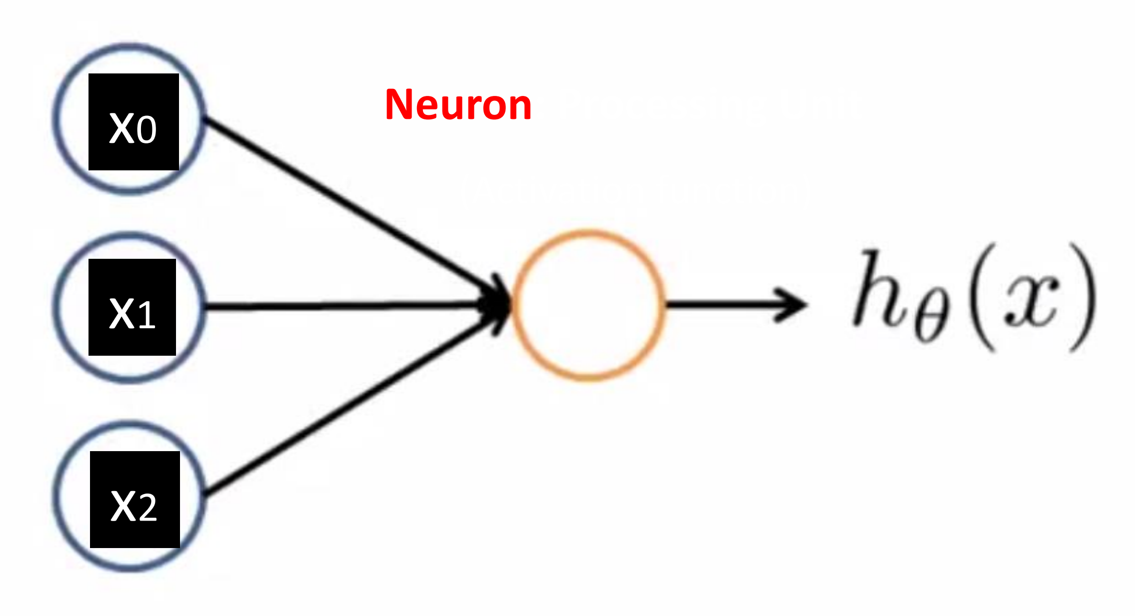
Recent resurgence: State-of-the-art technique for many applications

https://www.coursera.org/learn/machine-learning/lecture/IPmzw/neurons-and-the-brain

Neural network model

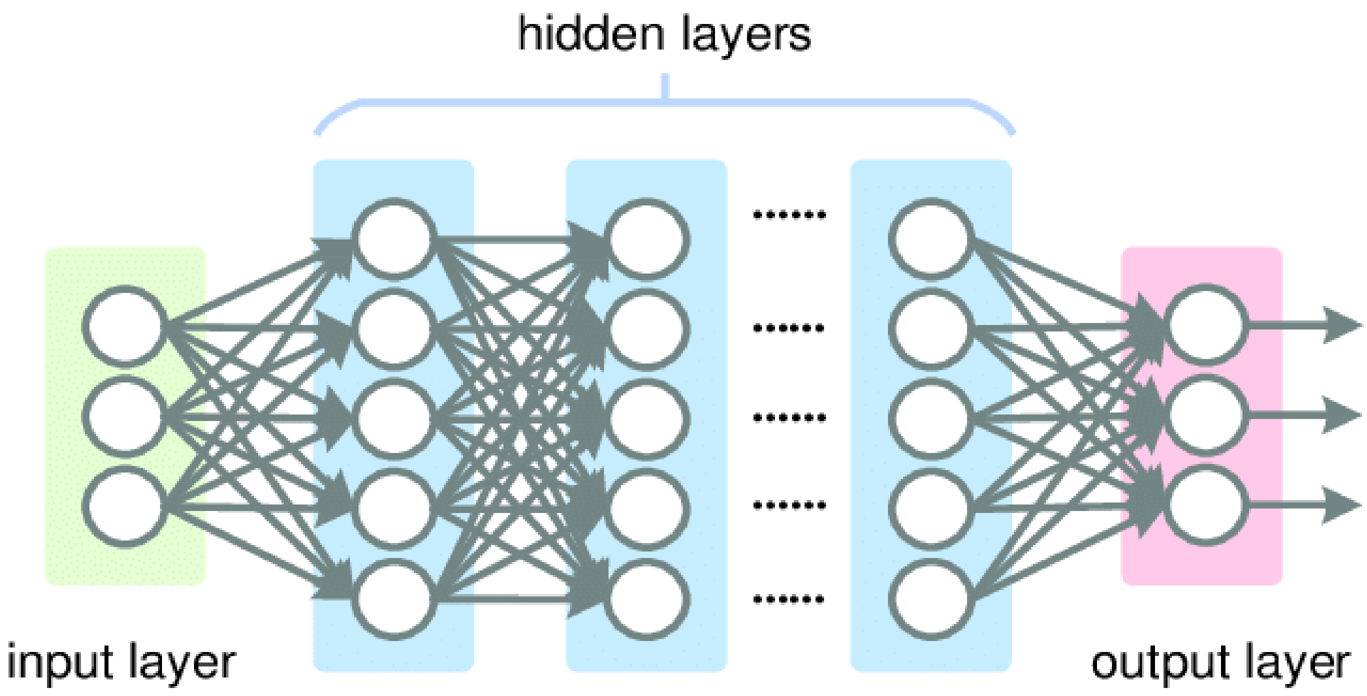


https://www.coursera.org/learn/machine-learning/lecture/IPmzw/neurons-and-the-brain



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

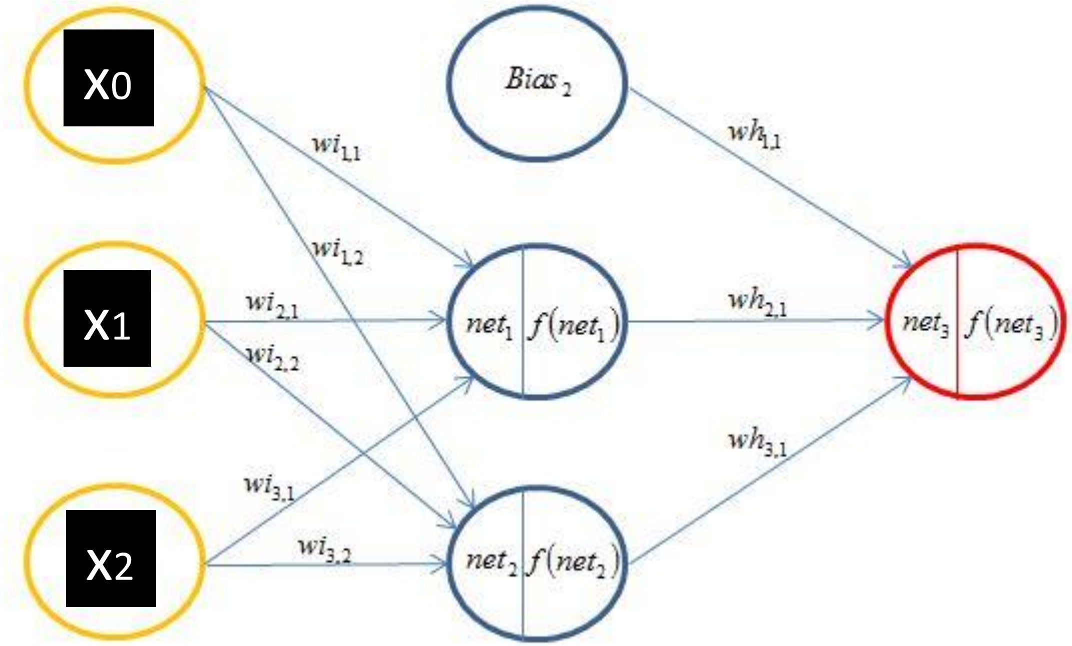
Neural network model



- Multi-Layer Perceptron (MLP)
  - Layer Types:
    - Input / hidden / output
  - Fully mesh connection
    - fully connected
    - All outputs of previous nodes are connected to each current node
    - Number of weights (zeta) for layer j
      - N(j-1) x N(j)

Neural network model

Neuron: Processing Unit



$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ matrix of weights controlling function mapping from layer j to layer $j+1$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$net_i = \theta^T x = w^T x$

$f(net_i) = a_i^j$

Which activation function can we use?

Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

net$_i$ = $\theta^T x = w^T x$

$f(net_i) = a_i^j$

- A function used to determine output of each neuron (node)
- Types of activation functions
  - Linear Activation
  - Non linear activation
    - Sigmoid (logistic) function
    - Hyperbolic Tangent function (Tanh)
    - Rectified Linear Unit (Relu)
    - Leaky ReLU

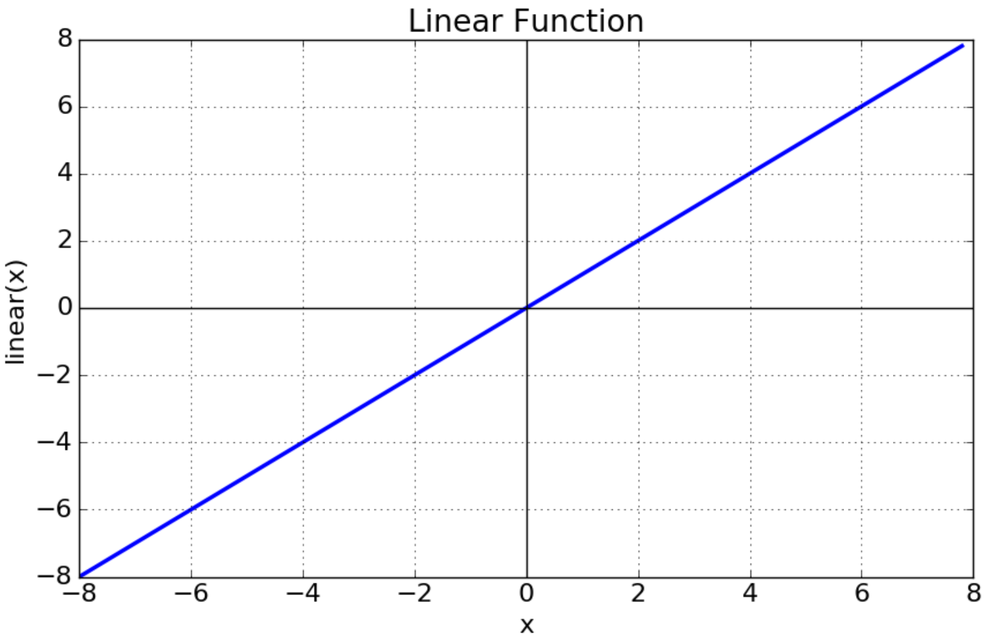
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

net$_i$ = $\theta^T x = w^T x$

$f(net_i) = a_i^j$



Equation : f(x) = x

Range : (-infinity to infinity)

It doesn’t help with the complexity or various parameters of usual data that is fed to the neural networks.

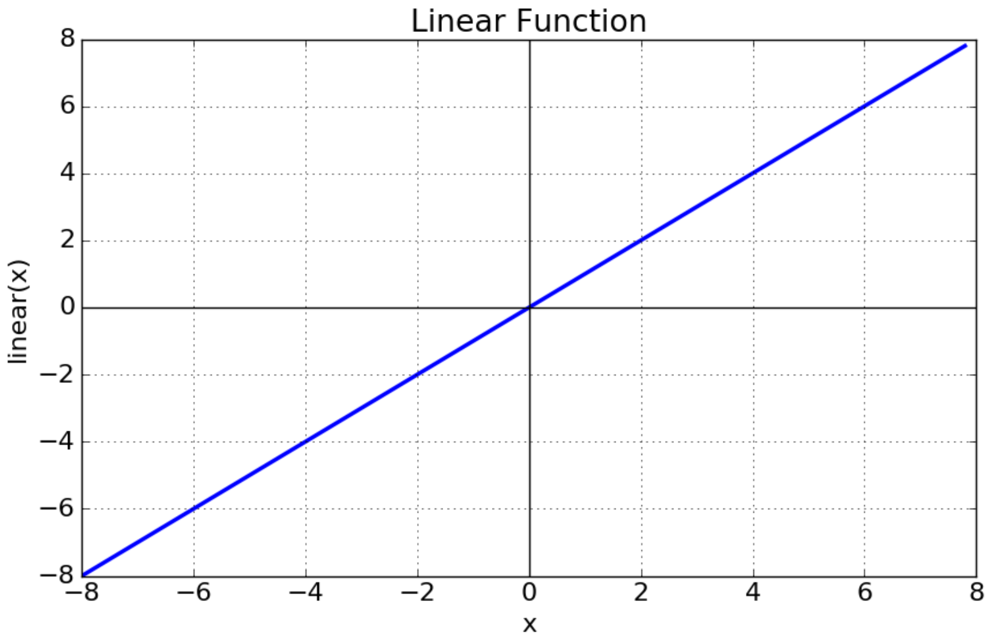
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Linear Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$net_i = \theta^T x = w^T x$

$f(net_i) = a_i^j$



Equation : f(x) = x

Range : (-infinity to infinity)

It doesn’t help with the complexity or various parameters of usual data that is fed to the neural networks.

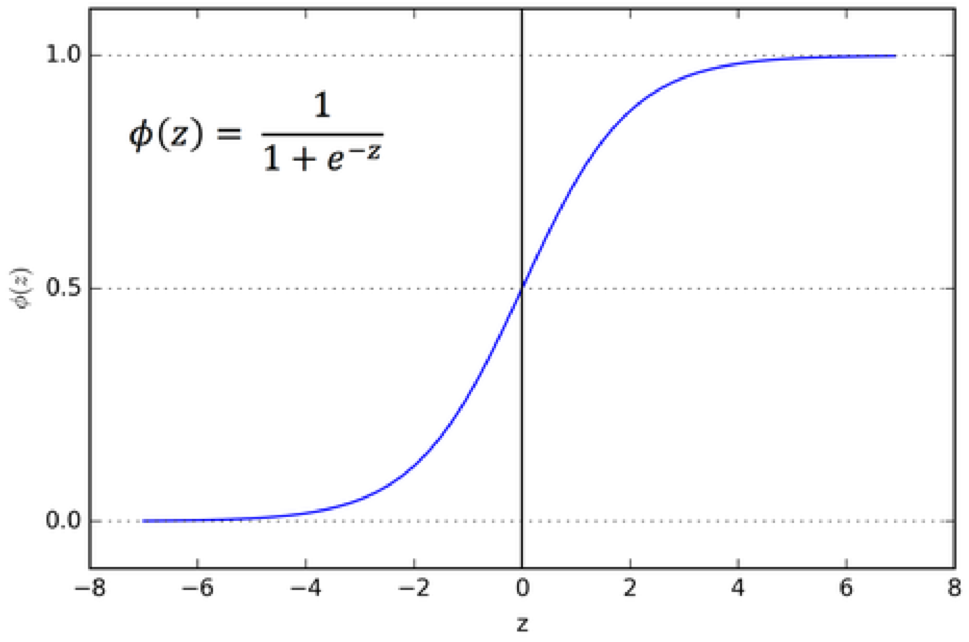
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Nonlinear Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

net$_i$ = $\theta^T x = w^T x$

$f(net_i) = a_i^j$



Equation : f(x) = $\frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta^T x}}$

zRange : (-infinity to infinity)

f(x) Range: [0,1]

- **Sigmoid Function curve looks like a S-shape.**
- **It is used for models where we have to predict the probability**
-
- **Negative of z gets mapping to positive f(z) <0.5**

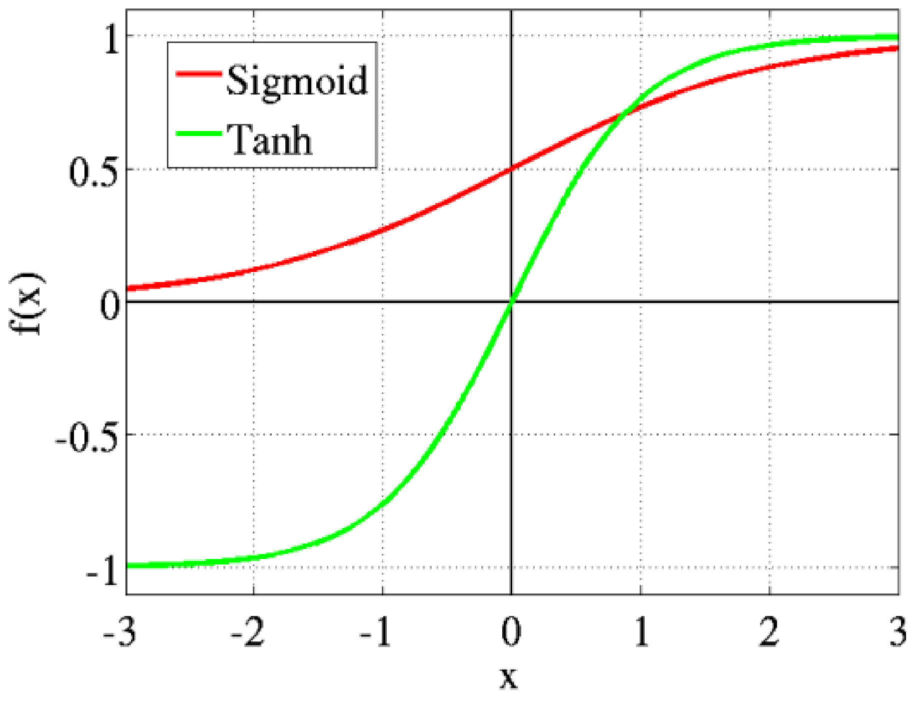
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Nonlinear Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$net_i = \theta^T x = w^T x$

$f(net_i) = a_i^j$



Equation : f(x) = tanh(x)

x Range : (-infinity to infinity)

f(x) Range: [-1,1]

- **tanh is also like logistic sigmoid but better.**
- **Negative of x gets mapping to negative f(x)**

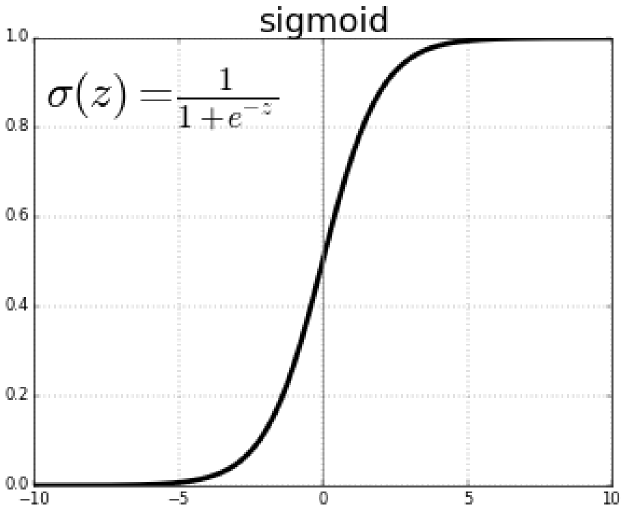
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

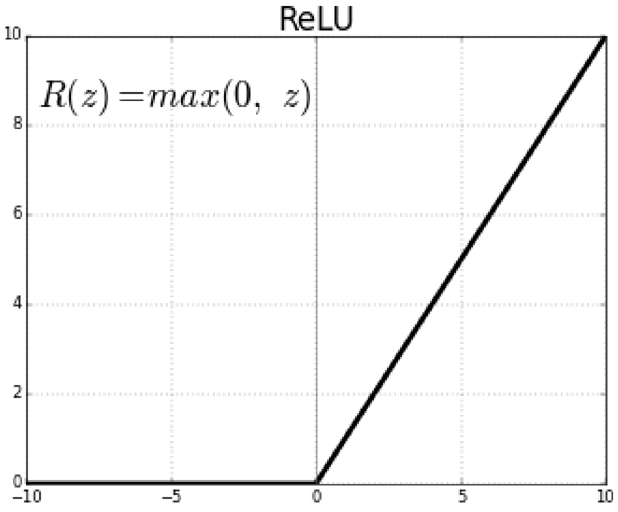
Nonlinear Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$net_i = \theta^T x = w^T x$

$f(net_i) = a_i^j$





Equation : f(z) = max(0, z)

z Range : (-infinity to infinity)

f(z) Range: [0,infinity)

- **ReLU is most used activation function in the world right now.**
- **f(z >=0) = z (positive z)**
- **Eliminate negative input z -> f(z<0) = 0**
  - **To reduce computational time**

All the negative values become zero immediately

- Decreases the ability of the model to fit or train from the data properly.

- Any negative input given to the ReLU activation function turns the value into zero, which results in not mapping the negative values appropriately.

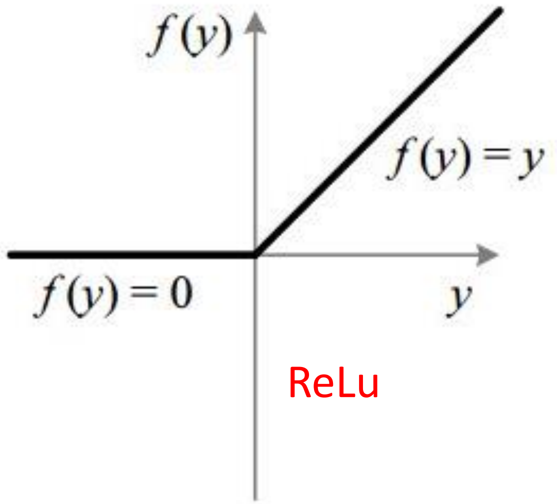
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

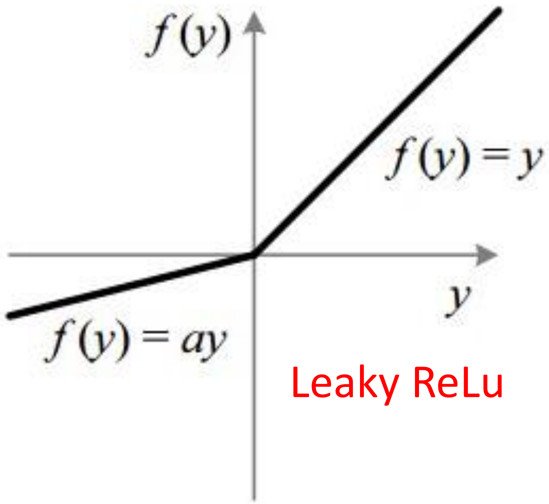
Nonlinear
Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

net$_i$ = $\theta^T x = w^T x$

$$f(net_i) = a_i^j$$





Equation : f(y) = $\begin{cases} ay & y < 0 \\ y & y \geq 0 \end{cases}$

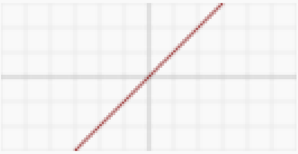
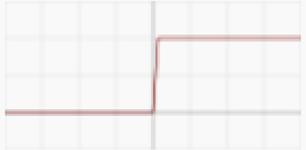
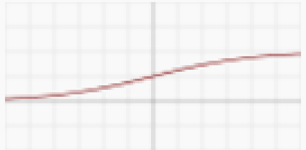
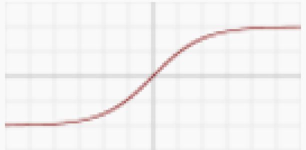
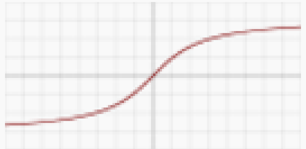
y Range : (-infinity to infinity)

a: small value (typically 0.01)

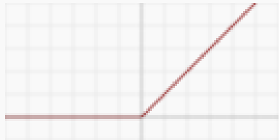
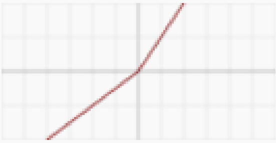
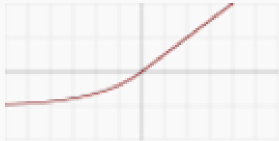
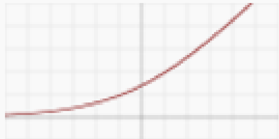
- **Leaky ReLU**
  - **Approximate negative input y = ay**
  - **Allow a very small leak f(y<0)**

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

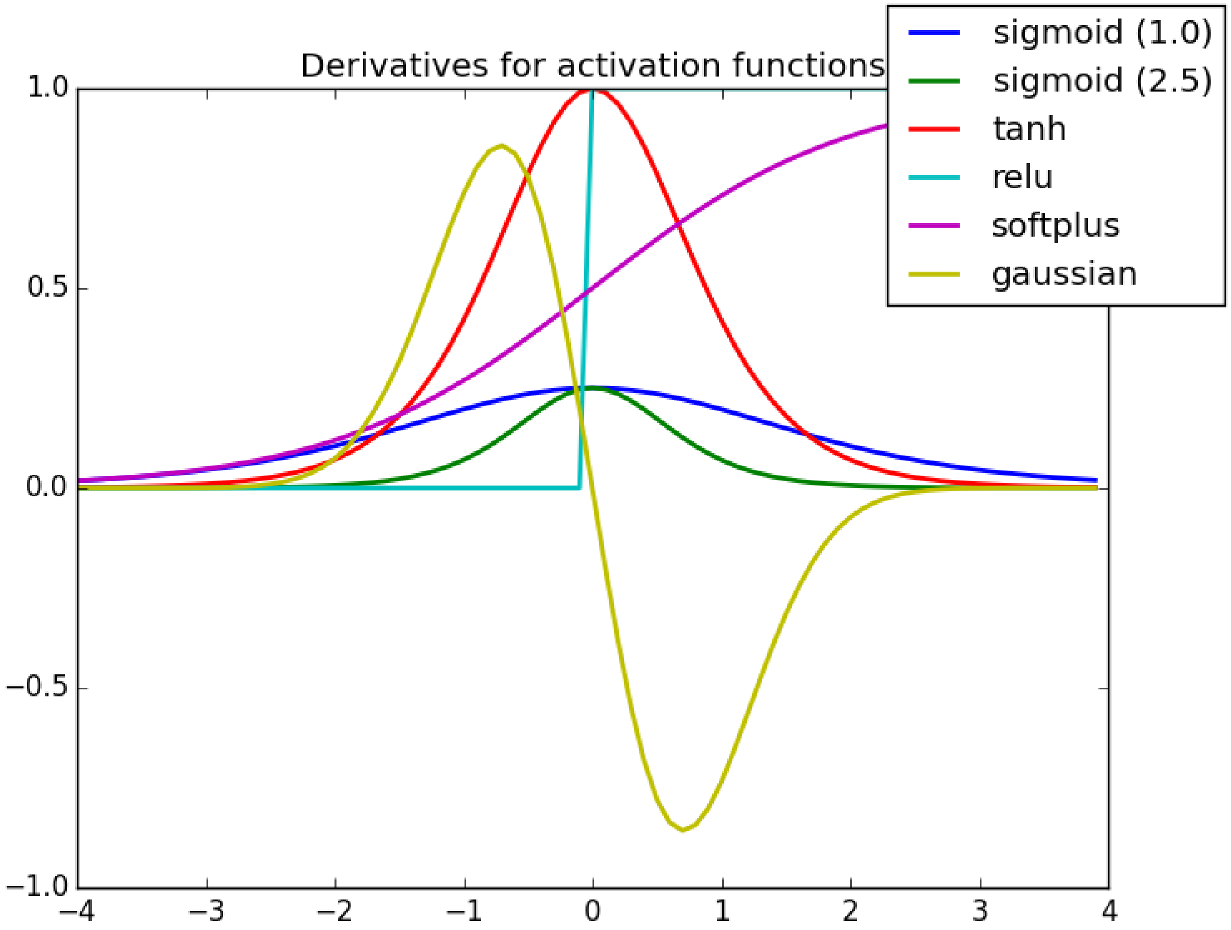
Activation function and its derivative

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$

Logistic regression for Classification

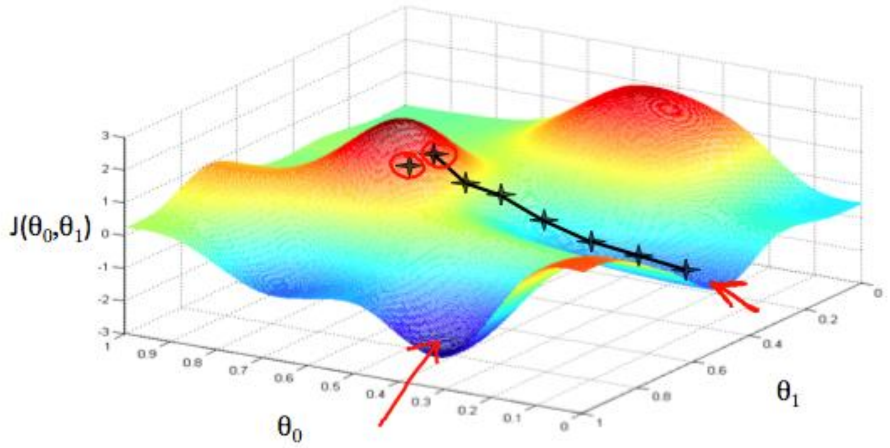
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \dfrac{1}{1 + e^{-x}}$

Logistic regression for Classification



Why do we need to know activation derivative function?

- Derivative form of activation function
  - Used in optimization process of gradient descent search
- Optimizer looks for **directions of weights to move** for next search iteration



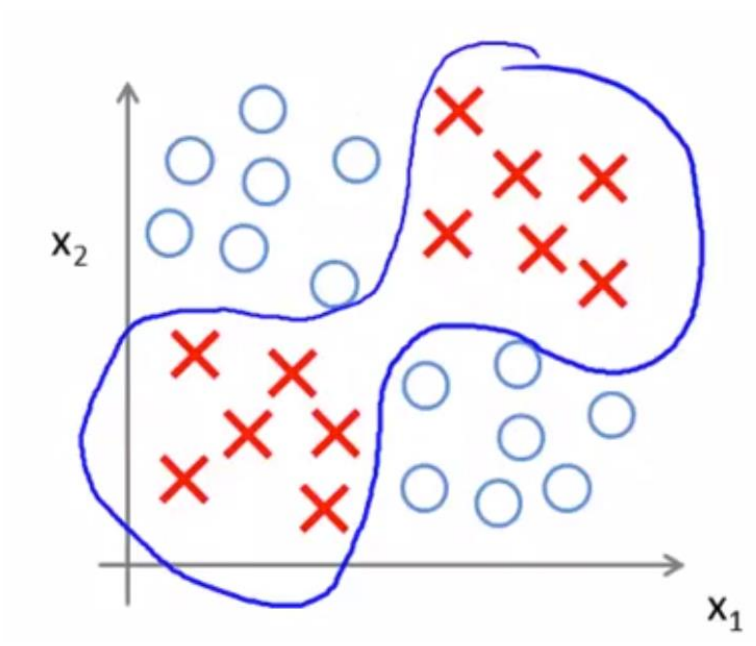
The gradient descent algorithm is:

repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

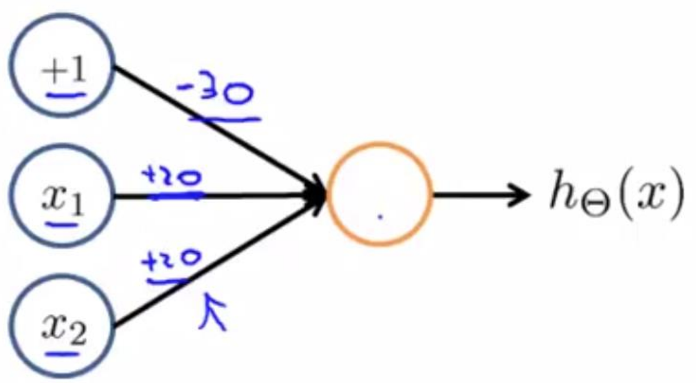
Neural model example

Neural model example

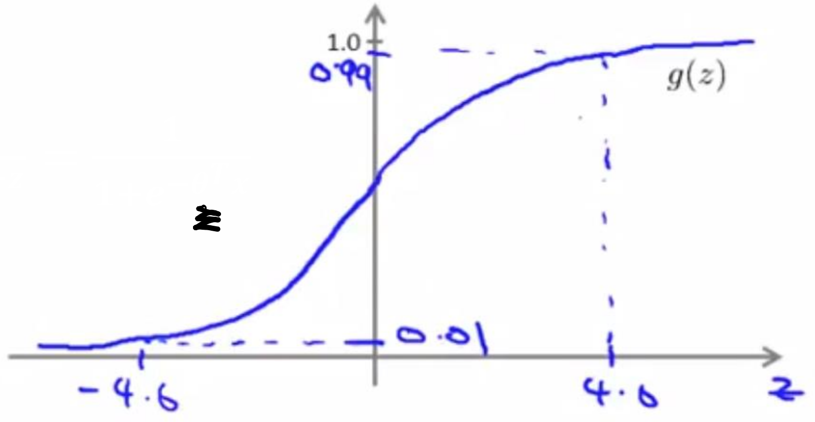


Simple example: AND

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$

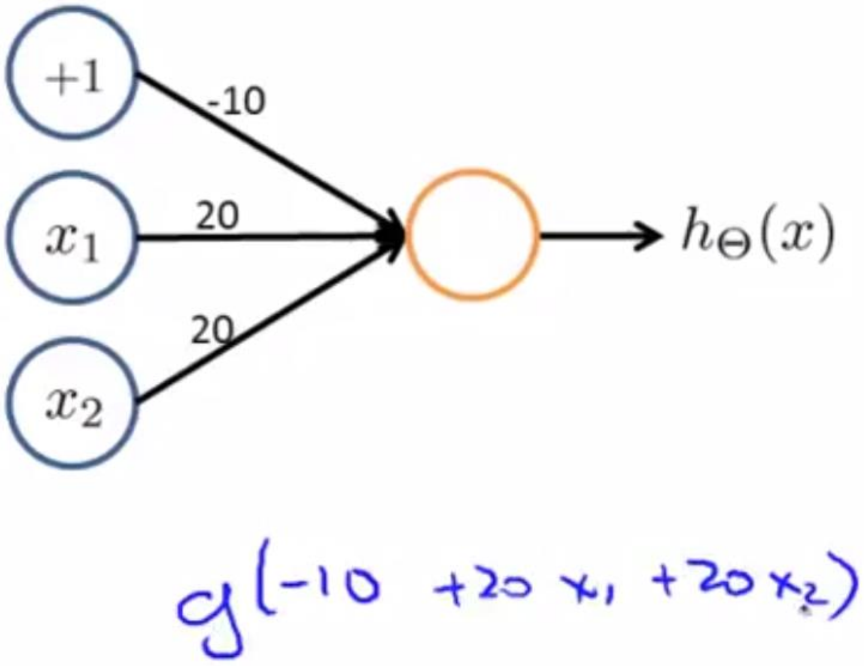


$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$



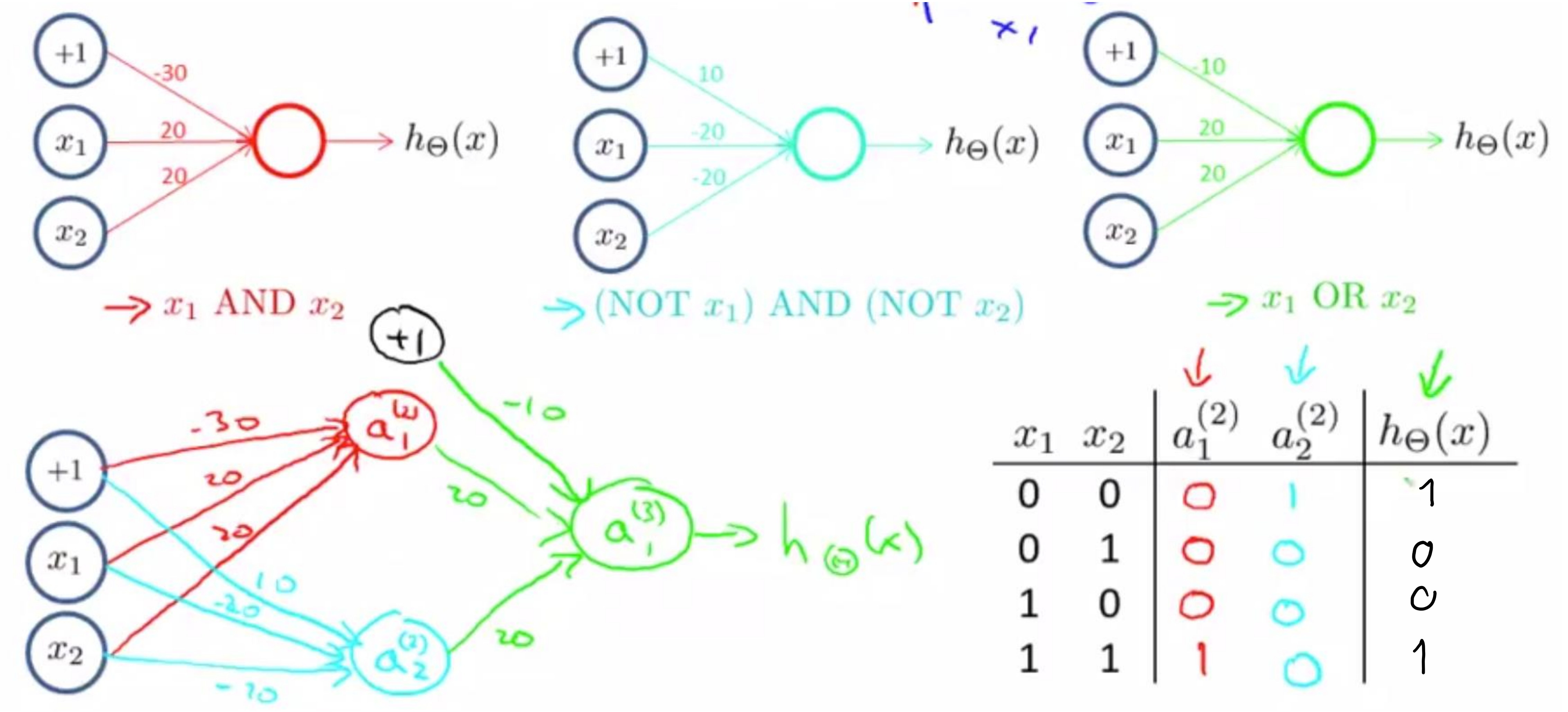
x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

Neural model example



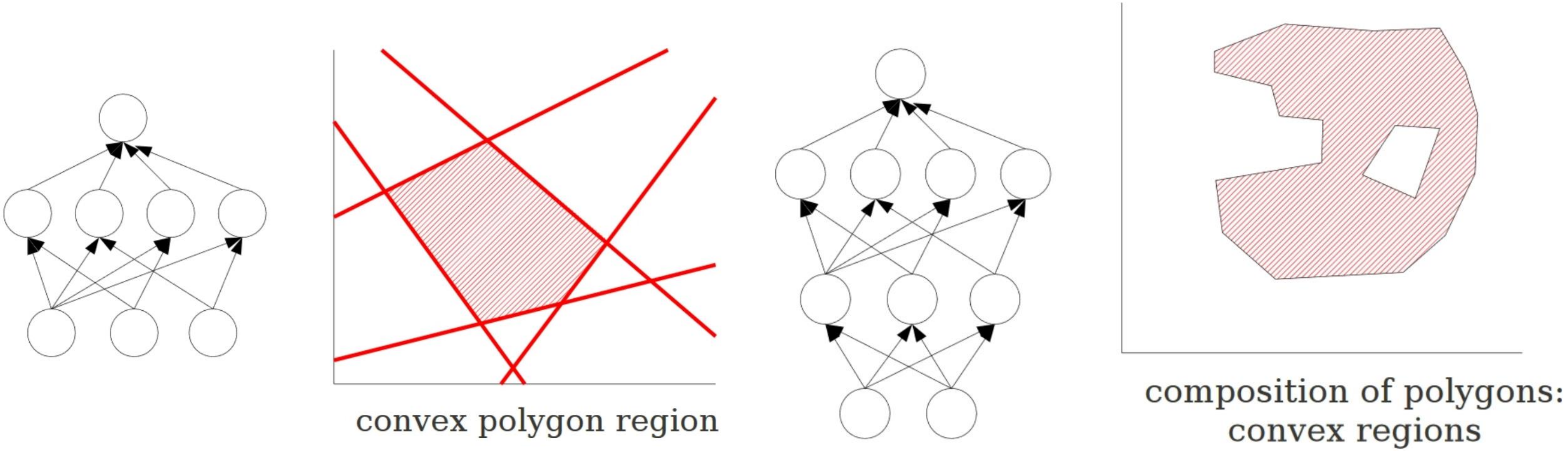
x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

Neural model example



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

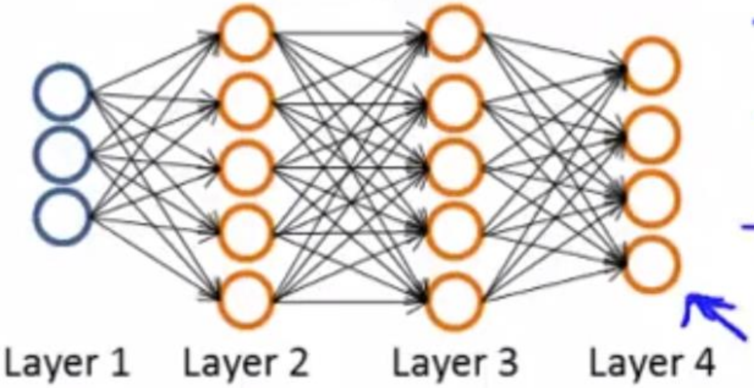
Neural model generate complex hyperplane decision



Multi-class classification

Binary vs multi-classification

Neural Network (Classification)



$\rightarrow \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$\rightarrow L =$ total no. of layers in network $\quad L = 4$

$\rightarrow s_l =$ no. of units (not counting bias unit) in layer l $\quad s_1 = 3, \; s_2 = 5, \; s_4 = s_L = 4$

Binary classification

$y = 0$ or $1 \leftarrow$

1 output unit $\leftarrow$

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units

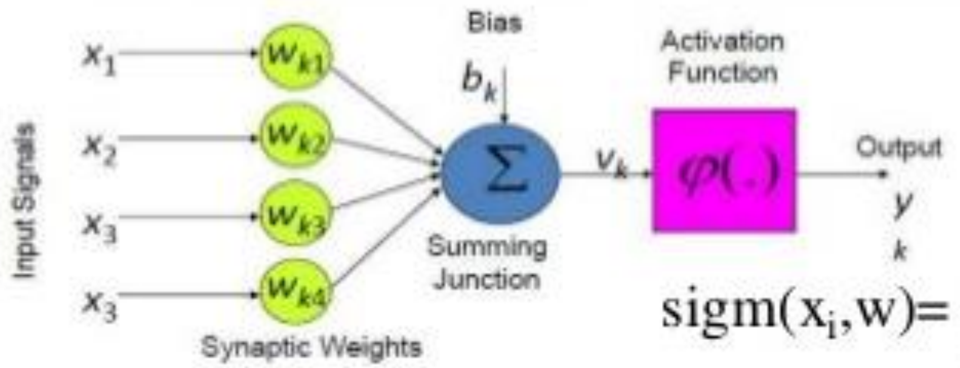
ONE-vs-ALL

output
Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$net_i = \theta^T x = w^T x$

$f(net_i) = a_i^j$



$$\text{sigm}(x_i, w) = \frac{1}{1+e^{-x_i w}}$$

Equation : $f(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta^T x}}$

z Range : (-infinity to infinity)

y: small value (typically 0.01)

- **Sigmoid is generally used for output activation function for Binary Classification**

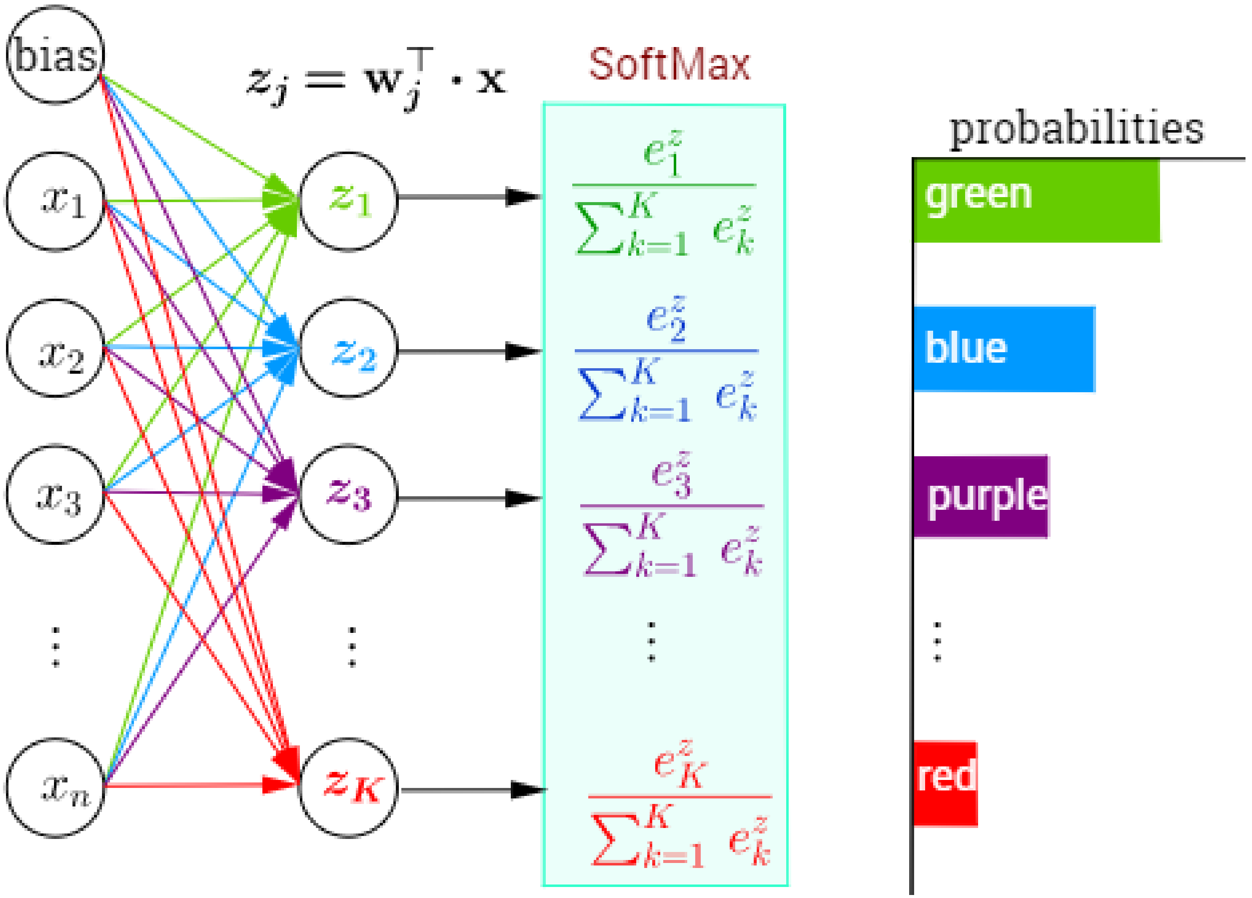
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

output
Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$net_i = \theta^T x = w^T x$

$f(net_i) = a_i^j$



Equation : $f(z) = \dfrac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

z Range : (-infinity to infinity)

y: small value (typically 0.01)

- **Generalized Logistic Regression to handle multiple classes**
- Calculate the **probabilities of each target class over all possible target classes**.
    - **Turns numbers into probabilities that sum to one.**

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

output
Activation functions

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

net$_i$ = $\theta^T x = w^T x$

$f(net_i) = a_i^j$

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \xrightarrow{\text{Softmax}} \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$

$$\sum_{k=1}^{3} e^{zk} = 0.5 + 2.3 + 0.8 = 3.6$$

$$\begin{bmatrix} 0.5 \\ 2.3 \\ 0.8 \end{bmatrix} \xrightarrow{\text{SOFTMAX}} \begin{bmatrix} \frac{0.5}{3.6} = 0.14 \\ \frac{2.3}{3.6} = 0.64 \\ \frac{0.8}{3.6} = 0.22 \end{bmatrix}$$

Equation : f(z) = $\dfrac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

y Range : (-infinity to infinity)

a: small value (typically 0.01)

- **Generalized Logistic Regression to handle multiple classes**
- Calculate the **probabilities of each target class over all possible target classes**.
  - **Turns numbers into probabilities that sum to one.**

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

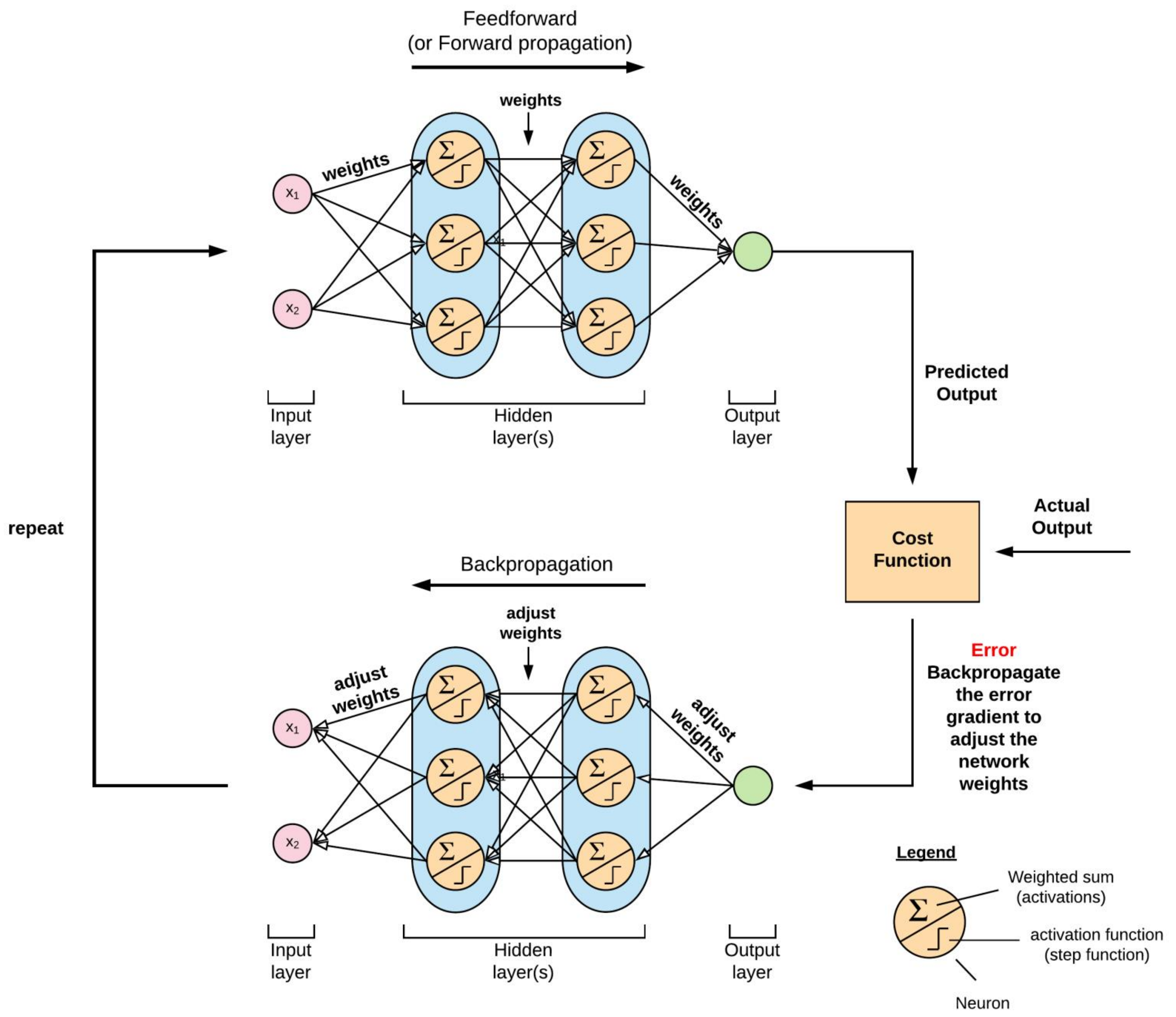
Neural network terminology

Neural network terminology

- Neuron
  - Neural network node (perceptron)
    - Types: Input / hidden / output
- Activation function
  - Neuron transfer function of weighted input from input / hidden layers
- Feed forward vs Backpropagation

Feed forward vs backpropagation

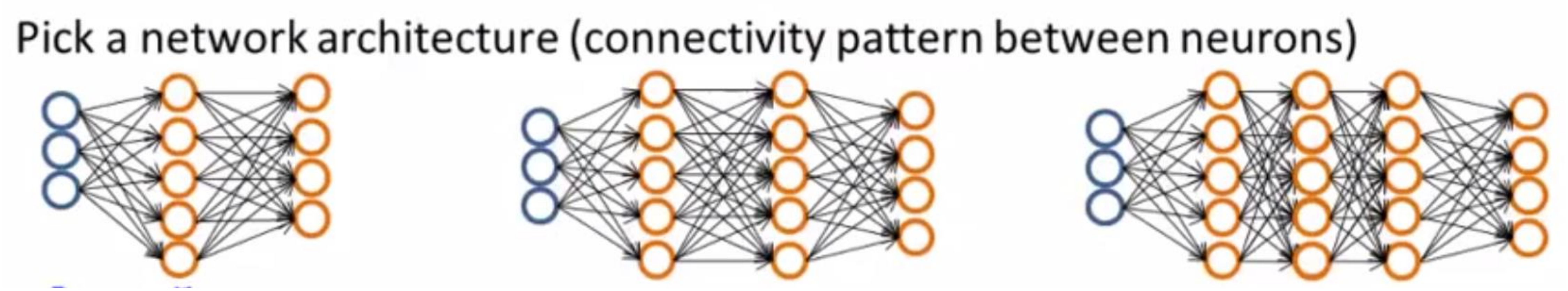
- Feed Forward NN
  - information flows in only one direction i.e. from input layer to output layer.
- Backpropagation
  - algorithm to train (adjust weight) of neural network.
  - Error in result is then communicated back to previous layers now.
  - Nodes get to know how much they contributed in the answer being wrong. Weights are re-adjusted.



https://medium.com/datadriveninvestor/math-neural-network-from-scratch-in-python-d6da9f29ce65

Neural network training process

Neural network training process



- **Initial NN model (architecture)**
  - n input nodes
  - h hidden layers
  - hj hidden nodes in jth layer

Neural network training process

- **1. Initial NN weights (θ_i^j) in all layers**
  - **ith node / jth layer**
- **2. Perform feed forward to get final output ($h_\theta(x)$)**
  - **Calculate for all training input dataset (x)**
- **3. Compute cost function and backpropagation**
  - **Error derivative of cost function**
- **4. Gradient Descent optimization with learning factor (α)**

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Iterative control parameters

- **hidden_layer_sizes**

- **Solver (Parameter Optimizer):**
  - **Adam (default: family of quasi-Newton methods)**
  - **SGD: Stochastic Gradient Descent**
  - **L-BFGS: Limited-memory BFGS (Broyden–Fletcher–Goldfarb–Shanno)**

- **Activation (*default ‘relu’, ‘logistic’, ‘tanh’)***
- **Learning rate factor**
- **Max. iteration (max_iter)**

Iterative control parameters

- **from sklearn.neural_network import MLPClassifier**
  - **clf = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)**

กำหนด Model Architecture

(#hidden layer, #node in each hidden layer)

- **Optimizer (solver):** Adam (default)

กำหนด Optimizer (solver) โดย default -> Adam

- **mlp.fit(X_train,y_train)**

Training

- **Training results**
  - **MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,**
  - **beta_2=0.999, early_stopping=False, epsilon=1e-08,**
  - **hidden_layer_sizes=(13, 13, 13), learning_rate='constant',**
  - **learning_rate_init=0.001, max_iter=500, momentum=0.9,**
  - **nesterovs_momentum=True, power_t=0.5, random_state=None,**
  - **shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,**
  - **verbose=False, warm_start=False)**

All parameters