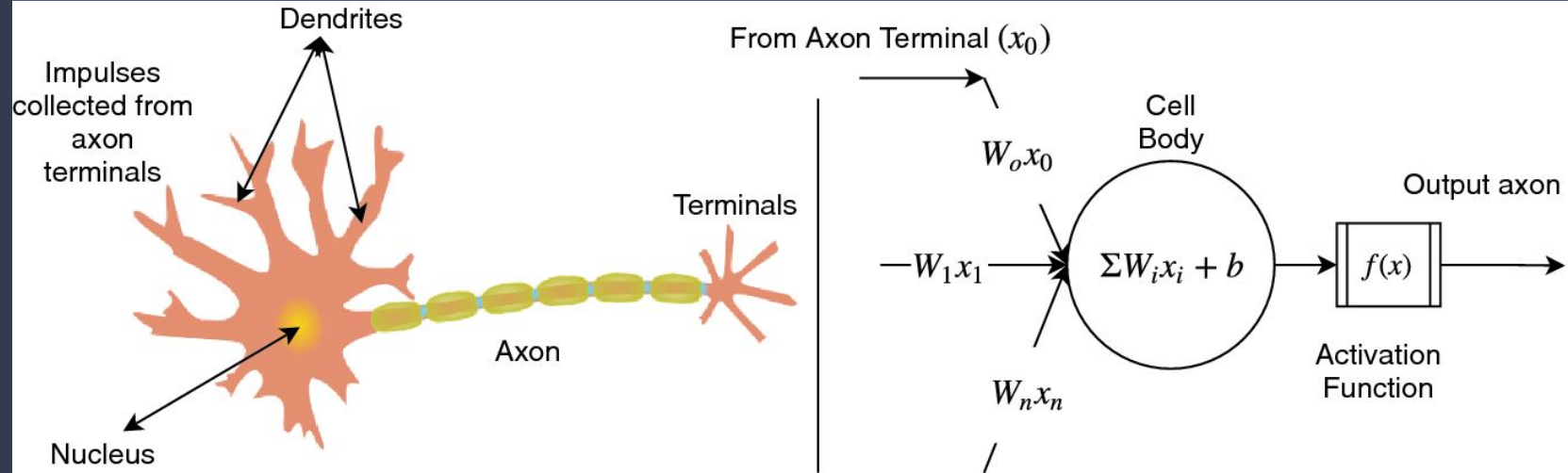
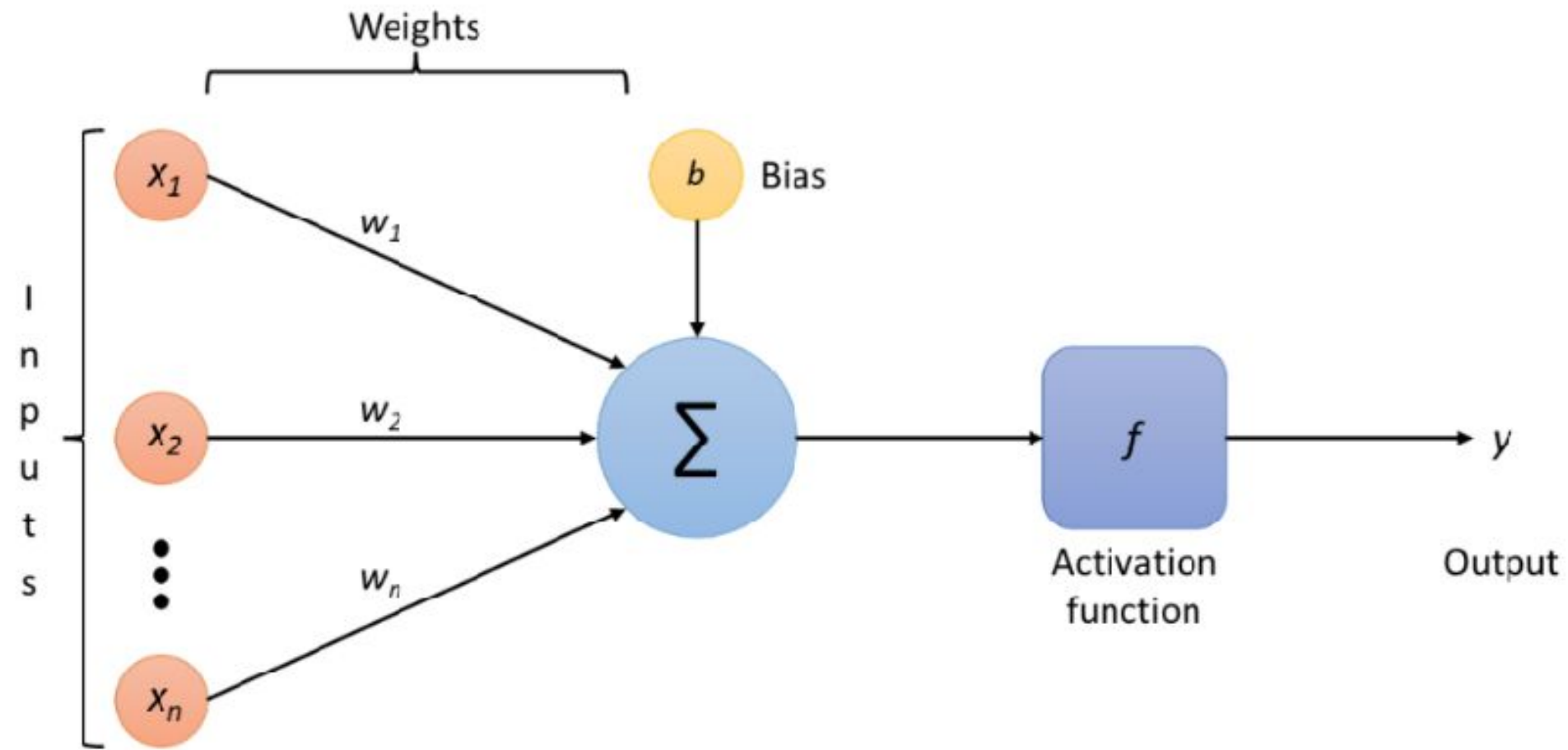


# Index Class - Neural Networks

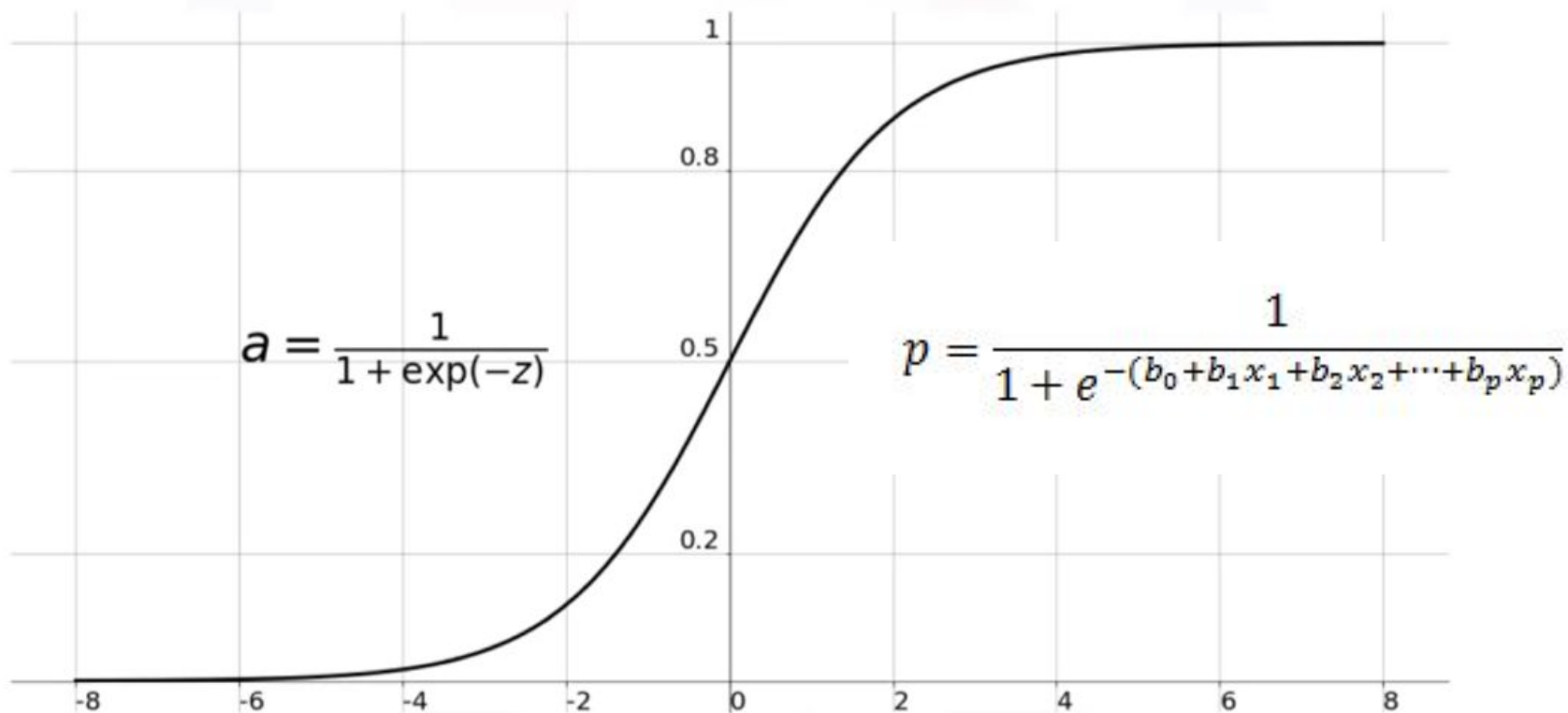
- Neural Networks, inspiration
- Units and hidden layers
- Activation function
- Boolean representations
- Stochastic Gradient Descent
- Batches & Epochs
- Momentum
- Regularization & Dropout

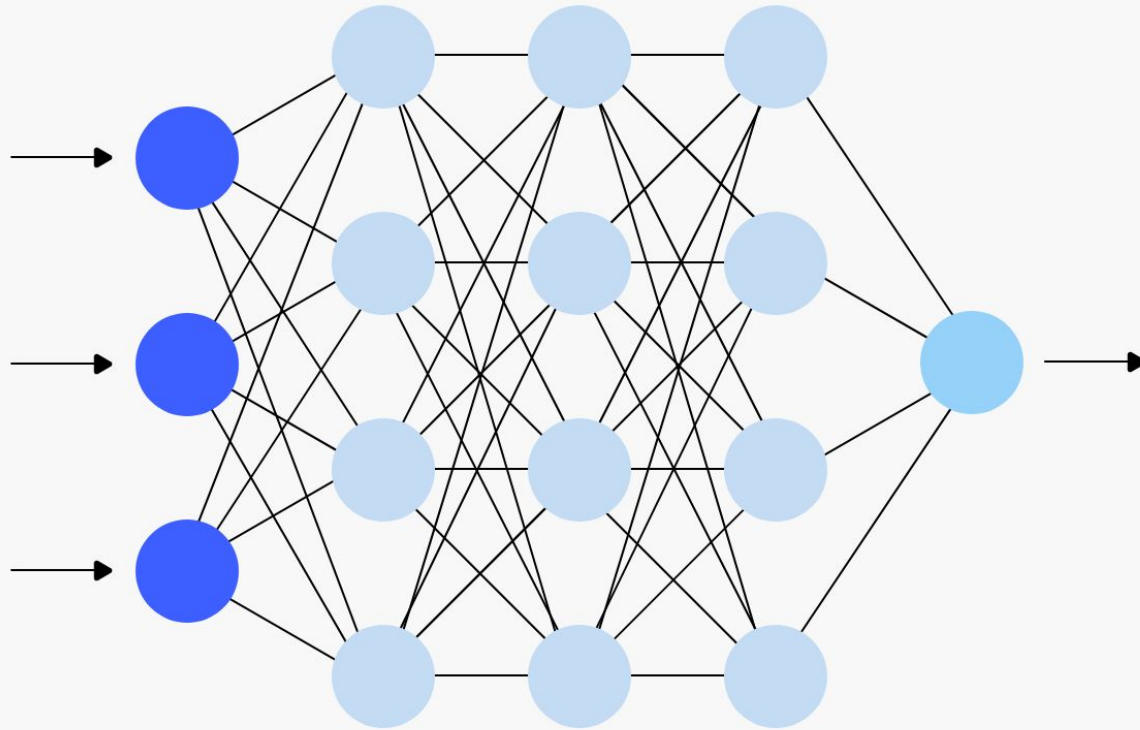
# Activation function






# Sigmoid Function

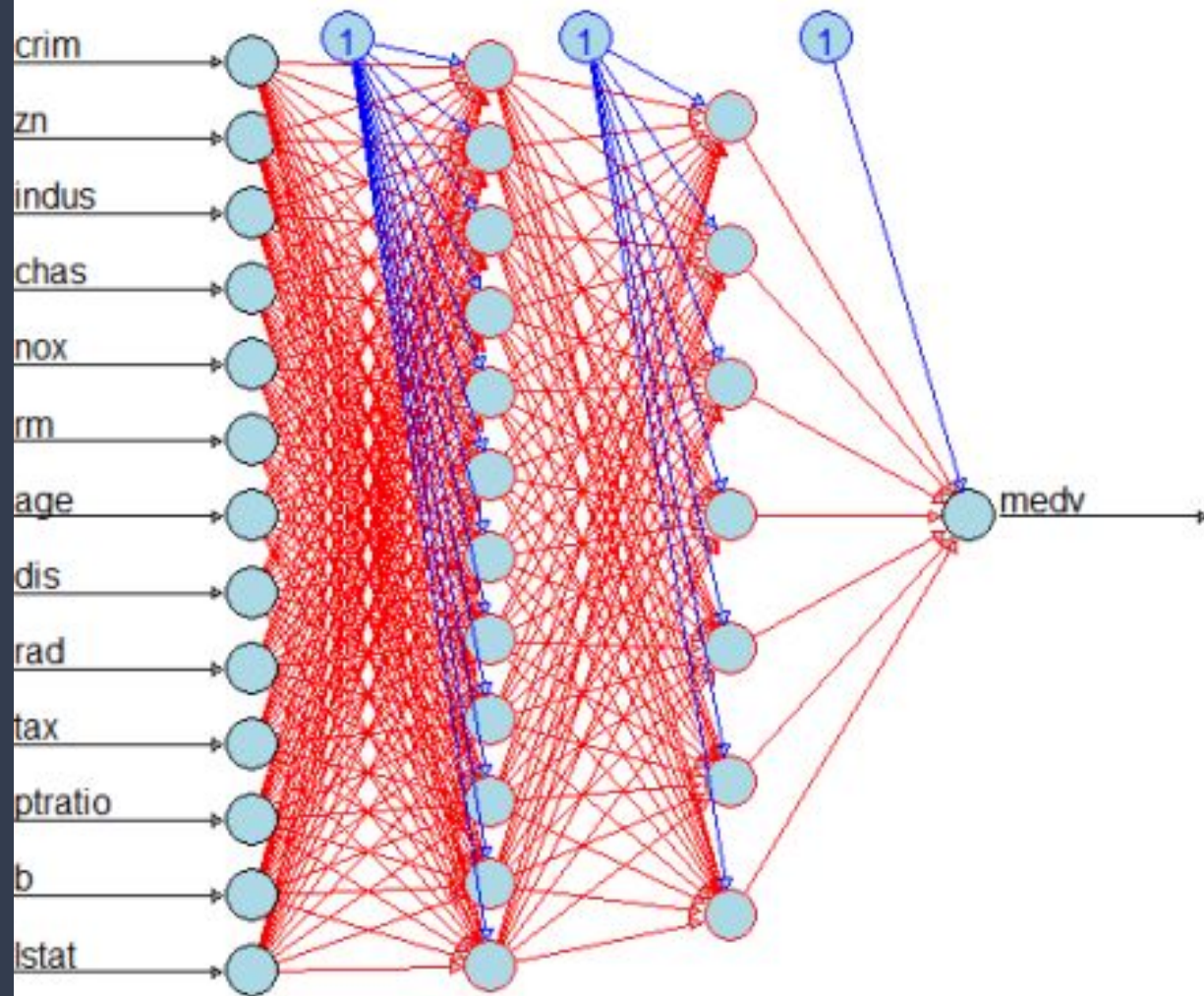




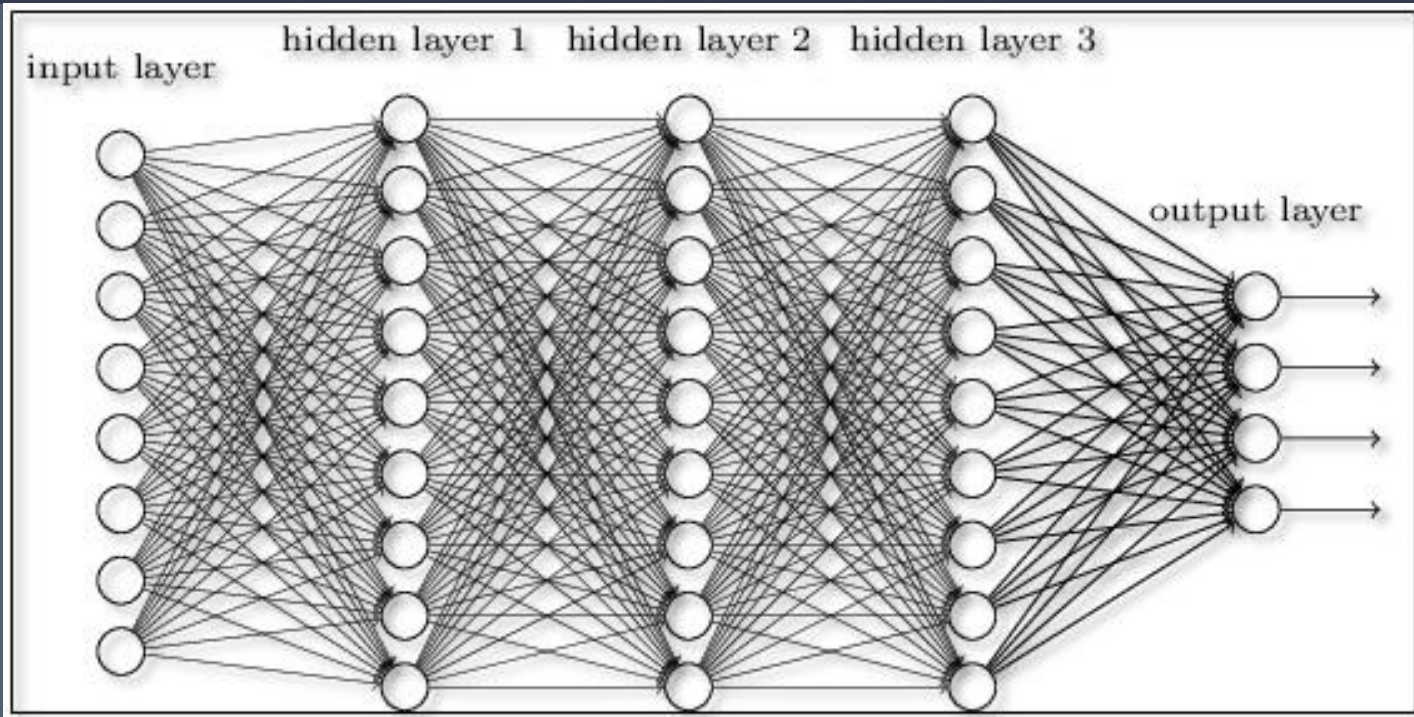
 Input  
Layer

 Hidden  
Layers

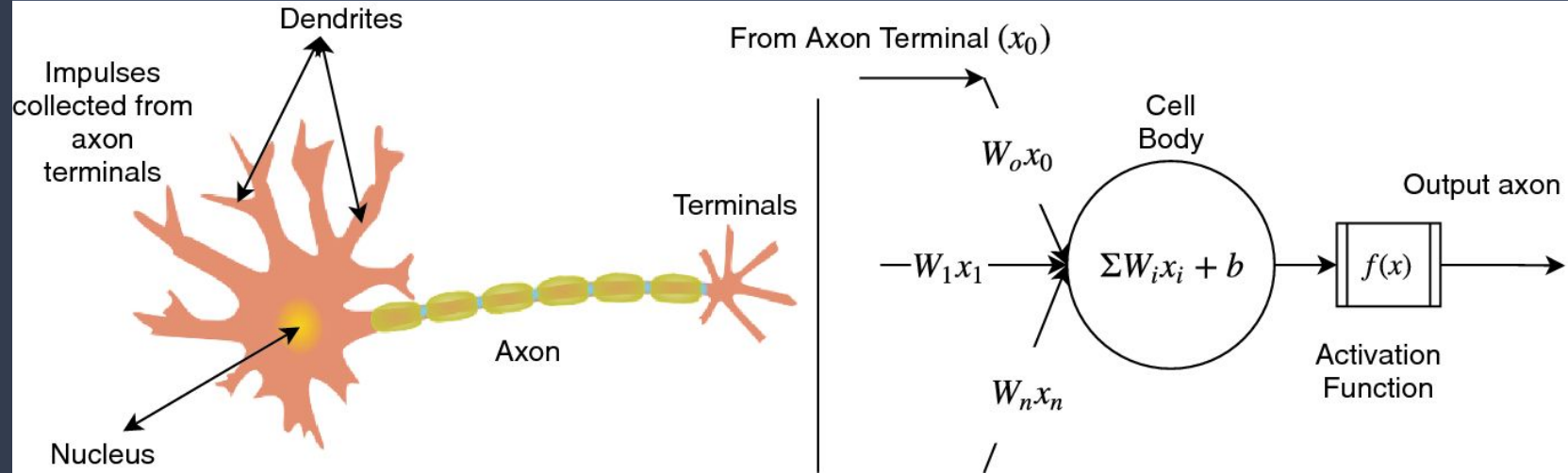
 Output  
Layer



# Neural Network

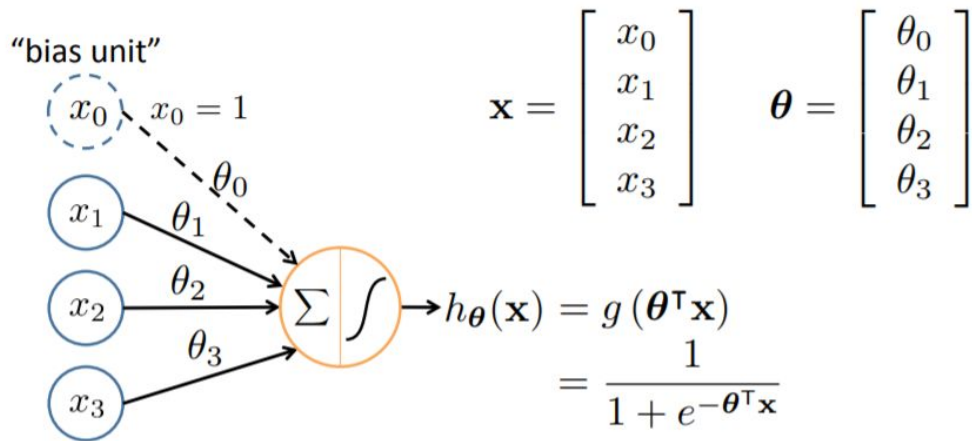


# Activation function





# Logistic Unit



Sigmoid (logistic) activation function:  $g(z) = \frac{1}{1 + e^{-z}}$

# Rectified Linear Unit

## 3. ReLU (Rectified Linear Unit) Activation Function

The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning.

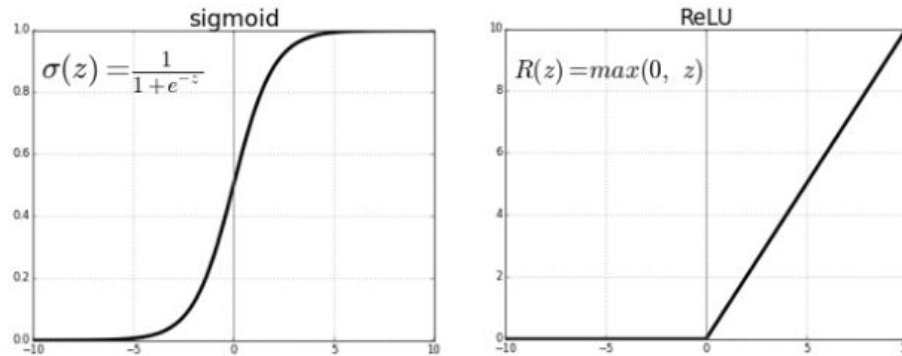
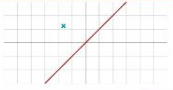

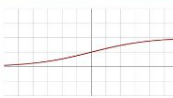
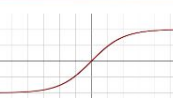




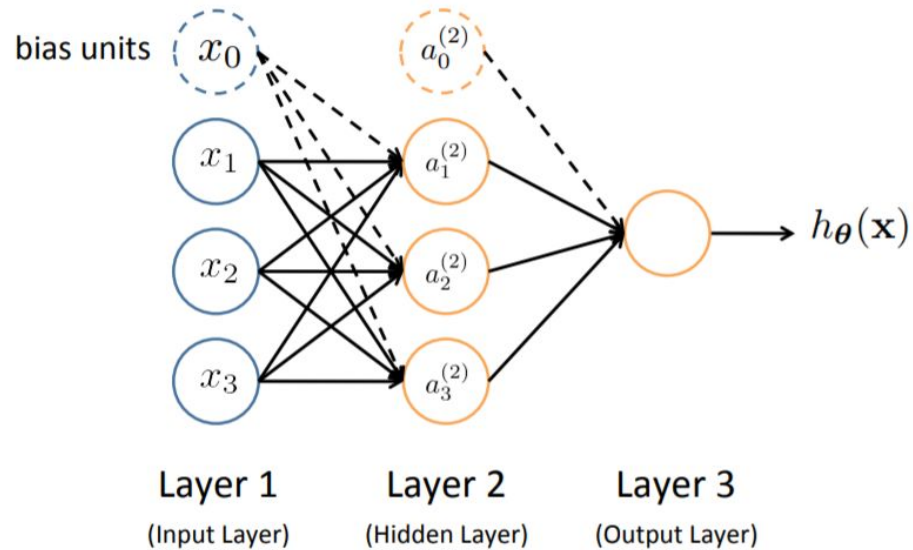


Fig: ReLU v/s Logistic Sigmoid

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

# Hidden Layers



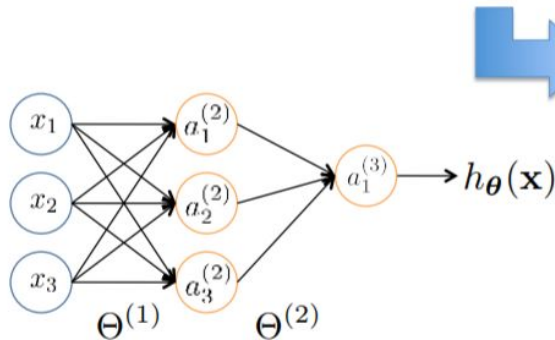
# Vectorization-Feed-Forward Process

$$a_1^{(2)} = g \left( \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3 \right) = g \left( z_1^{(2)} \right)$$

$$a_2^{(2)} = g \left( \Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3 \right) = g \left( z_2^{(2)} \right)$$

$$a_3^{(2)} = g \left( \Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3 \right) = g \left( z_3^{(2)} \right)$$

$$h_{\Theta}(\mathbf{x}) = g \left( \Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)} \right) = g \left( z_1^{(3)} \right)$$



## Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)} \mathbf{x}$$

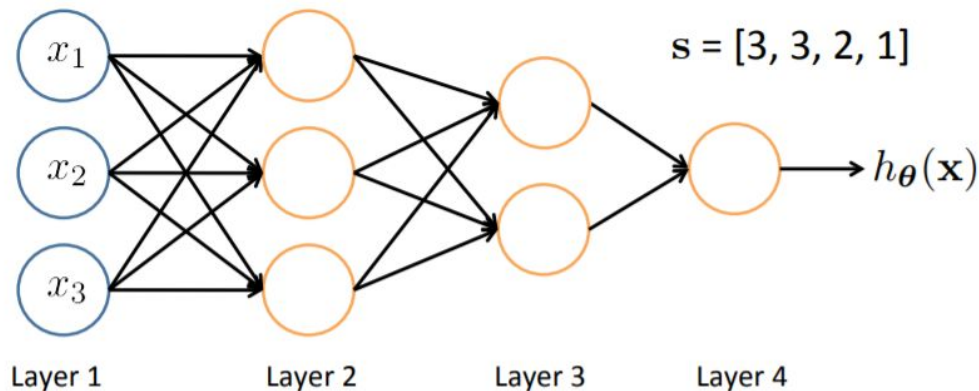
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

Add  $a_0^{(2)} = 1$

$$\mathbf{z}^{(3)} = \Theta^{(2)} \mathbf{a}^{(2)}$$

$$h_{\Theta}(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

# Another structure

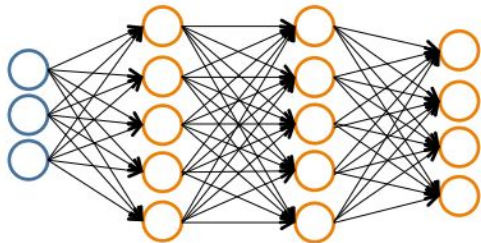


$L$  denotes the number of layers

$\mathbf{s} \in \mathbb{N}^{+L}$  contains the numbers of nodes at each layer

- Not counting bias units
- Typically,  $s_0 = d$  (# input features) and  $s_{L-1} = K$  (# classes)

# Binary vs Multi-class



## Binary classification

$y = 0$  or  $1$

1 output unit ( $s_{L-1} = 1$ )

## **Given:**

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$  contains # nodes at each layer

–  $s_0 = d$  (# features)

## Multi-class classification ( $K$ classes)

$\mathbf{y} \in \mathbb{R}^K$  e.g.  $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ ,  $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ ,  $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ ,  $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$   
pedestrian car motorcycle truck

$K$  output units ( $s_{L-1} = K$ )

# Links

<https://nnplayground.com/>

[Tensorflow playground](#)

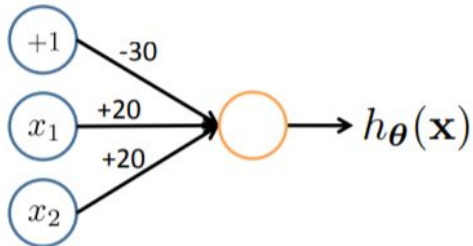


# Boolean representation

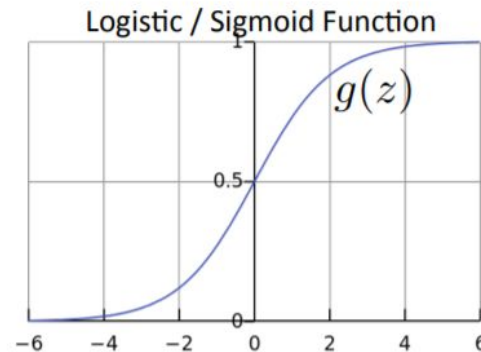
## Simple example: AND

$$x_1, x_2 \in \{0, 1\}$$

$$y = x_1 \text{ AND } x_2$$

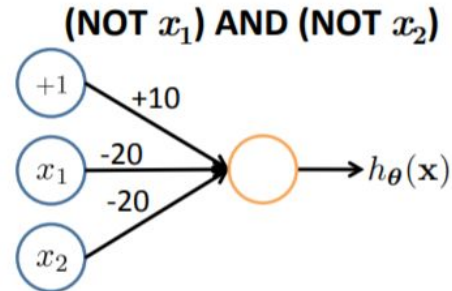
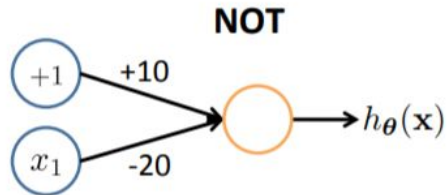
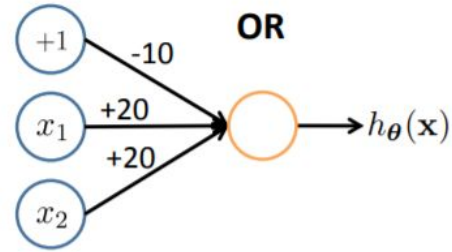
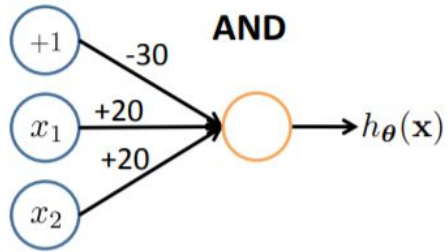


$$h_{\theta}(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

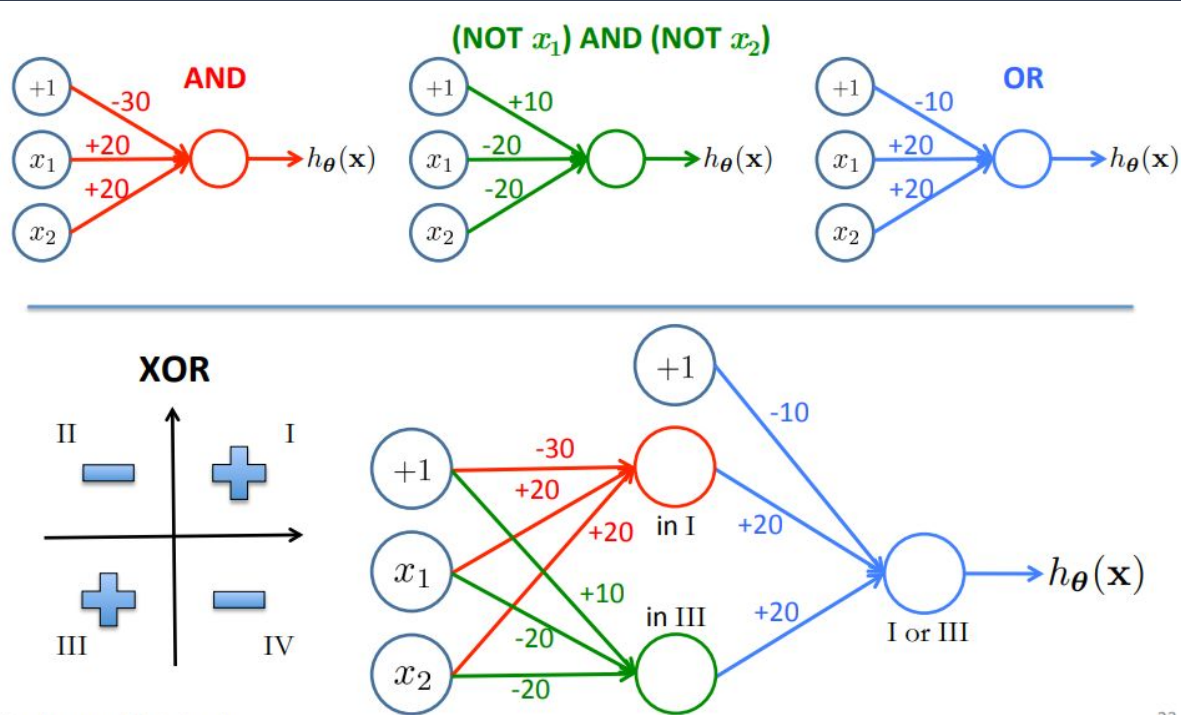


$x_1$	$x_2$	$h_{\theta}(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

# Boolean representation



# Boolean representation



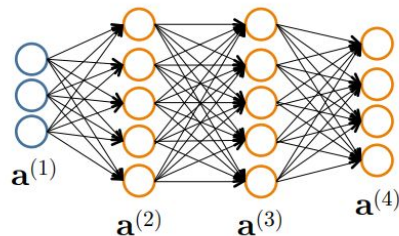
# Forward propagation

## Forward Propagation

- Given one labeled training instance  $(\mathbf{x}, y)$ :

### Forward Propagation

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$  [add  $a_0^{(2)}$ ]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$  [add  $a_0^{(3)}$ ]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_{\Theta}(\mathbf{x}) = g(\mathbf{z}^{(4)})$



# Backpropagation

## Backpropagation Intuition

- Each hidden node  $j$  is “responsible” for some fraction of the error  $\delta_j^{(l)}$  in each of the output nodes to which it connects
- $\delta_j^{(l)}$  is divided according to the strength of the connection between hidden node and the output node
- Then, the “blame” is propagated back to provide the error values for the hidden layer

# Stochastic Gradient Descent

1. Draw a batch of samples.
  2. Compute the gradient for those samples
  3. Move parameters a little in the opposite direction of the gradient.
- In practice you'll use Stochastic gradient descent in highly dimensional spaces.
  - Reduces Overfitting

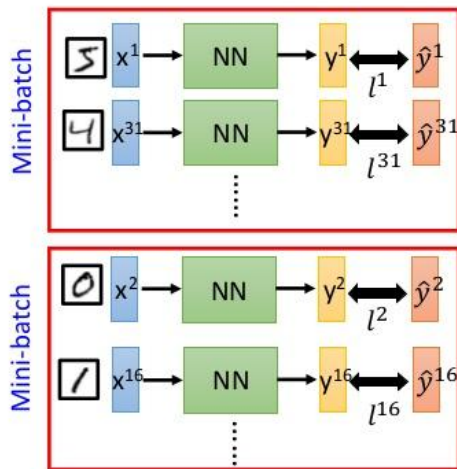
# Batches & Epochs

- one epoch = one forward pass and one backward pass of all the training examples
- batch size = the number of samples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of iterations = number of passes, each pass using [batch] (one forward pass + one backward pass)

# Batches & Epochs

We do not really minimize total loss!

## Mini-batch



- Randomly initialize network parameters

- Pick the 1<sup>st</sup> batch  
 $L' = l^1 + l^{31} + \dots$   
Update parameters once

- Pick the 2<sup>nd</sup> batch  
 $L'' = l^2 + l^{16} + \dots$   
Update parameters once

⋮

- Until all mini-batches have been picked

one epoch

Repeat the above process



# Momentum

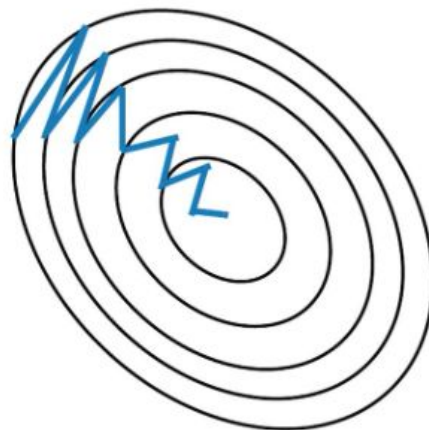
Momentum  $\alpha$  is used to diminish the fluctuations in weight changes over consecutive iterations:

$$\Delta\omega_i(t+1) = -\eta \frac{\partial E}{\partial w_i} + \alpha \Delta\omega_i(t),$$

where  $E(\mathbf{w})$  is the error function,  $\mathbf{w}$  - the vector of weights,  $\eta$  - learning rate.



Stochastic Gradient  
Descent **without**  
Momentum



Stochastic Gradient  
Descent **with**  
Momentum

# RMSprop

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction.

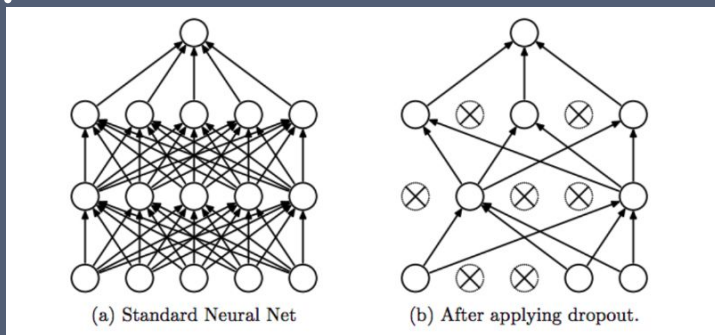
The difference between RMSprop and gradient descent is on how the gradients are calculated

# Regularizations

- L1 regularization —  $|w|$
- L2 regularization —  $w^2$

# Dropout

Individual nodes are either dropped out of the net with probability  $1-p$  or kept with probability  $p$ , so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.



# Why Dropout?

A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data.

Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.