Learning With Neural Networks

By Nithish Divakar
Video Analytics Lab
Dept. Computational and Data Sciences
IISc Bangalore

What is the following number ?

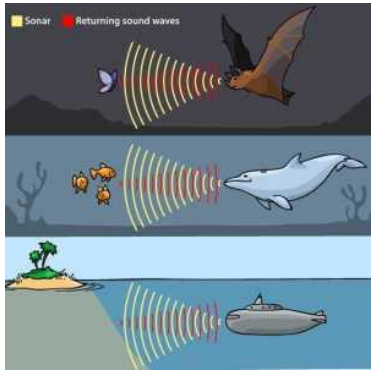
Inspiration

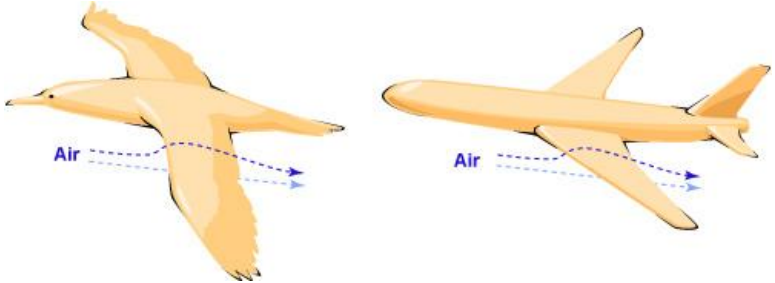
What is the following number ?

Why not copy the best 'natural' learning system ...

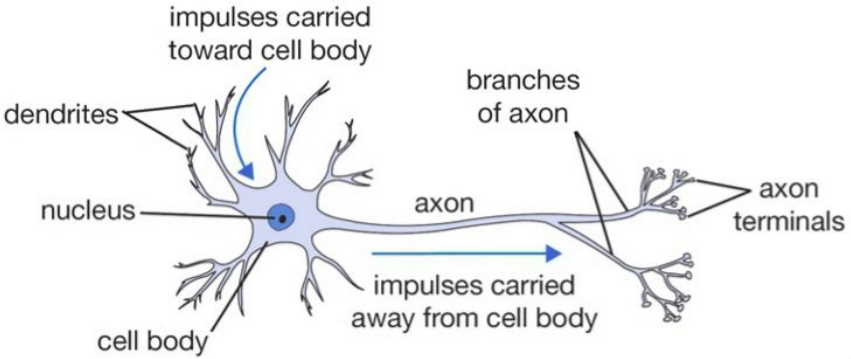
... to create the best 'artificial' learning system ?

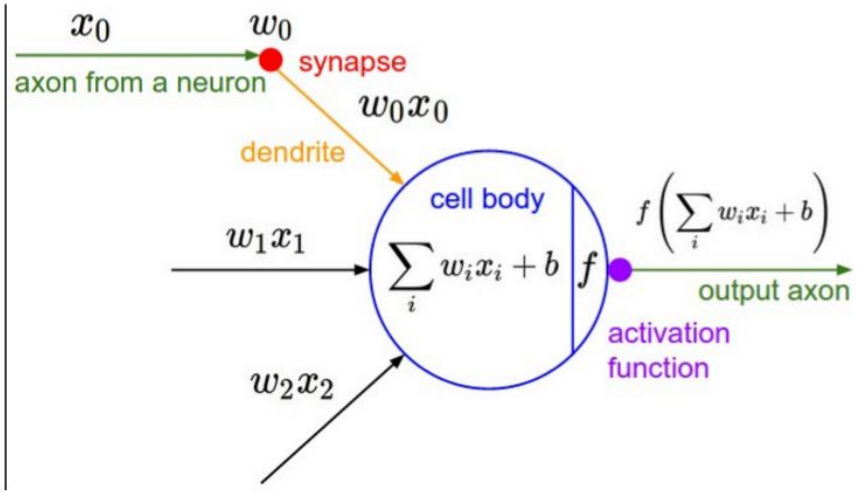
Copying from Nature -- Nothing New, Really !!!





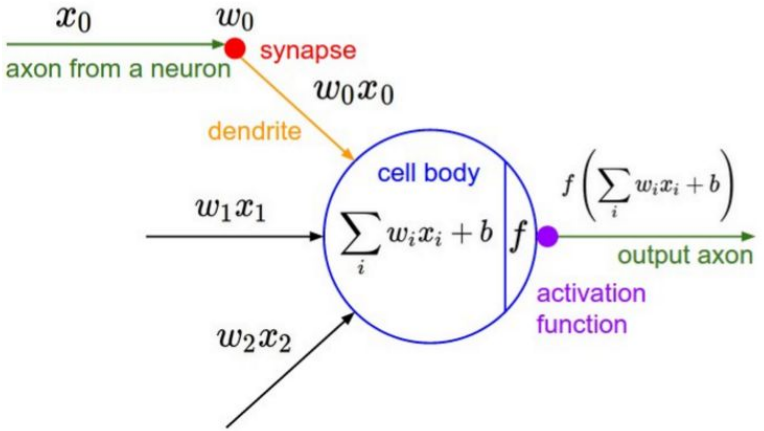
Basic Building block :: A Neuron





Mapping to Our Problem

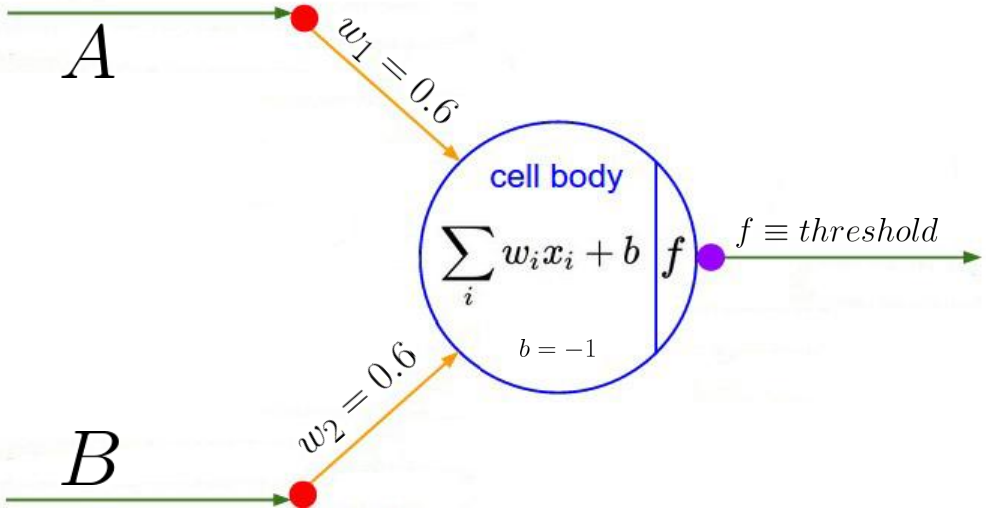
$$y = f(\mathbf{x};\mathbf{w})$$



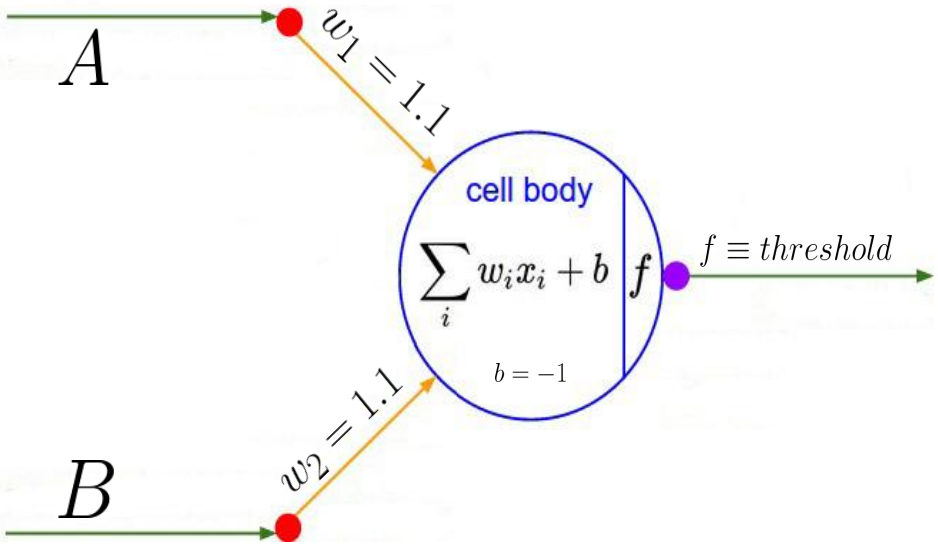
- **x**,y : known
- **w** : weights (need to be “learnt”)


PENGUINS ARE BLACK AND WHITE.
SOME OLD TV SHOWS ARE BLACK AND WHITE.
THEREFORE, SOME PENGUINS ARE OLD TV SHOWS.
GLASBERGEN
LOGIC
One thing penguins aren't good at

Logical Functions ...First NN Implementations (1943)



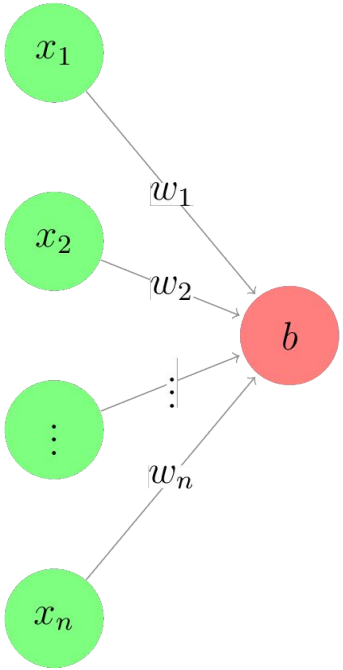
A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

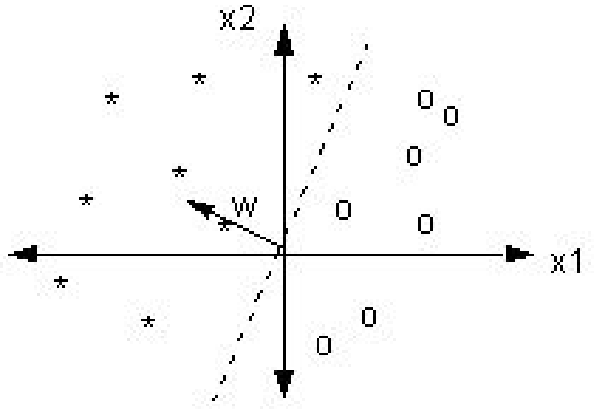


A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE ?

RosenBlatt's "perceptron" (1953)



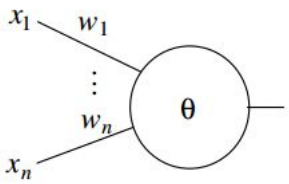


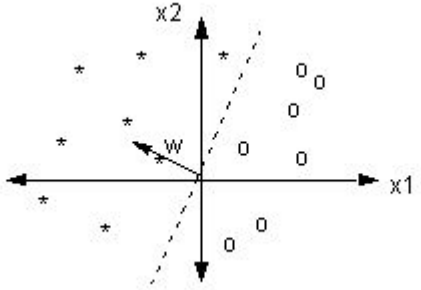
Linearly Separable Classes

$$output = \begin{cases} 0 & \sum_i w_i x_i + b \leq 0 \\ 1 & \sum_i w_i x_i + b > 0 \end{cases}$$

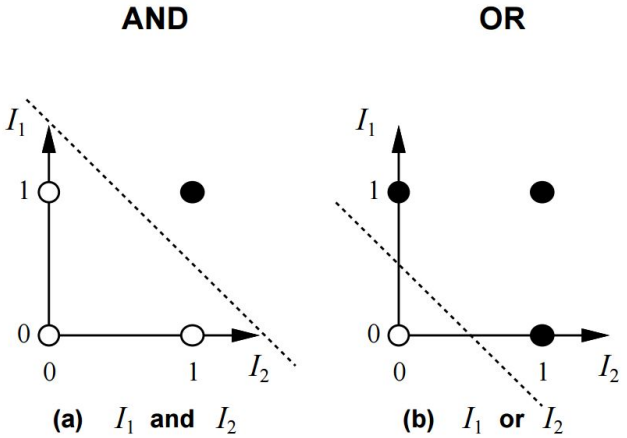
Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.

RosenBlatt's "perceptron" (1953)





Linearly Separable Classes

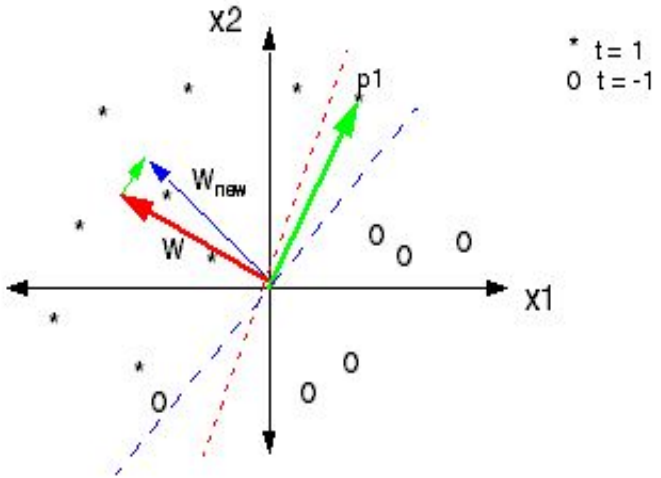


Perceptron Algorithm

```
Initialize weights w ; // Random or 0

Until [all examples correctly classified]
    For each training sample (x,y)
        Compute yt := x^T w
         if y == yt // Correctly classified
            continue ;
        else // Update weights
            dw = (y - yt) * x ;
            w   = w + dw;
        EndIf
    EndFor
EndUntil
```

Perceptron Algorithm

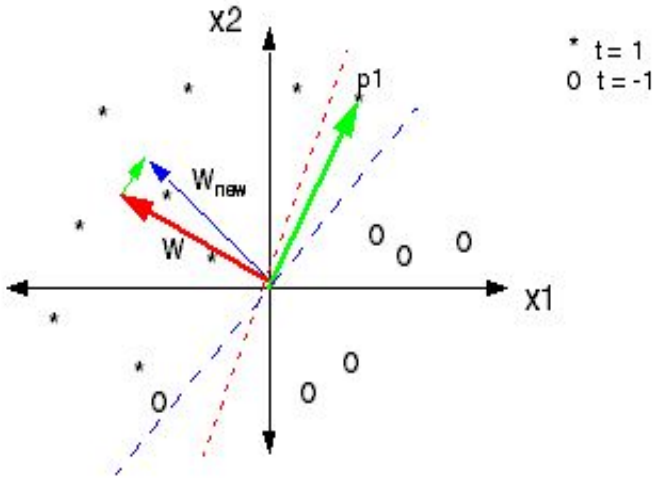
```
Initialize weights w ; // Random or 0

Until [all examples correctly classified]
     For each training sample (x,y)
          Compute yt := x^T w
           if y == yt // Correctly classified
               continue ;
          else // Update weights
               dw = (y - yt) * x ;
               w   = w + dw;
          EndIf
     EndFor
EndUntil
```



Perceptron Algorithm

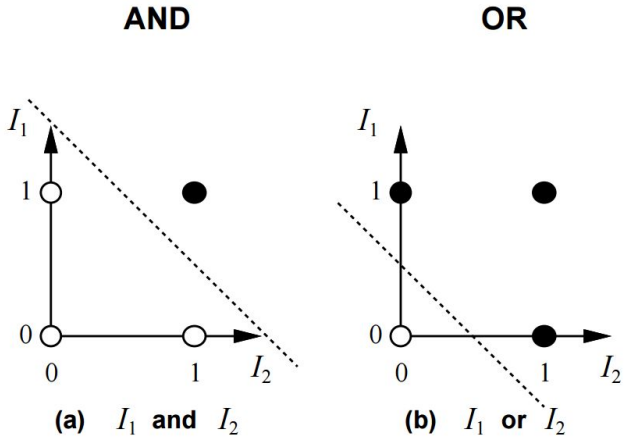
Demo ...

visualize_PLA.html

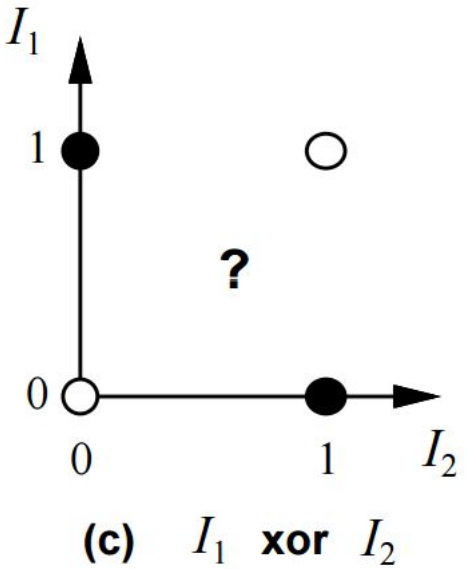


https://github.com/Xelio/Machine-Learning

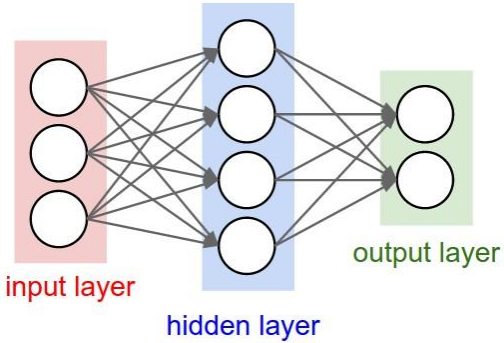
Can a Perceptron Separate Distributions Which are Not Linearly Separable ???

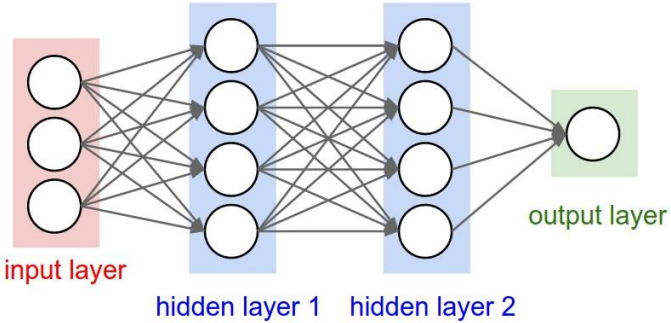


Linearly Separable Classes

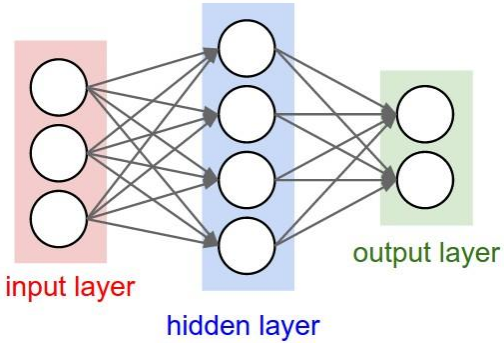


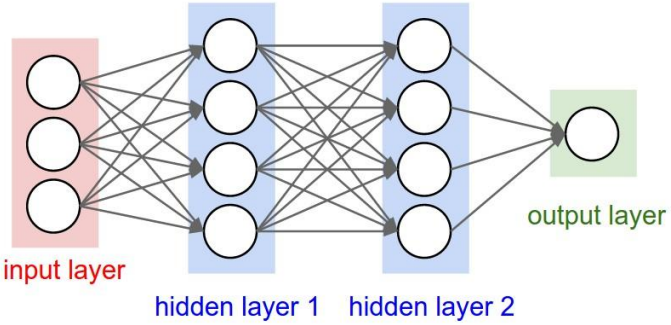
Why Use Only One Neuron ?

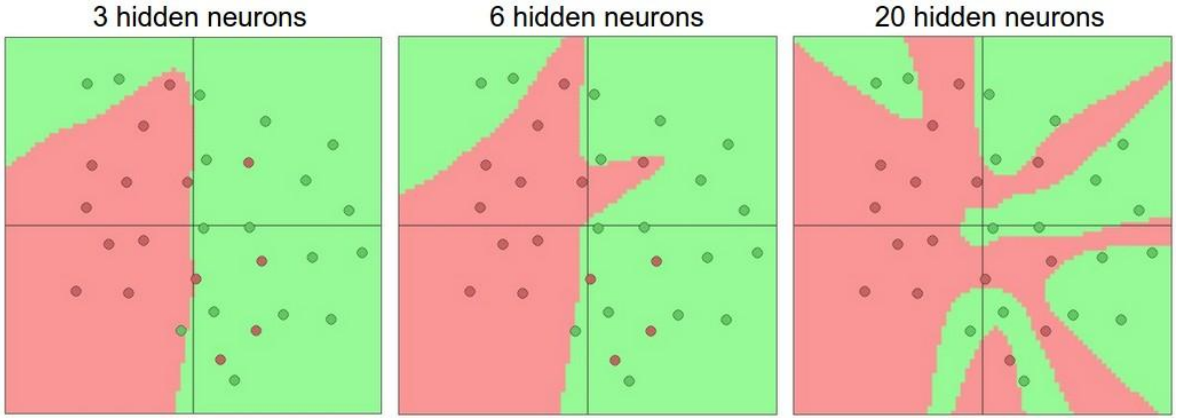




Why Use Only One Neuron ?

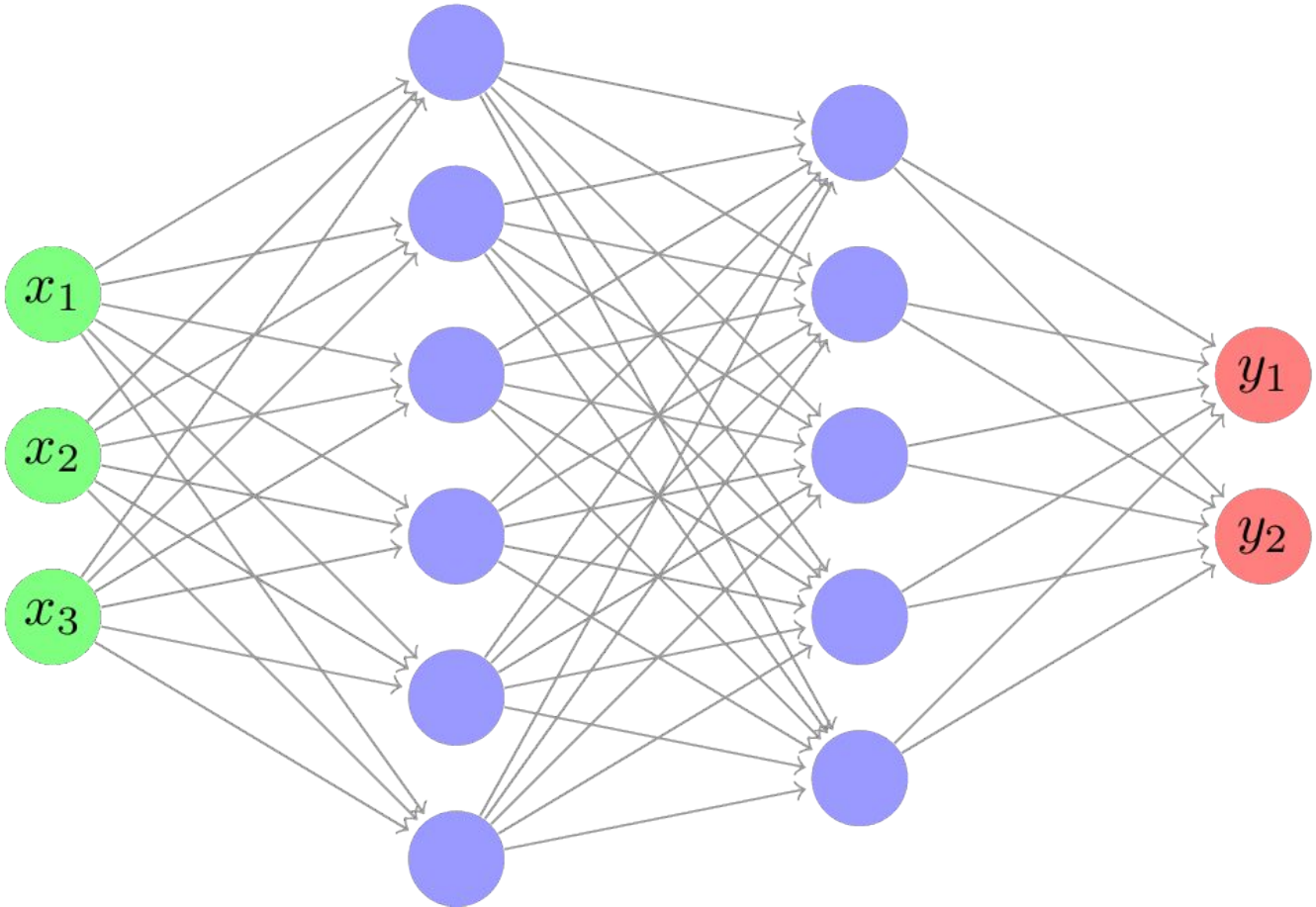




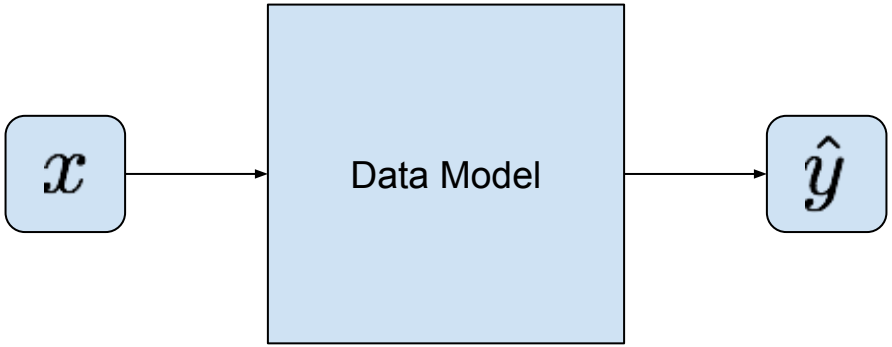


Neural Networks

Input layer Hidden layer 1 Hidden layer 2 Output layer



Data model



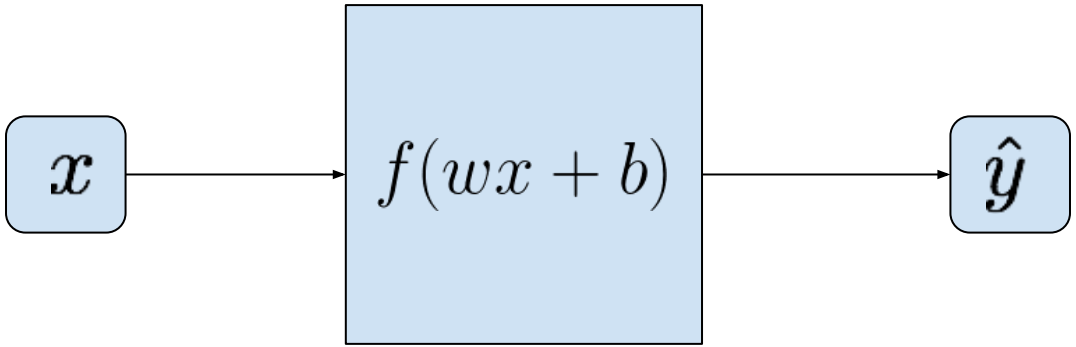
$y \equiv \hat{y}$

Good model

$y \not\equiv \hat{y}$

Bad model

A Simple data model

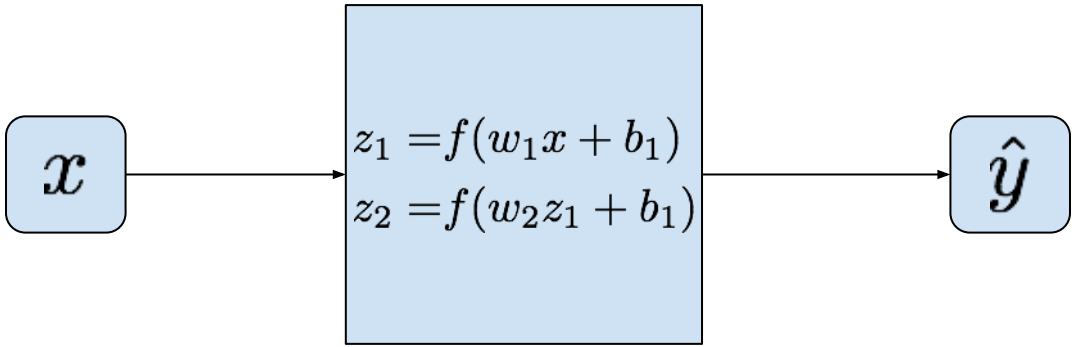


$\{w, b\}$ parameters of the model

DEMO:

fitting_point.html

A ~~Simple~~ Complicated data model



$\{w_1, w_2, b_1, b_2\}$ parameters of the model

DEMO:

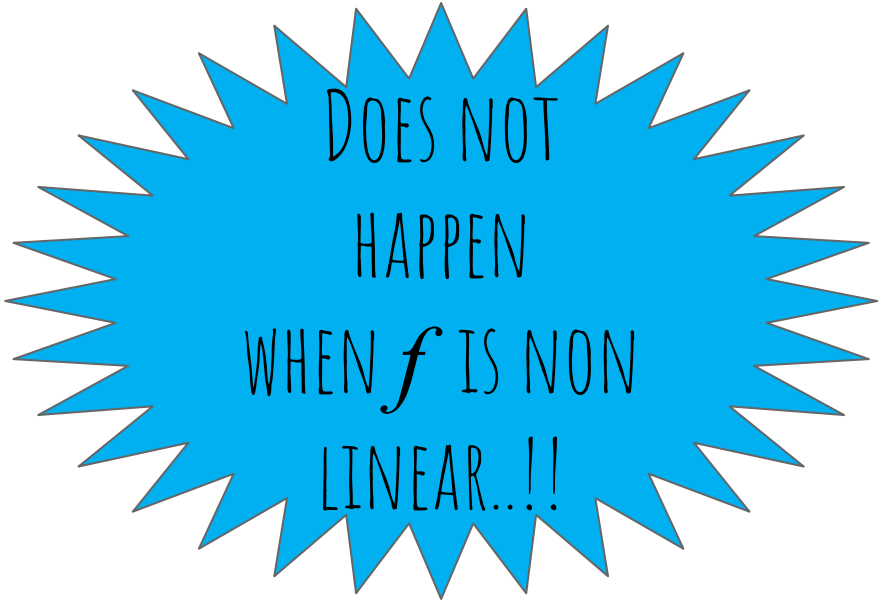
fitting_point2.html

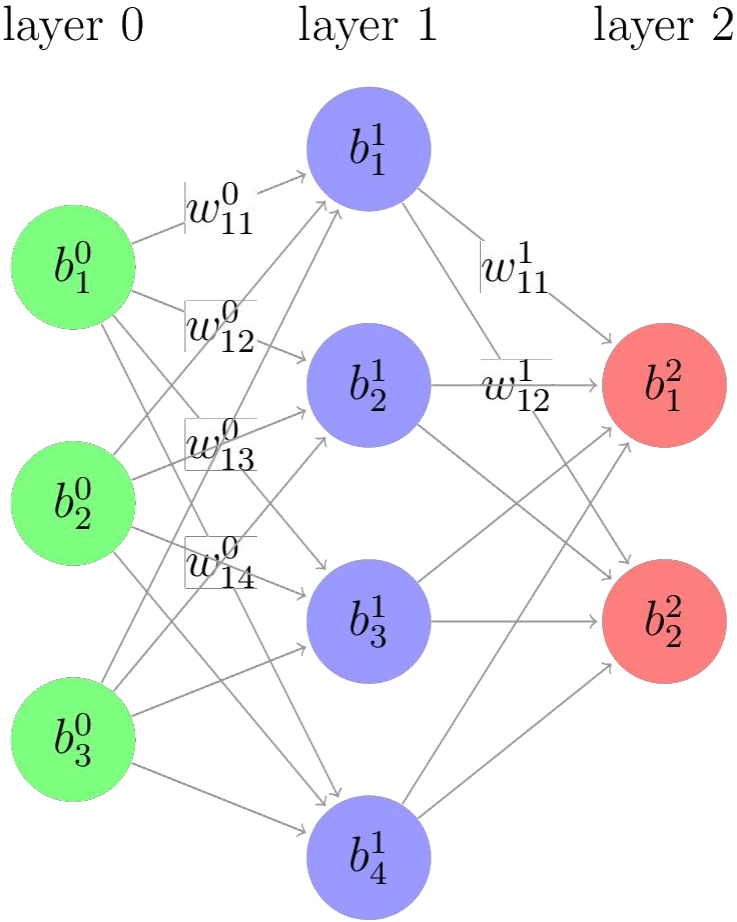
Why do we need no linearity ?

$$z_1 = f(z_0)$$
$$z_2 = f(z_1)$$

But if f is linear

$$y = f(x) = ax$$
$$\implies z_2 = f(z_1)$$
$$= f(f(z_0))$$
$$z_2 = f(az_0)..!!$$





Things of the network

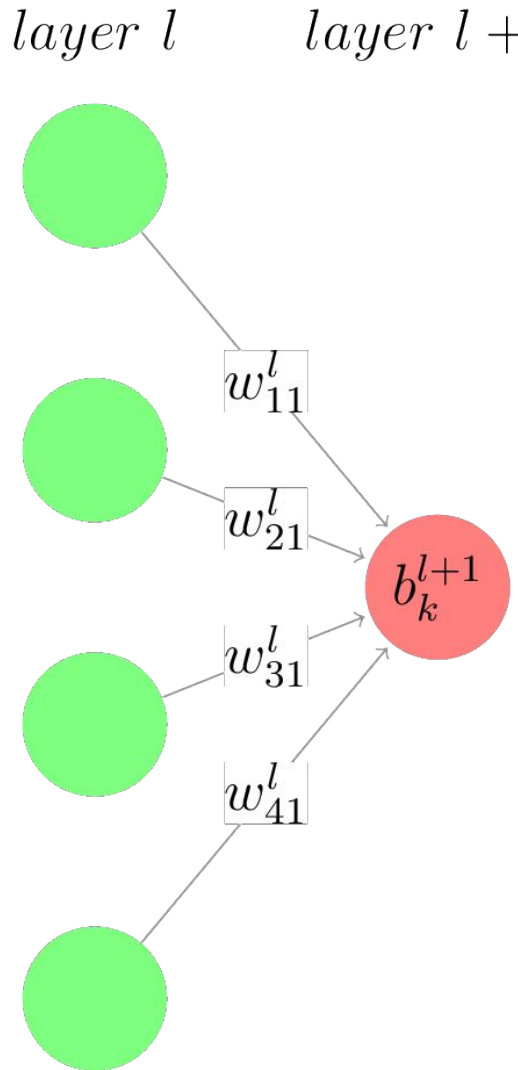
$w_{ji}^l \mapsto$ weight btw $node_j^l$ and $node_i^{l+1}$

$b_{lj} \mapsto$ bias of $node_j^l$

Output of a single neuron

$$z_k^{l+1} = f\left(\sum_i w_{ji}^l z_i^l + b_i^{l+1}\right)$$

f is the non linearity
In perceptron it was threshold function



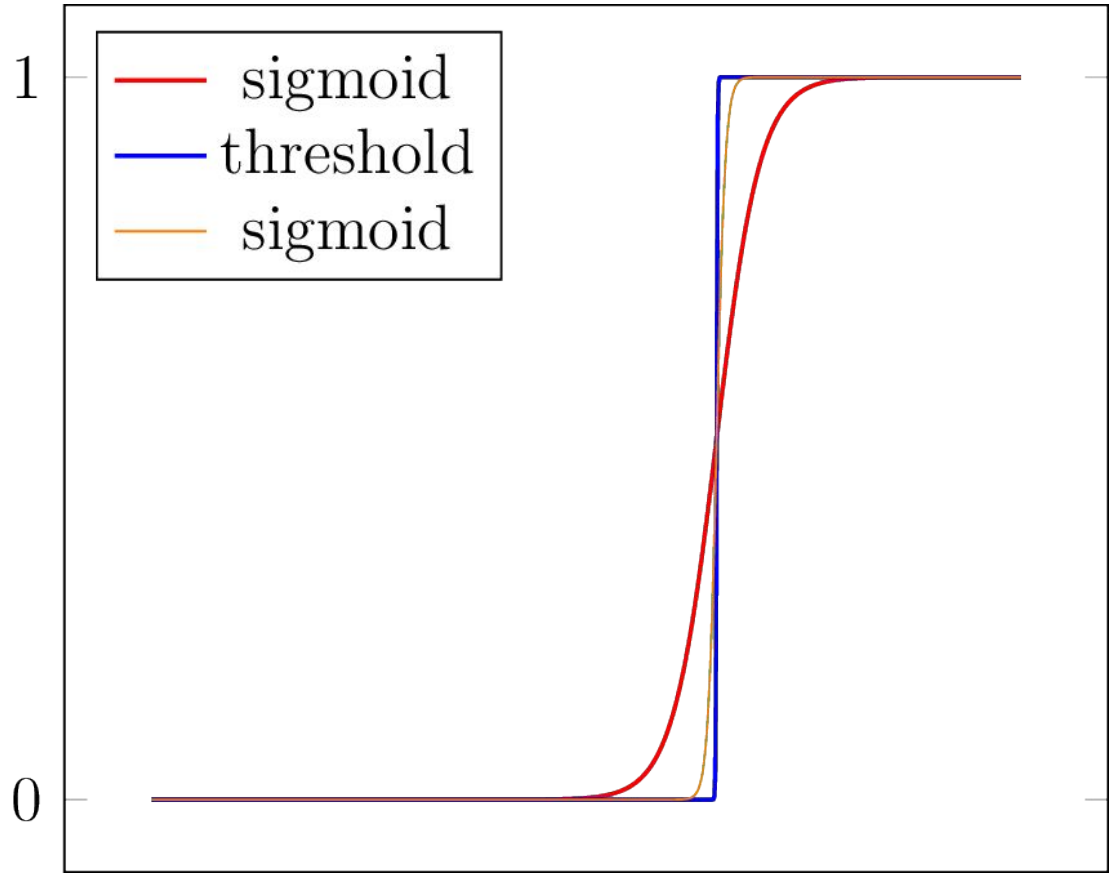
Non Linearity

$$z_k^{l+1} = f\left(\sum_i w_{ji}^l z_i^l + b_i^{l+1}\right)$$

$$w \leftarrow w + \nabla w$$

$$z = \ ?$$

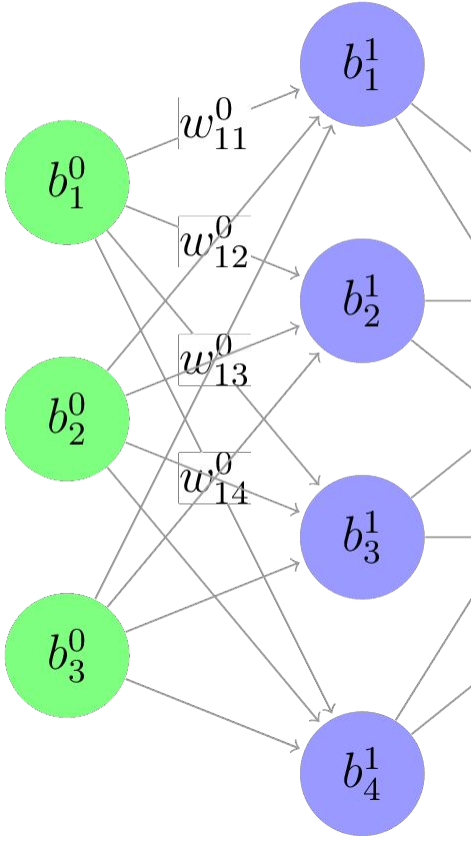
$$simoid(z) = \frac{1}{1+e^{-z}}$$



Output of a layer

$$z_k^{l+1} = f\left(\sum_i w_{ji}^l z_i^l + b_i^{l+1}\right)$$

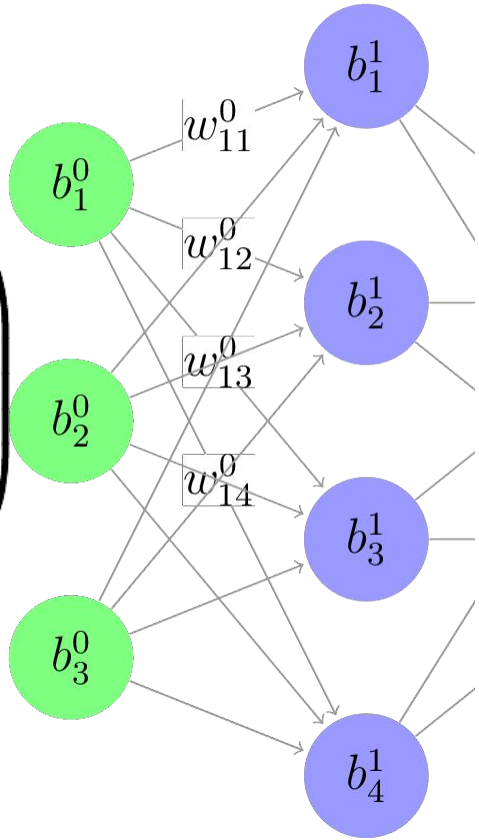
$$\begin{aligned}
z_1^{l+1} &\quad \sum_i w_{1i}^l z_i^l + b_1^{l+1} \\
z_2^{l+1} &\quad \sum_i w_{2i}^l z_i^l + b_2^{l+1} \\
z_3^{l+1} &\quad \sum_i w_{3i}^l z_i^l + b_3^{l+1} \\
\vdots &\quad \vdots \\
z_n^{l+1} &\quad \sum_i w_{3i}^l z_i^l + b_n^{l+1}
\end{aligned}$$



Output of a layer

$$z_k^{l+1} = f\left(\sum_i w_{ji}^l z_i^l + b_i^{l+1}\right)$$

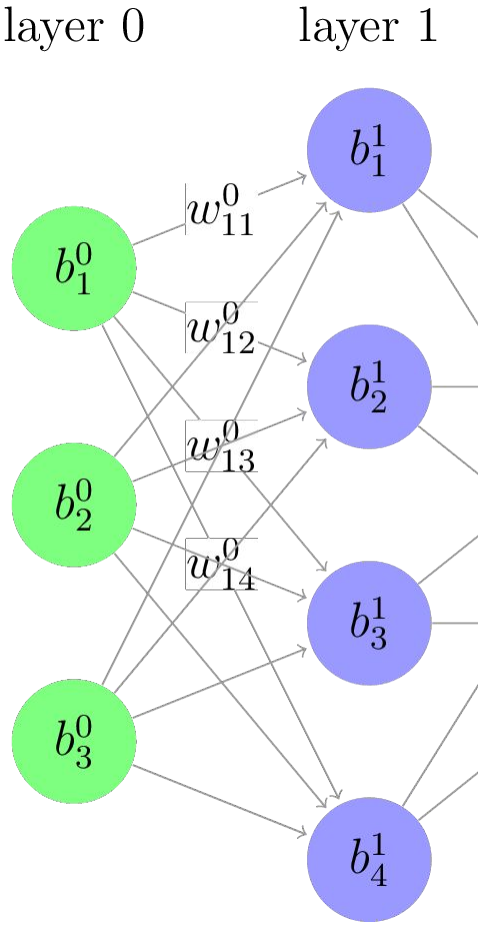
$$\begin{bmatrix} z_1^{l+1} \\ z_2^{l+1} \\ \vdots \\ z_m^{l+1} \end{bmatrix} = f\left(\begin{bmatrix} w_{11}z_1 + w_{12}z_2 \dots w_{1m}z_m \\ w_{21}z_1 + w_{22}z_2 \dots w_{2m}z_m \\ \vdots \\ w_{n1}z_1 + w_{n2}z_2 \dots w_{nm}z_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right)$$



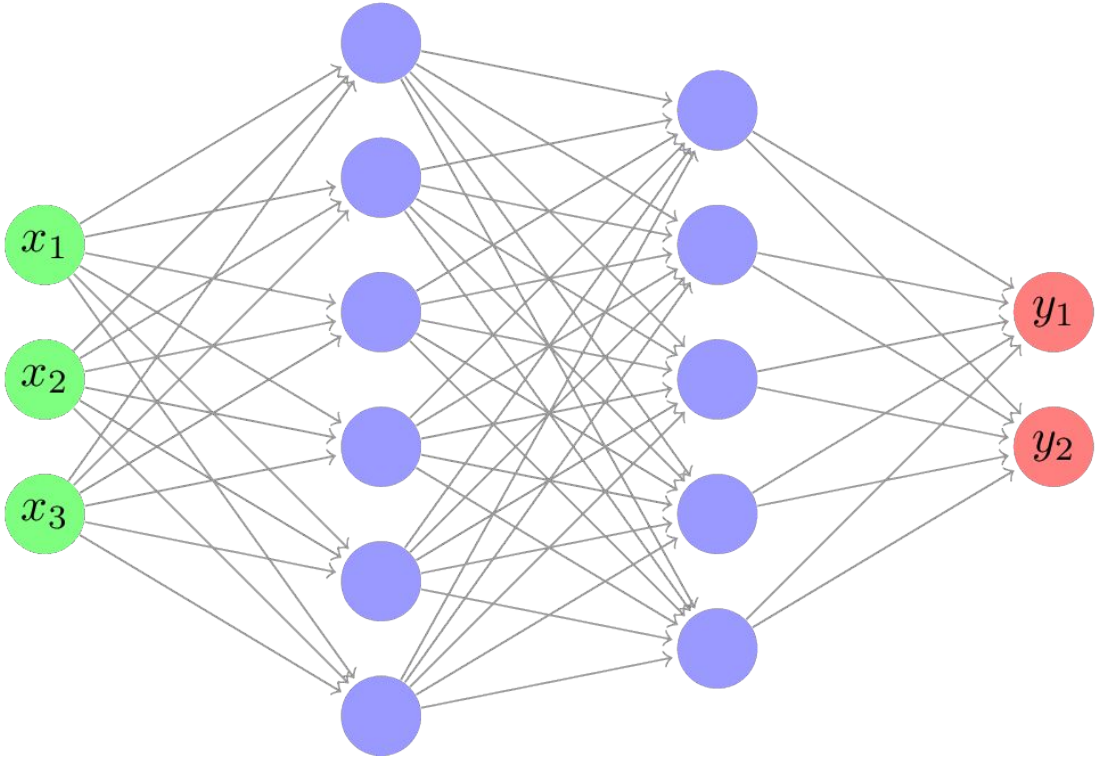
Output of a layer

$$z_k^{l+1} = f\left(\sum_i w_{ji}^l z_i^l + b_i^{l+1}\right)$$

$$z^{l+1} = f(W^l z^l + b^{l+1})$$



Neural network

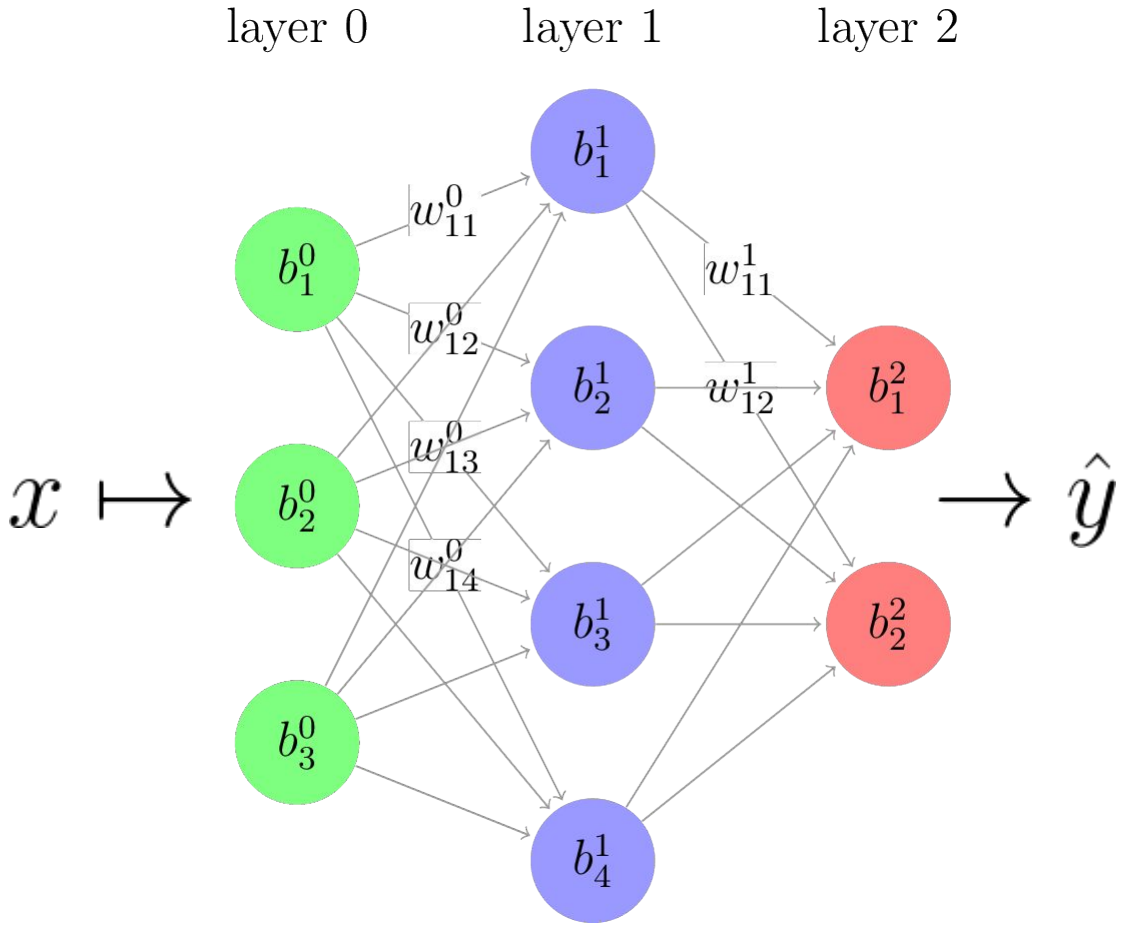


$$\hat{y} = f(f(f(f(f(W^1 x + b^2)))))$$

Learning things!!

Just another name for finding the right parameters

?

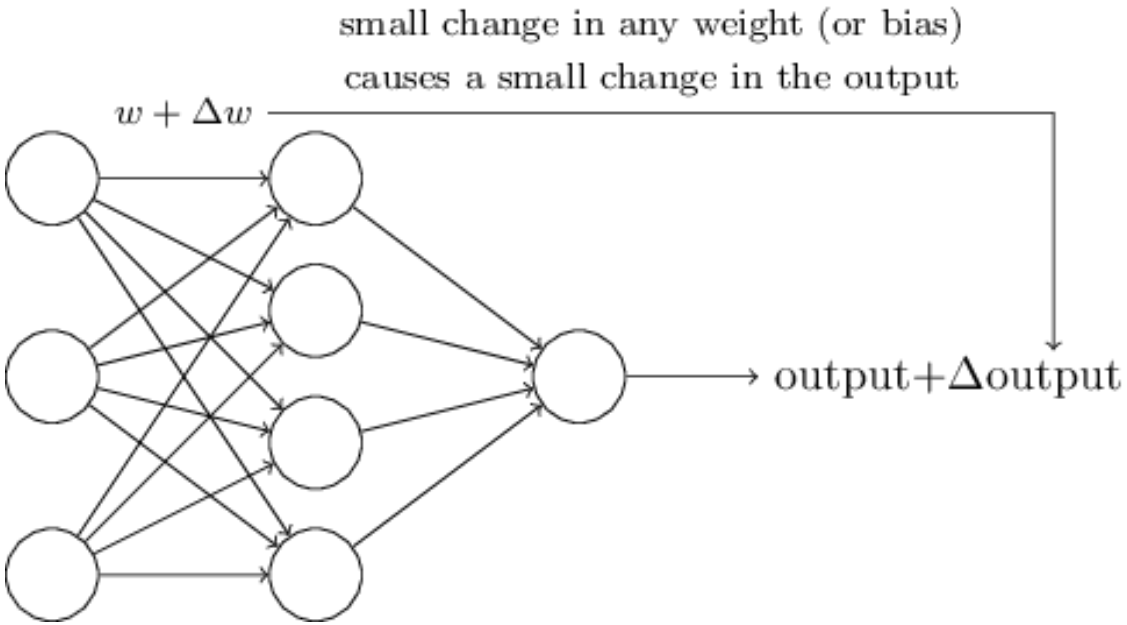


$$\mathrm{er}[y, \hat{y}]$$

error

$$(y - \hat{y})^2$$

REDUCE ERROR

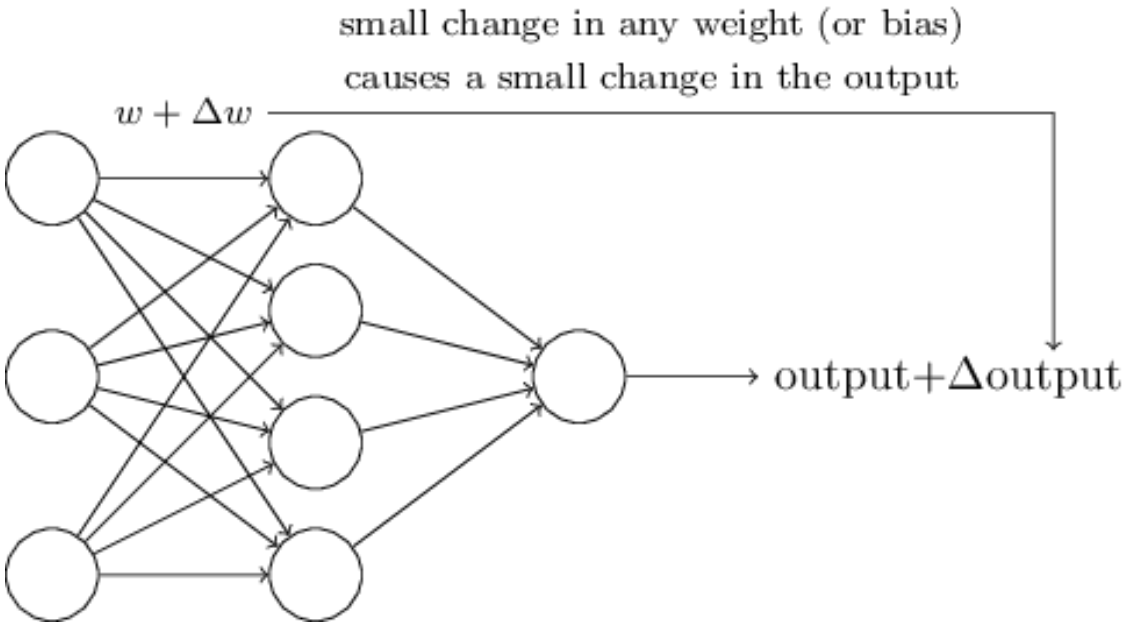


What happens to error ?

$$(y - \hat{y})^2$$

http://neuralnetworksanddeeplearning.com/chap1.html

Show demo fitting_point2.html which weights to tune



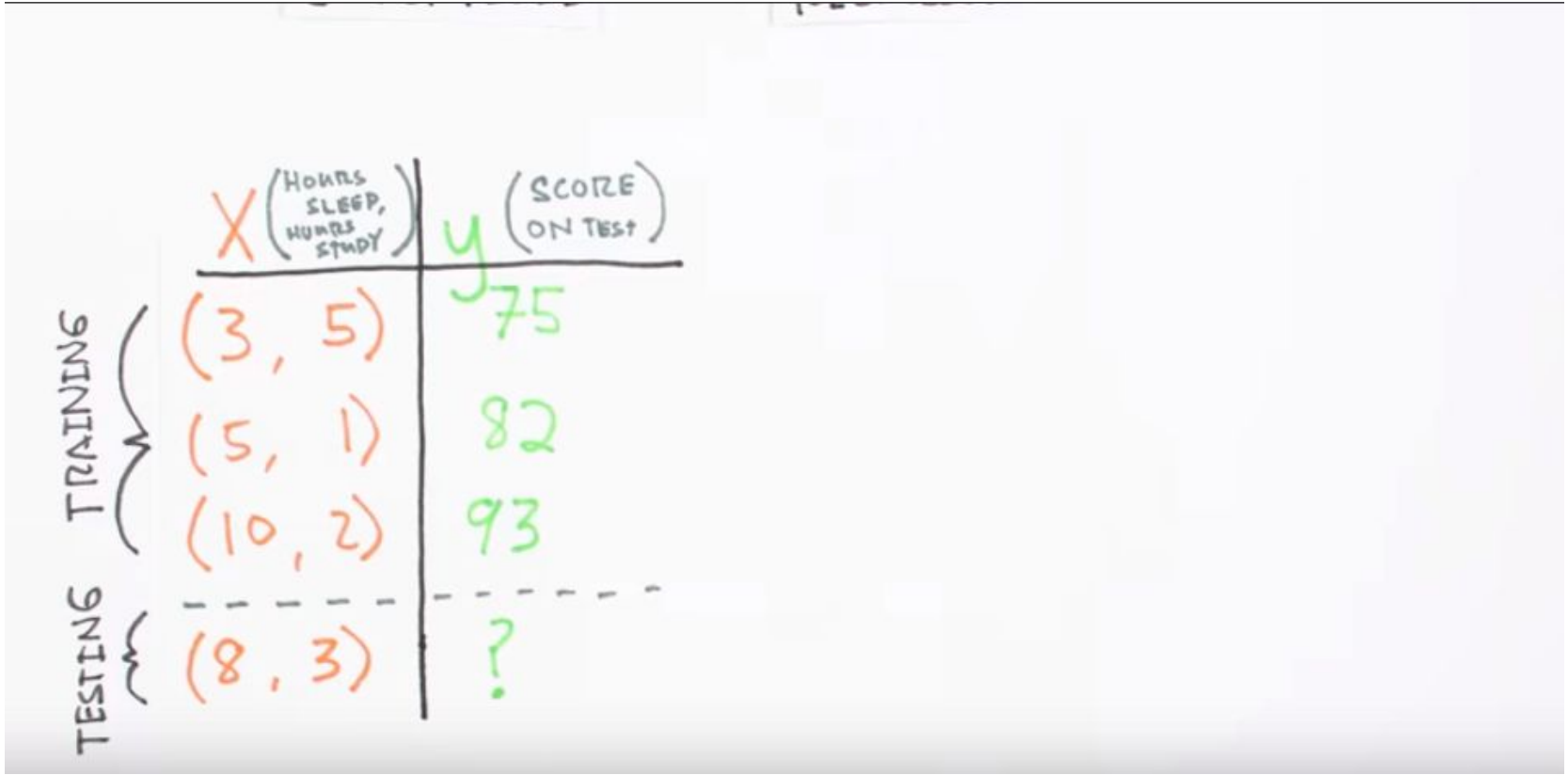
Any way to automate the process ?

$$(y - \hat{y})^2$$

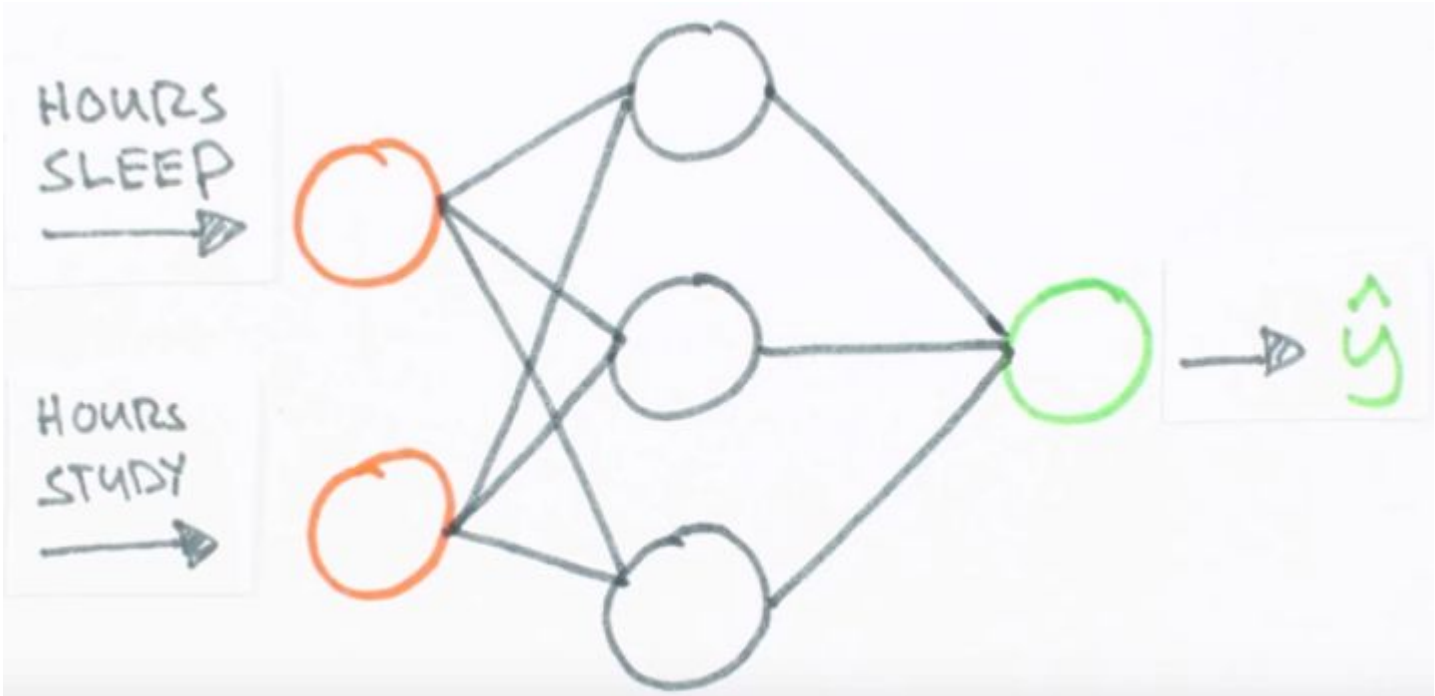
http://neuralnetworksanddeeplearning.com/chap1.html

Show demo fitting_point2.html which weights to tune

Neuron Networks training



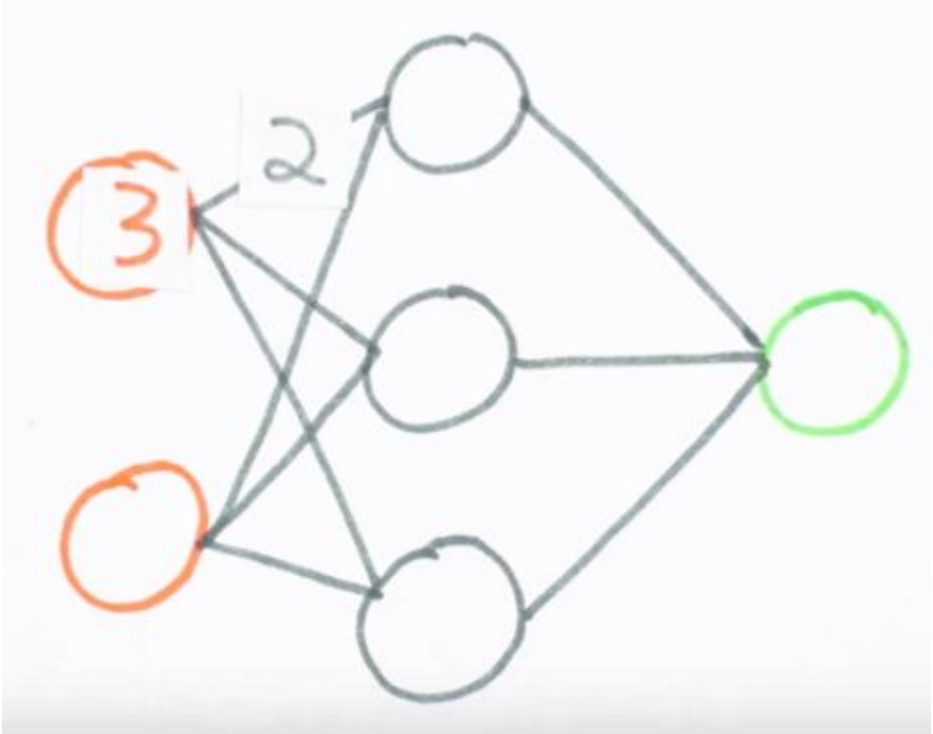
Multi-Neuron Networks :: Architecture



Multi-Neuron Networks :: Architecture

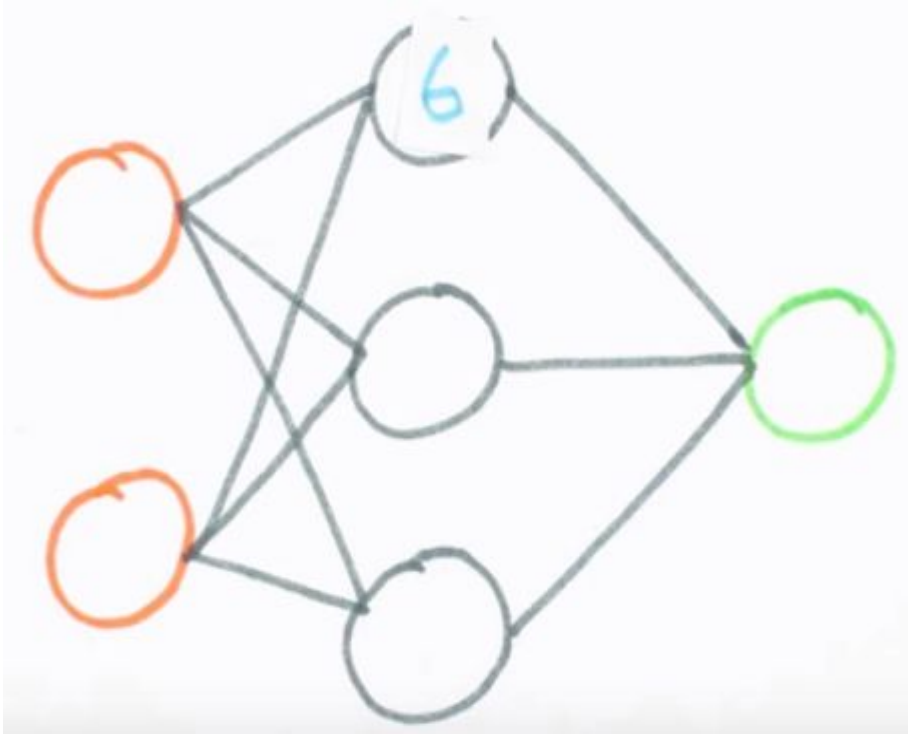
Synapse

Weight on Synapse = 2



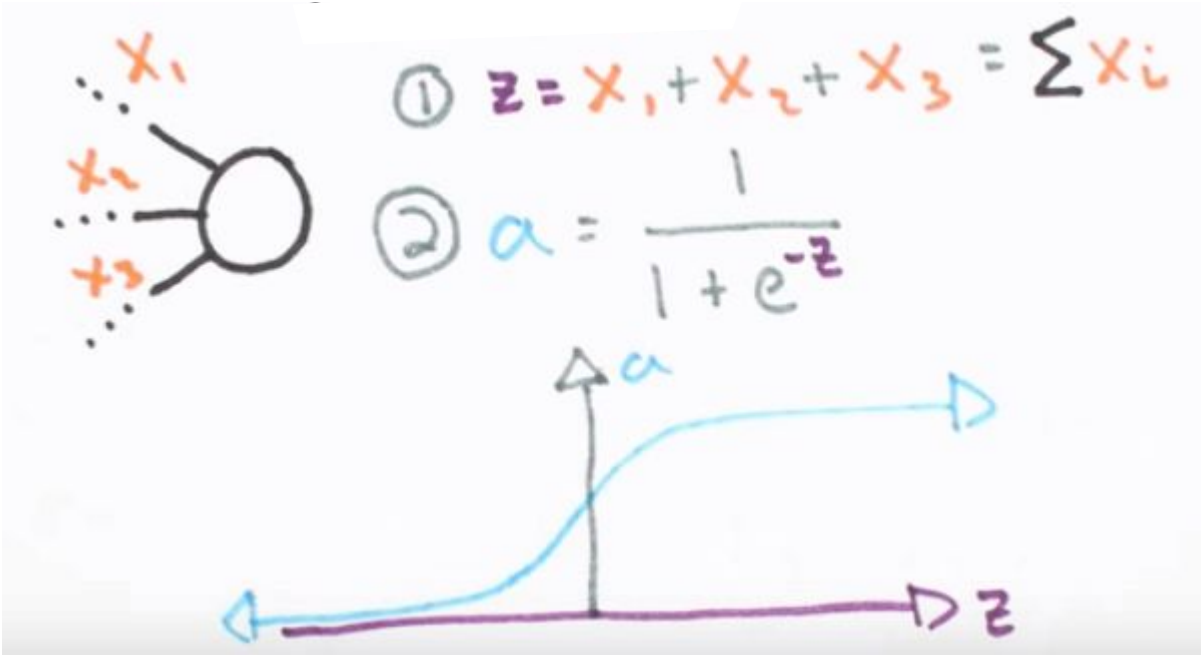
Multi-Neuron Networks :: Architecture

Synapse

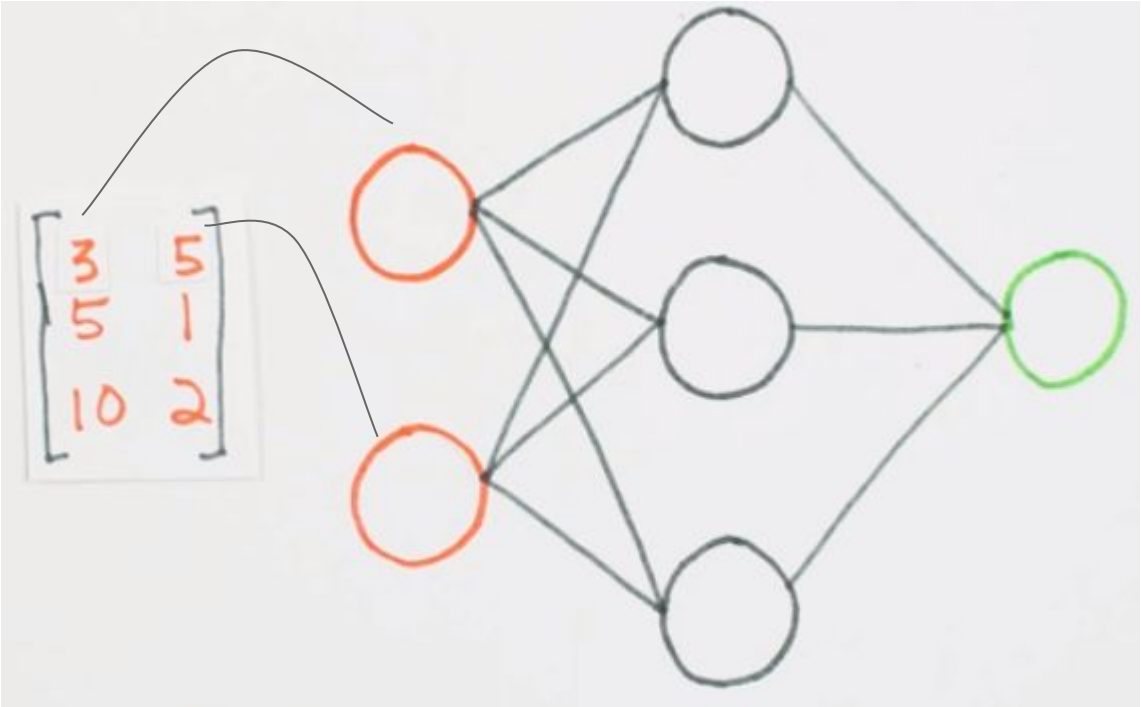


Multi-Neuron Networks :: Architecture

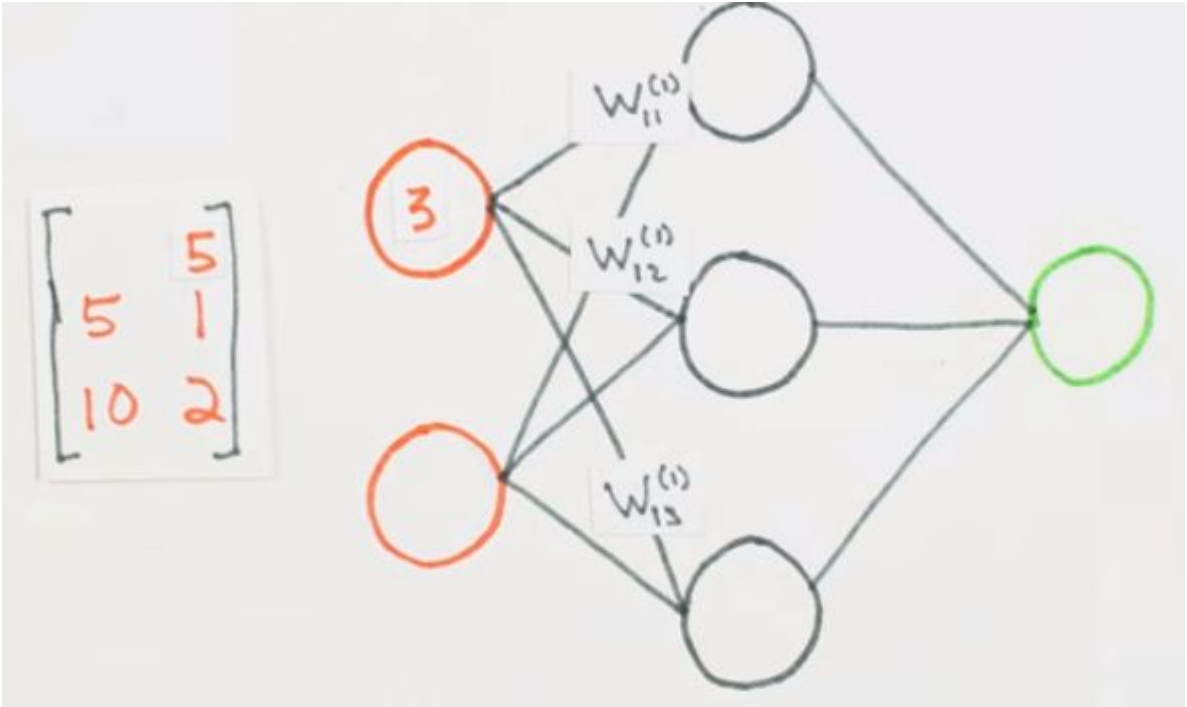
Neuron



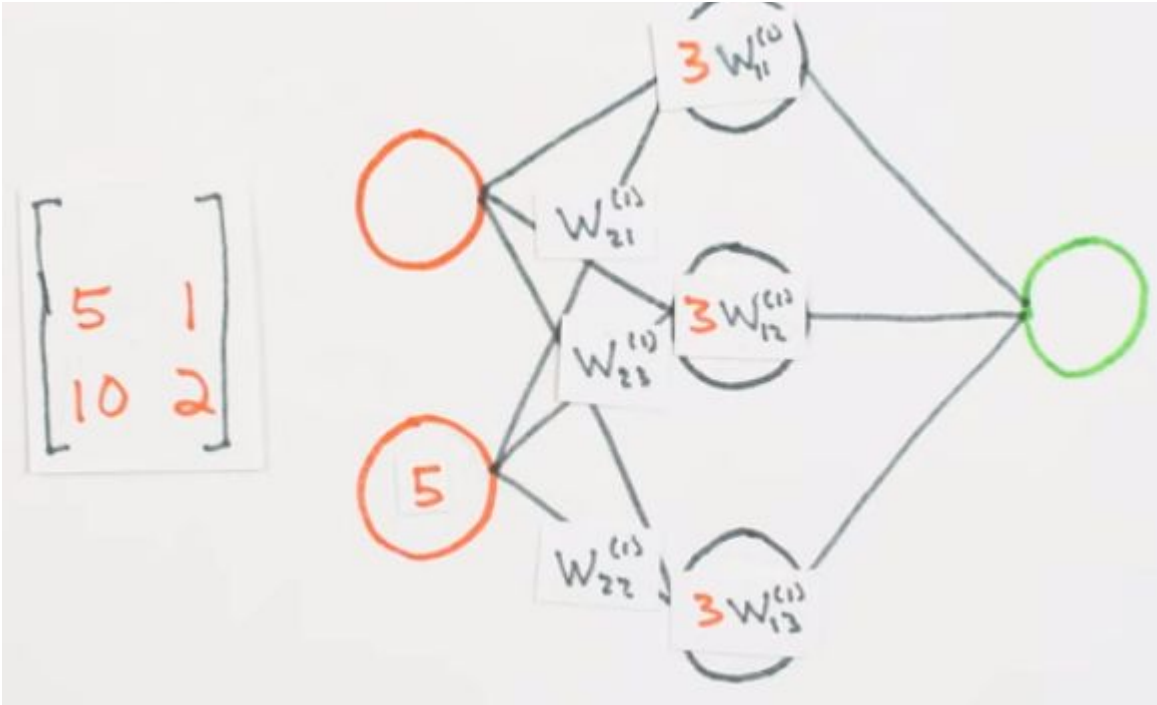
Learning Step 1: Forward propagation



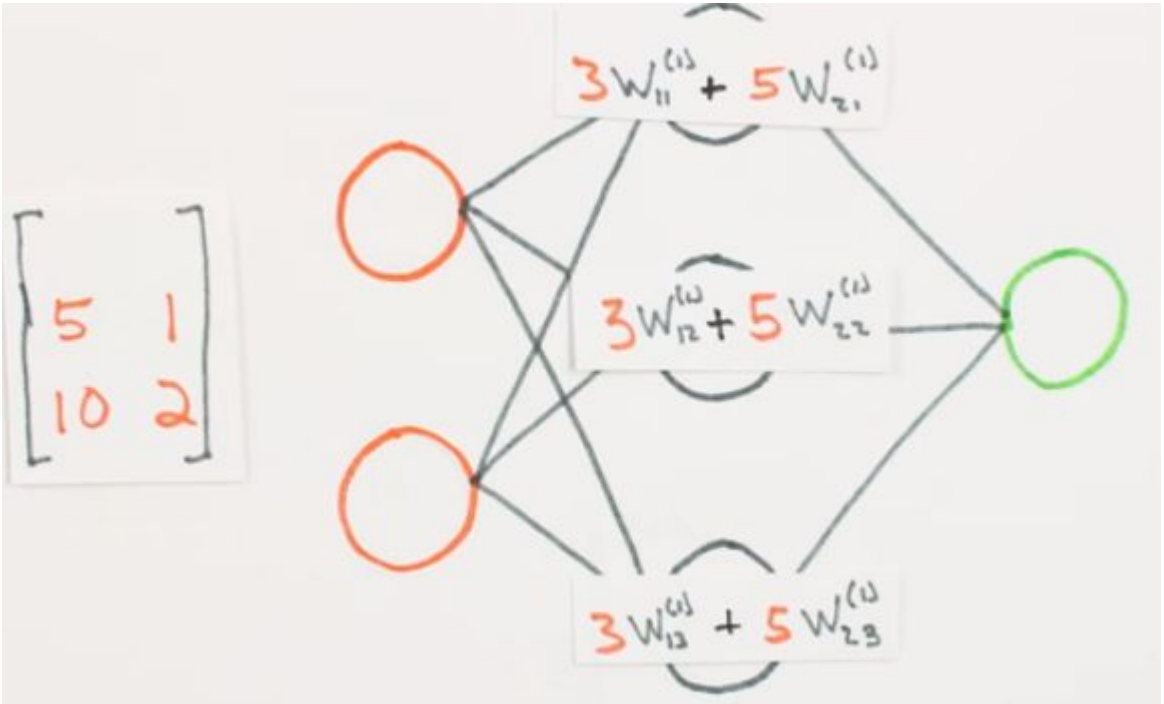
Multi-Neuron Networks :: forward propagation



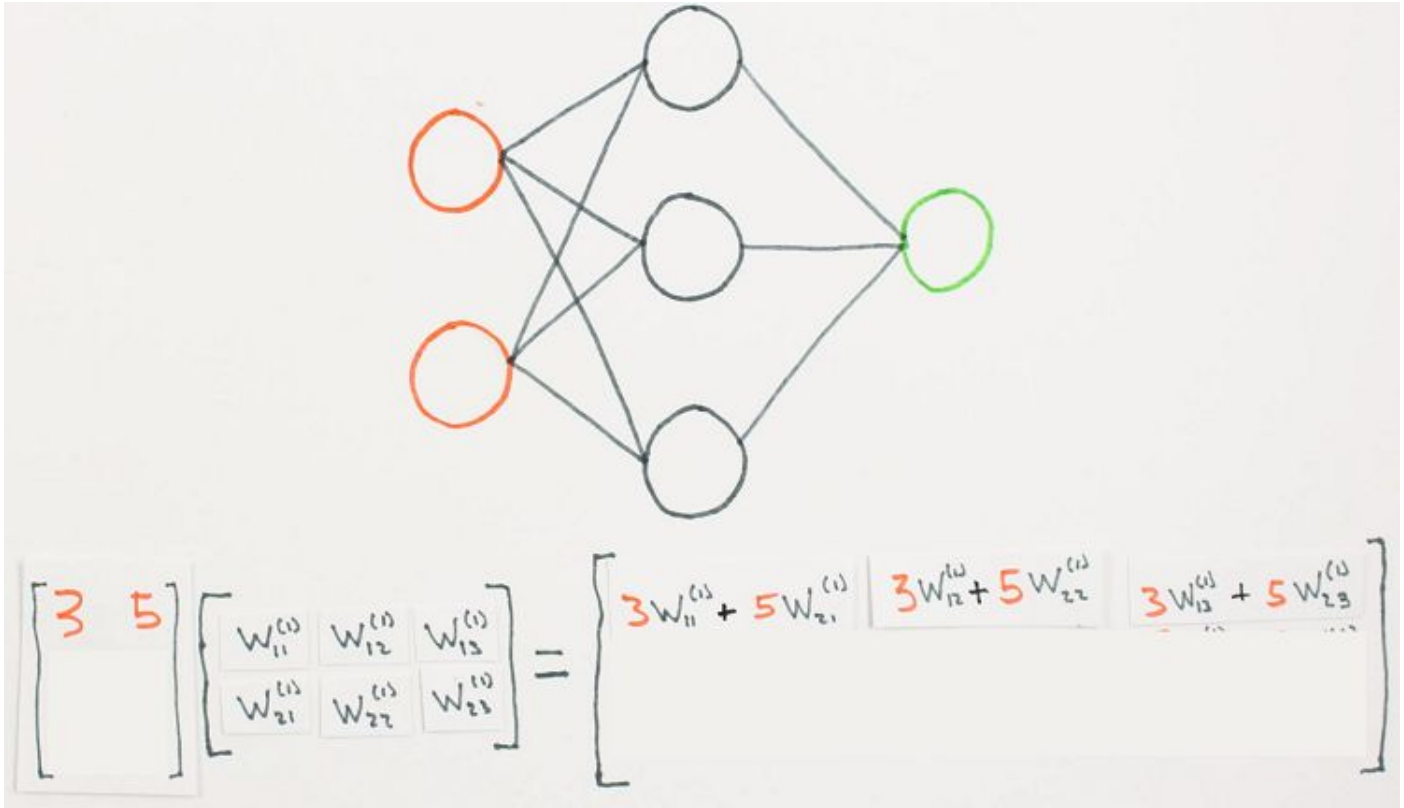
Multi-Neuron Networks :: forward propagation



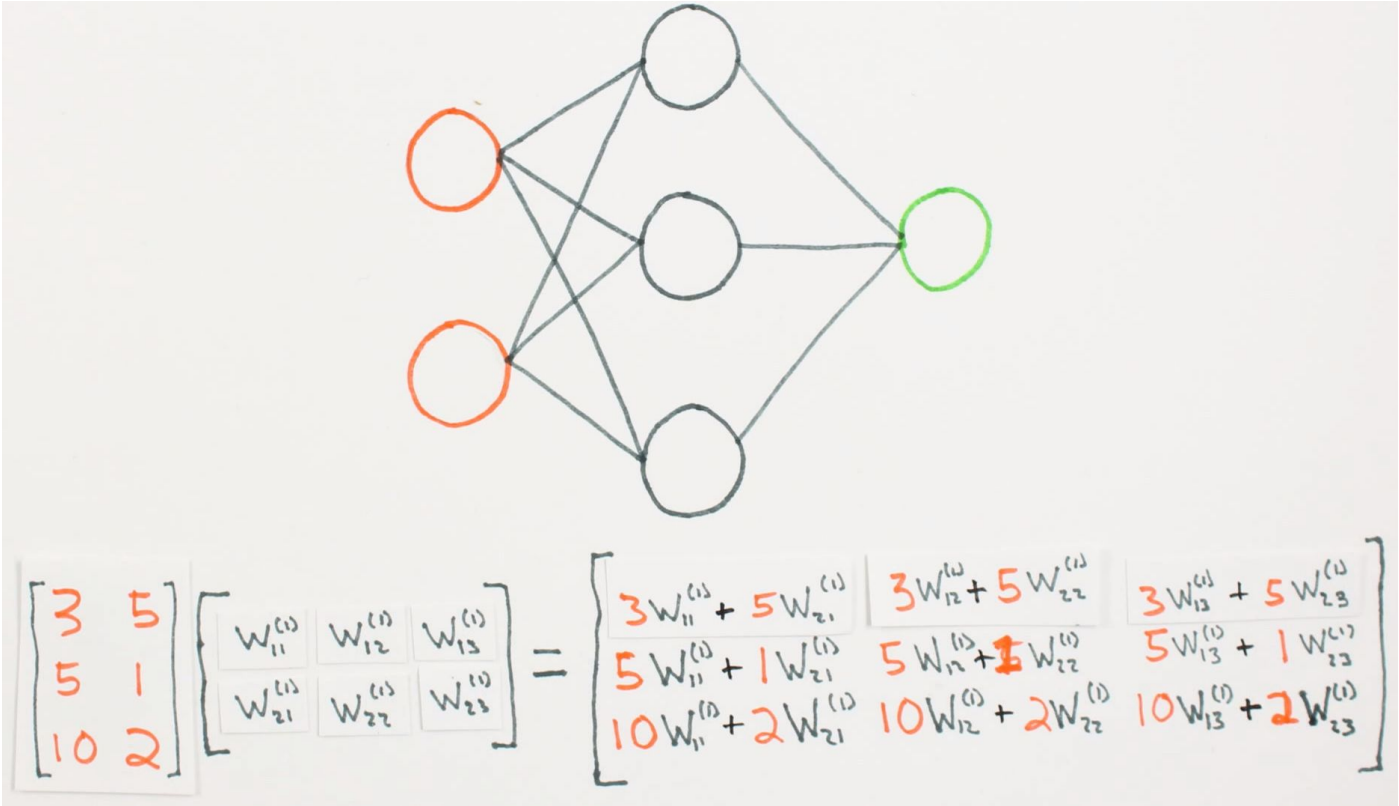
Multi-Neuron Networks :: forward propagation



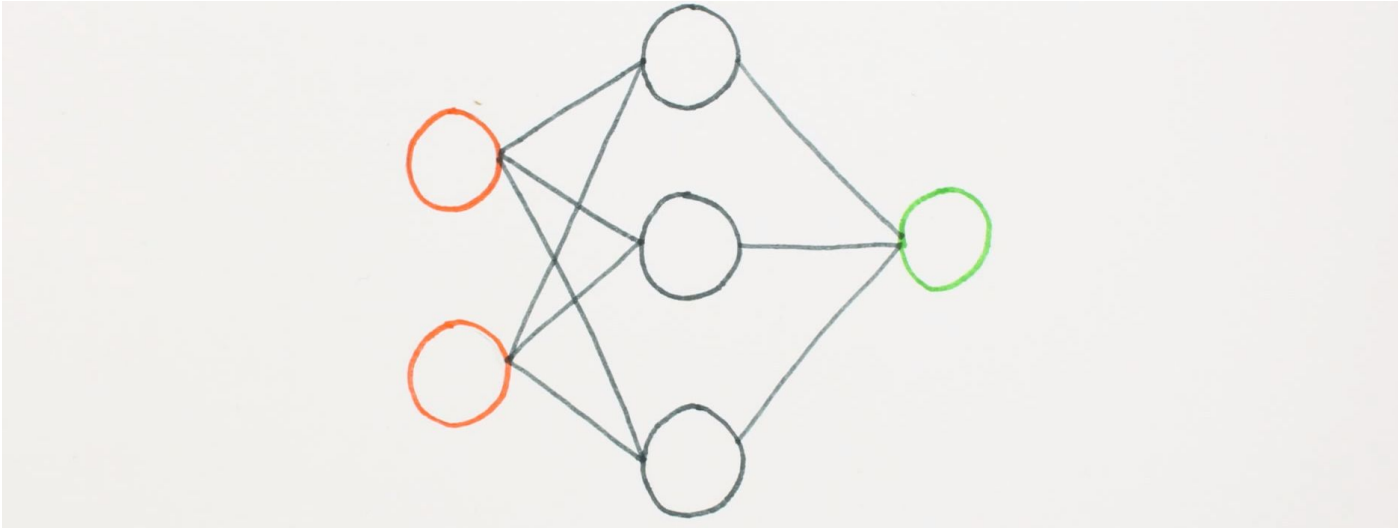
Multi-Neuron Networks :: forward propagation



Multi-Neuron Networks :: forward propagation

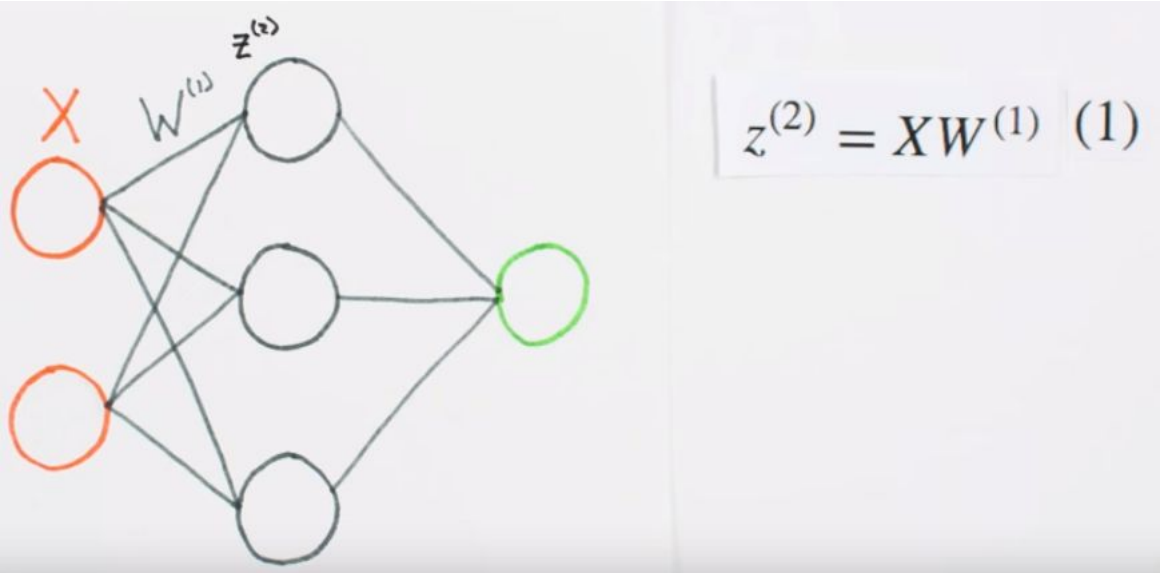


Multi-Neuron Networks :: forward propagation

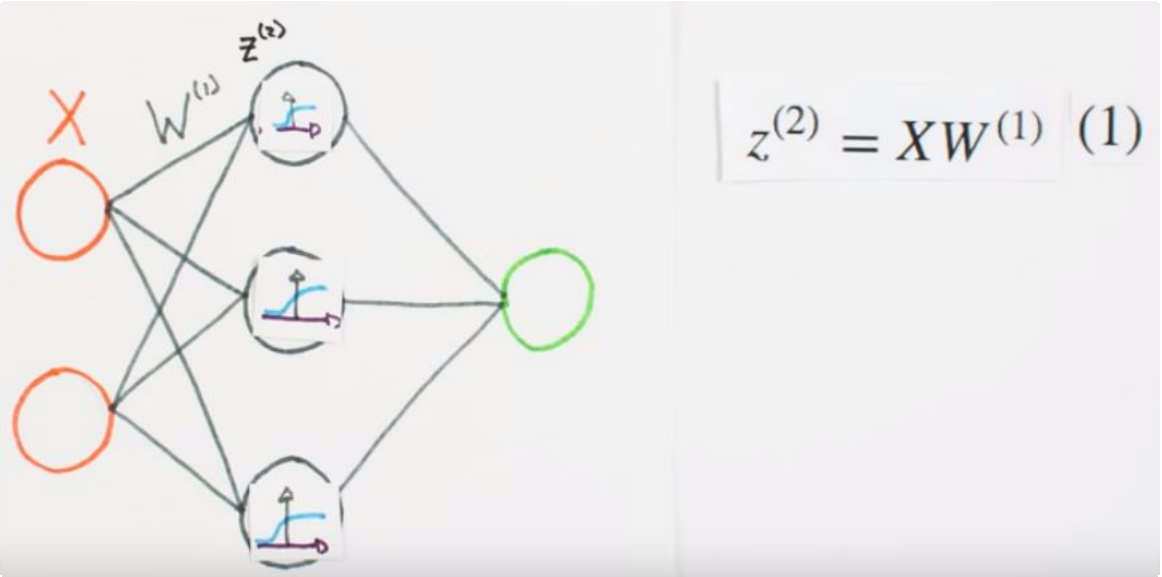


$$X \mid W^{(1)} = \mid Z^{(2)}$$

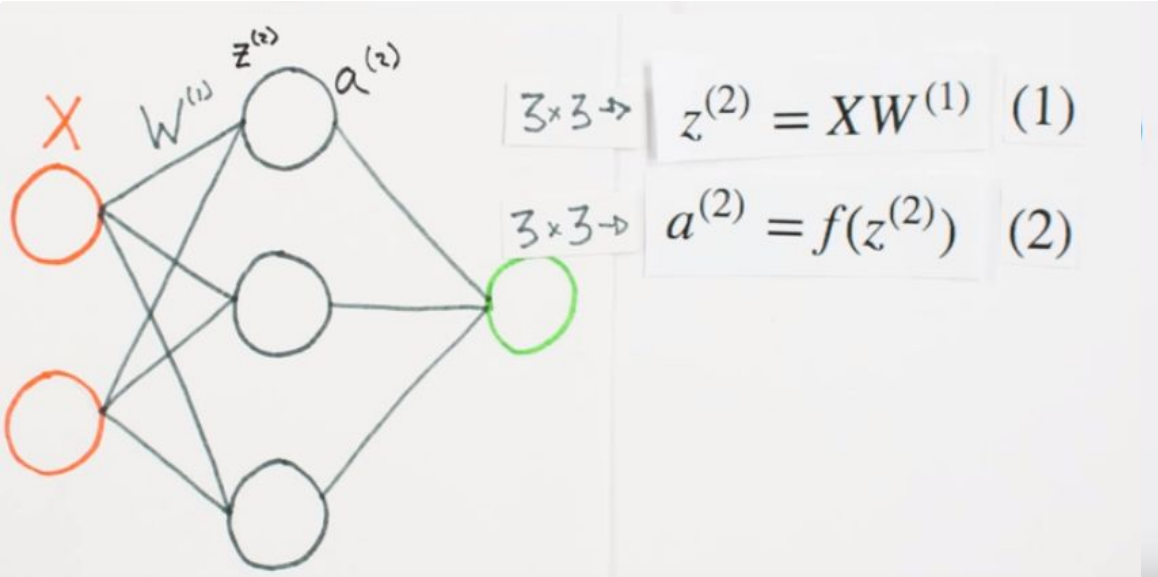
Multi-Neuron Networks :: forward propagation



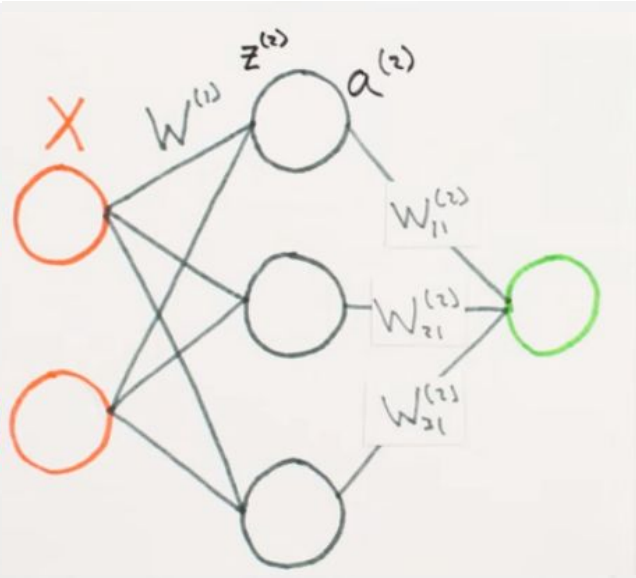
Multi-Neuron Networks :: forward propagation



Multi-Neuron Networks :: forward propagation



Multi-Neuron Networks :: forward propagation

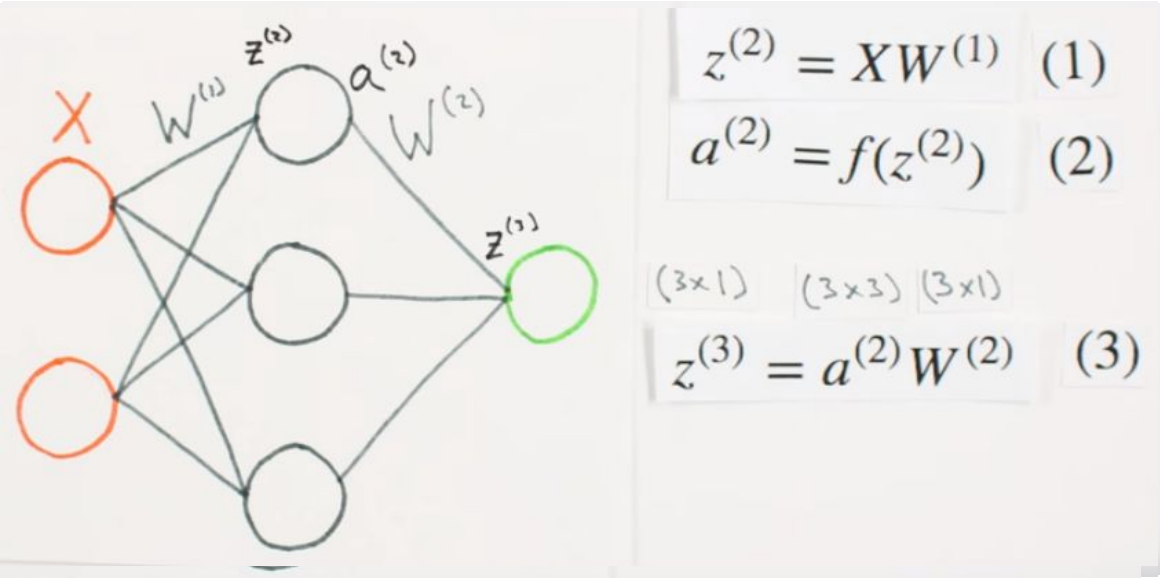


$$z^{(2)} = XW^{(1)} \quad (1)$$

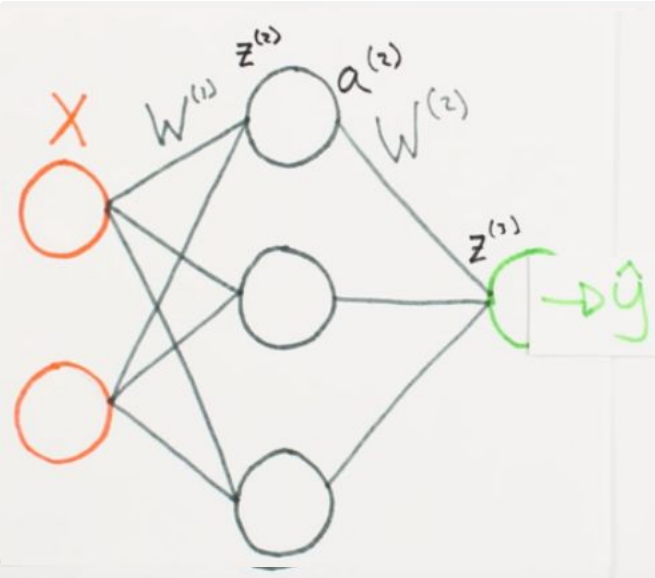
$$a^{(2)} = f(z^{(2)}) \quad (2)$$

$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

Multi-Neuron Networks :: forward propagation



Multi-Neuron Networks :: forward propagation



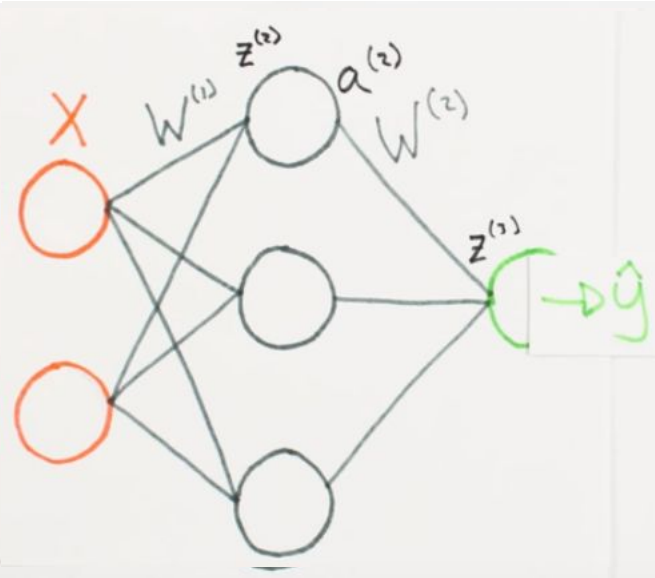
$$z^{(2)} = XW^{(1)} \quad (1)$$

$$a^{(2)} = f(z^{(2)}) \quad (2)$$

$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

$$\hat{y} = f(z^{(3)}) \quad (4)$$

Multi-Neuron Networks :: forward propagation



$$z^{(2)} = XW^{(1)} \quad (1)$$

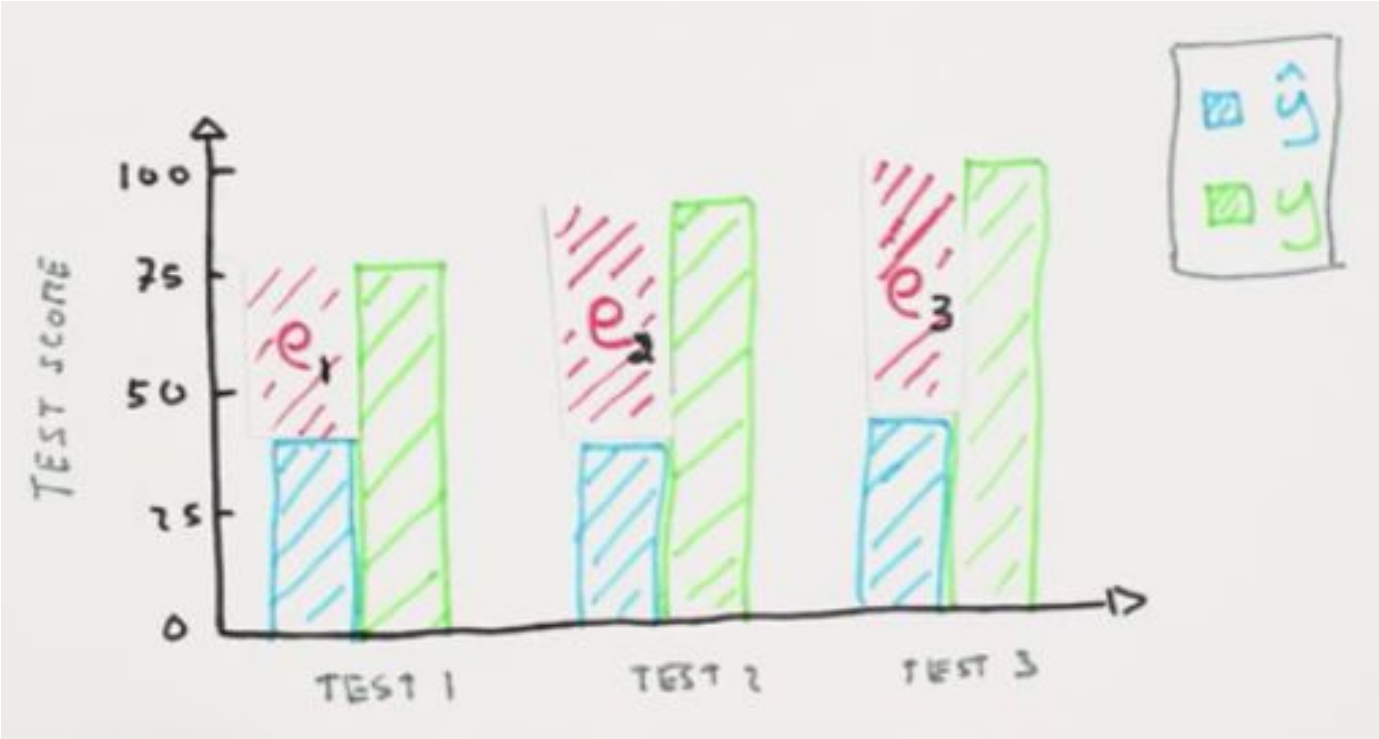
$$a^{(2)} = f(z^{(2)}) \quad (2)$$

$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

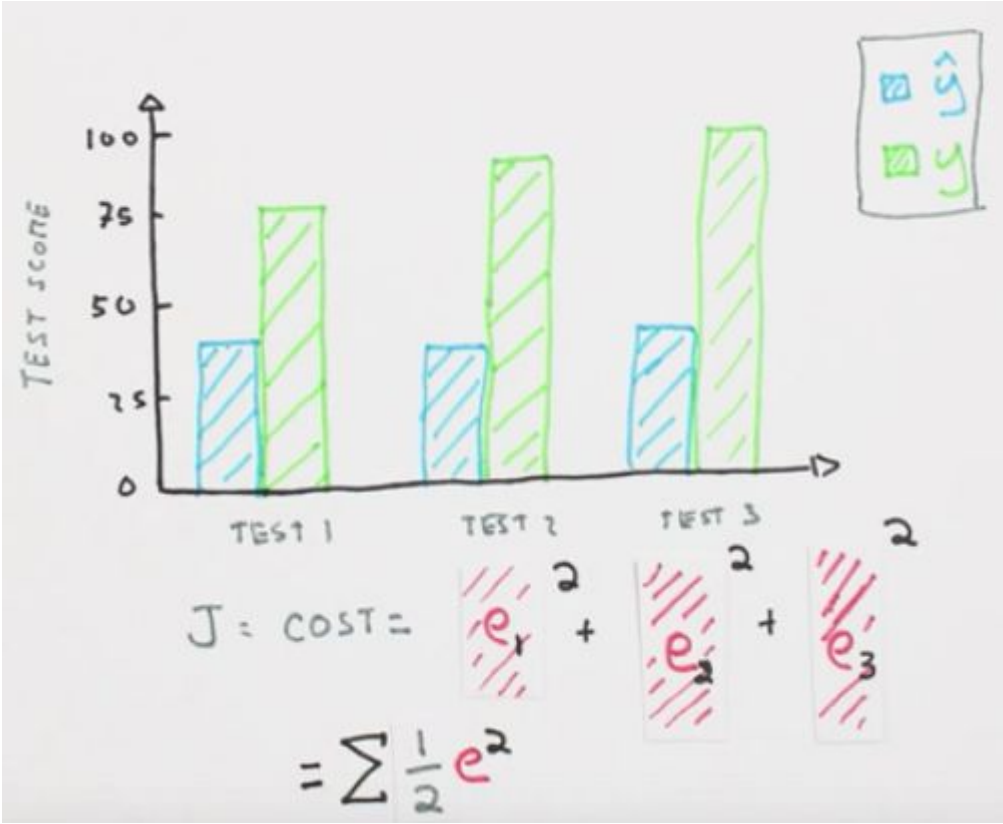
$$\hat{y} = f(z^{(3)}) \quad (4)$$

Multi-Neuron Networks :: Gradient Descent

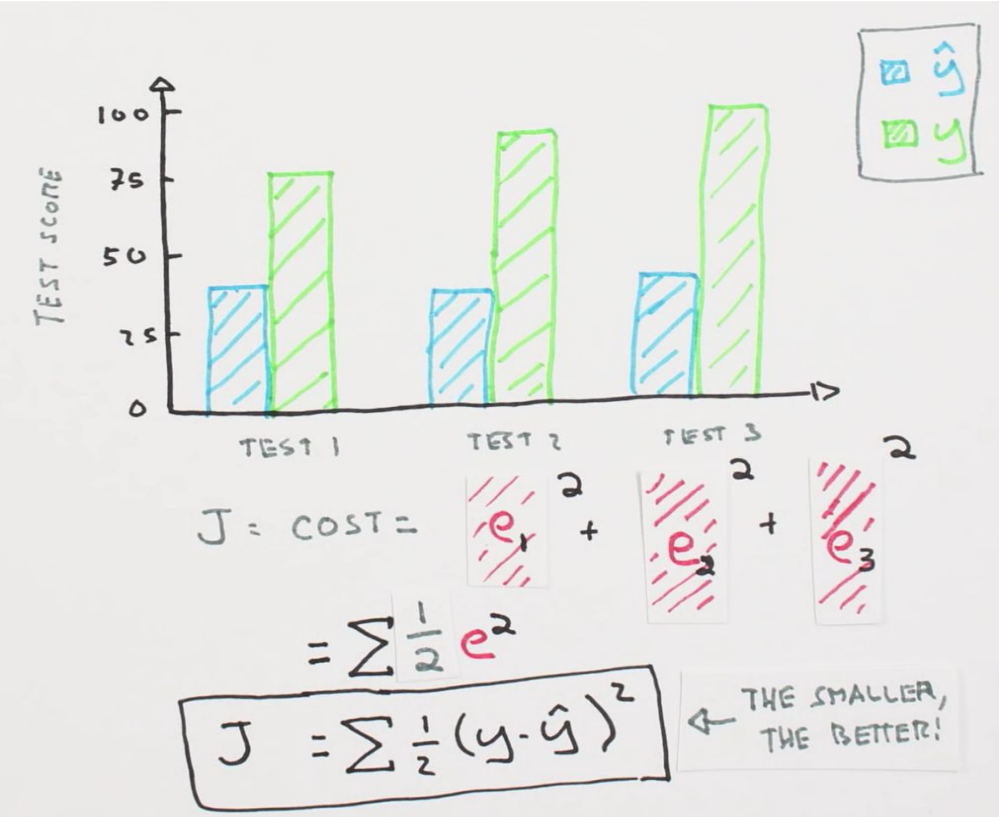
Multi-Neuron Networks :: Gradient Descent



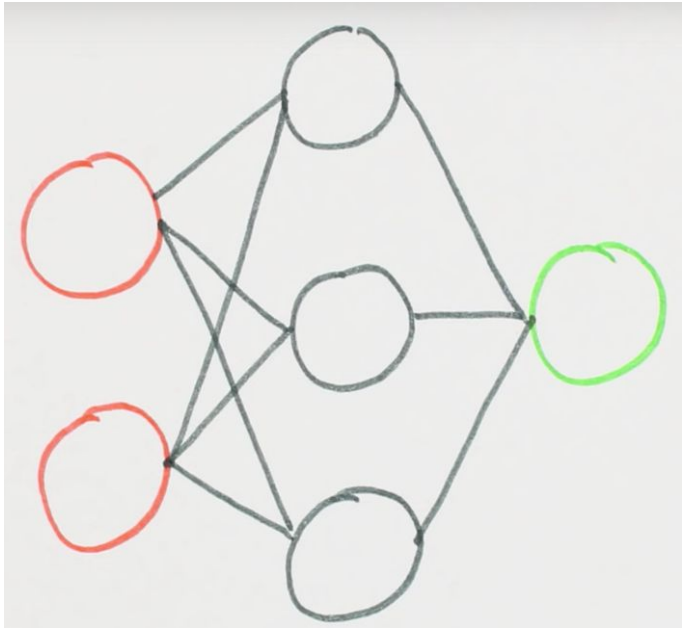
Multi-Neuron Networks :: Gradient Descent



Multi-Neuron Networks :: Gradient Descent



Multi-Neuron Networks :: Gradient Descent

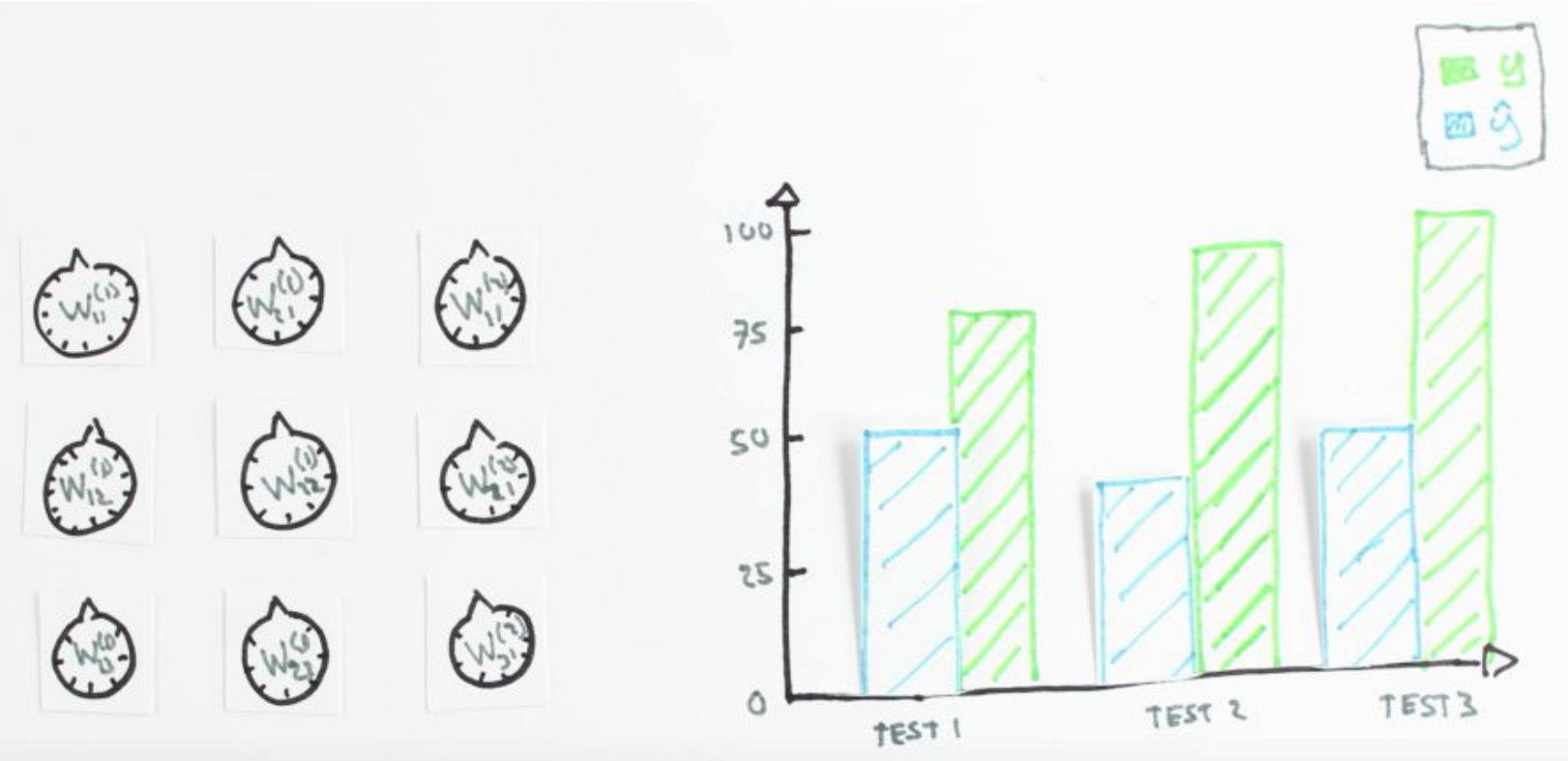


Training a Network

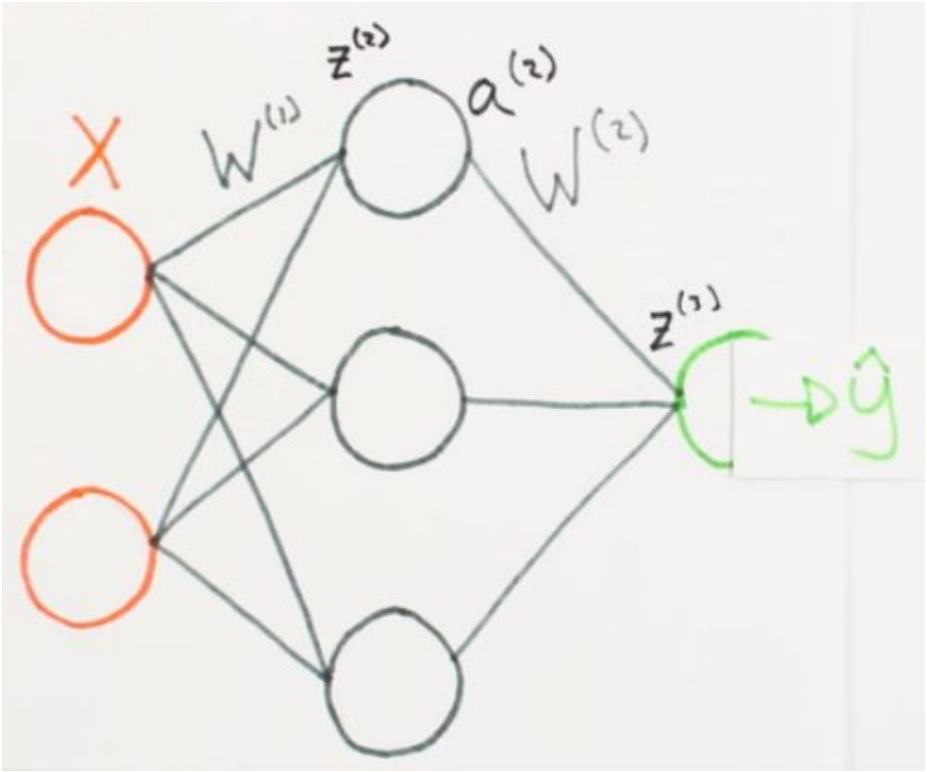
=

Minimizing a Cost Function

Multi-Neuron Networks :: Gradient Descent



Multi-Neuron Networks :: Gradient Descent



$$z^{(2)} = XW^{(1)} \quad (1)$$

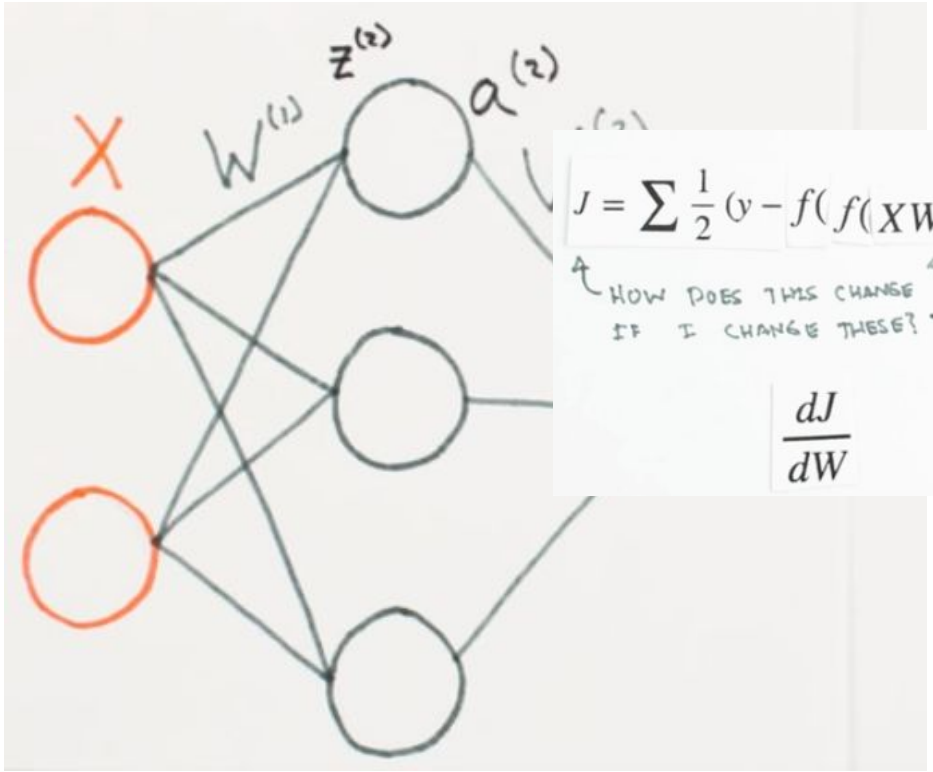
$$a^{(2)} = f(z^{(2)}) \quad (2)$$

$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

$$\hat{y} = f(z^{(3)}) \quad (4)$$

$$J = \sum \frac{1}{2}(y - \hat{y})^2 \quad (5)$$

Multi-Neuron Networks :: Gradient Descent



$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

HOW DOES THIS CHANGE IF I CHANGE THESE?

$$\frac{dJ}{dW}$$

$$z^{(2)} = XW^{(1)} \quad (1)$$

$$a^{(2)} = f(z^{(2)}) \quad (2)$$

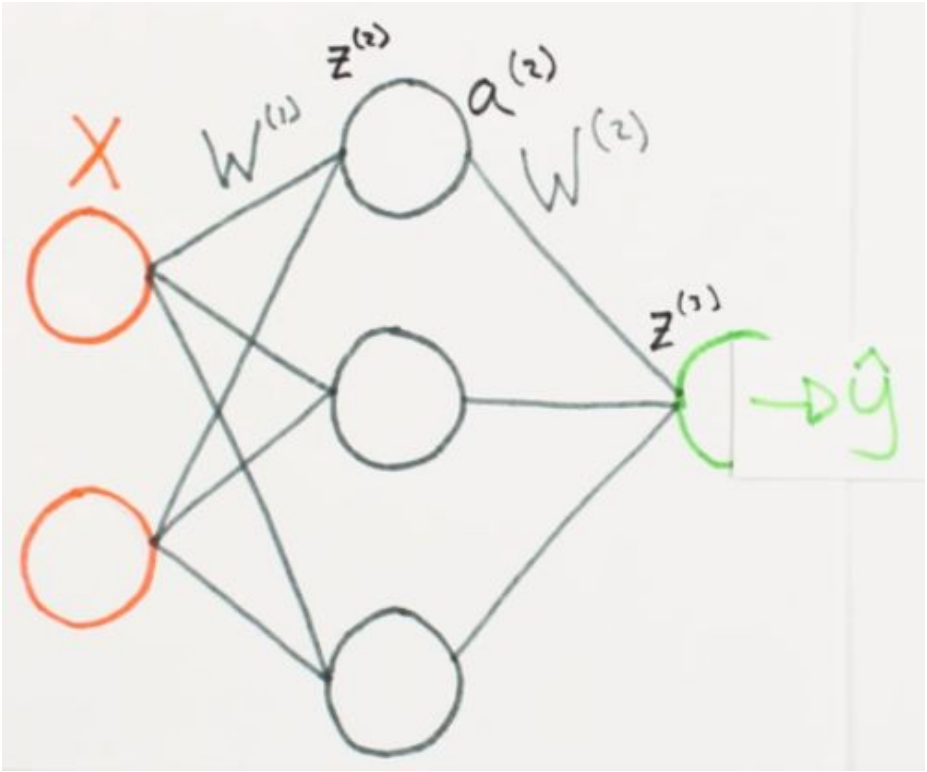
$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

$$\hat{y} = f(z^{(3)}) \quad (4)$$

$$J = \sum \frac{1}{2}(y - \hat{y})^2 \quad (5)$$

$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

Multi-Neuron Networks :: Gradient Descent

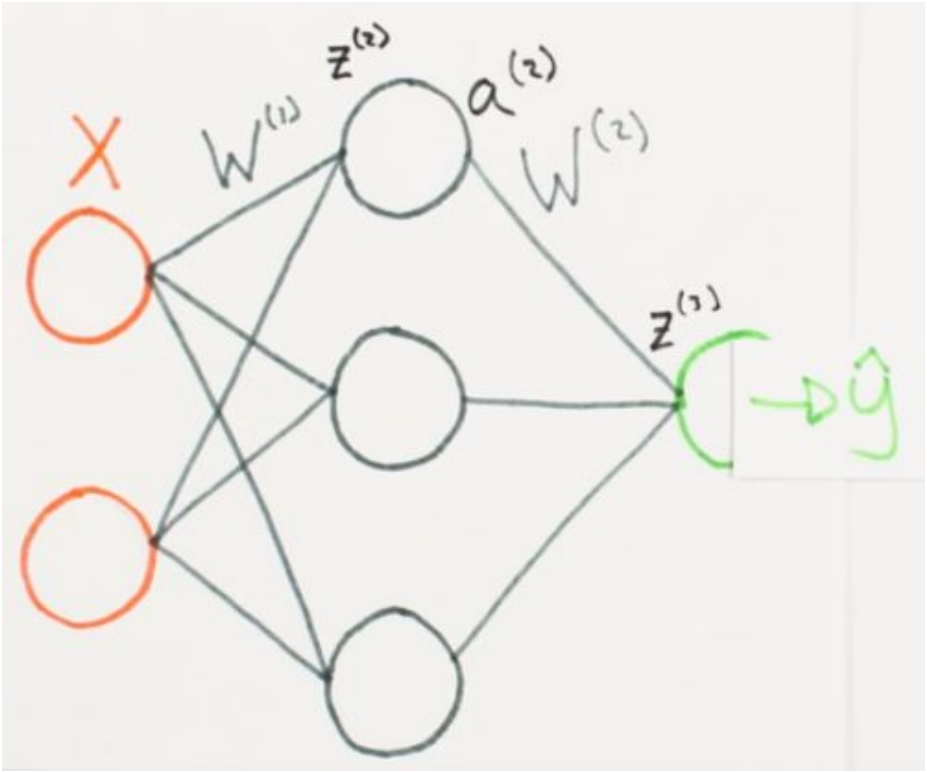


$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

HOW DOES THIS CHANGE IF I CHANGE THESE?

$$\frac{dJ}{dW}$$

Multi-Neuron Networks :: Gradient Descent

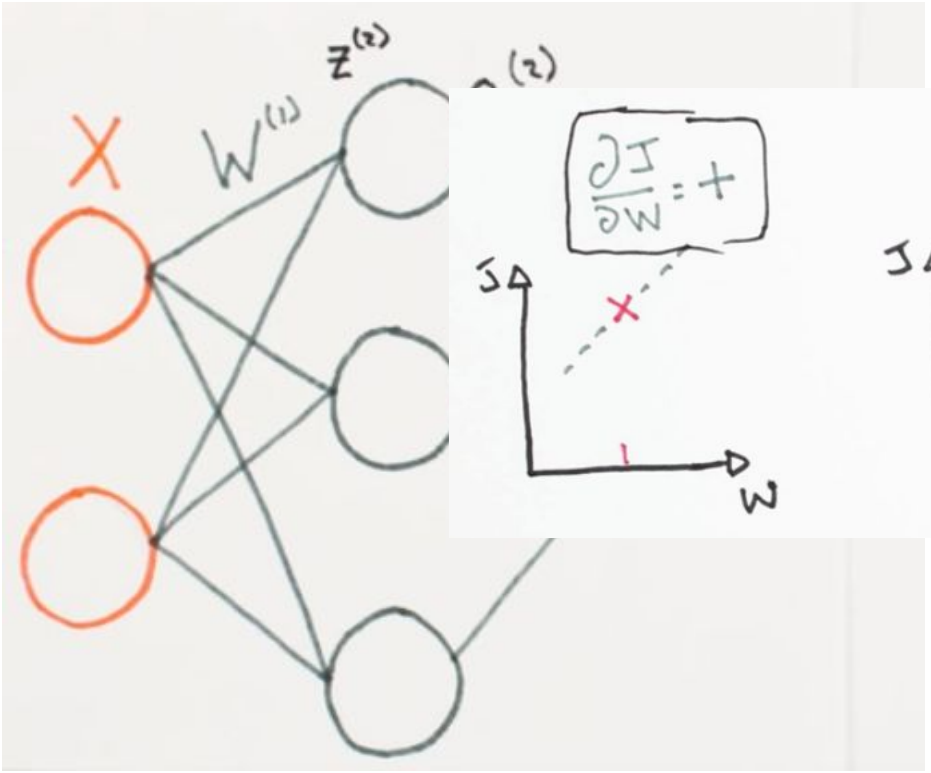


$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

HOW DOES THIS CHANGE IF I CHANGE THESE?

$$\frac{\partial J}{\partial W}$$

Multi-Neuron Networks :: Gradient Descent



$$f(f(XW^{(1)})W^{(2)})^2$$

IS CHANGE
GE THESE?

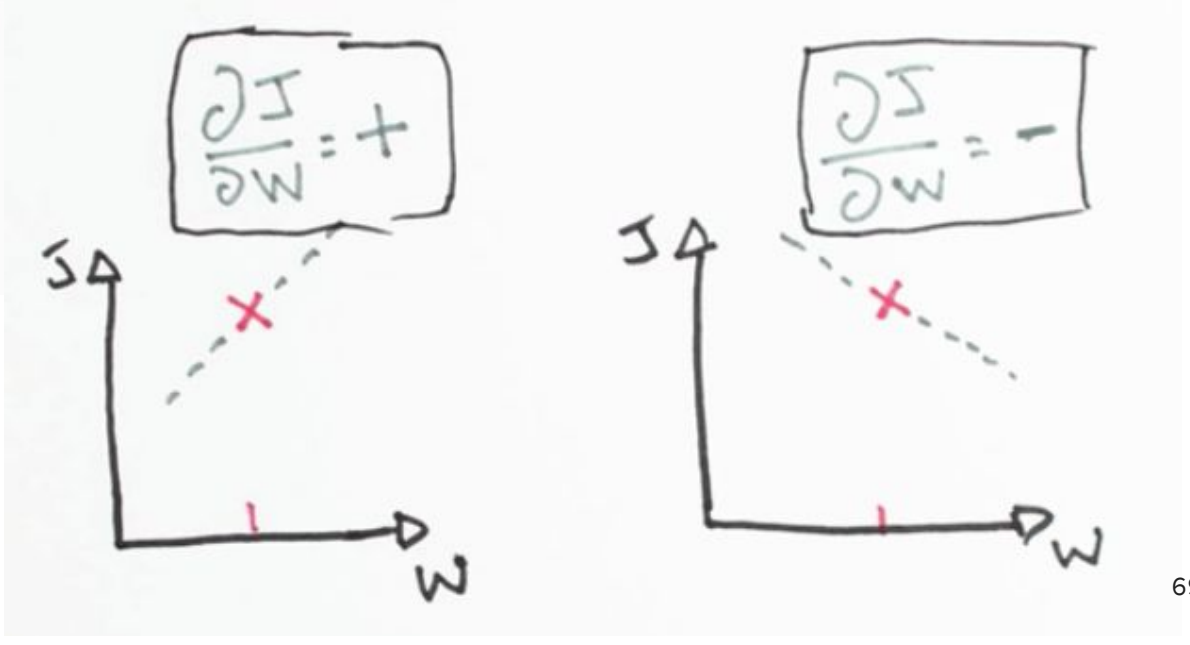
$$\frac{\partial J}{\partial W}$$

Multi-Neuron Networks :: Gradient Descent

$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

HOW DOES THIS CHANGE IF I CHANGE THESE?

$$\frac{\partial J}{\partial W}$$

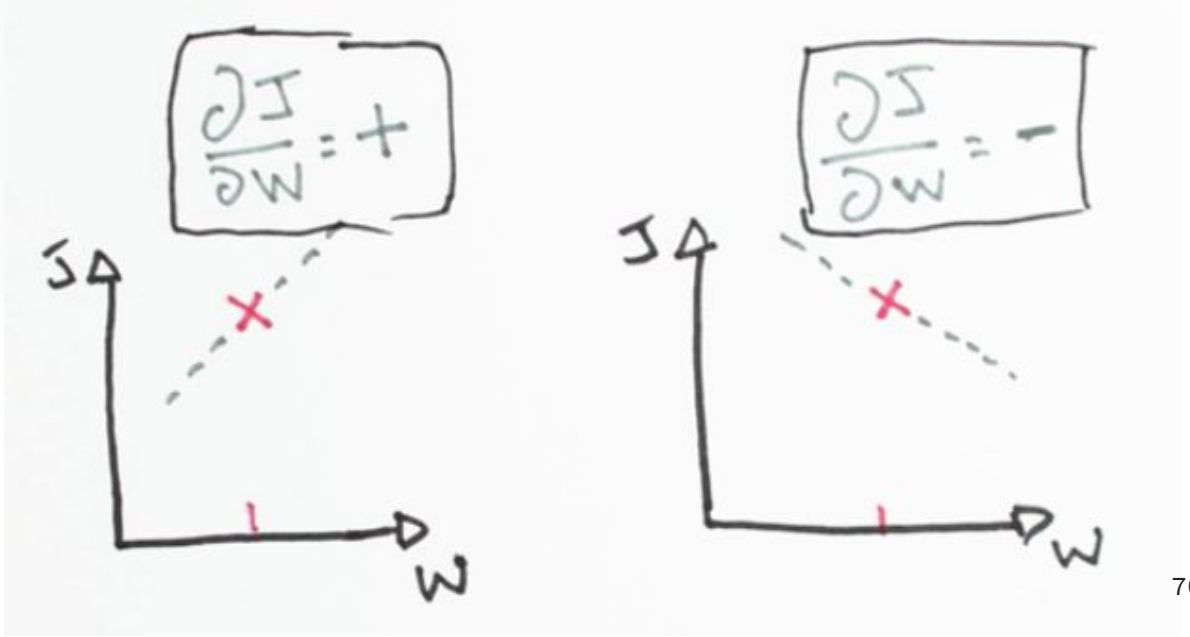


Multi-Neuron Networks :: Gradient Descent

$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

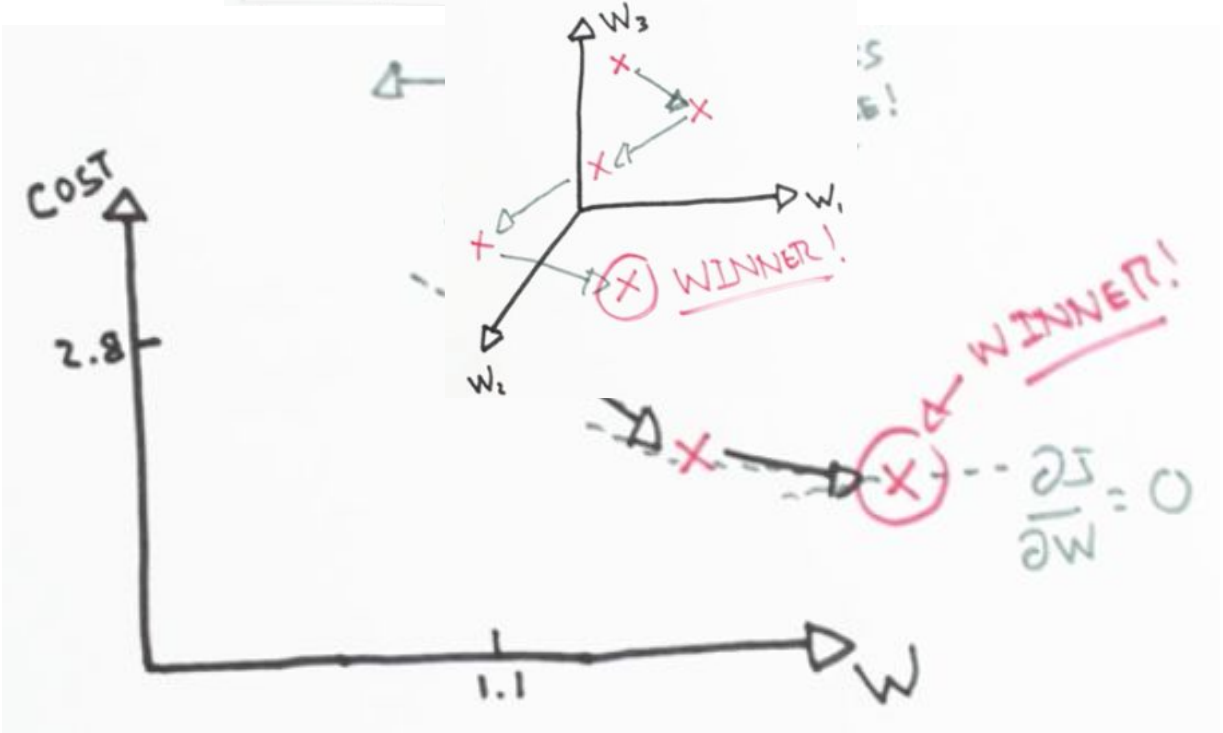
HOW DOES THIS CHANGE IF I CHANGE THESE?

$$\frac{\partial J}{\partial W}$$

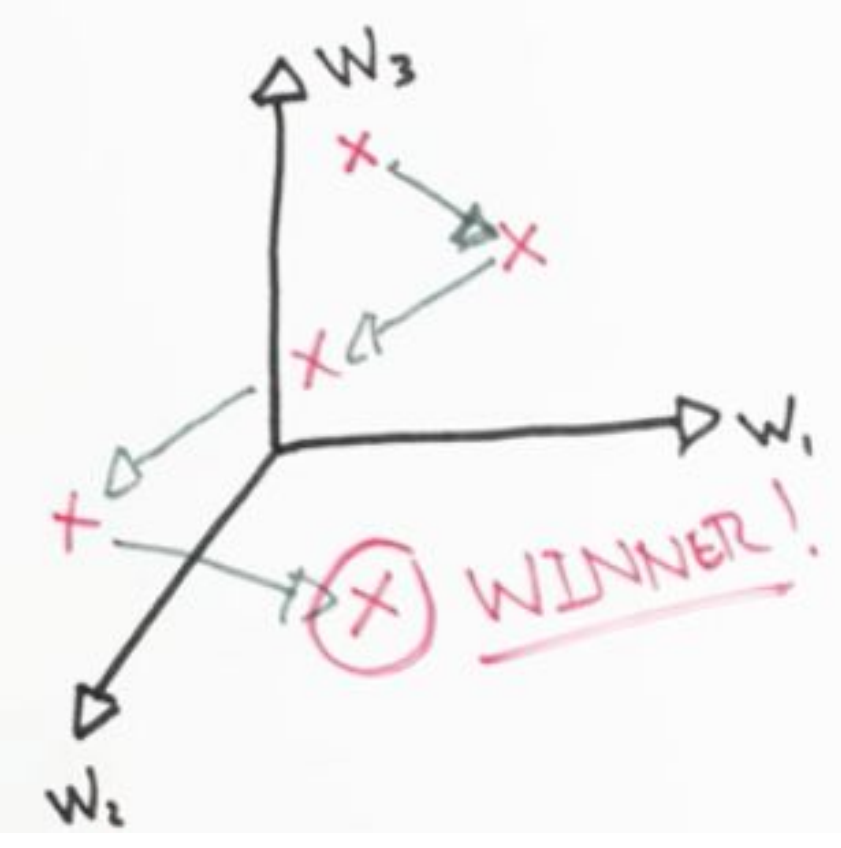


Multi-Neuron Networks :: Gradient Descent

$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$



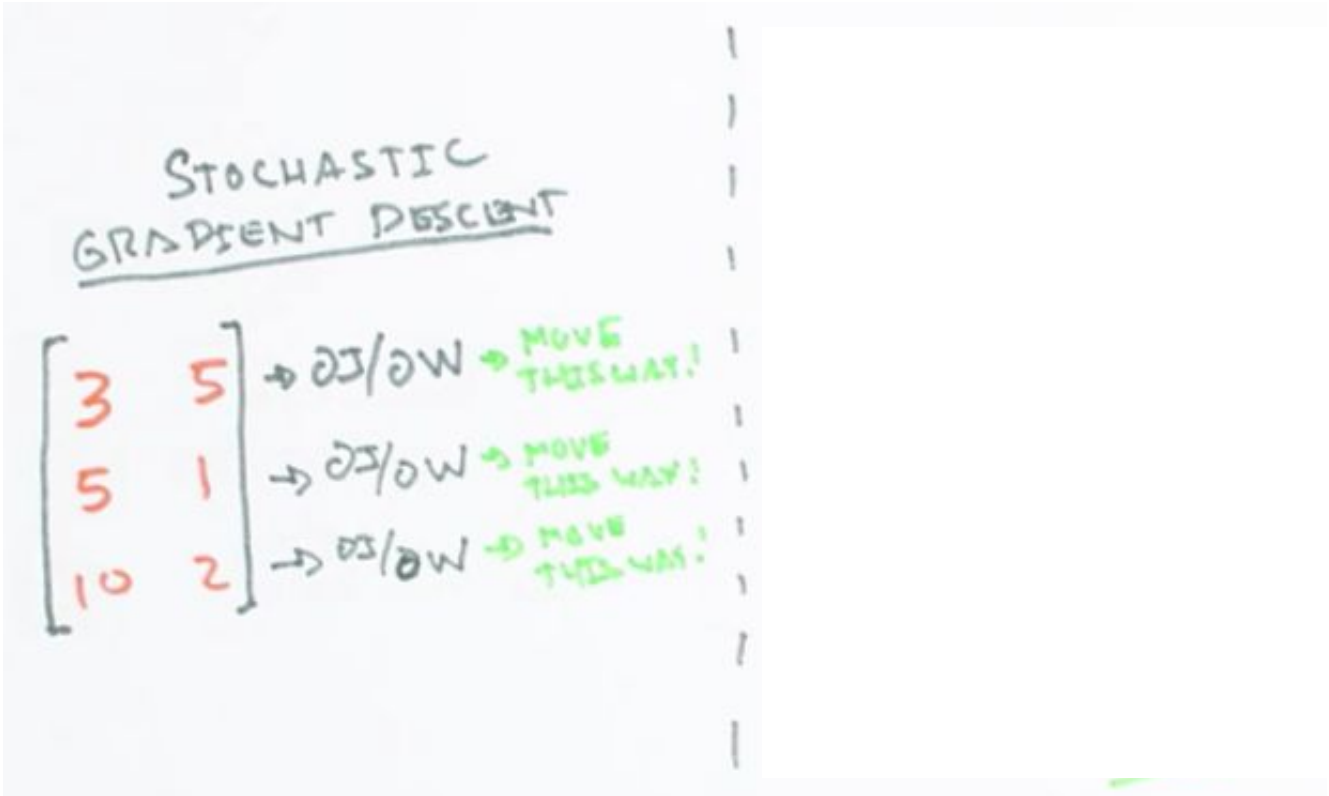
Multi-Neuron Networks :: Gradient Descent



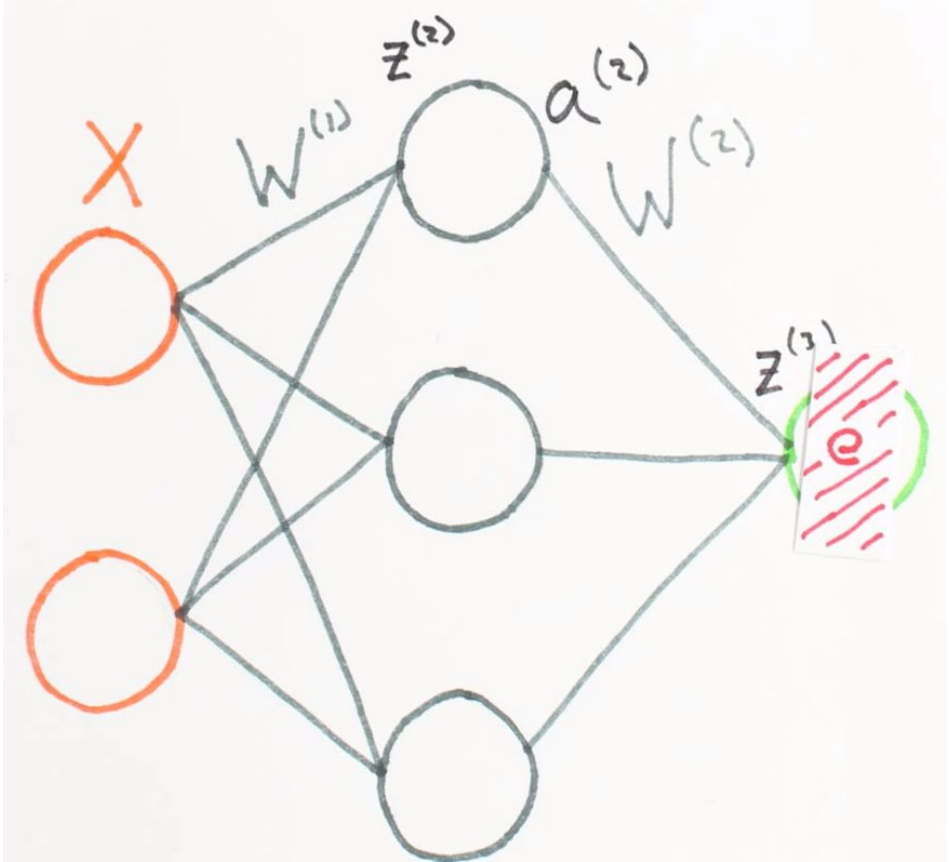
Multi-Neuron Networks :: Gradient Descent

$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

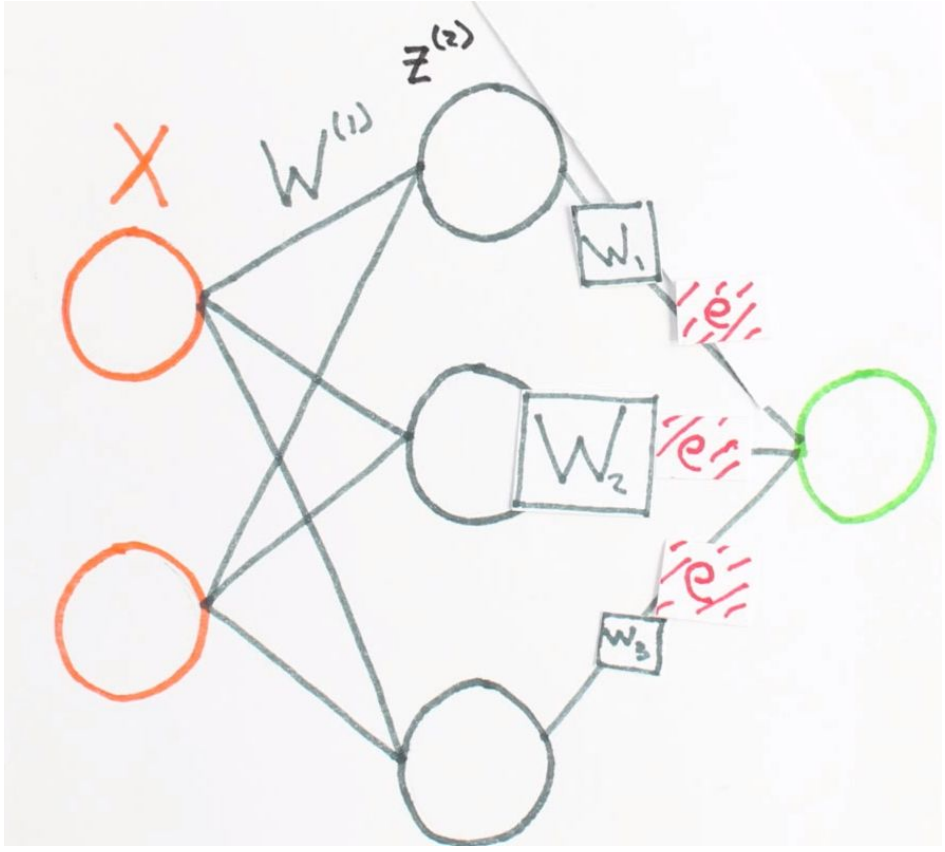
Multi-Neuron Networks :: Gradient Descent



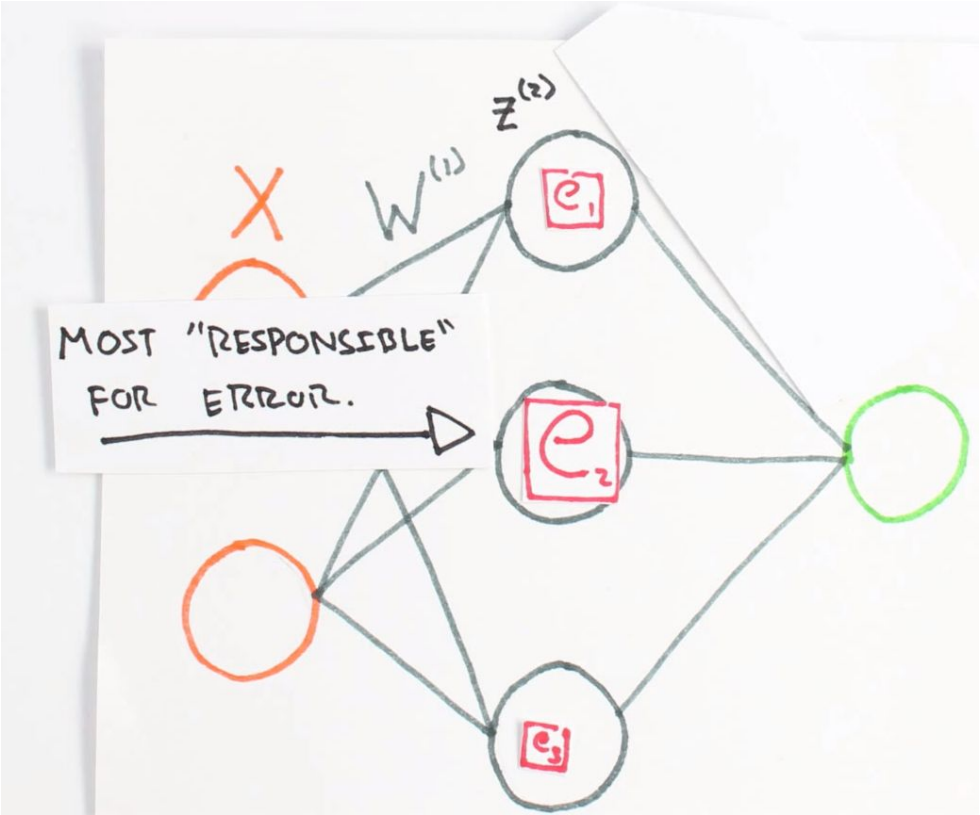
Multi-Neuron Networks :: Gradient Descent



Multi-Neuron Networks :: Gradient Descent



Multi-Neuron Networks :: Gradient Descent



Multi-Neuron Networks :: training

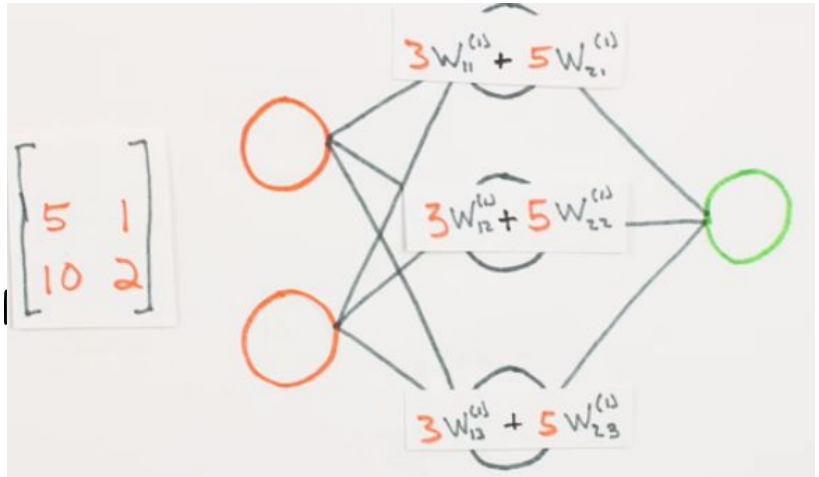
Initialize network with random weights

While [not converged]

Do forward prop

Do backprop and determine chai

Update ALL weights in ALL layers



Multi-Neuron Networks :: training

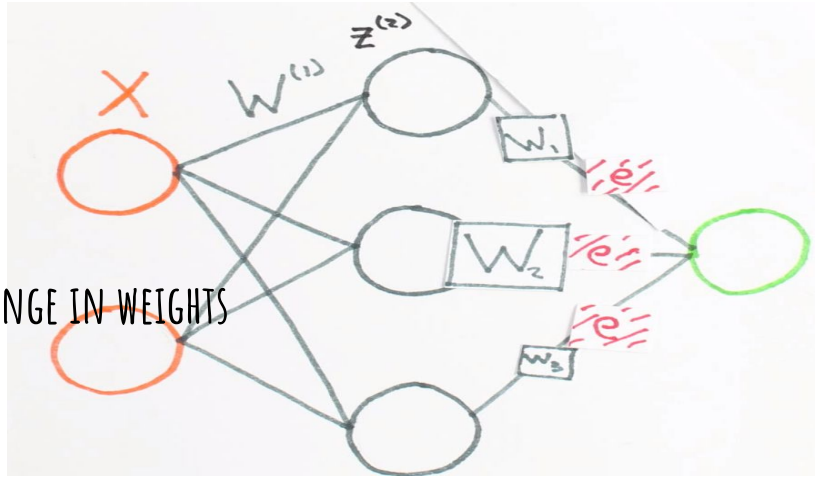
Initialize network with random weights

While [not converged]

Do forward prop

Do backprop and determine change in weights

Update ALL weights in ALL layers



Multi-Neuron Networks :: training

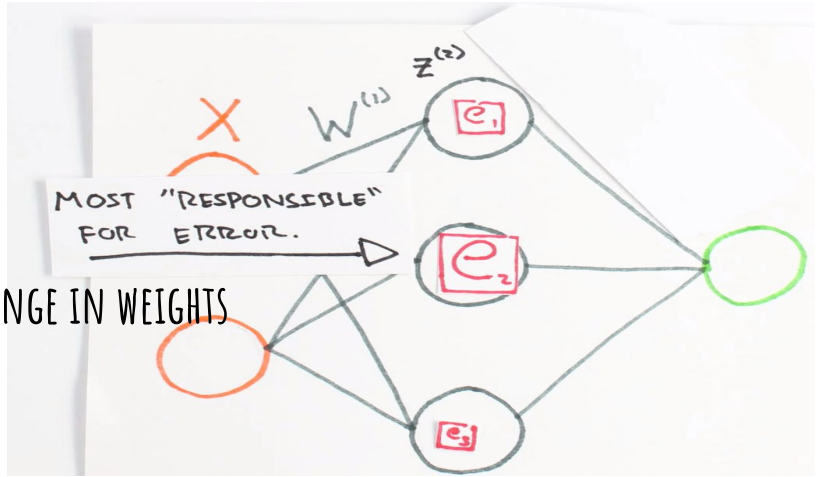
Initialize network with random weights

While [not converged]

Do forward prop

Do backprop and determine change in weights

Update ALL weights in ALL layers



Multi-Neuron Networks :: training

Initialize network with random weights

While [not converged]

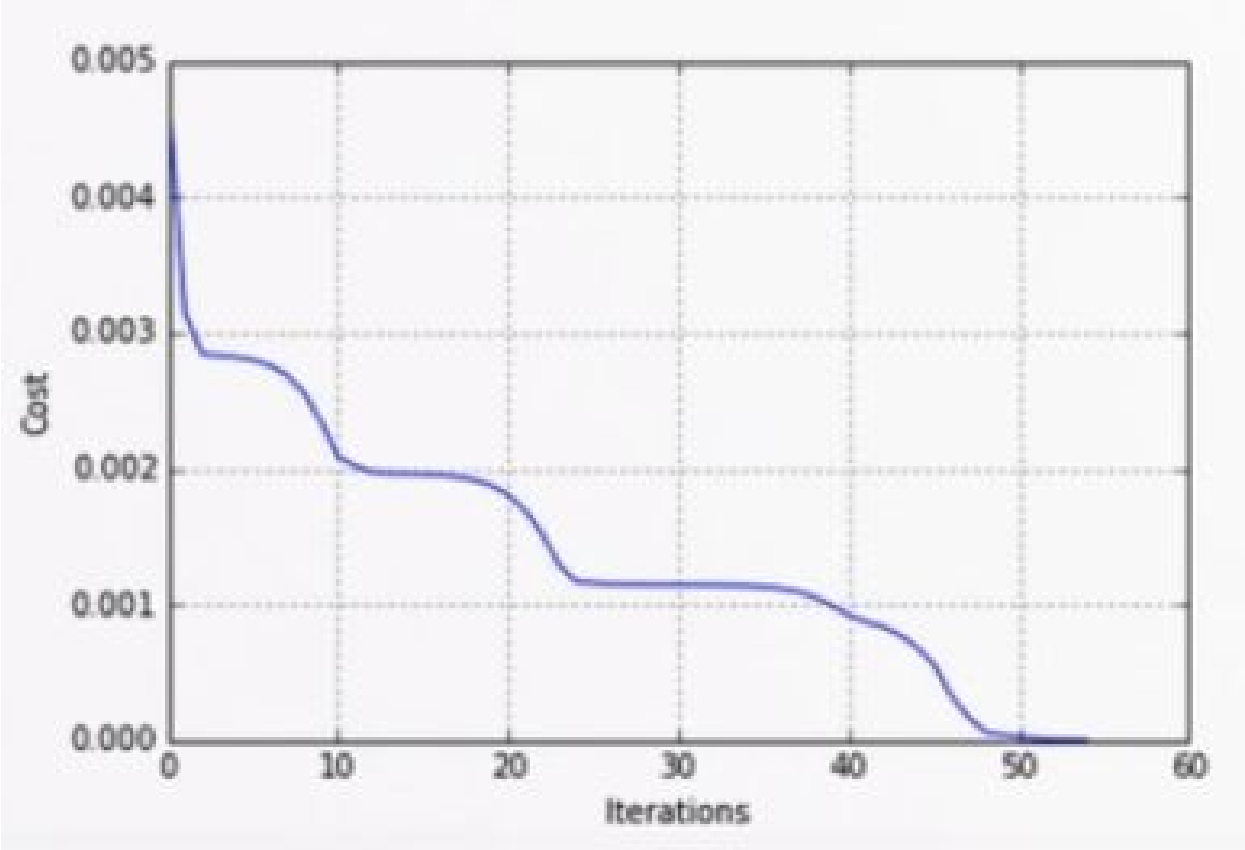
Do forward prop

Do backprop and determine change in weights

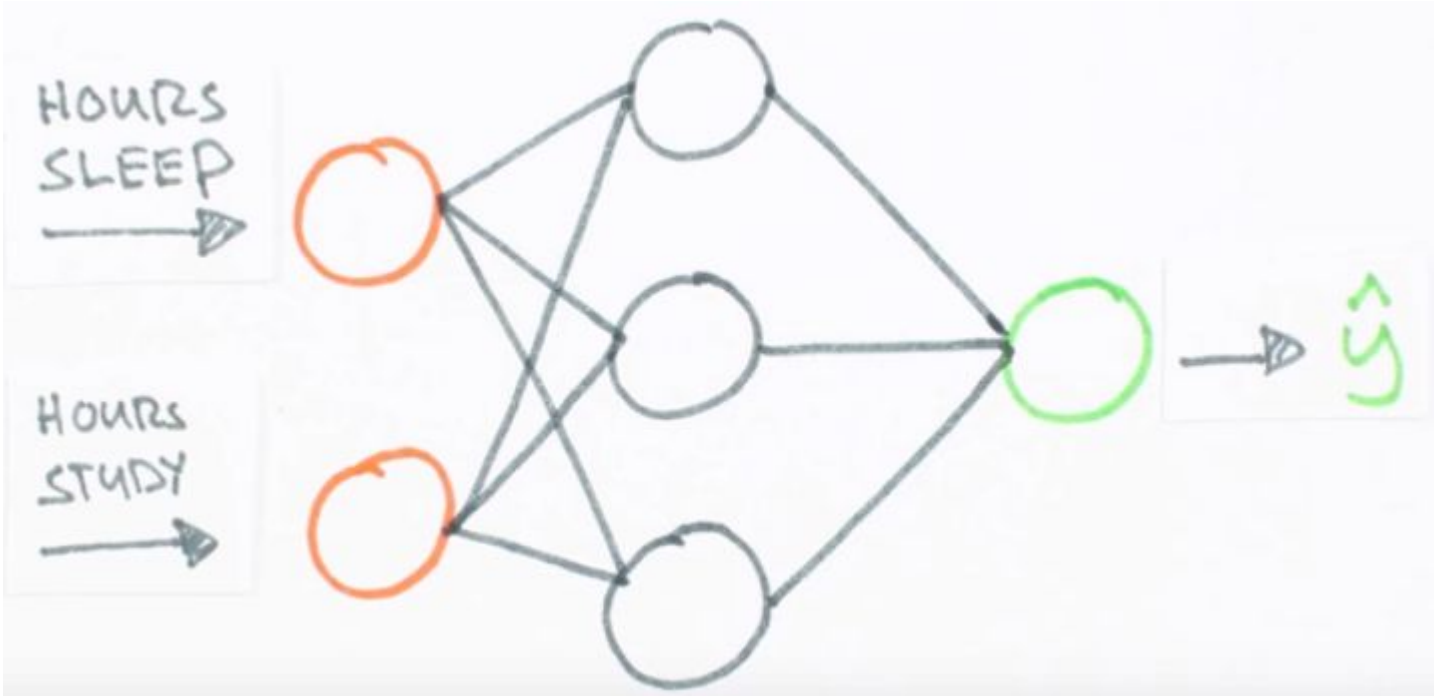
Update All weights in All layers

One Iteration

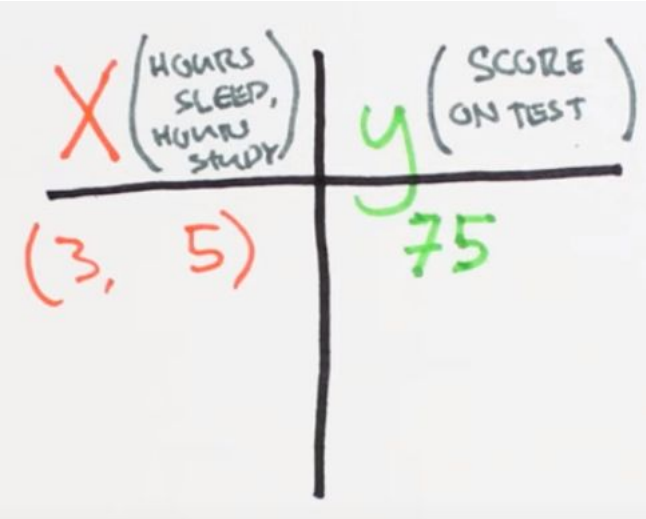
Multi-Neuron Networks :: training



Multi-Neuron Networks :: testing

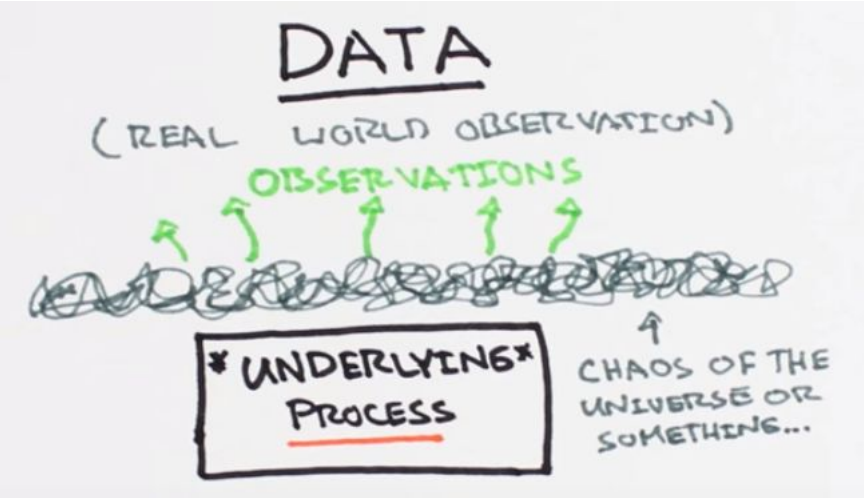


ALL IZZ WELL ???

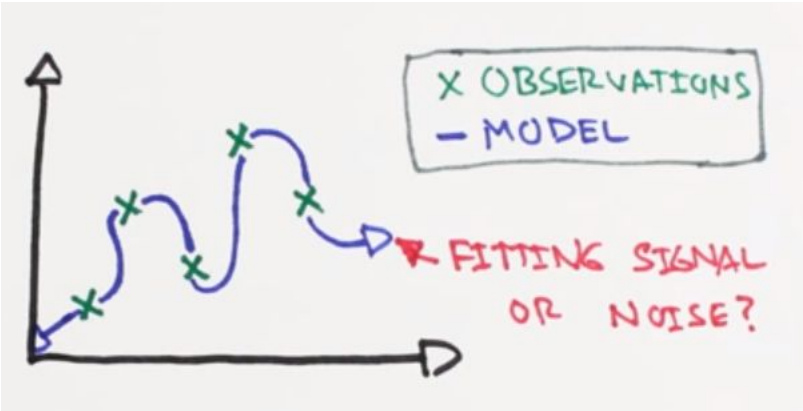


Demo ???

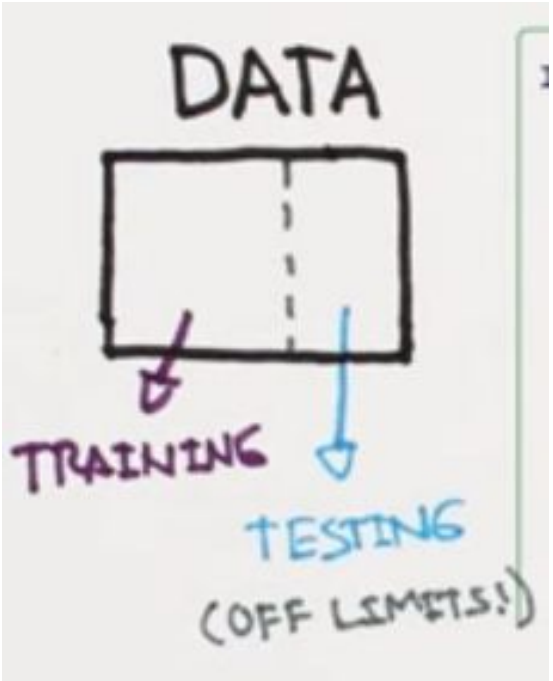
General Situation



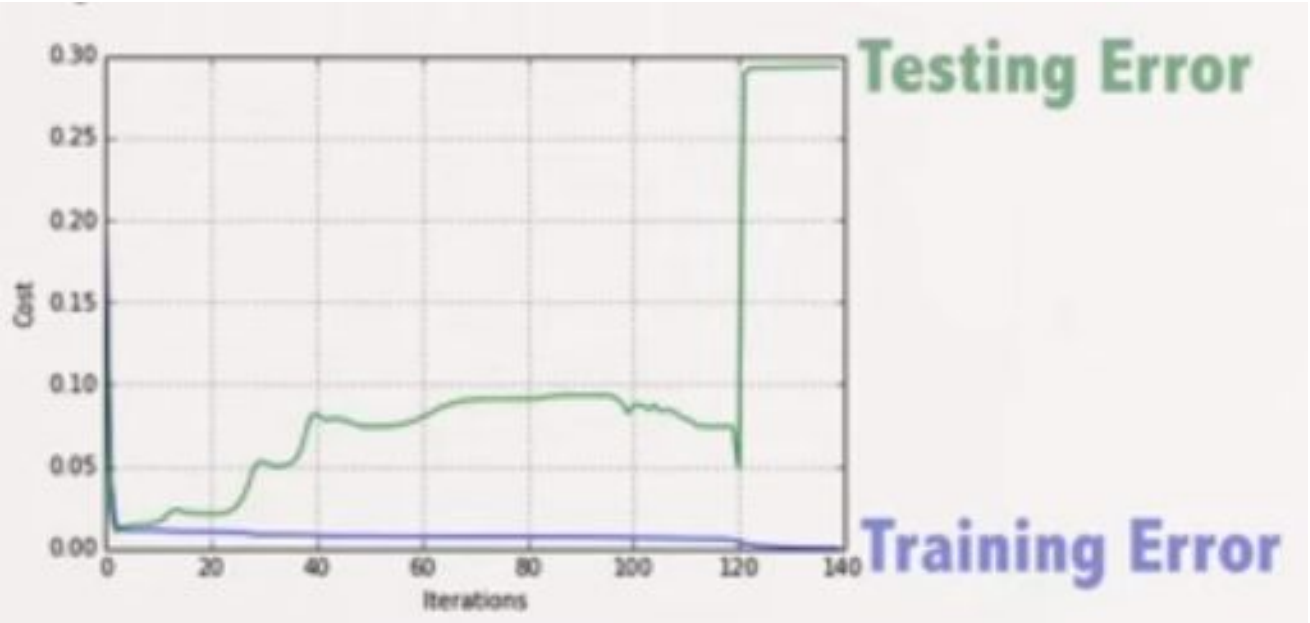
Observation = Signal + Noise



How to Avoid Overfitting ?



How To Avoid Overfitting ?

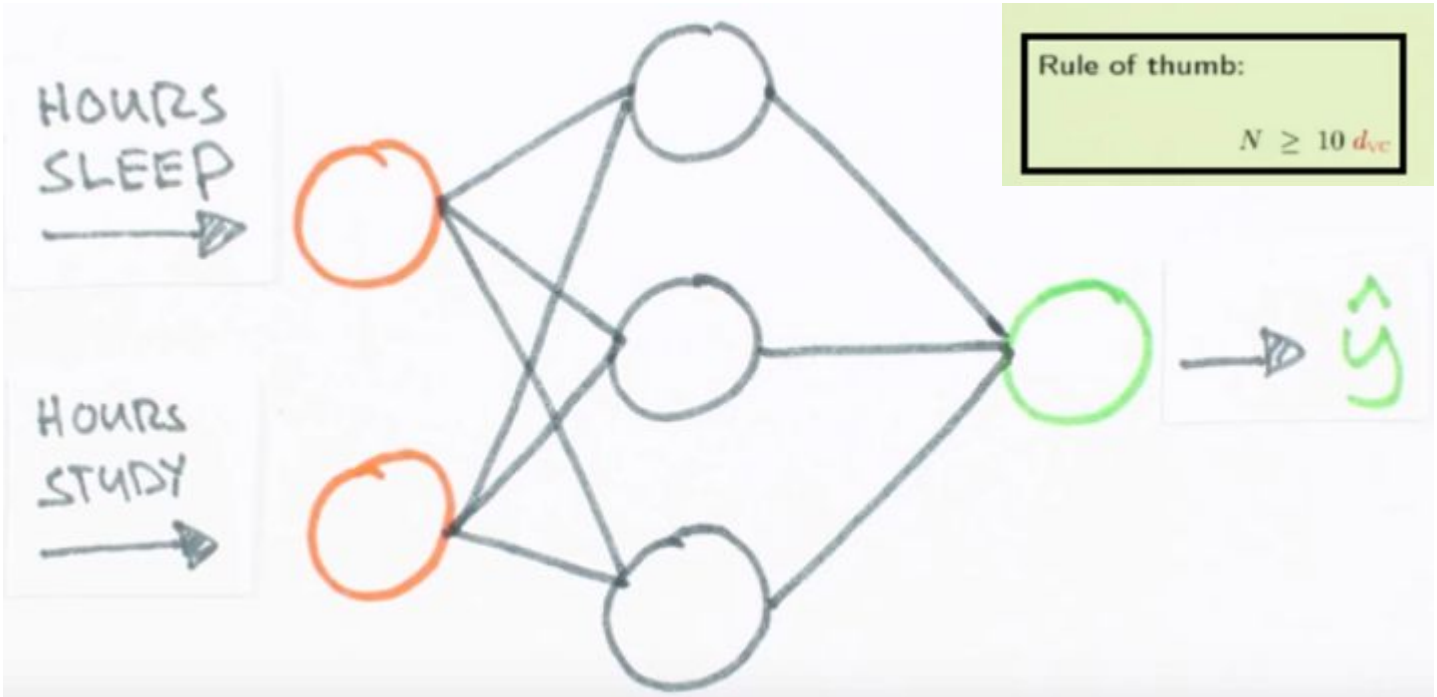


How Much Data Do we Need ?

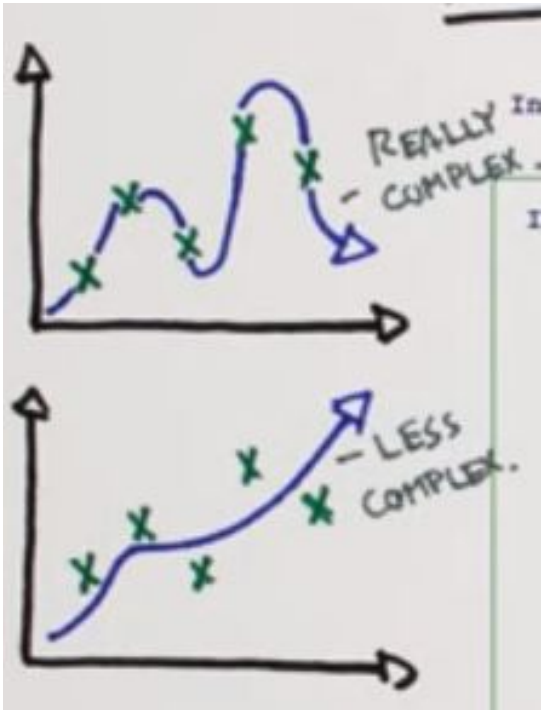
Rule of thumb:

$$N \geq 10\, d_{VC}$$

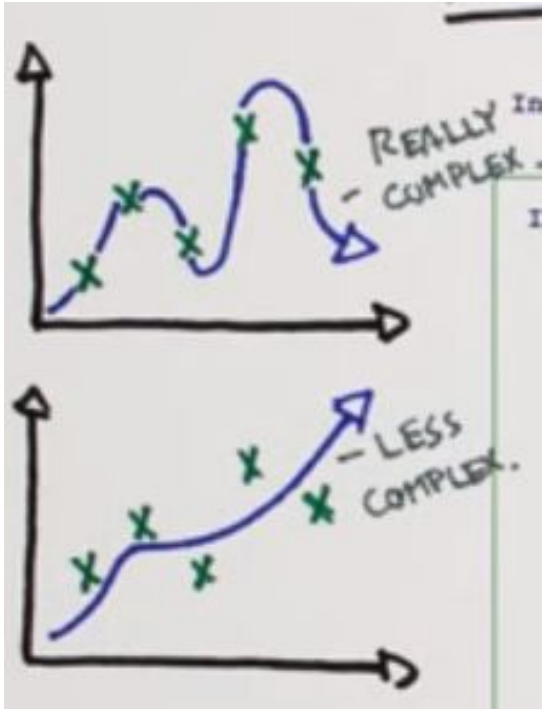
How Much Data Do we Need ?



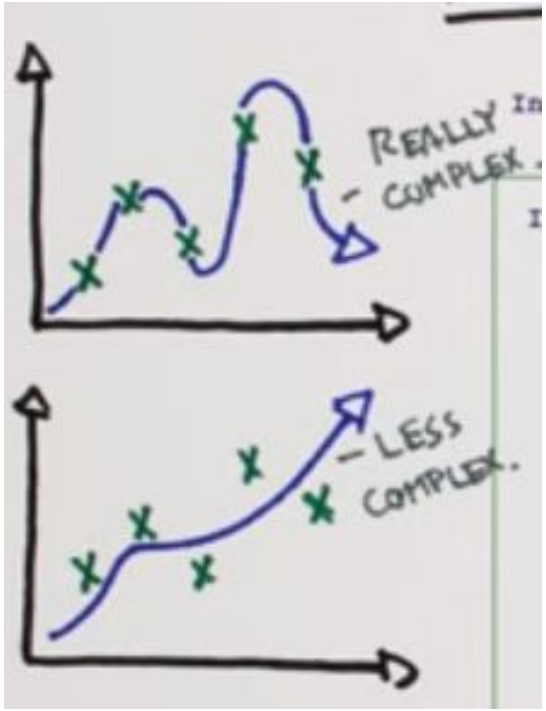
Avoiding Overfitting : Technique #2 : Regularization



Avoiding Overfitting : Technique #2 : Regularization

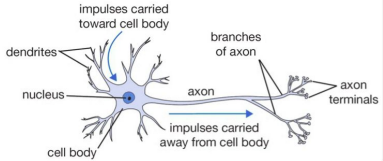


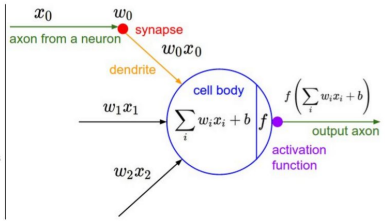
Avoiding Overfitting : Technique #2 : Regularization

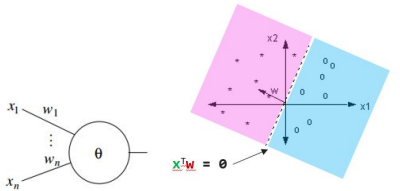


$$J = \sum \frac{1}{2}(y - f(\, f(\, XW^{(1)}\,)\, W^{(2)}\,)\,)^2 + ||W||^2$$

Neural Networks



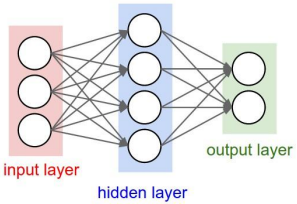


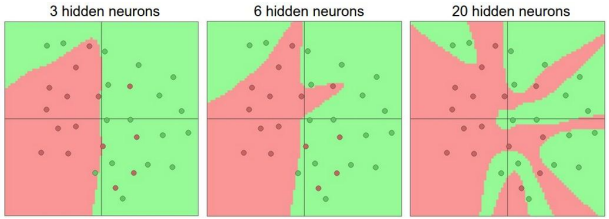


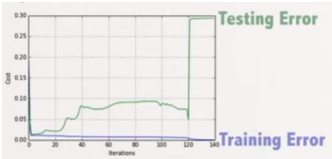
Linearly Separable Classes

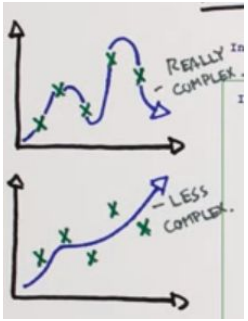
$x^T w > 0$ => Class 1

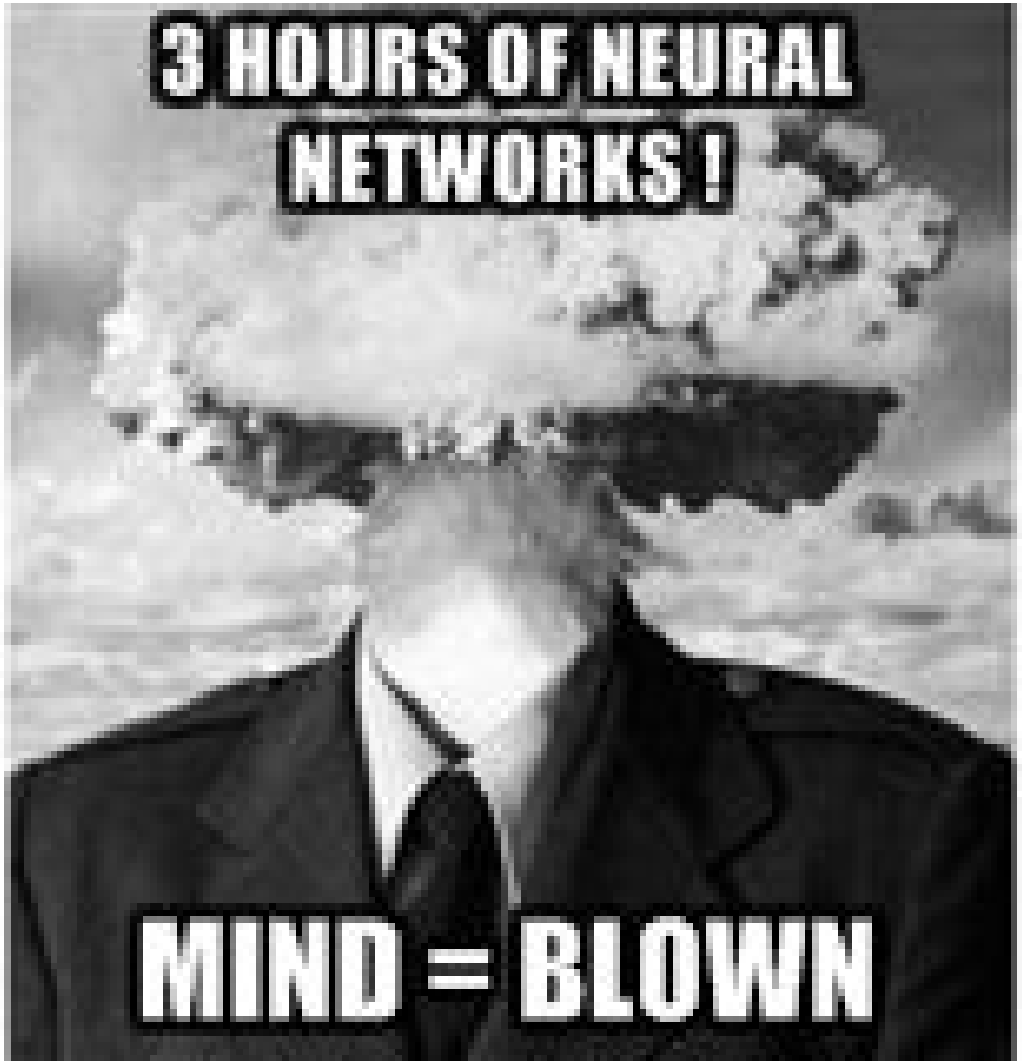
$x^T w < 0$ => Class 2










3 HOURS OF NEURAL NETWORKS !
MIND = BLOWN

References

- Neural network diagrams https://github.com/battlesnake/neural
- ‘Neural Networks Demystified’ - YouTube channel
- Andrej Karpathy + Stanford + Neural Networks
- http://neuralnetworksanddeeplearning.com/