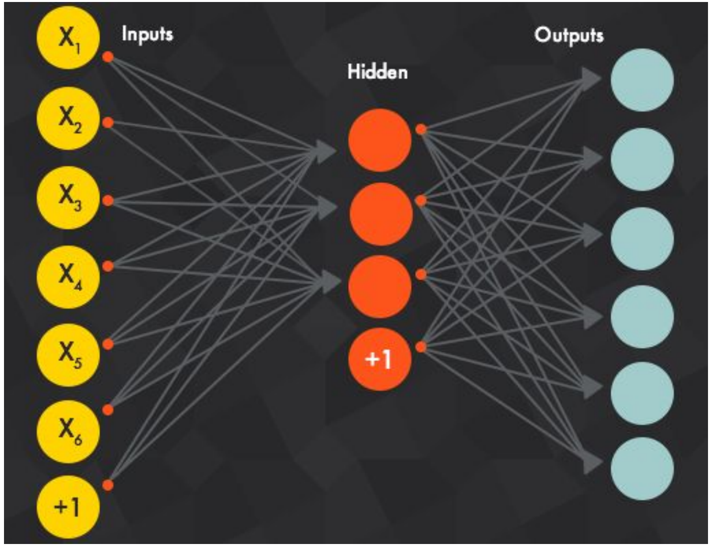
Deep Learning KEP

ISTE

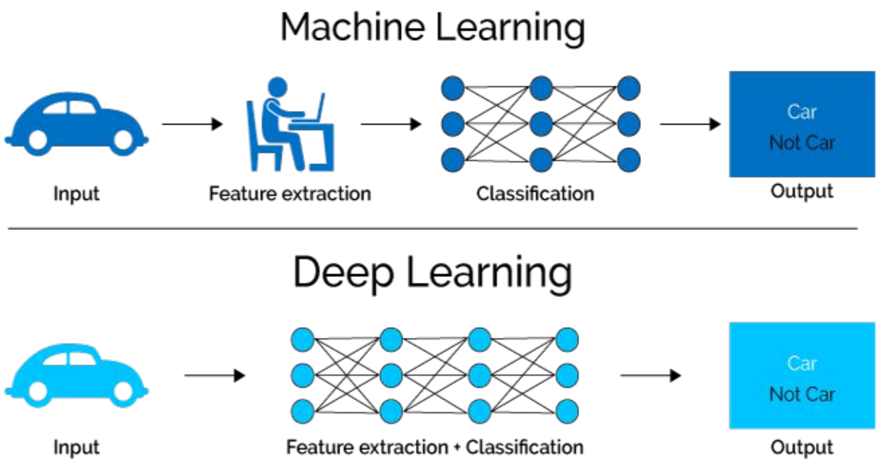
Date: 09 March 2020



Why Deep Learning?

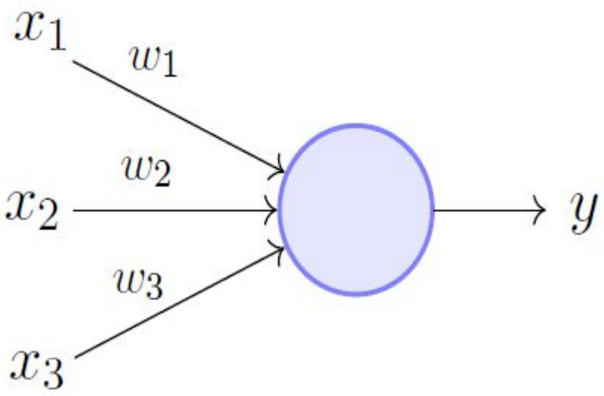
1. Computer Vision
2. NLP
3. Generative Networks

Deep Learning vs Machine Learning

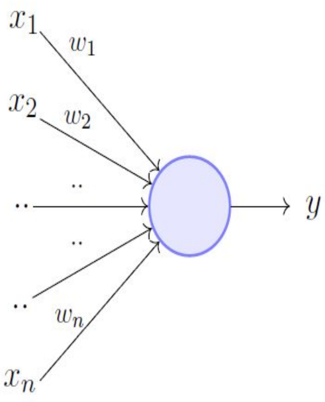


Introduction to Deep Learning

Perceptron



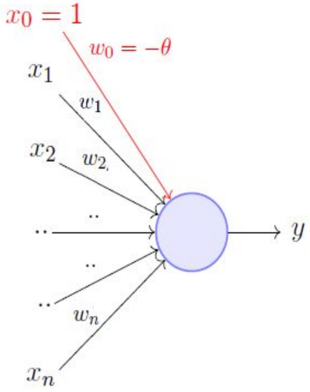
Perceptron Model (Minsky-Papert in 1969)



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

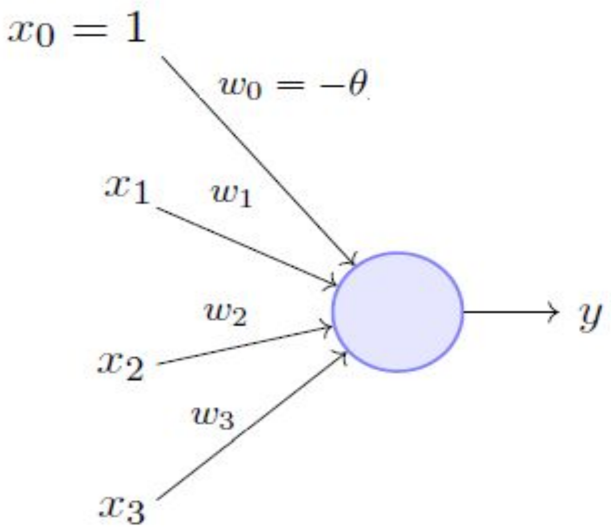
$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

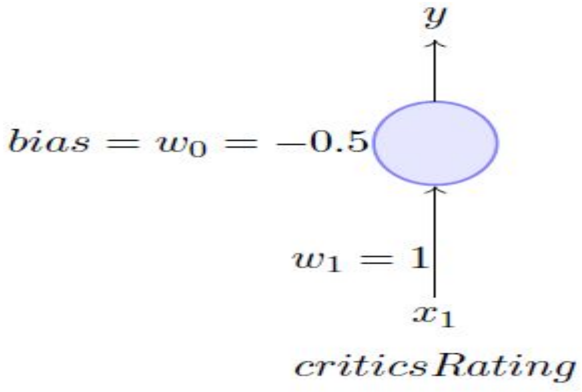
$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$



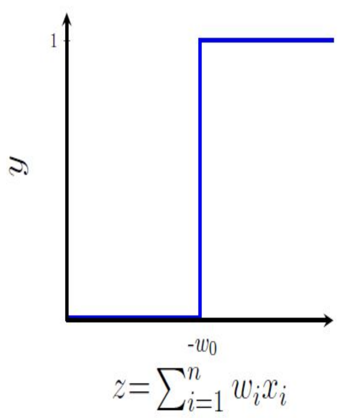
$$x_1 = isPremierLeagueOn$$
$$x_2 = isManUnitedPlaying$$
$$x_3 = isFriendlyGame$$

Artificial Neurons

if you look at a problem of deciding if I will be watching a movie or not, based only on one real-valued input (***x_1*** = *criticsRating*) and if the threshold we set is 0.5 (***w_0*** = -0.5) and ***w_1***= 1 then our setup would look like this:



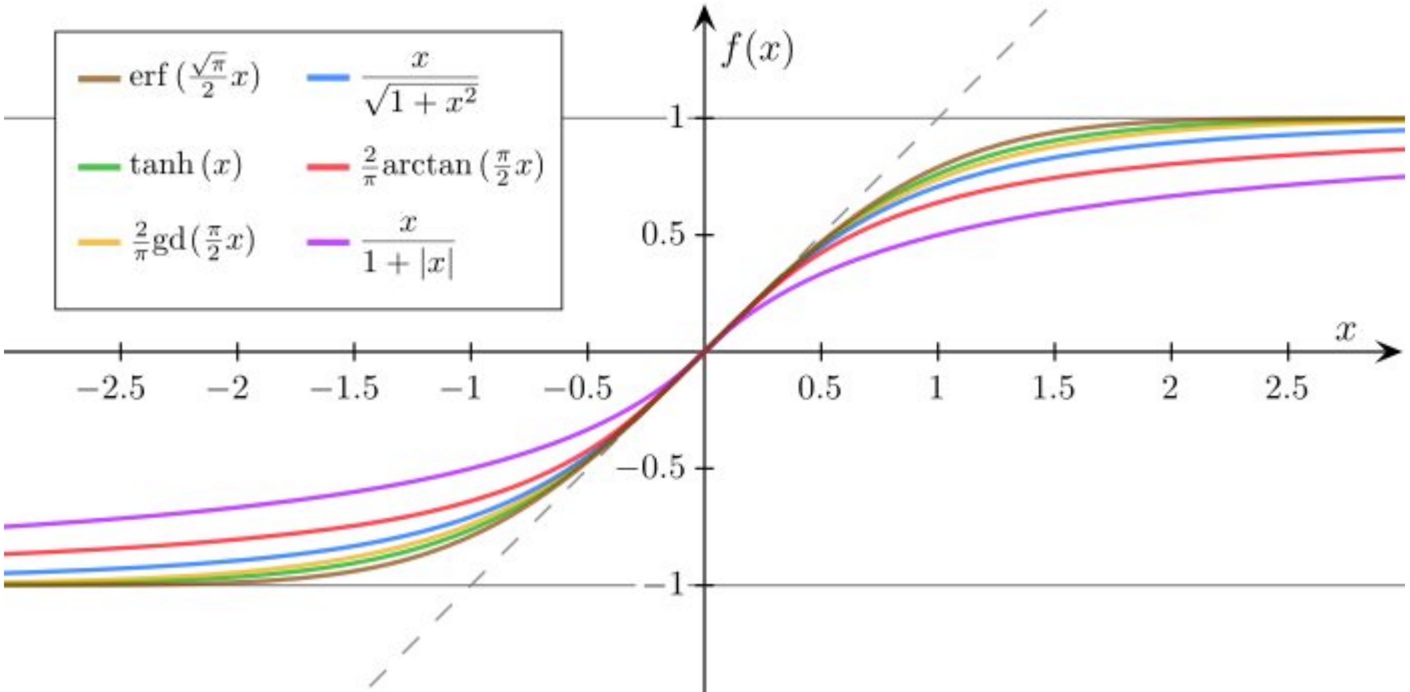
$$
\begin{aligned}
y &= 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0 \\
&= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0
\end{aligned}
$$



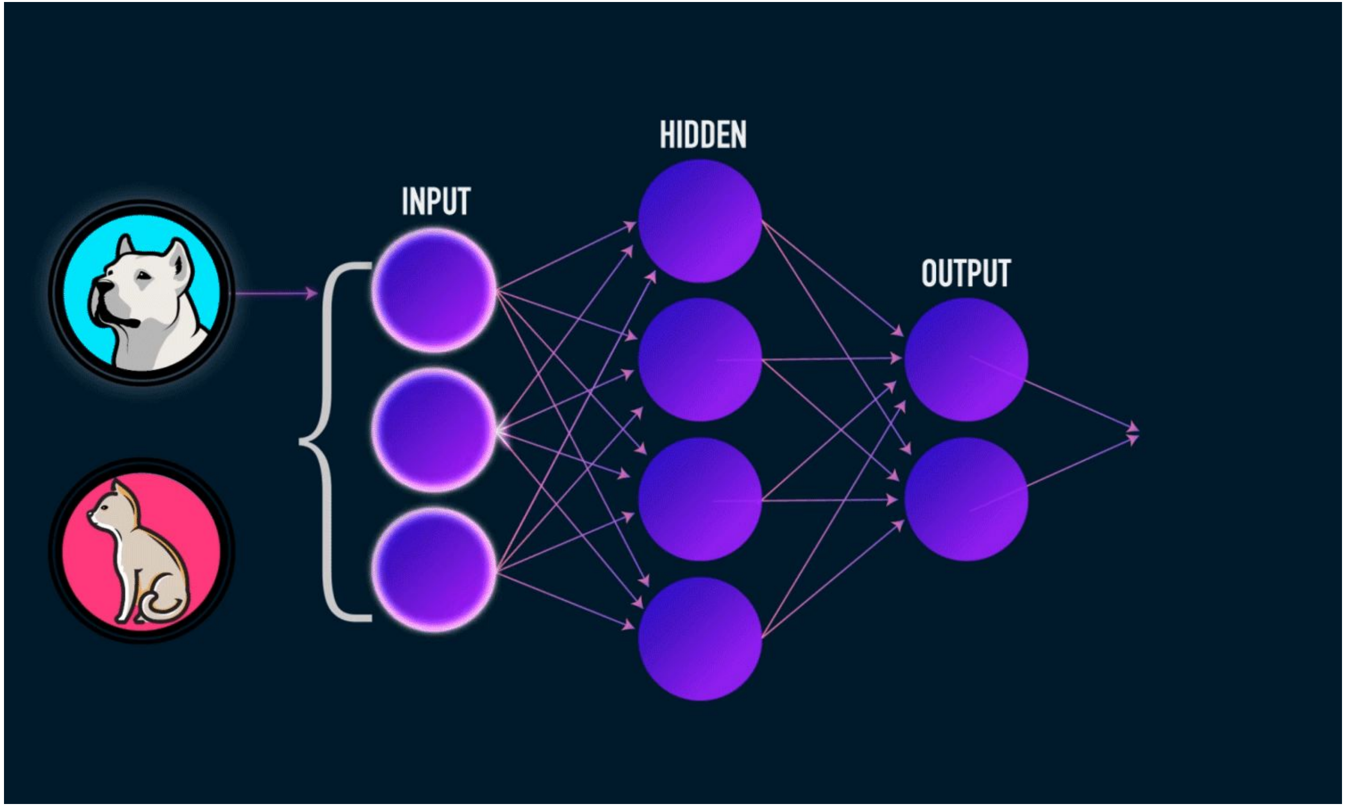
What would be the decision for a movie with *criticsRating* = 0.51? *Yes!*

What would be the decision for a movie with *criticsRating* = 0.49? *No!*

Motivation For Sigmoid Neurons



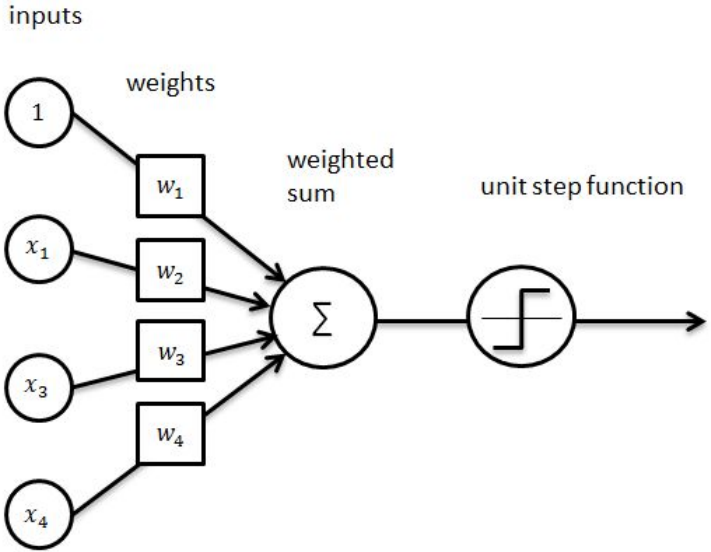
Neural Networks

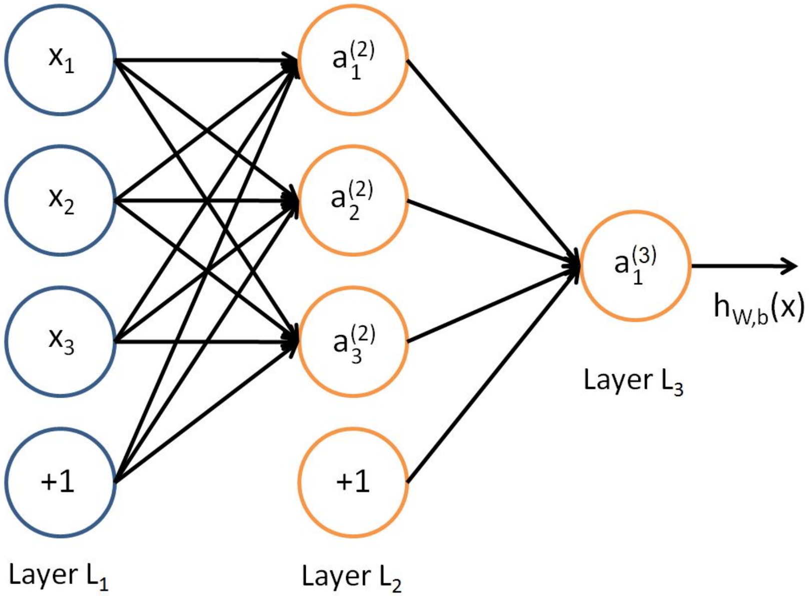


Model Representation

The Neural Network is constructed from 3 type of layers:

1. Input layer — initial data for the neural network.
2. Hidden layers — intermediate layer between input and output layer and place where all the computation is done.
3. Output layer — produce the result for given inputs.


inputs
weights
1
x_1
x_3
x_4
w_1
w_2
w_3
w_4
weighted sum
Σ
unit step function


x_1
x_2
x_3
+1
$a_1^{(2)}$
$a_2^{(2)}$
$a_3^{(2)}$
+1
$a_1^{(3)}$
$h_{W,b}(x)$
Layer L_1
Layer L_2
Layer L_3

We are going to mark the “bias” nodes as x_0 and a_0 respectively. So, the input nodes can be placed in one vector *X* and the nodes from the hidden layer in vector *A*.

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad A = \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

The weights (arrows) are usually noted as *θ* or *W.* In this case I will note them as *θ.* The weights between the input and hidden layer will represent *3x4* matrix. And the weights between the hidden layer and the output layer will represent *1x4* matrix.

$$\theta^{(1)} = \begin{bmatrix} \theta_{10} & \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{20} & \theta_{21} & \theta_{22} & \theta_{23} \\ \theta_{30} & \theta_{31} & \theta_{32} & \theta_{33} \end{bmatrix}$$

$$
\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)
\end{aligned}
$$

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

$$
\mathrm{a}_n^L = \left[\sigma\left(\sum\nolimits_m \theta_{nm}^L \left[\cdots\left[\sigma\left(\sum\nolimits_j \theta_{kj}^2 \left[\sigma\left(\sum\nolimits_i \theta_{ji}^1 x_i + b_j^1\right)\right] + b_k^2\right)\right]\cdots\right]_m + b_n^L\right)\right]_n
$$

Forward Propagation

This process of Forward propagation is actually getting the Neural Network output value based on a given input. This algorithm is used to calculate the cost value.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

Back Propogation

What we want to do is minimize the cost function ***J(θ)*** using the optimal set of values for ***θ*** (weights). Backpropagation is a method we use in order to **compute the partial derivative of *J(θ)***.

This partial derivative value is then used in Gradient descent algorithm (*"Image 23"*) for calculating the ***θ*** values for the Neural Network that minimize the cost function ***J(θ)***.

$$
\begin{aligned}
&Repeat\ \{\\
&\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\\
&\}
\end{aligned}
$$

Backpropagation algorithm has 5 steps:

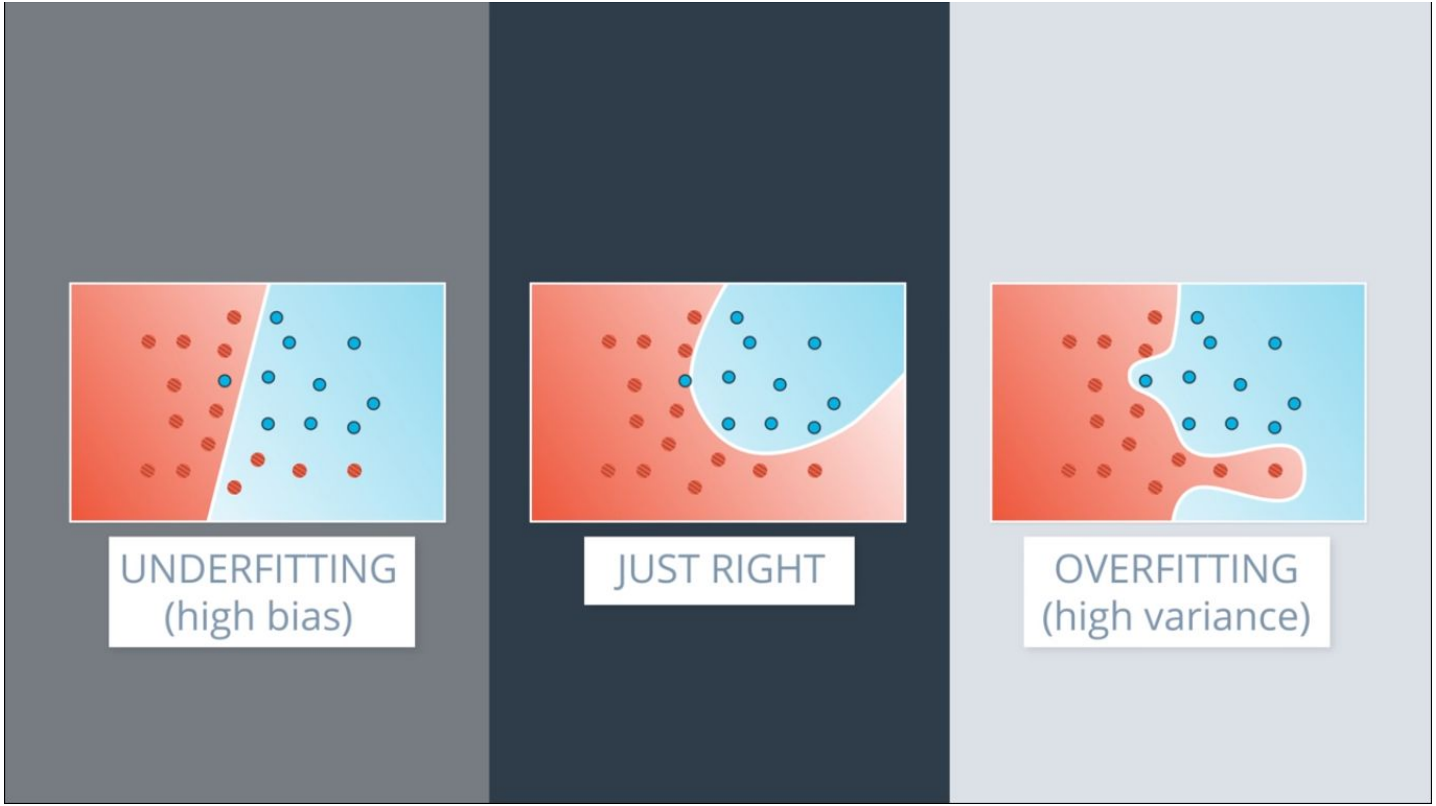
1. Set ***a(1) = X;*** for the training examples
2. Perform forward propagation and compute ***a(l)*** for the other layers ***(l = 2…L)***
3. Use ***y*** and compute the delta value for the last layer ***δ(L) = h(x) — y***
4. Compute the ***δ(l)*** values backwards for each layer (described in “Math behind Backpropagation” section)
5. Calculate derivative values ***Δ(l) = (a(l))^T ∘ δ(l+1)*** for each layer, which represent the derivative of cost ***J(θ)*** with respect to ***θ(l)*** for layer ***l***

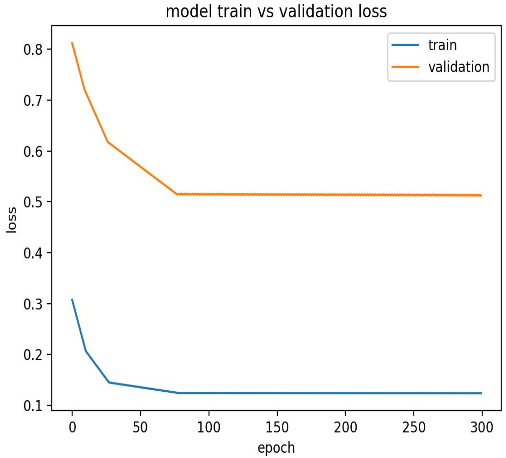
Backpropagation is about determining how changing the weights impact the overall cost in the neural network.

Why derivatives ?

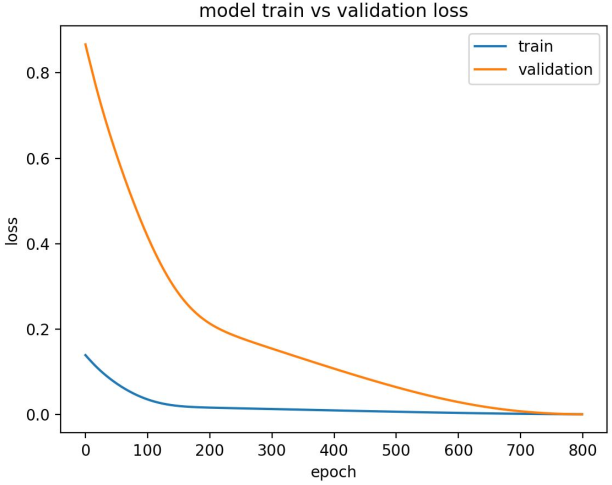
The derivative of a function (in our case ***J(θ)***) on each variable (in our case weight ***θ***) tells us the **sensitivity of the function with respect to that variable** or **how changing the variable impacts the function value**.

Overfitting and Underfitting

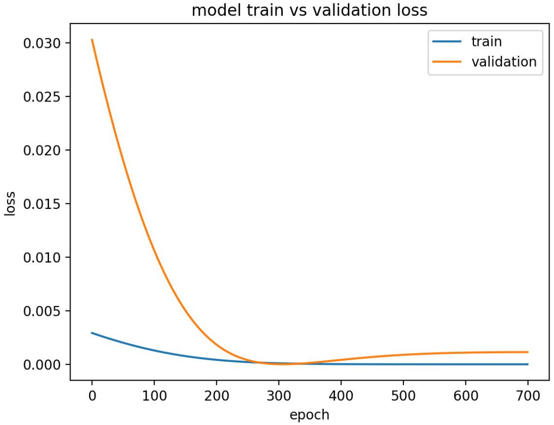




Underfit



Goodfit



Overfit