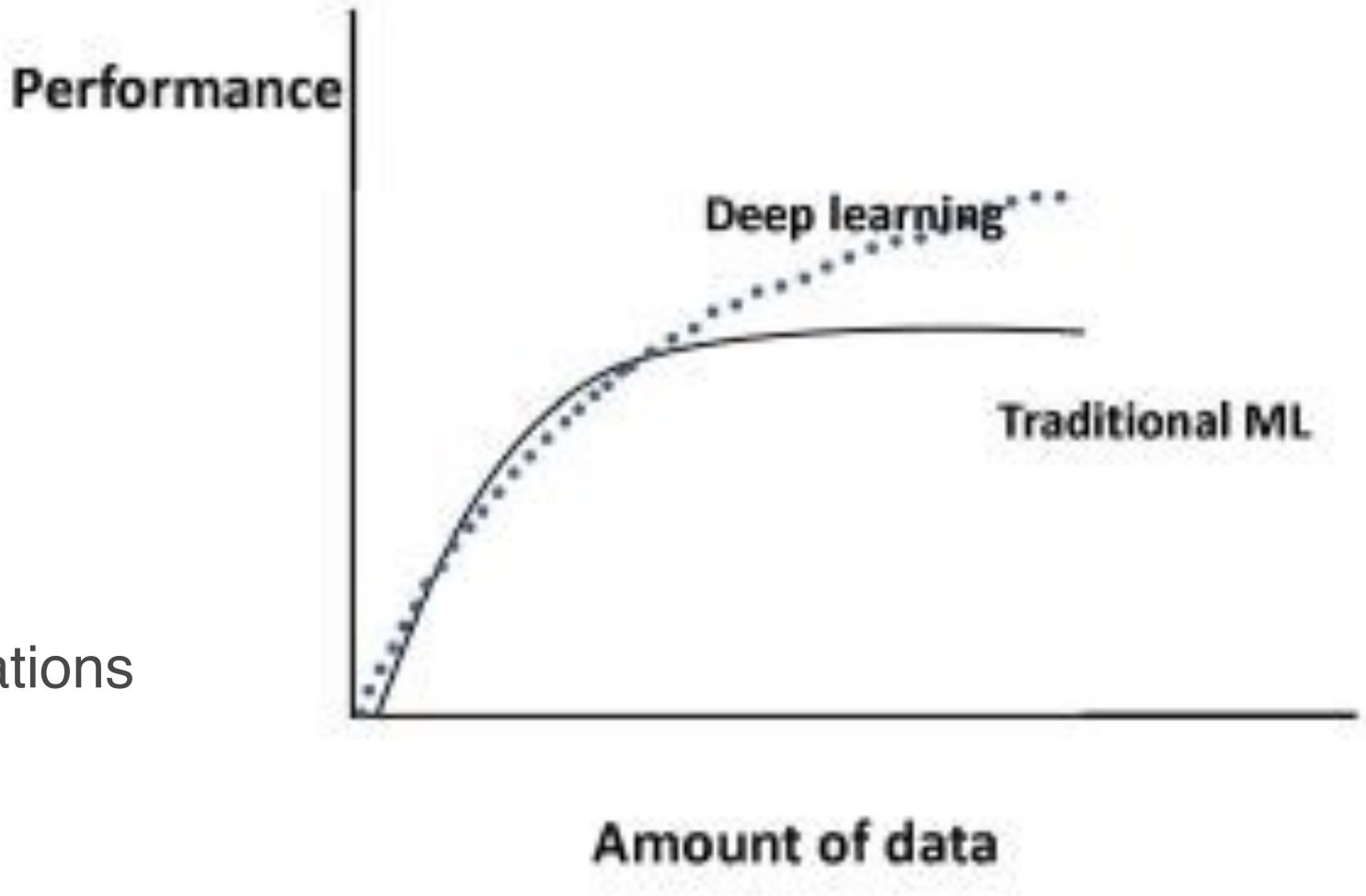
Artificial Neural Networks

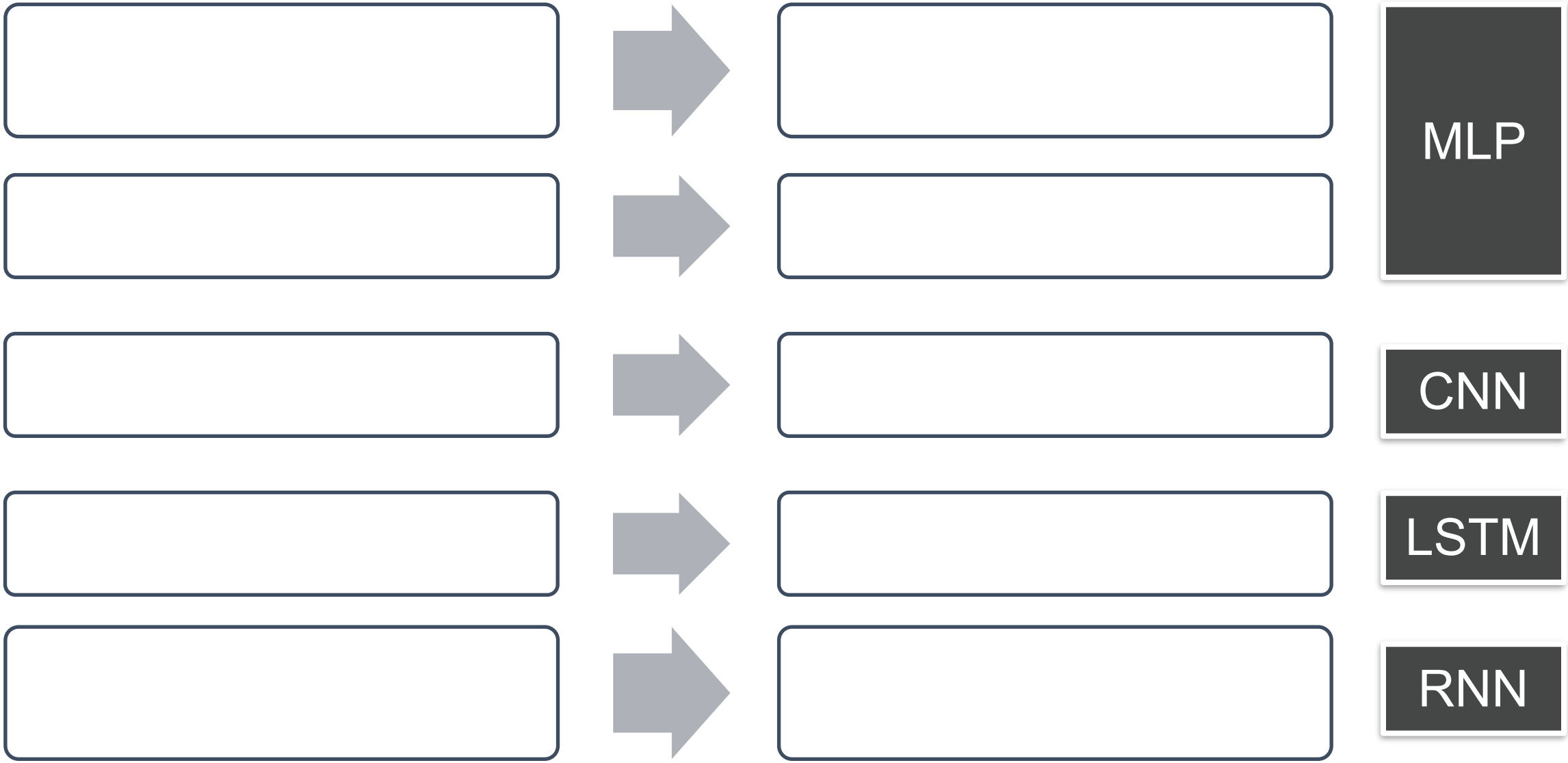
Parag Mantri, PhD

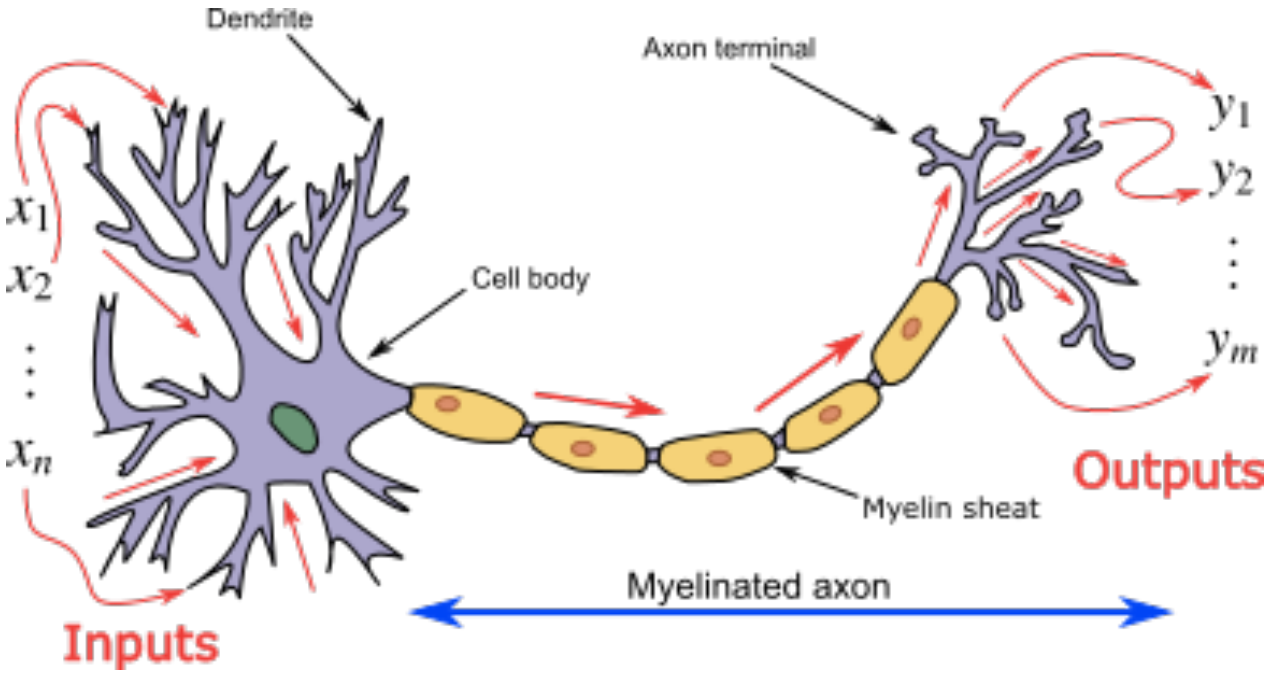
Neural Networks

- Basic idea behind ANN and why we need it now
- Understanding of single neuron or perceptron
- XOR example of why linear classification does not work
- Intuition behind need for activation functions and types. (RELU, Sigmoid, etc)
- Gradient descent review
- Equations of loss and cost function.
- Basic idea of forward and backward propagation
- Mathematical details behind backward propagation

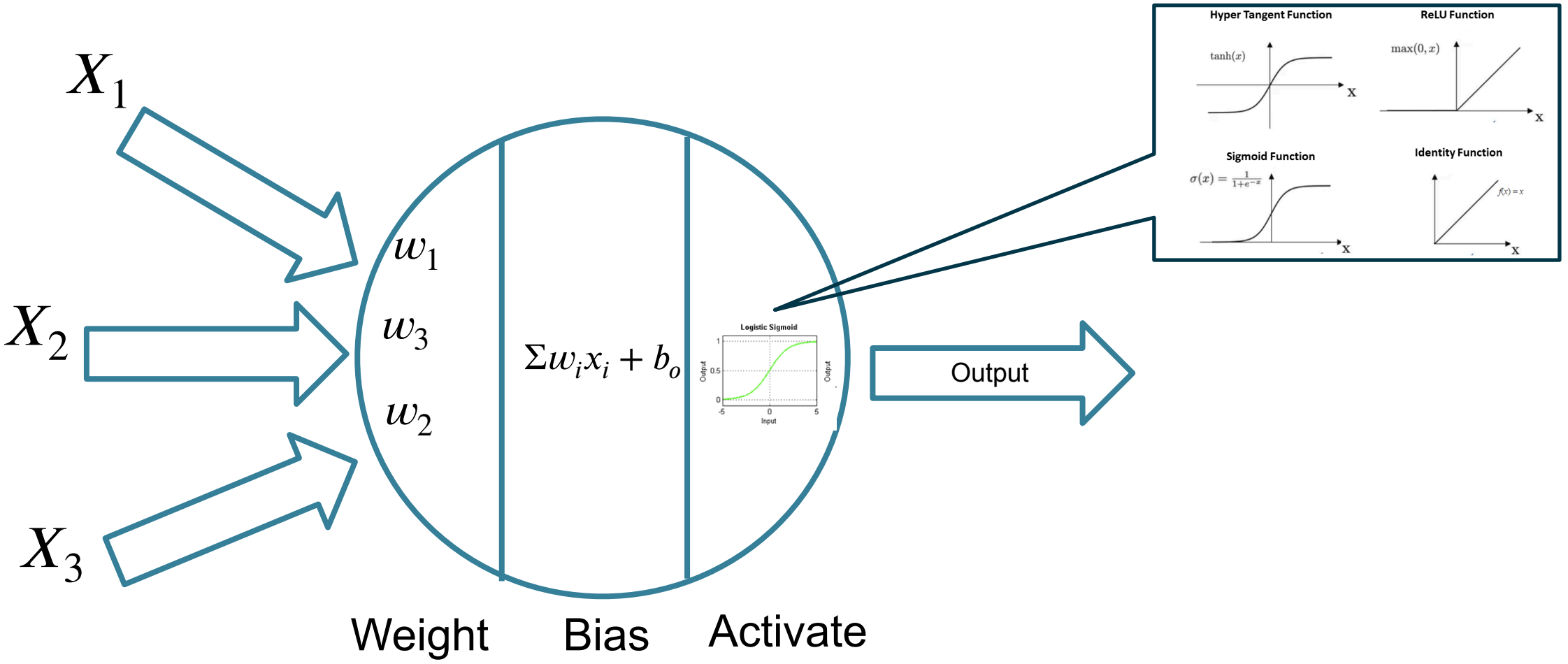
- Data
  - Cell phones
  - Sensors
  - Camera
  - Wearables
  - Digitalization
- Computer Power
  - Smaller and faster
- Algorithms
  - Problem Specific innovations



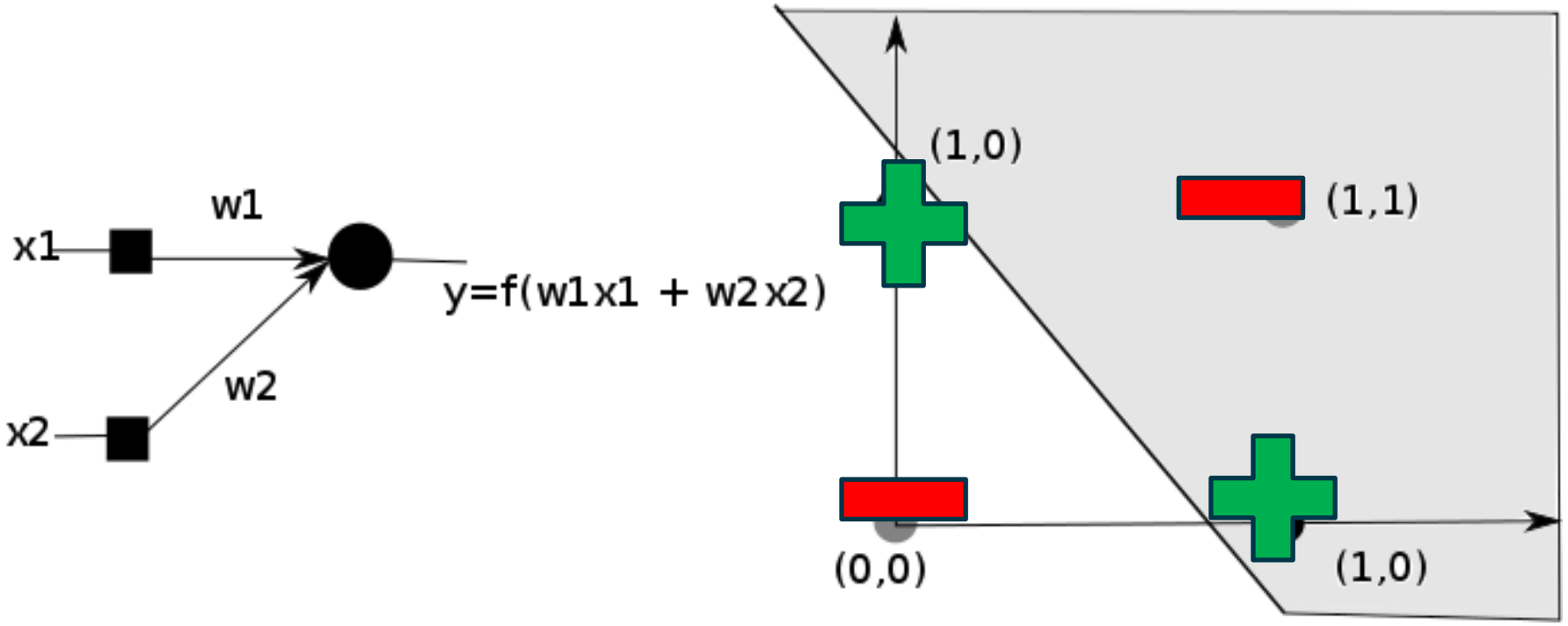

MLP
CNN
LSTM
RNN

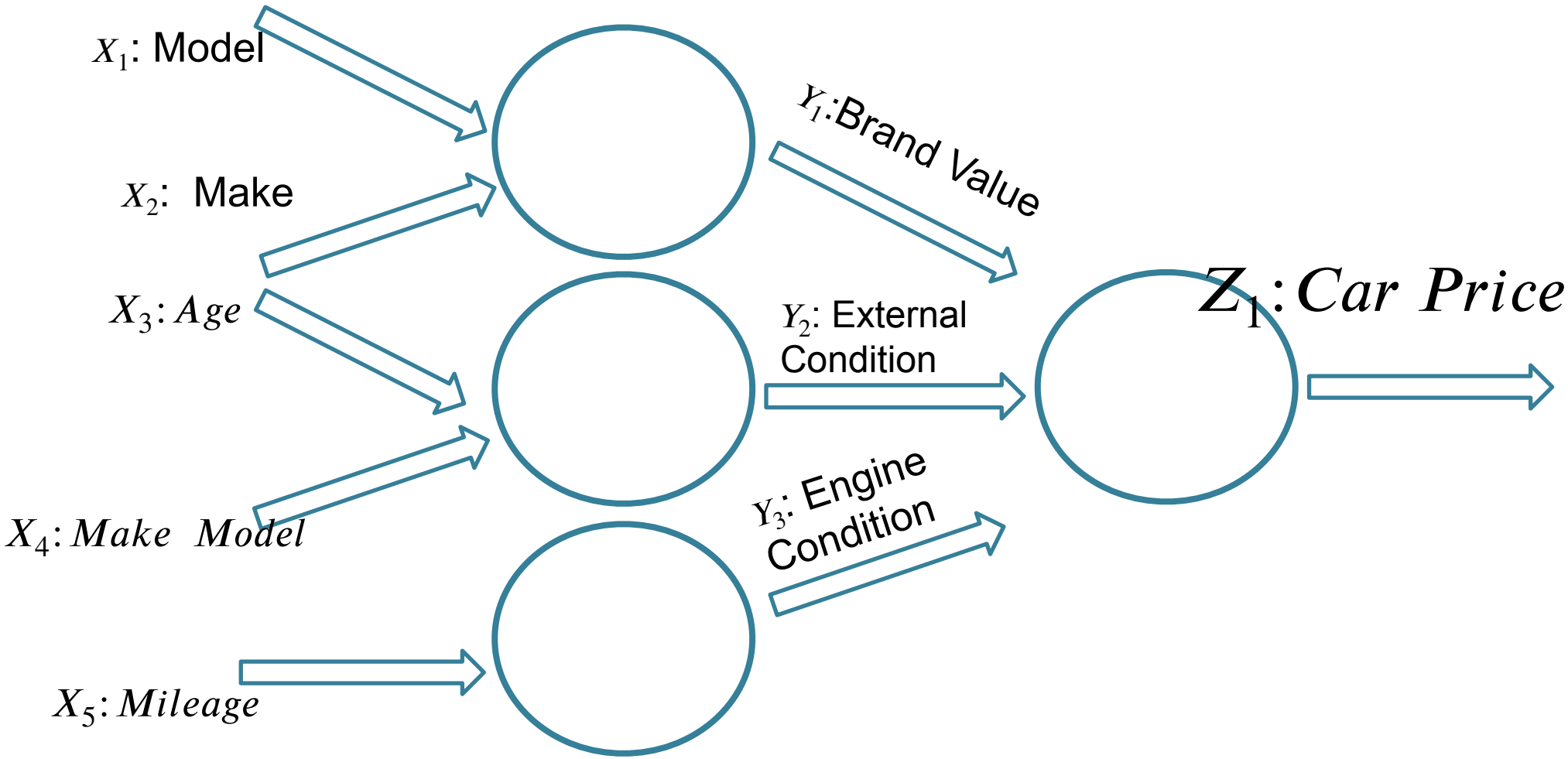


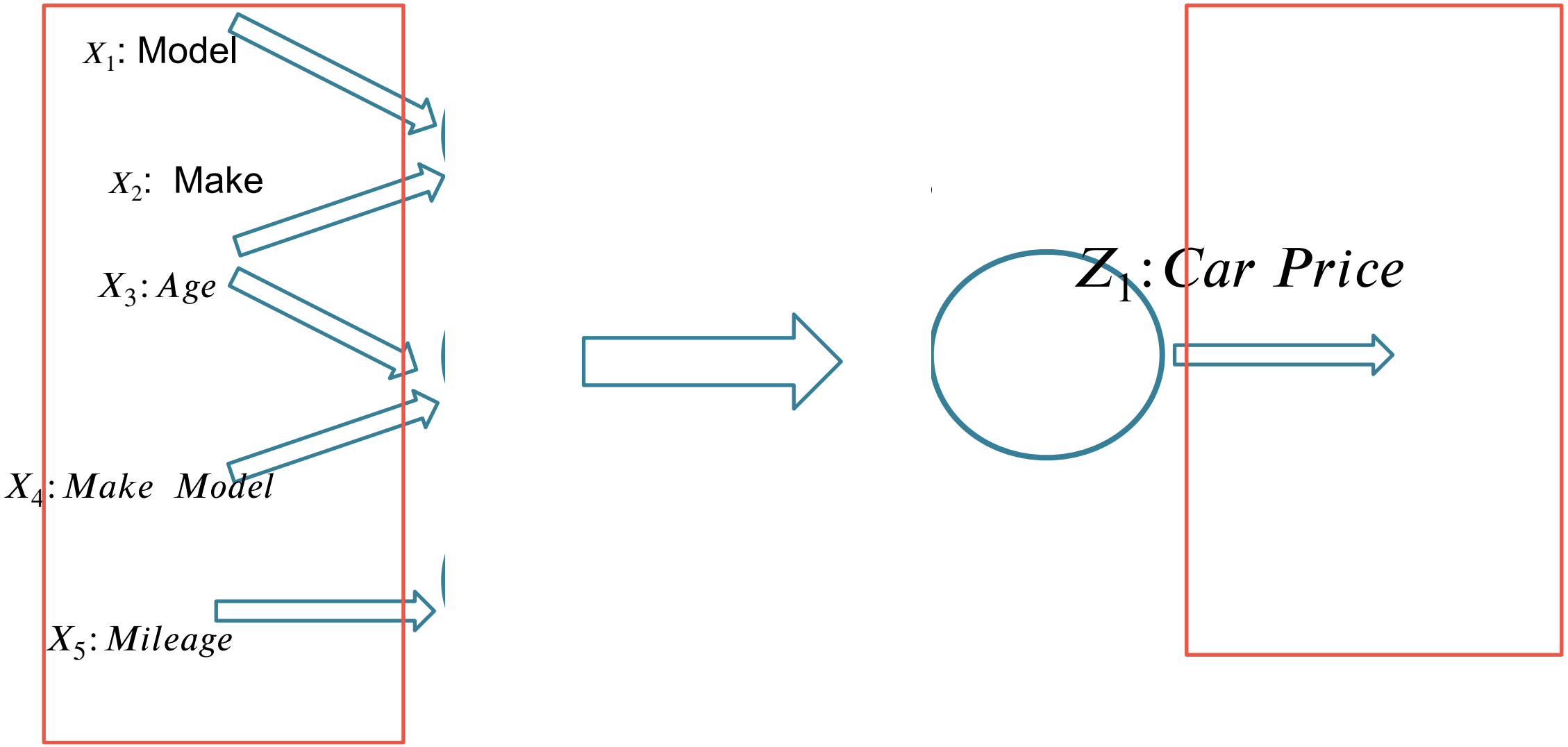
Goal	Understand the brain
	Emulate the Brain
ANN	Did we achieve the goal?


X_1
X_2
X_3
w_1
w_3
w_2
$\Sigma w_i x_i + b_o$
Logistic Sigmoid
Output
Input
Weight
Bias
Activate
Hyper Tangent Function
$\tanh(x)$
ReLU Function
$\max(0, x)$
Sigmoid Function
$\sigma(x) = \frac{1}{1+e^{-x}}$
Identity Function
$f(x) = x$
x

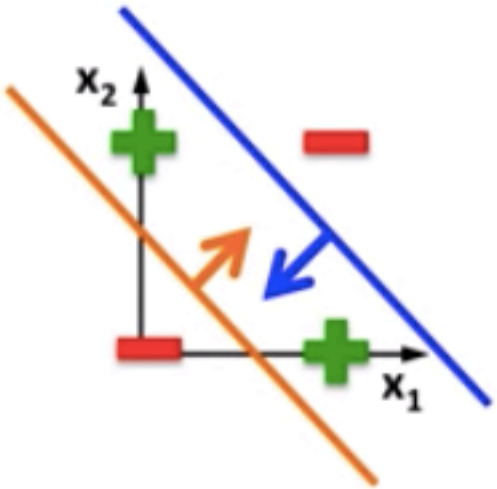
x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	0

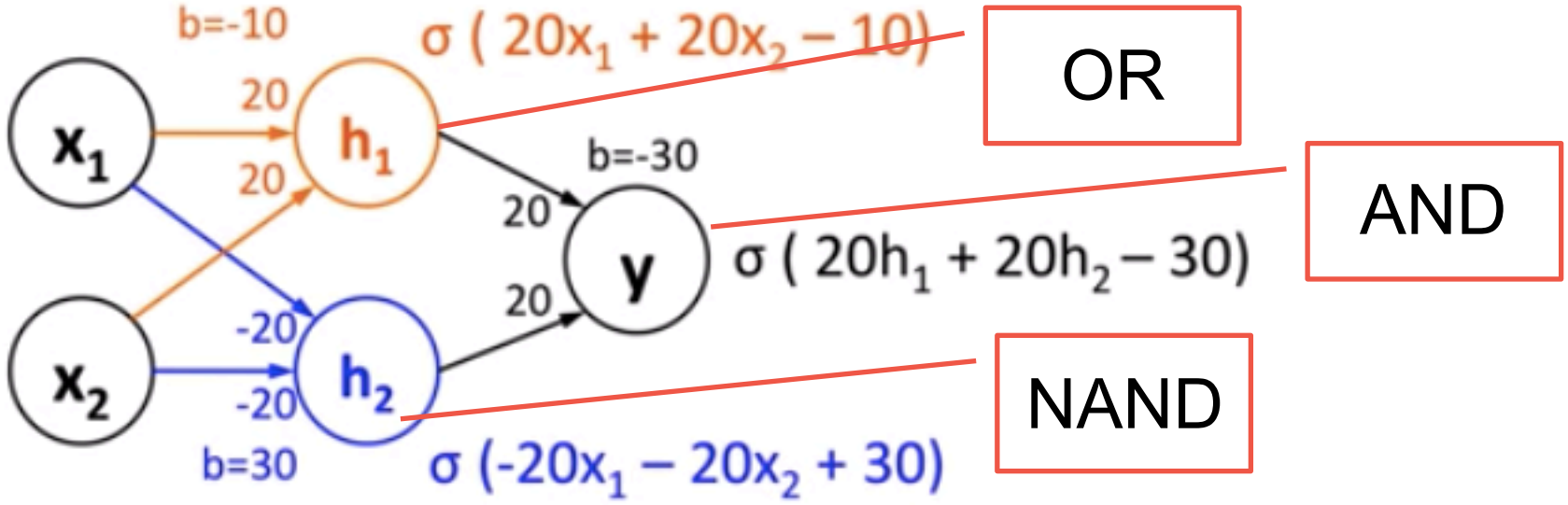



X_1: Model
X_2: Make
X_3: Age
X_4: Make Model
X_5: Mileage
Y_1: Brand Value
Y_2: External Condition
Y_3: Engine Condition
Z_1: Car Price


X_1: Model
X_2: Make
X_3: Age
X_4: Make Model
X_5: Mileage
Z_1: Car Price

Linear classifiers cannot solve this

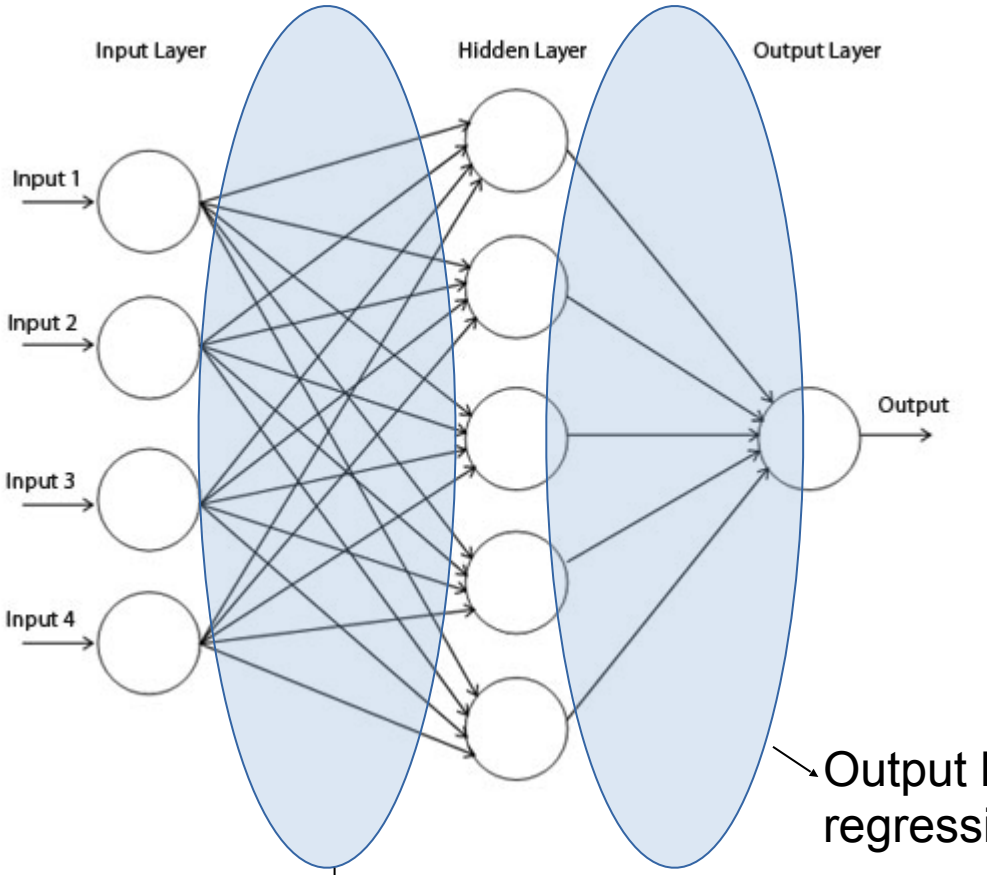




σ(20*0 + 20*0 – 10) ≈ 0	σ (-20*0 – 20*0 + 30) ≈ 1	σ (20*0 + 20*1 – 30) ≈ 0
σ(20*1 + 20*1 – 10) ≈ 1	σ (-20*1 – 20*1 + 30) ≈ 0	σ (20*1 + 20*0 – 30) ≈ 0
σ(20*0 + 20*1 – 10) ≈ 1	σ (-20*0 – 20*1 + 30) ≈ 1	σ (20*1 + 20*1 – 30) ≈ 1
σ(20*1 + 20*0 – 10) ≈ 1	σ (-20*1 – 20*0 + 30) ≈ 1	σ (20*1 + 20*1 – 30) ≈ 1

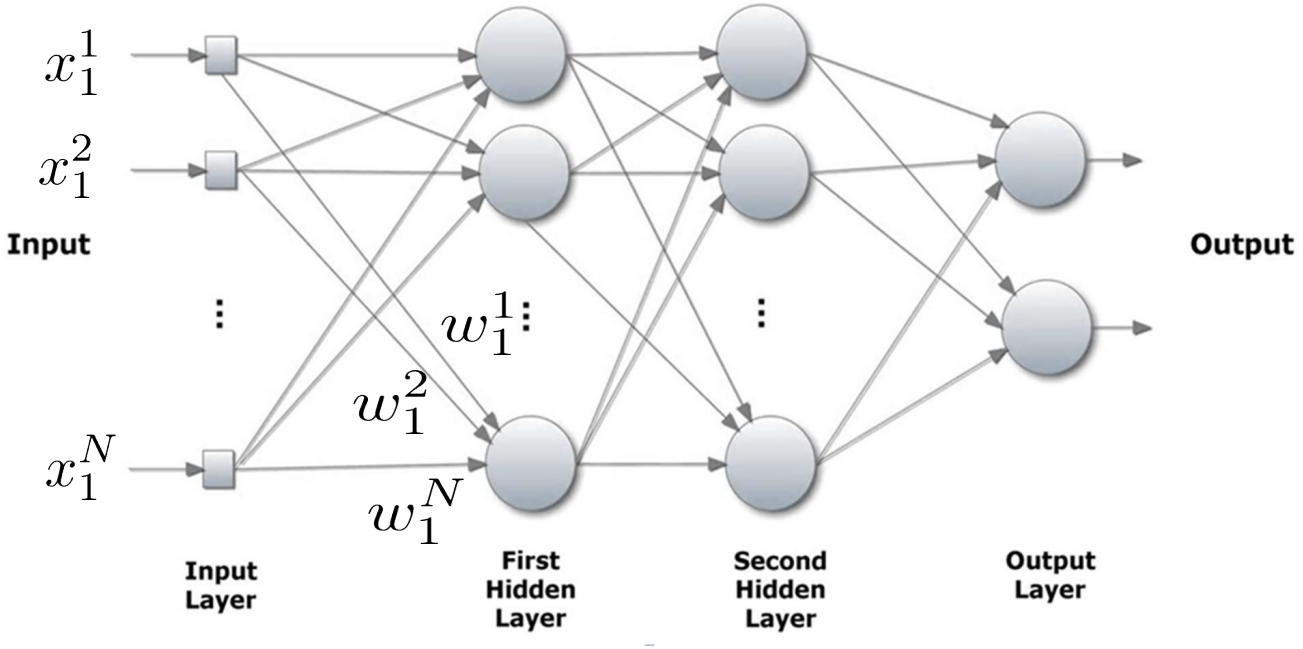
Each layer is a transformation into new space

Ref: Neural Networks Video by Victor Lavrenko



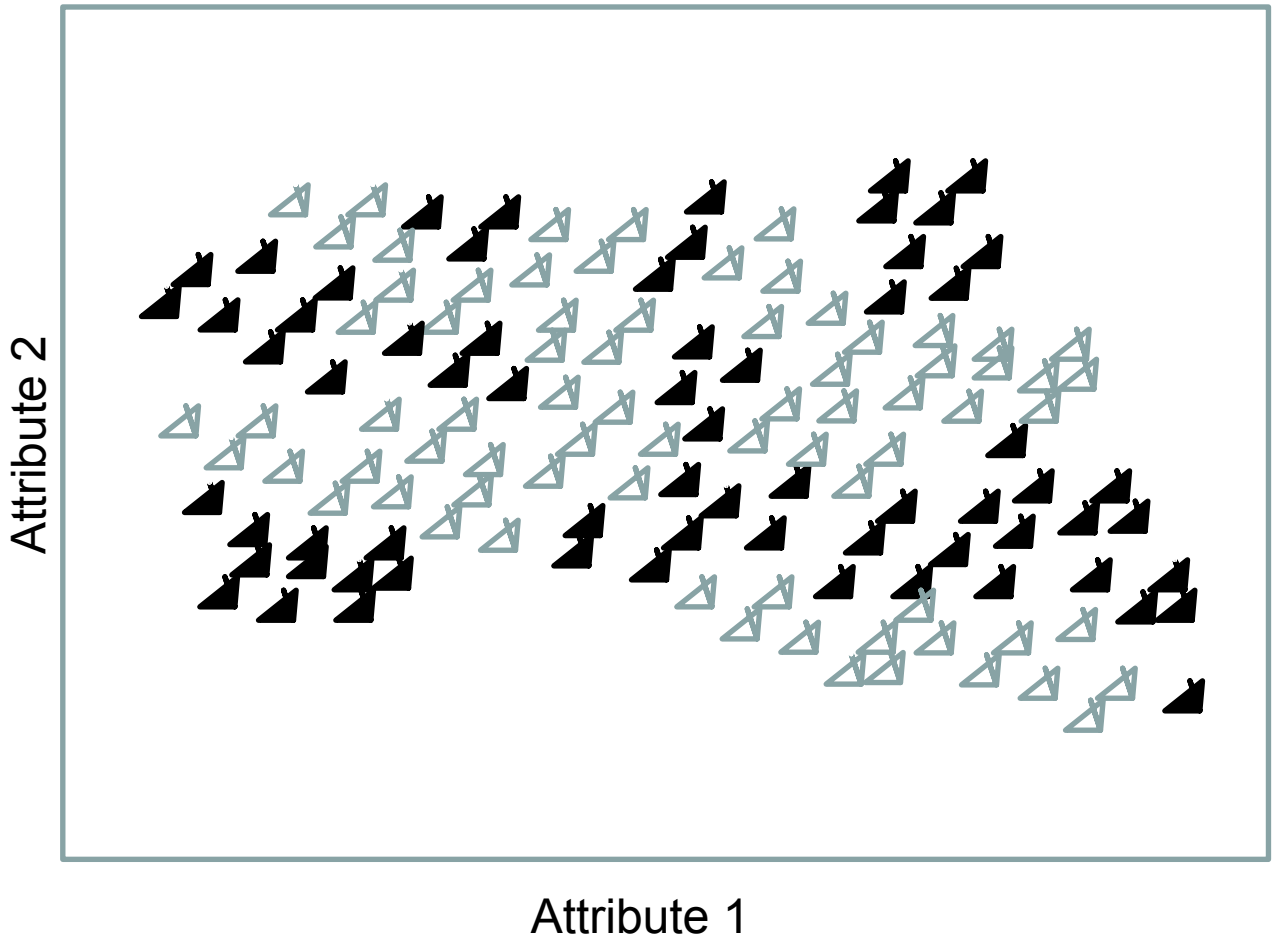
Output layer is a simple regression/classification layer

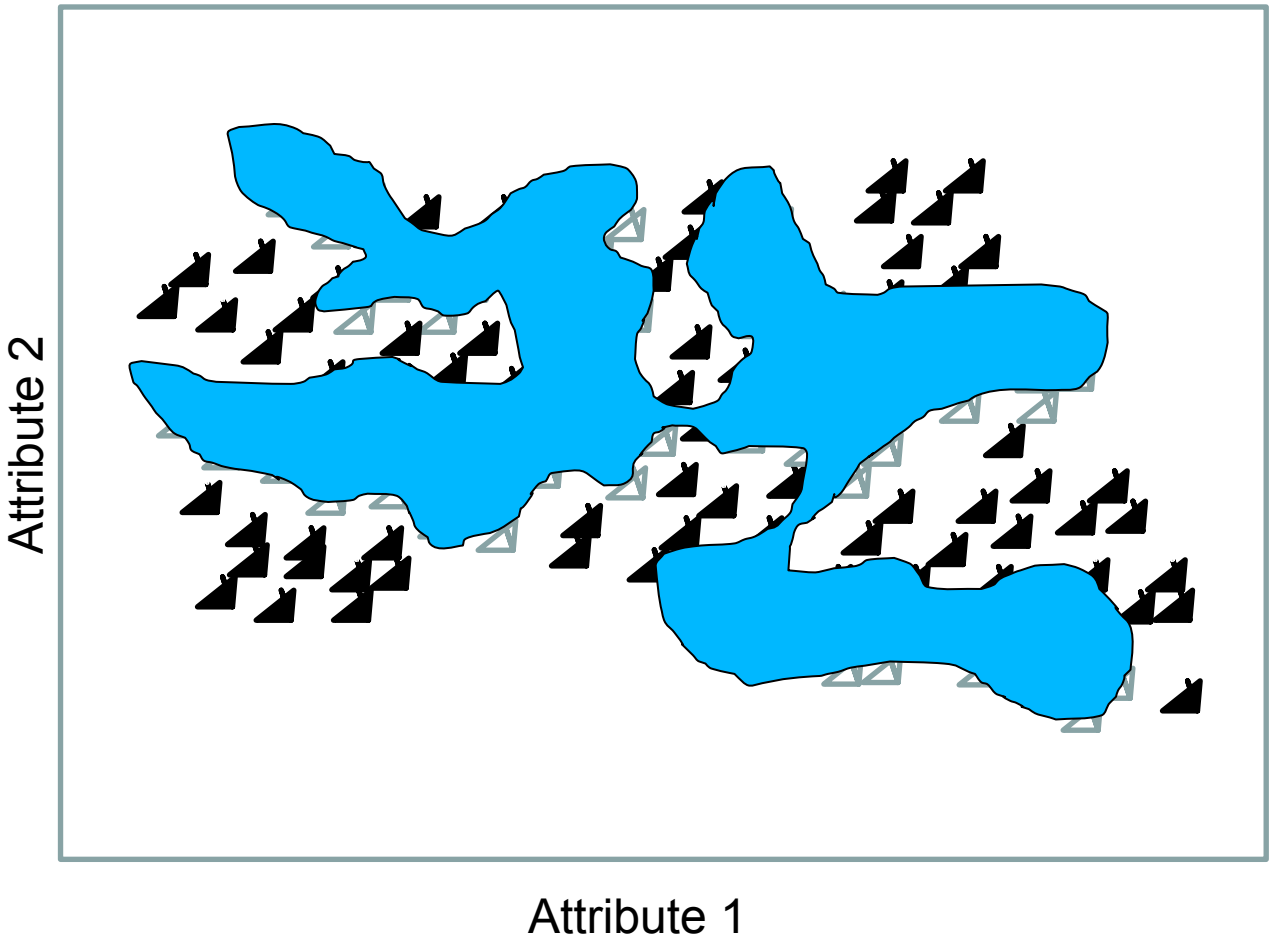
All the connections represent a Matrix W

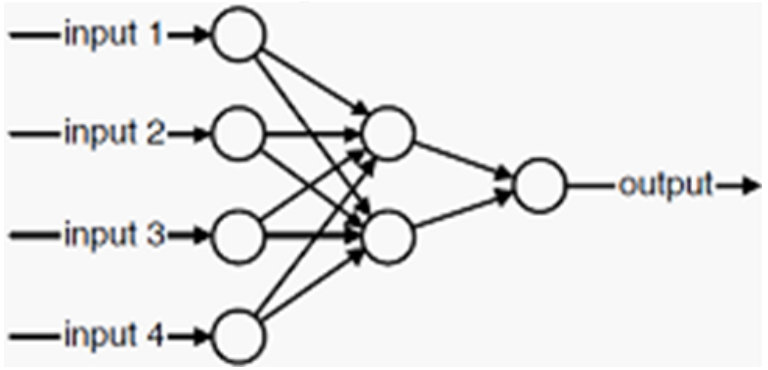


A multilayer perceptron

- Feed forward network
- Fully connected


Attribute 2
Attribute 1


Attribute 2
Attribute 1


input 1
input 2
input 3
input 4
output

- 90%: One hidden layer
- 10%: Two hidden layers

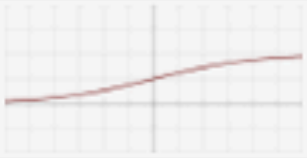
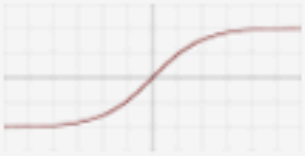
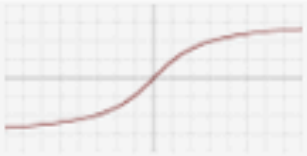
- Rarely more than that! It might lead to overfitting!

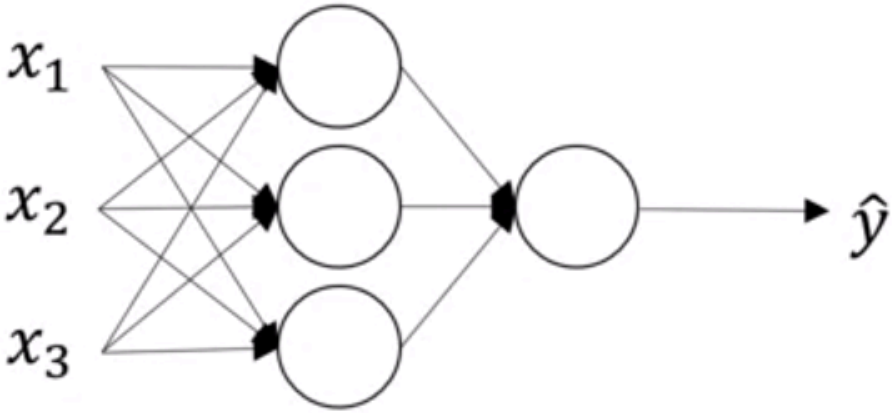
- Most structured-data problems can be handled with at-most 2 layers. Rare are problems where higher layers can be justified.

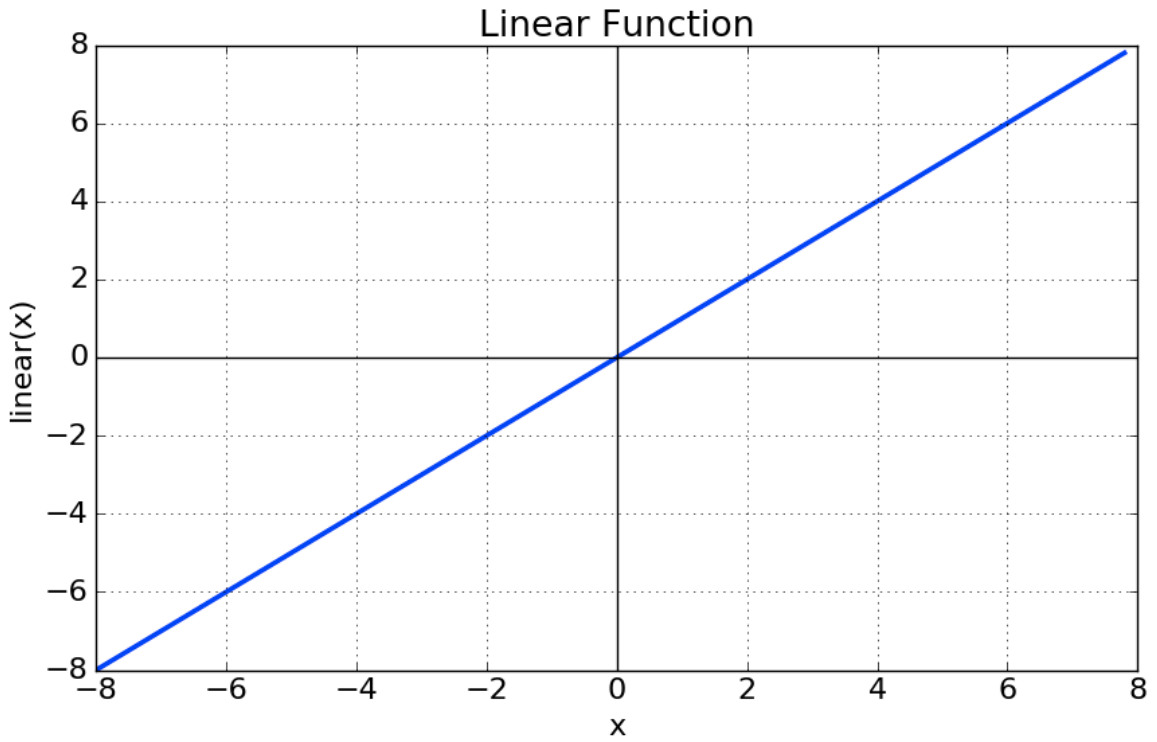
- “rules of thumb” exist but are absolute rules. Experiment and validate and then cross validate like any other hyper parameter

See ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hu

Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
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		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$





Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = g^{[1]}(z^{[1]})$$

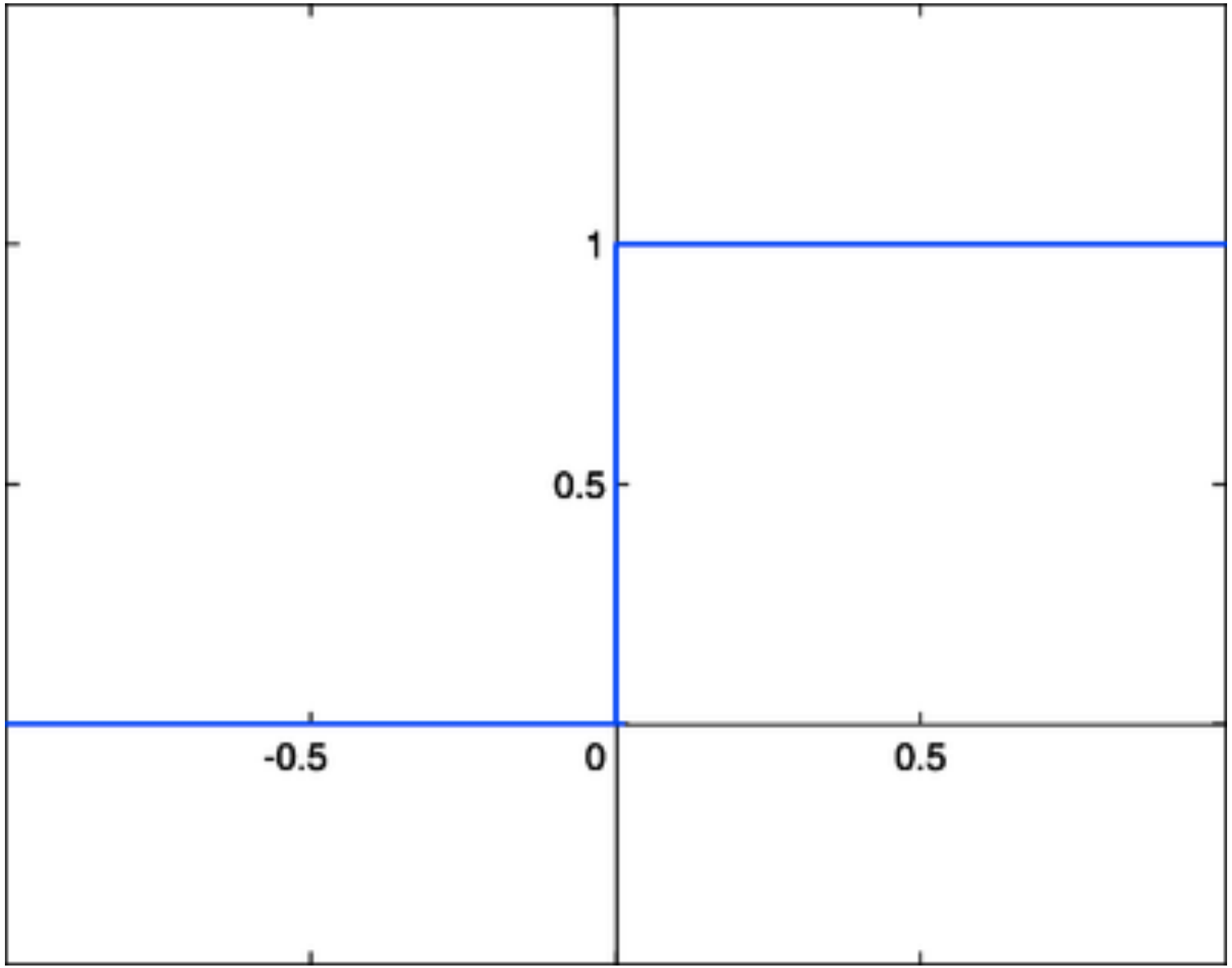
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]})$$

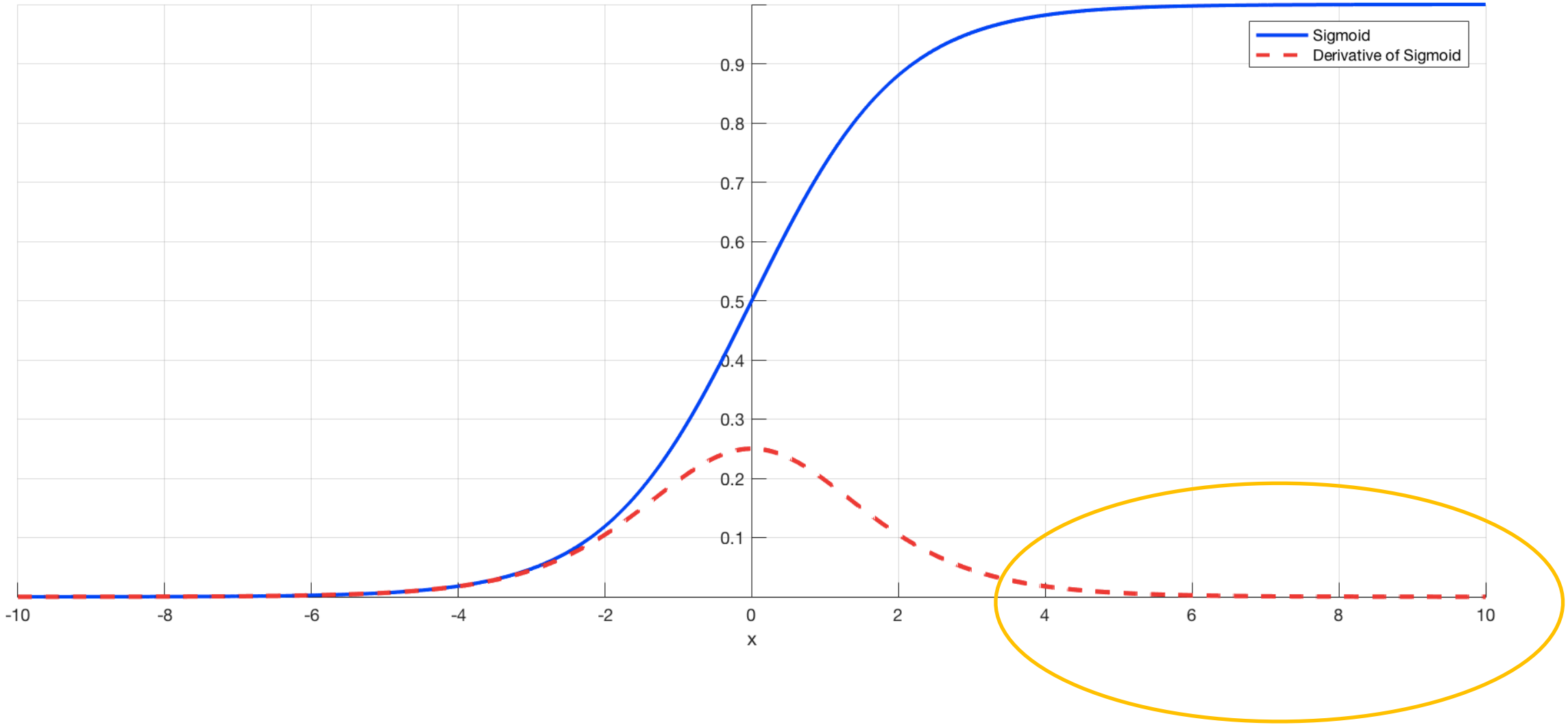
Activation Function

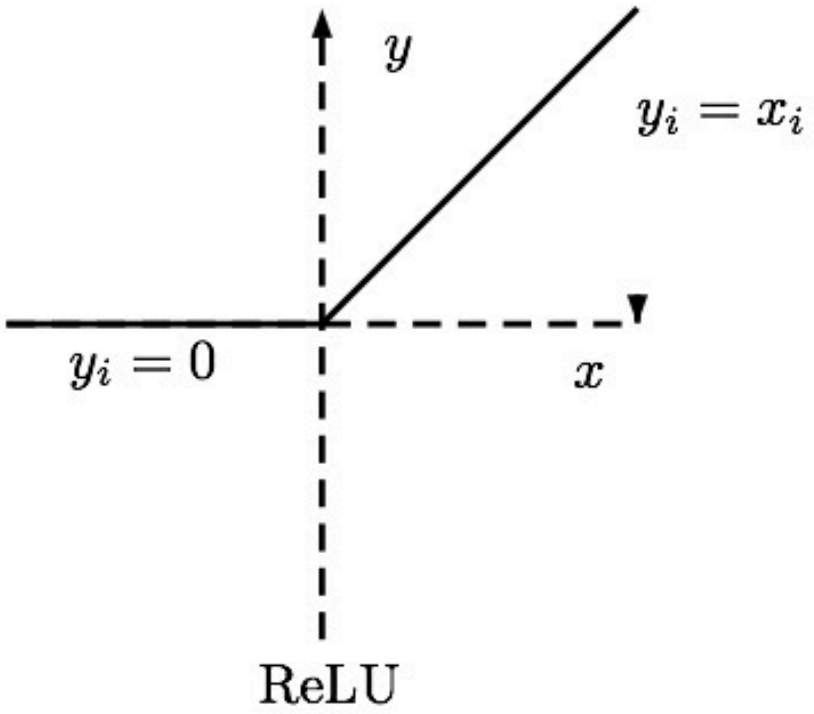
Q: What will be Z[2] if we have linear activation function ?

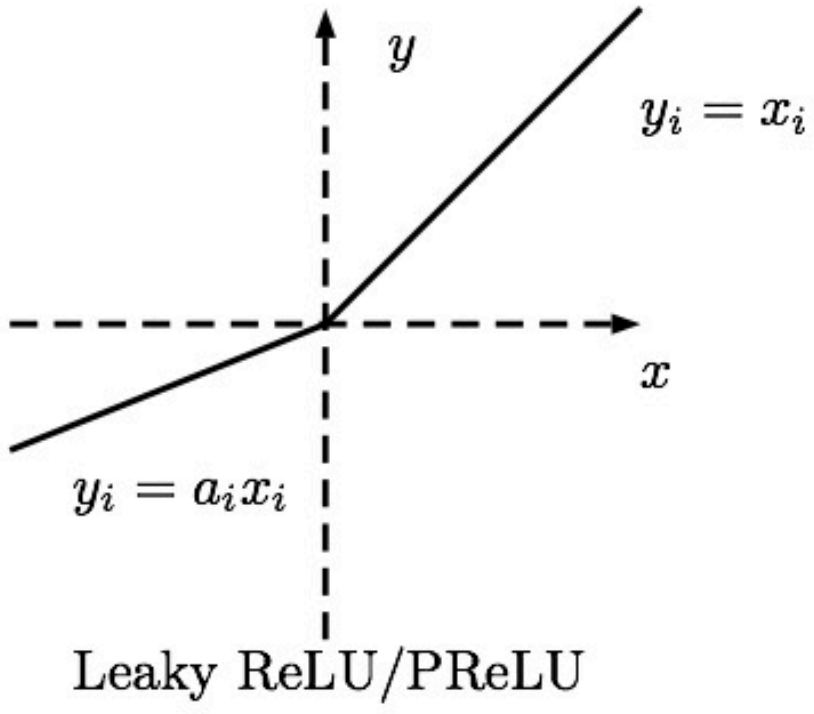
- Linear Activation Function Output
  - $W^{[2]}W^{[1]}X + W^{[2]}b^{[1]} + b^{[2]}$
- Linear activation function defeats the purpose of multiple layer
- i.e. All the layers can be expressed as single neuron



- Does not represent derivative learning.
- Suitable for output layer only


Sigmoid
Derivative of Sigmoid
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-10
-8
-6
-4
-2
0
2
4
6
8
10
x

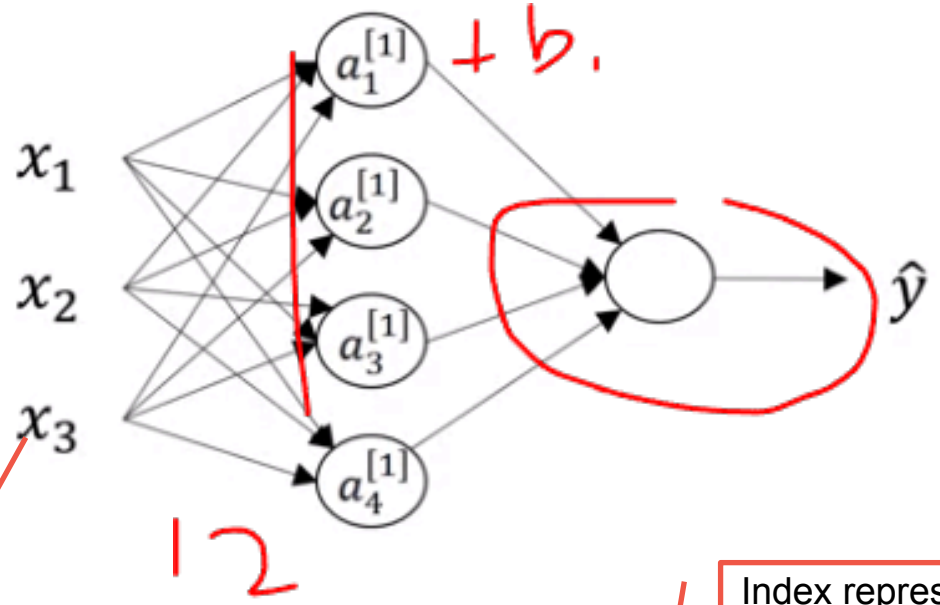




$$f(X) = \max((0,x))$$

Linear Algebra Convention

- n: Number of Features
- m: Number of training samples
- $X \epsilon \mathbb{R}^{n}$, $y \epsilon (0,1)$. (Classification Problem)
- Shape of X: n rows by m columns (What is the shape of a data frame?)
- Shape of y: 1 row by m columns



a[0]

Given input x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

Size 4 X 1

Size 4 X 1

Size 4 X 3

Size 3 X 1

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

$$W^{[1]} = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \\ W_{41} & W_{42} & W_{43} \end{bmatrix}$$

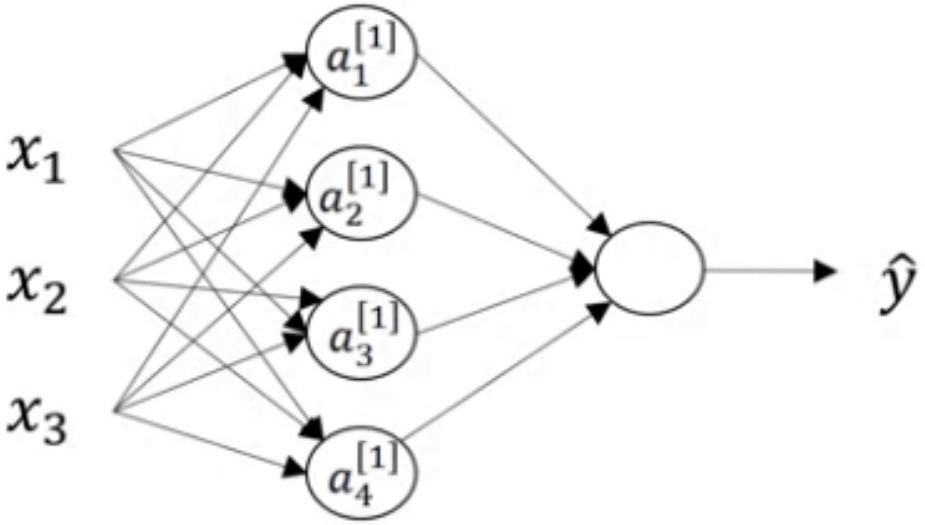
Index representing neuron in layer "From"

Index representing neuron in layer "To"

$$Z_1^{(2)}$$

Layer Number

Neuron Number



What will be the shape of all the elements in this layer ?

Given input x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

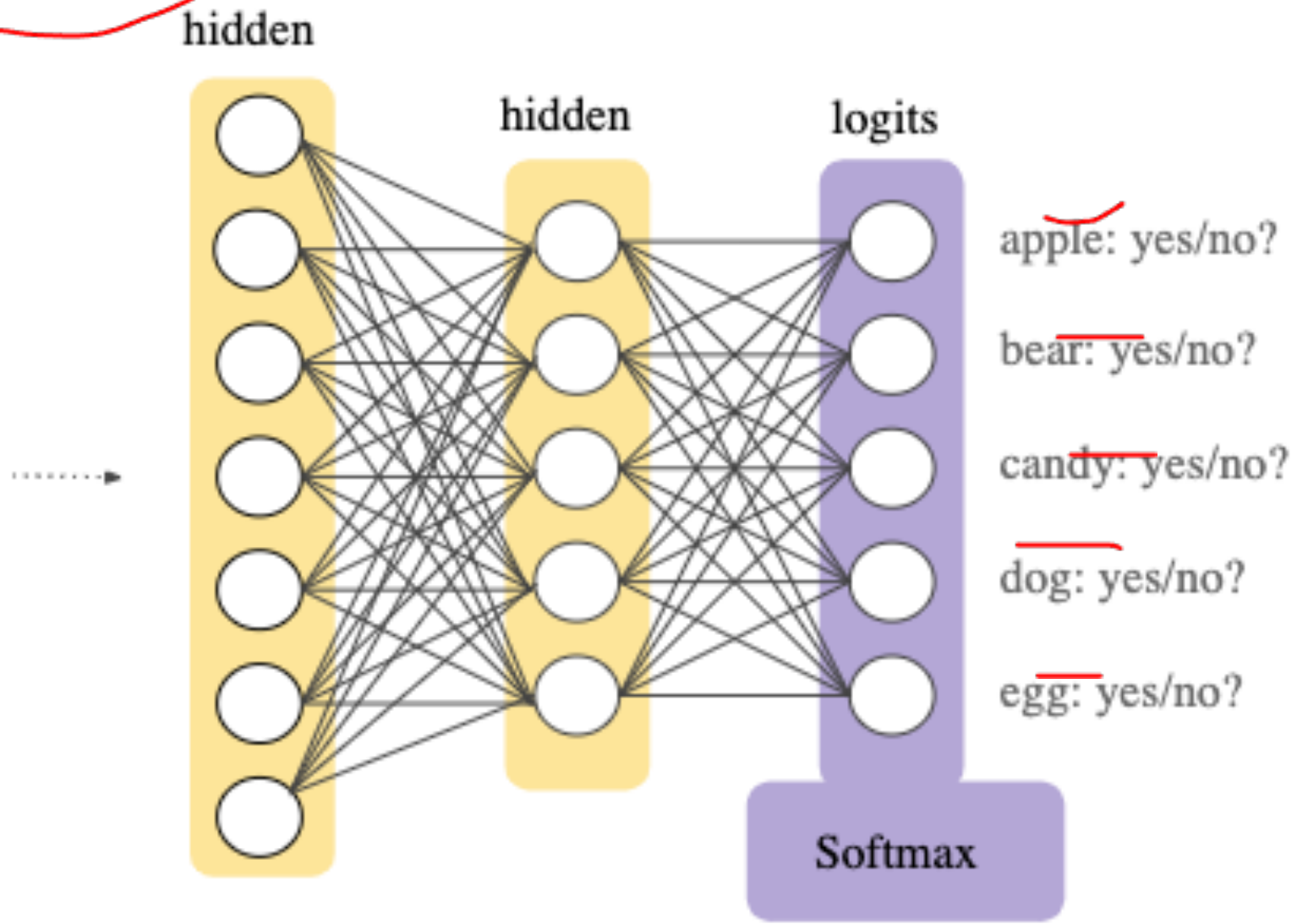
$$a^{[2]} = \sigma(z^{[2]})$$

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

Desired Output

NN Output

$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)],$$


hidden
hidden
logits
apple: yes/no?
bear: yes/no?
candy: yes/no?
dog: yes/no?
egg: yes/no?
Softmax

Gradient Descent

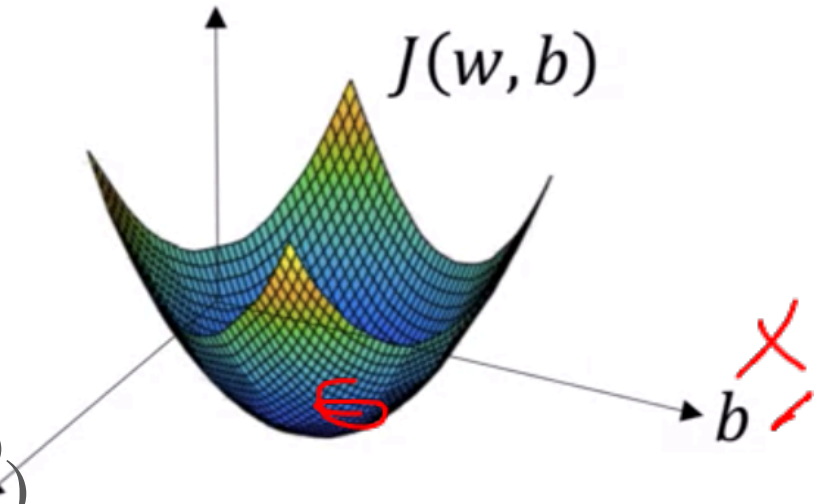
- $\hat{y} = \sigma(w^T x + b) where\ \sigma(z) = \frac{1}{1 + e^{-z}}$

- Loss function (Individual training example)

$$L = -y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Cost function (over entire data set)

$$J(w, b) = \frac{1}{m} \sum y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$



W is actually an dimensional vector

- $w = w - \alpha \frac{dJ}{dw}$
- $b = b - \alpha \frac{dJ}{db}$

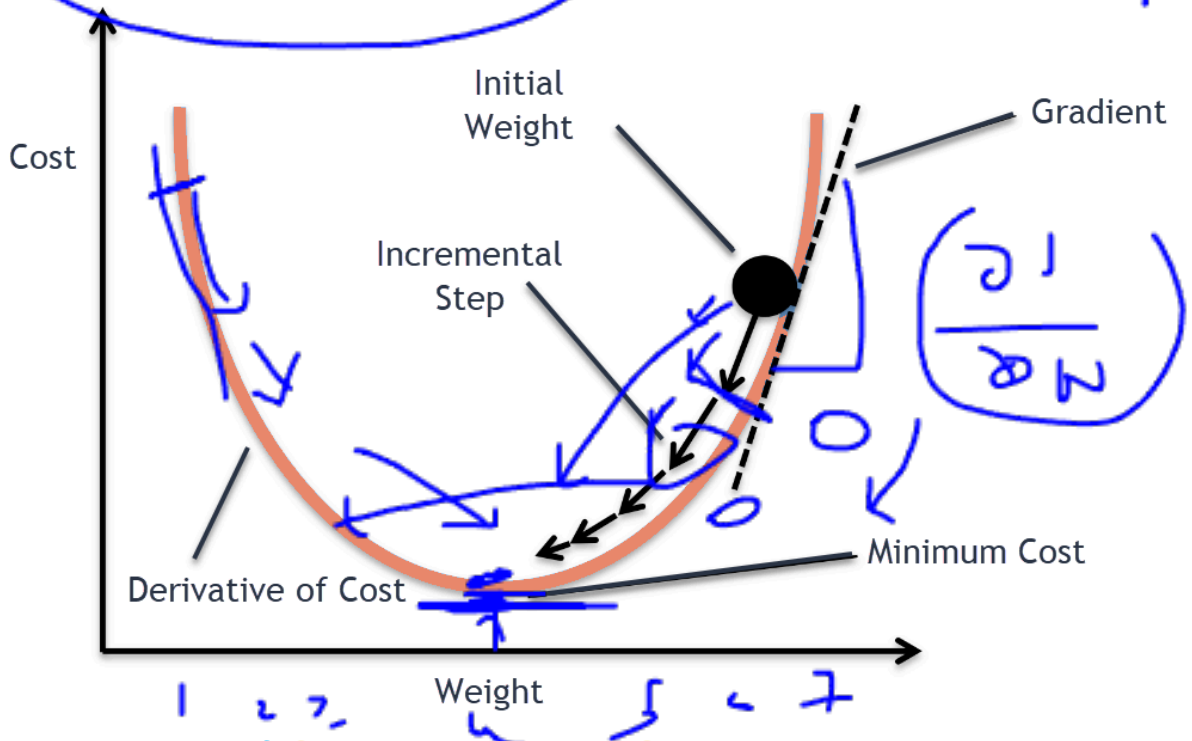
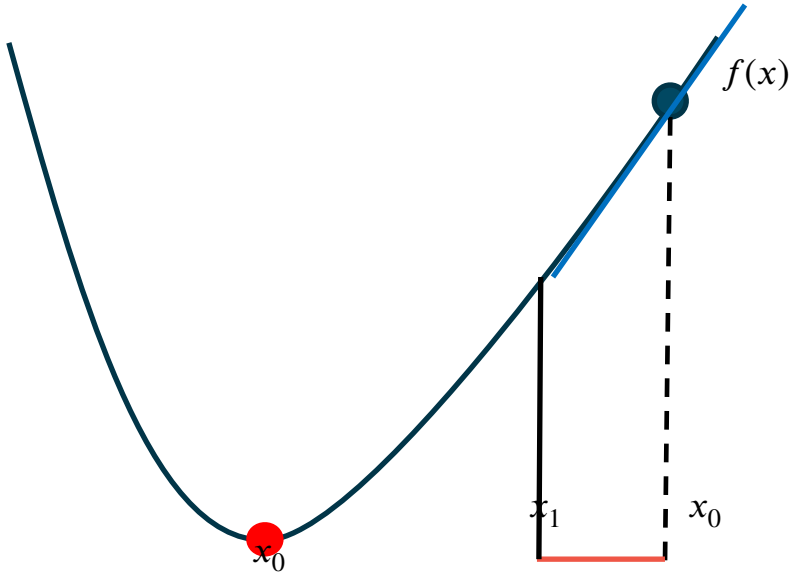
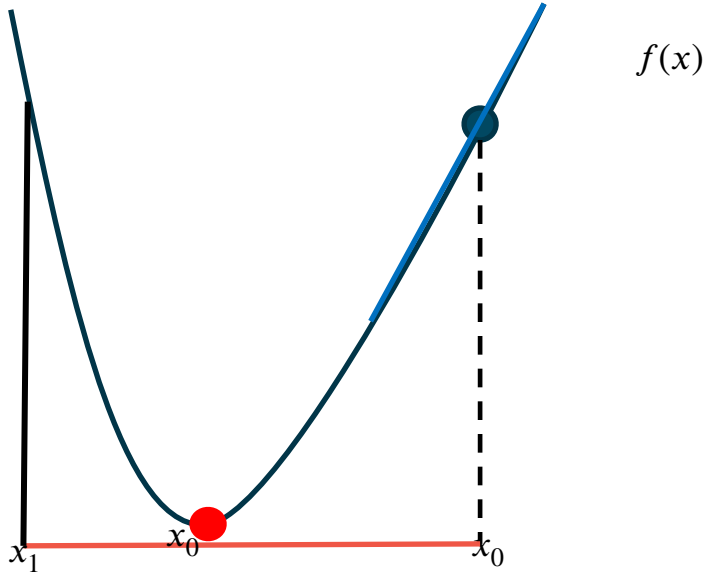


Image Ref: Stochastic Vs Batch Gradient Descent, Kapil Divakar

- Learning rate α, is between 0 and 1

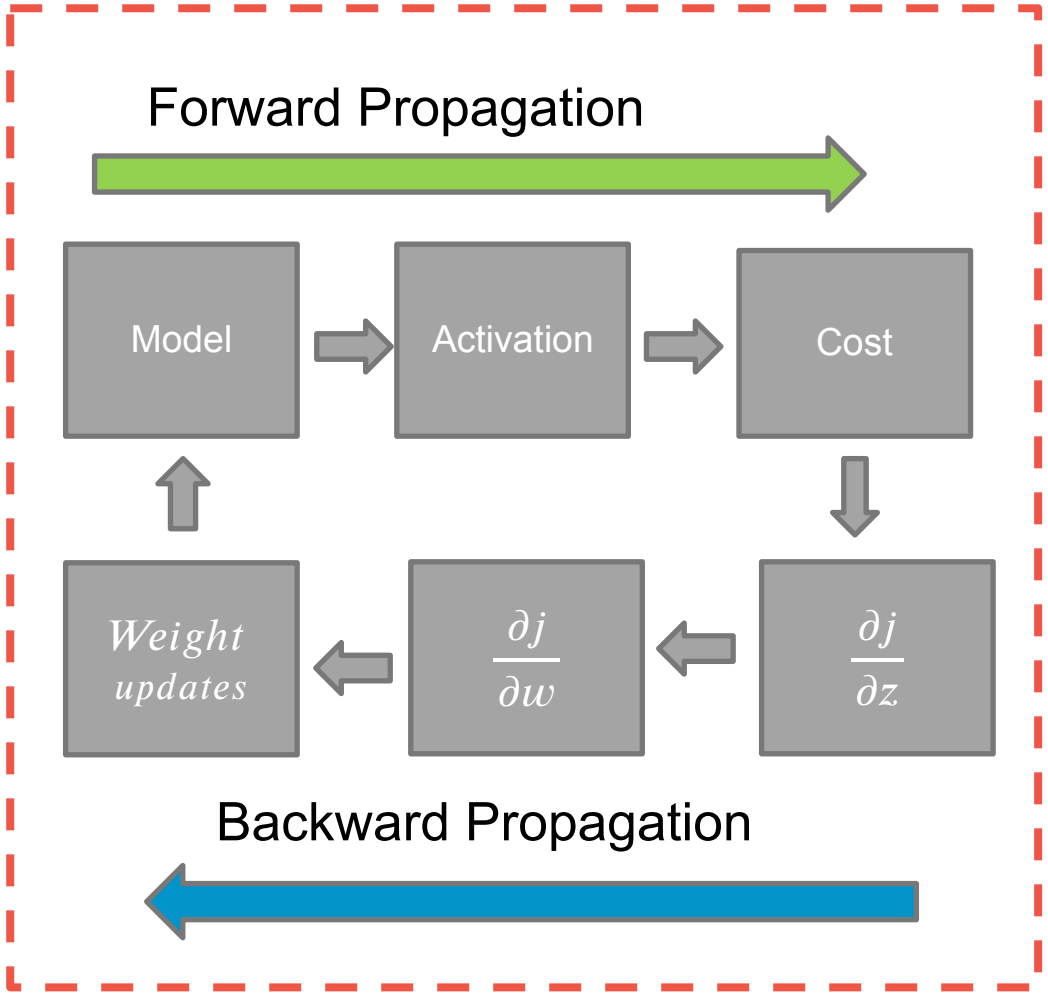
- The choice of learning rate can be critical to how soon the algorithm converges.



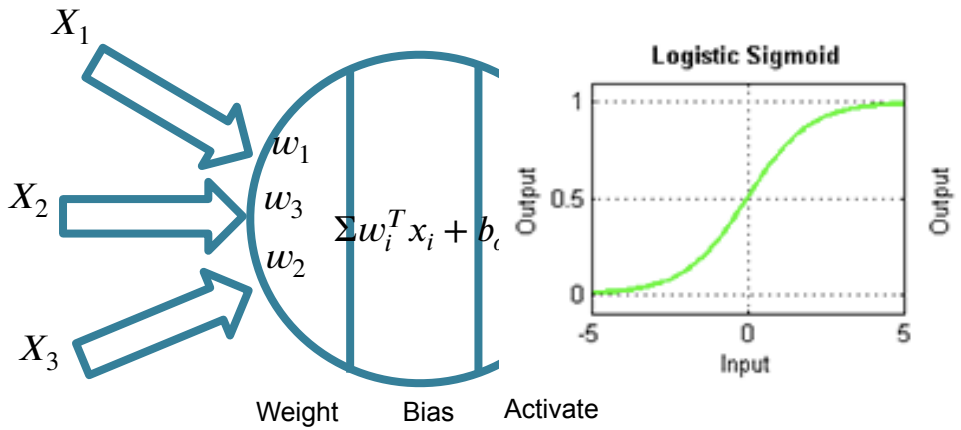


Excel

Forward & Backward Propagation



One step in gradient descent



Cost function (over entire data set)

$$J(w,b) = \frac{1}{m}\Sigma\,[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})]$$

Weight update equations

$$\rightarrow w = w - \alpha\frac{dJ}{dw}$$

$$\rightarrow b = b - \alpha\frac{dJ}{db}$$

Learning rate α, is between 0 and 1

- *We need derivatives for weight updates. (Gradient descent equations)*
- *Derivates of cost with respect to weights is not easily obtained (analytically or numerically)*
- *Computational graph along with chain rule allows you to take derivatives at each step backwards, and then multiply all intermediate derivates.*

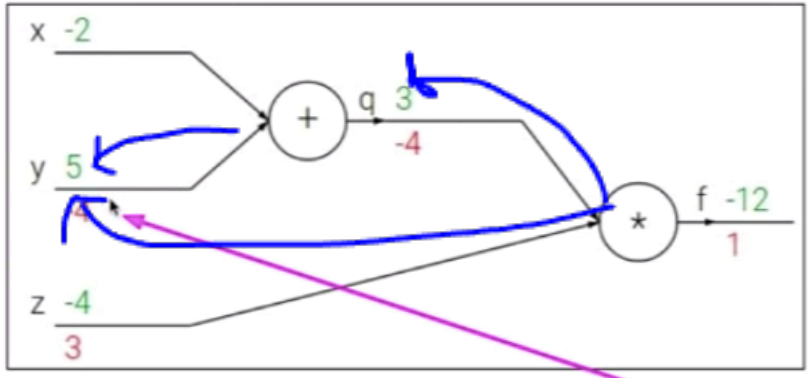
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\frac{\partial f}{\partial y}$

Chain rule:

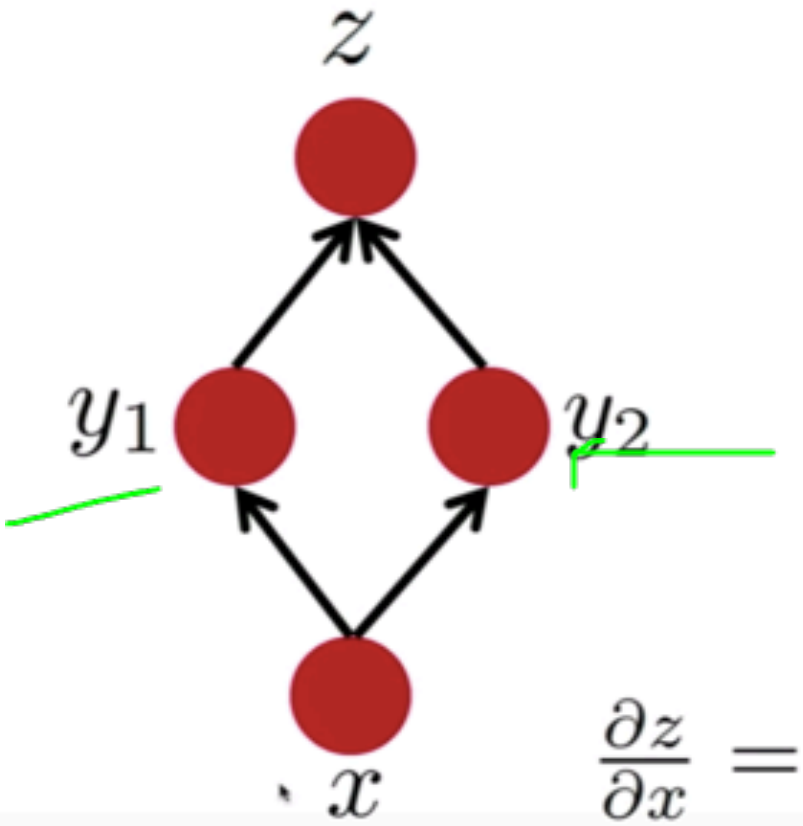
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

$\frac{\partial f}{\partial x}$

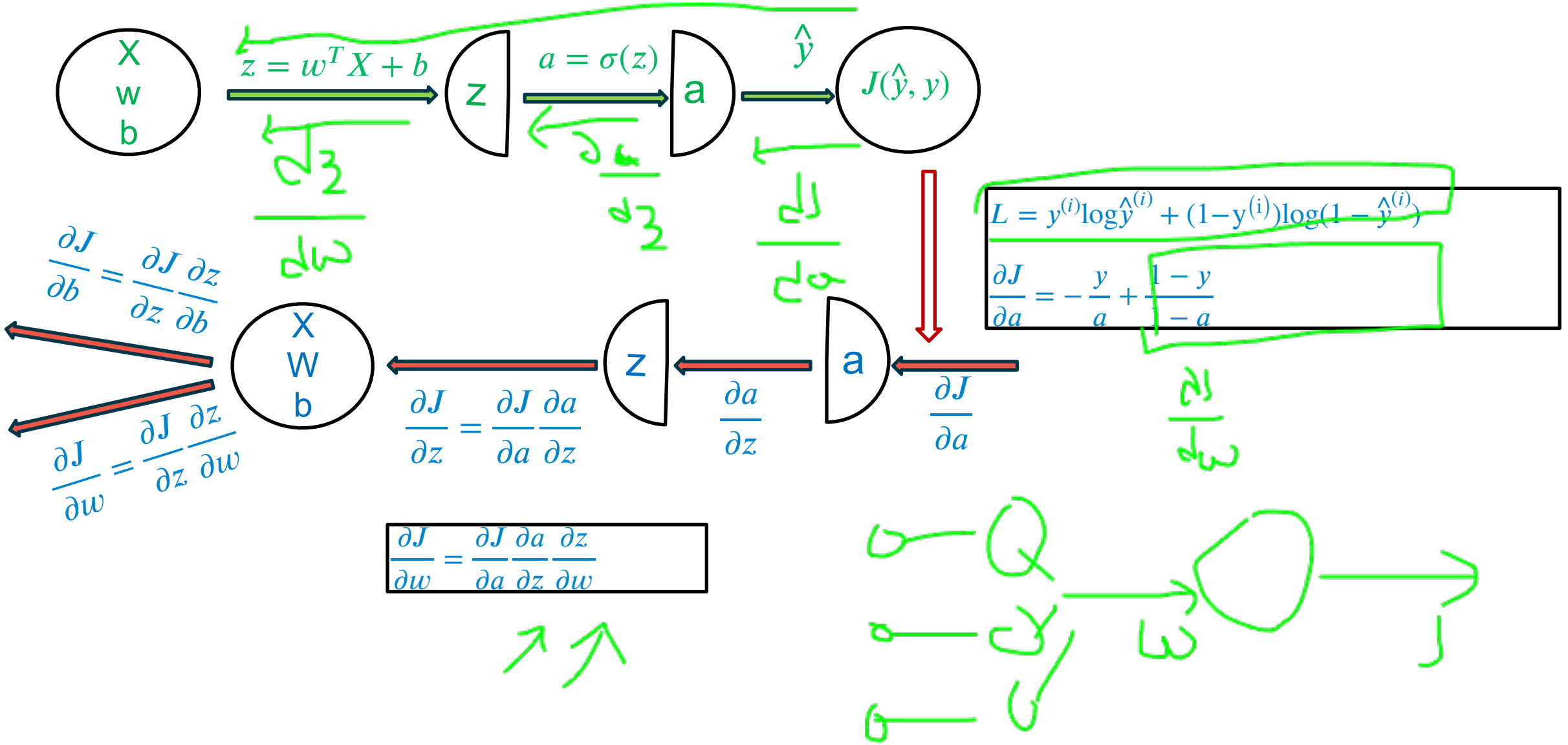
$\frac{\partial f}{\partial x} \frac{\partial q}{\partial z} = \frac{\partial f}{\partial y}$

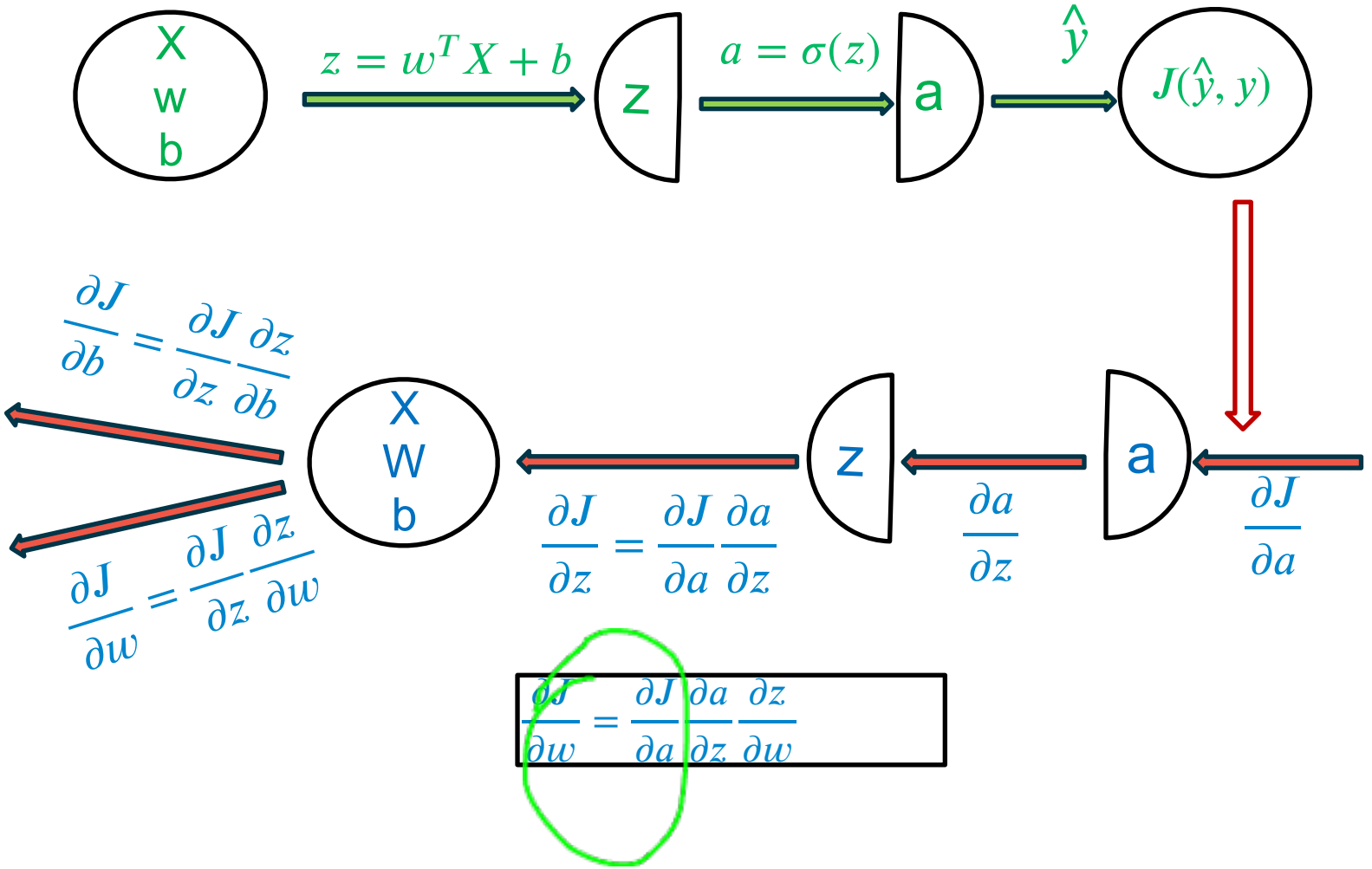
Ref: Backpropagation and Project Advice, Stanford University

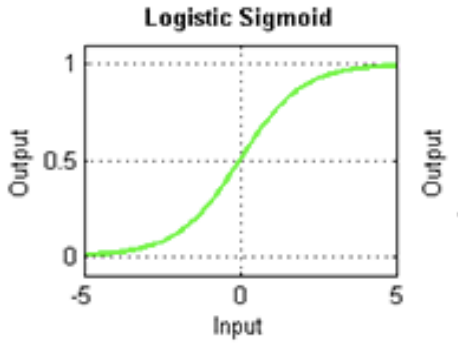
Multiple Paths Chain Rule



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$


X
w
b
$z = w^T X + b$
z
$a = \sigma(z)$
a
$\hat{y}$
$J(\hat{y}, y)$
$L = y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$
$\frac{\partial J}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$
$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial b}$
$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial w}$
X
W
b
$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z}$
z
$\frac{\partial a}{\partial z}$
a
$\frac{\partial J}{\partial a}$
$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w}$

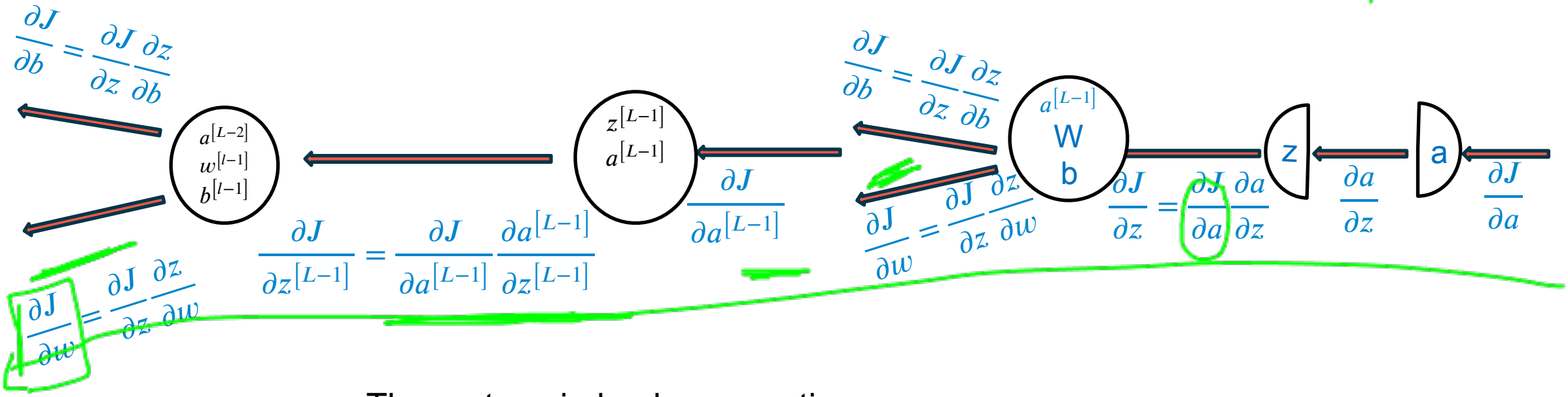




$$a = \frac{1}{1+e^{-z}}$$

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

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w}$$



Three steps in backpropagation

1. Cost to Activation
2. Activation to Z (Linear combination and bias)
3. Z to weights

$$\frac{\partial J}{\partial a^{[L-1]}} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial a^{[l-1]}} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial a^{[l-1]}}$$

$$\frac{\partial J}{\partial w^{[L-1]}} = \frac{\partial J}{\partial z^{[L-1]}}\frac{\partial z^{[L-1]}}{\partial w^{[L-1]}}$$

$$= \frac{\partial J}{\partial a^{[L-1]}}\frac{\partial a^{[L-1]}}{\partial z^{[L-1]}}\frac{\partial z^{[L-1]}}{\partial w^{[L-1]}}$$

$$= \frac{\partial J}{\partial z}\frac{\partial z}{\partial a^{[l-1]}}\frac{\partial a^{[L-1[}}{\partial z^{\{L-1]}}\frac{\partial z^{[L-1]}}{\partial w^{[L-1]}}$$

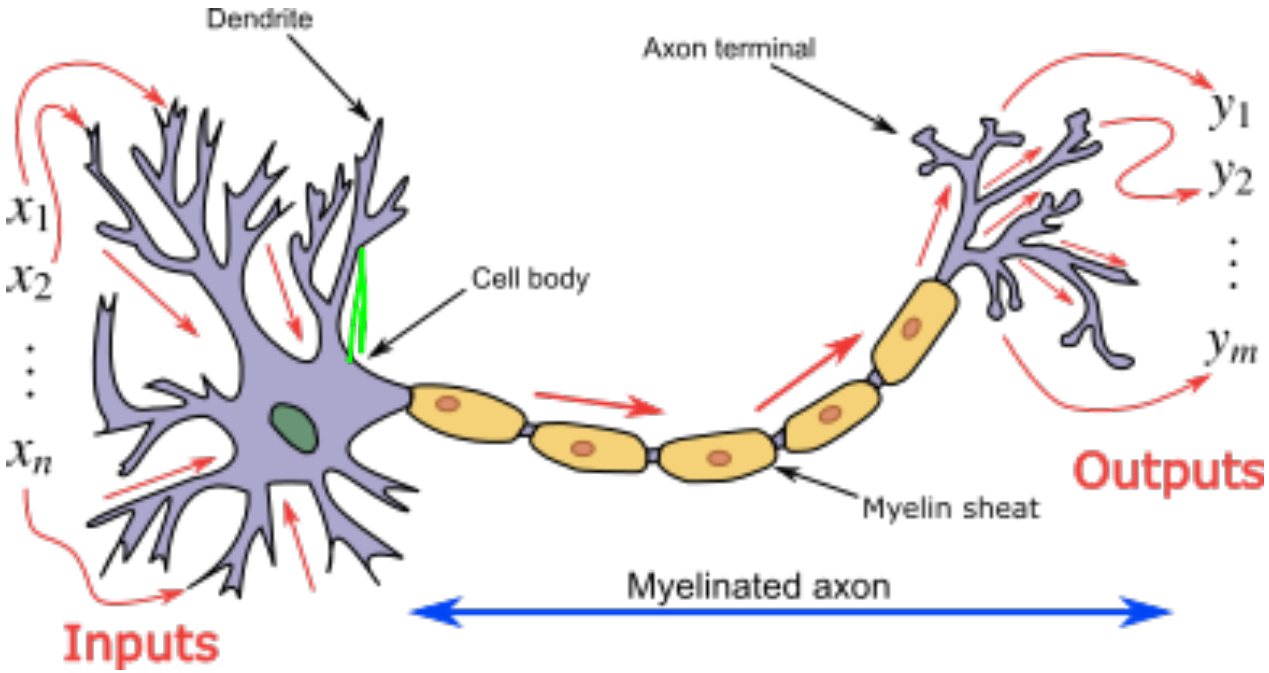
$$= \frac{\partial J}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial a^{[l-1]}}\frac{\partial a^{[L-1[}}{\partial z^{[L-1]}}\frac{\partial z^{[L-1]}}{\partial w^{[L-1]}}$$

1. $\frac{\partial J}{\partial z} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z} = a - y$ (see note 1)
2. $\frac{\partial J}{\partial w} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial J}{\partial z} x^T.$ (see note 2)
3. $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial z}$
4. $\frac{\partial J}{\partial z^{[L-1]}} = w\frac{\partial J}{\partial z} * \frac{\partial a}{\partial z}$
5. $\frac{\partial J}{\partial w^{[L-1]}} = \frac{\partial J}{\partial z^{[L-1]}}\frac{\partial z^{[L-1]}}{\partial w^{[L-1]}} = \frac{\partial J^{[L-1]}}{\partial z^{[L-1]}} x^T$
6. $\frac{\partial J}{\partial b^{[L-1]}} = \frac{\partial J}{\partial z^{[L-1]}}$

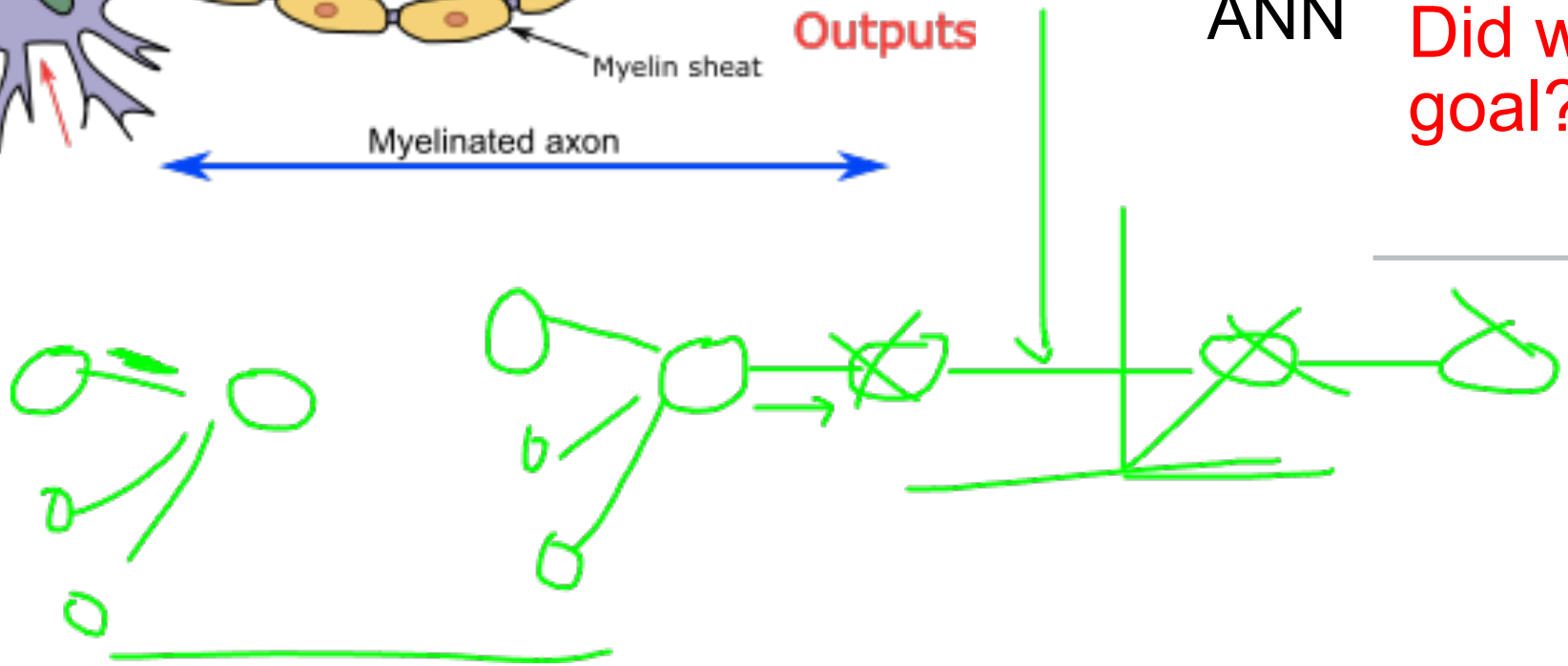
(*Note 1: Only for sigmoid. Will differ for other activation functions*)

(Note 2: If this not the input layer, use (a) of previous layer, instead of x)

- Each step of gradient descent requires gradient to be calculated over the entire data
- Sending ALL the data at once (if in millions), could be computationally infeasible
- Alternate is to select a 'random' data point, calculate the derivative and update the weight. Then take another point and repeat the process.
- Since only one observation is taken, the gradient descent may fluctuate a lot and may take more iterations (but less computations) to converge.
- Sometimes, instead of one data point, a **"mini-batch"** of data points is selected which is faster than stochastic gradient descent and at the same time less computation than "batch" (full) gradient descent
- When the entire data set is passed through the network once, it is called an **Epoch**,



Goal	Understand the brain
	Emulate the Brain
ANN	Did we achieve the goal?

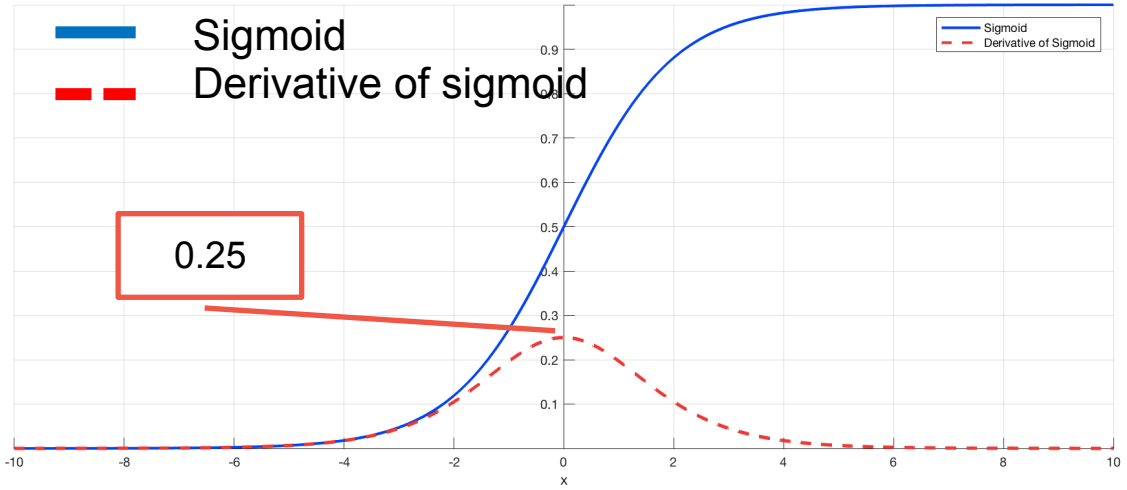


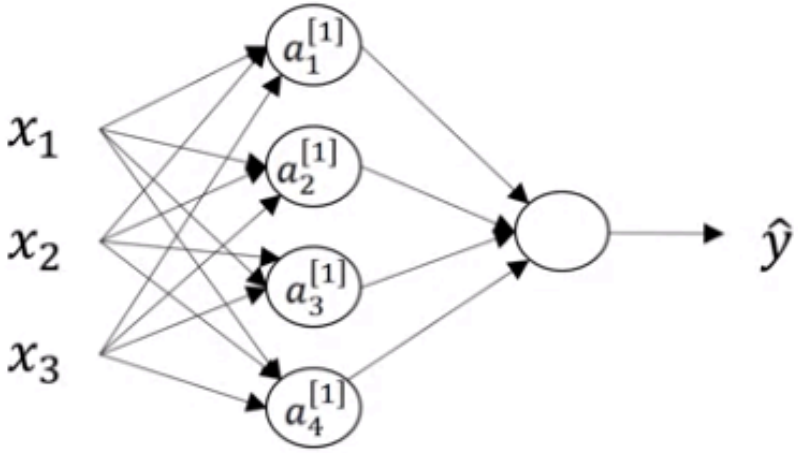
Exploding & Vanishing Gradient

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w}$$

$$\frac{\partial J}{\partial w^{[L-1]}} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial a^{[l-1]}}\frac{\partial a^{[L-1]}}{\partial z^{\{L-1\}}}\frac{\partial z^{[L-1]}}{\partial w^{[L-1]}}$$

- Deeper you go in the network, weights of initial layers will have several gradients multiplied (due to backpropagation)
- Gradient of sigmoid activation function with max value of 0.25, can vanish (and hence not update weights) in a deeper network.
- Typical problem in sigmoid but not necessarily in RELU





- In some cases gradients may explode or vanish if weights are not initialized properly. This is not because of sigmoid function but similar reasoning apply (i.e. several weight matrices are multiplied over the layers in a deep network)

- $$\frac{\partial J}{\partial z^{[L-1]}} = w \frac{\partial J}{\partial z} * \frac{\partial a}{\partial z}$$
- $$\frac{\partial J}{\partial w^{[L-1]}} = \frac{\partial J}{\partial z^{[L-1]}} \frac{\partial z^{[L-1]}}{\partial w^{[L-1]}} = \frac{\partial J}{\partial z}$$
- $$\frac{\partial J}{\partial b^{[L-1]}} = \frac{\partial J}{\partial z^{[L-1]}}$$

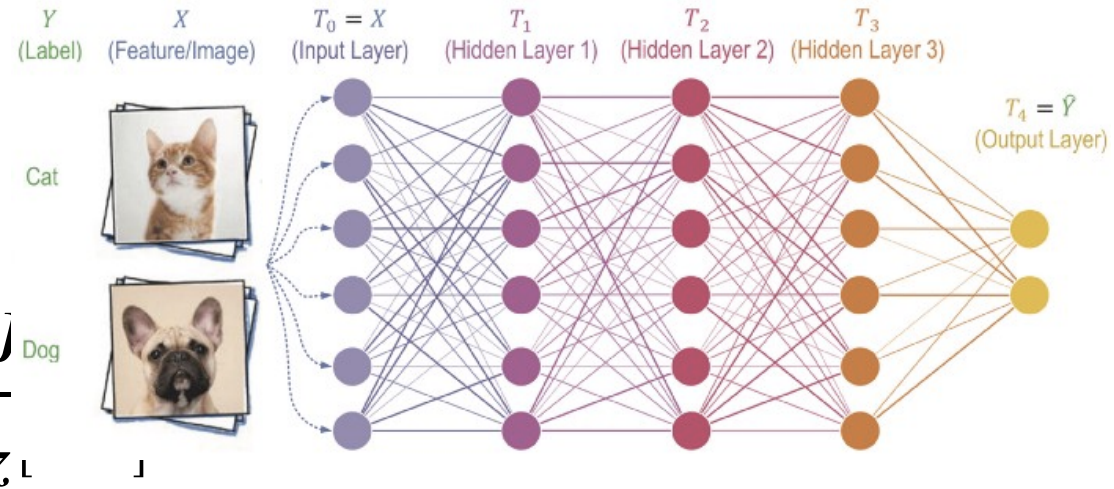
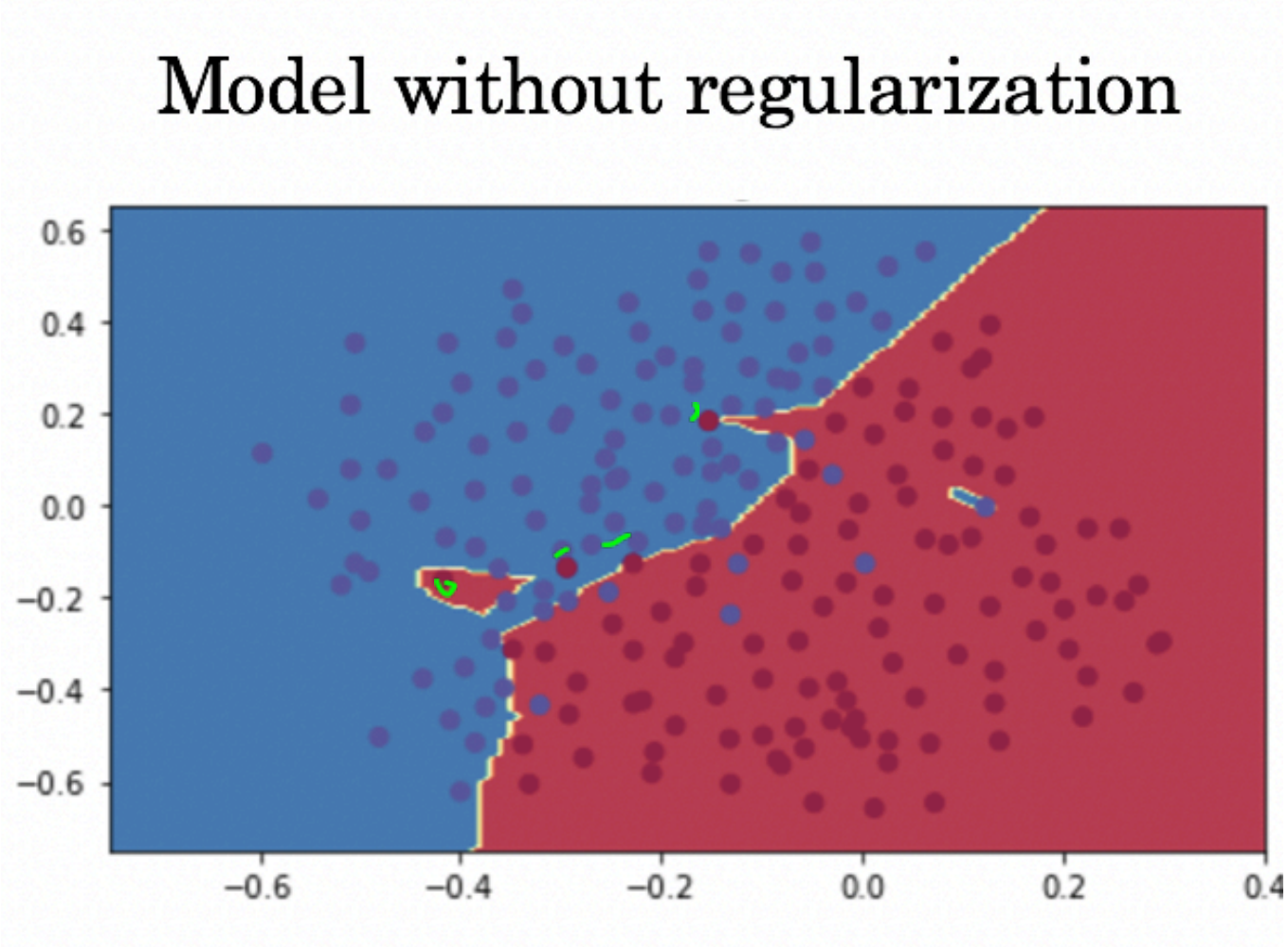


Image source

Regularization

- Neural Networks can overfit!

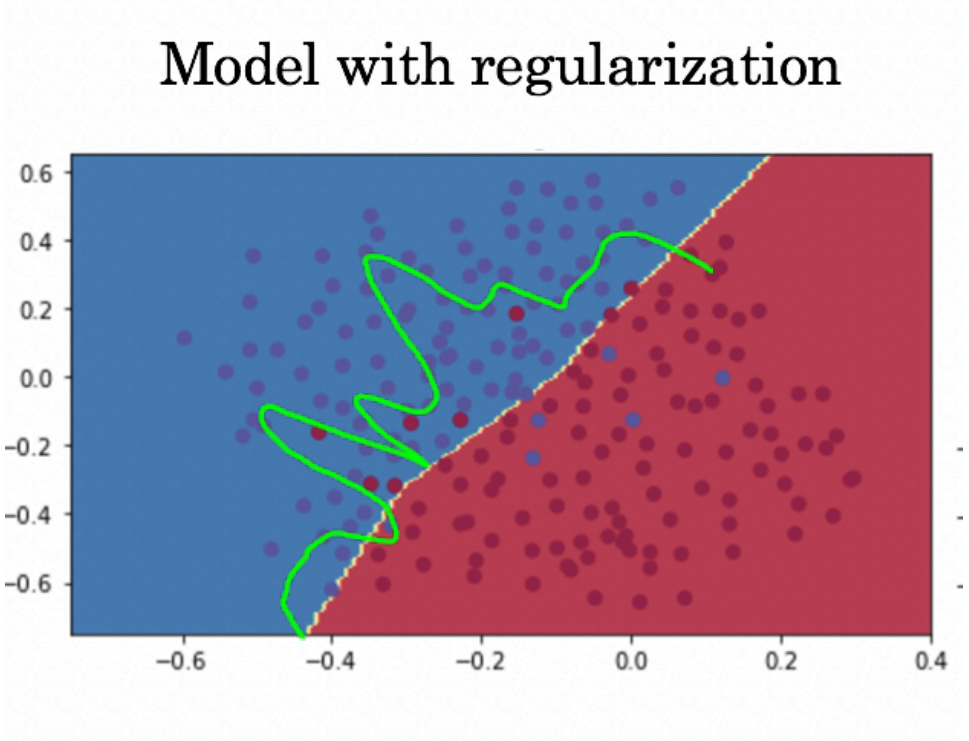


$$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[L]}\|_F^2$$

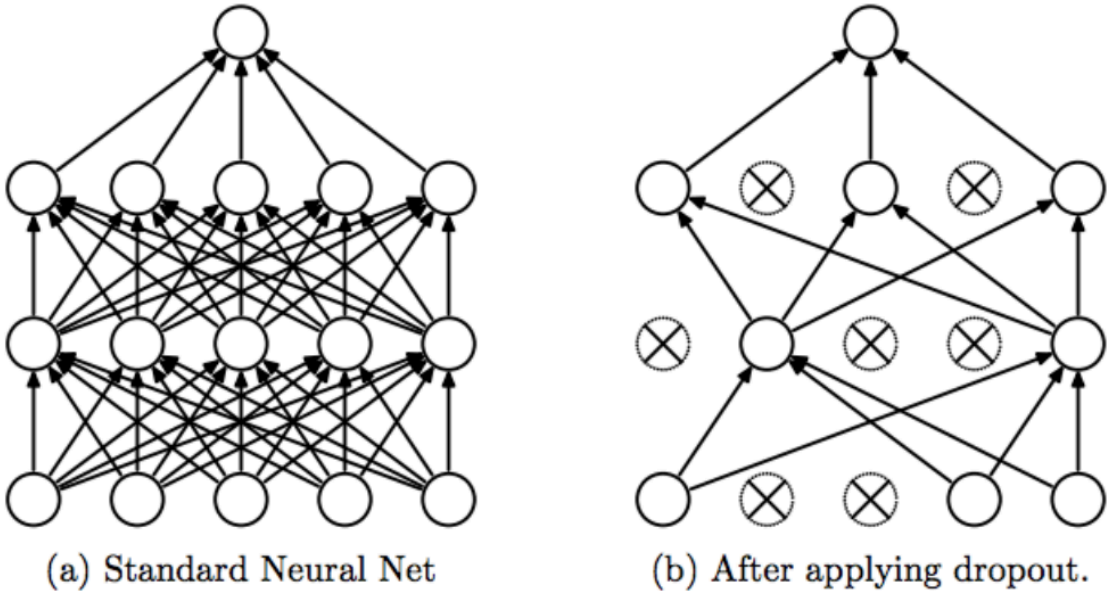
L2 regularization

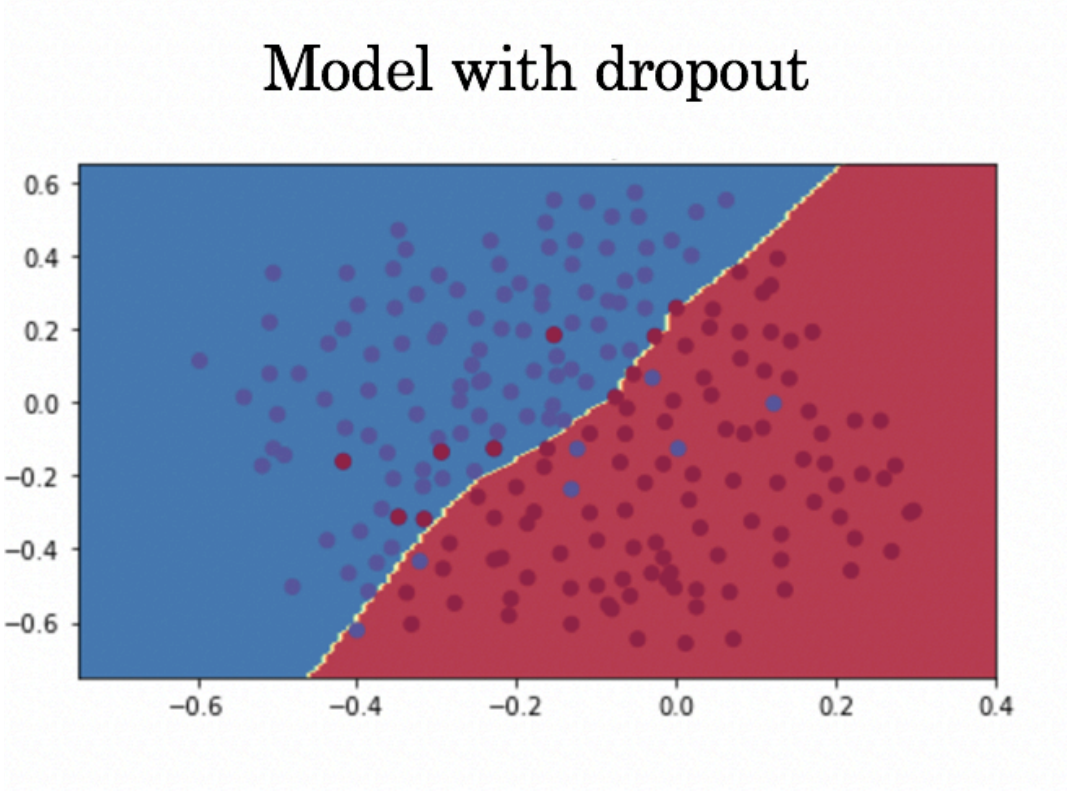
Image source

Regularization Parameter



As the weights increase (i.e high weight is given to any neuron, regularization term also increases and adds to the cost. Since, gradient descent tries to minimize the cost, it will try to keep the weights low, hence regularized)





Dropout Regularization

Random neurons are dropped from the network with some probability. Since a neuron has certain chances of being dropped, network will not give high weight to any of it during training providing the effect of regularization.

➢ **Epoch:**

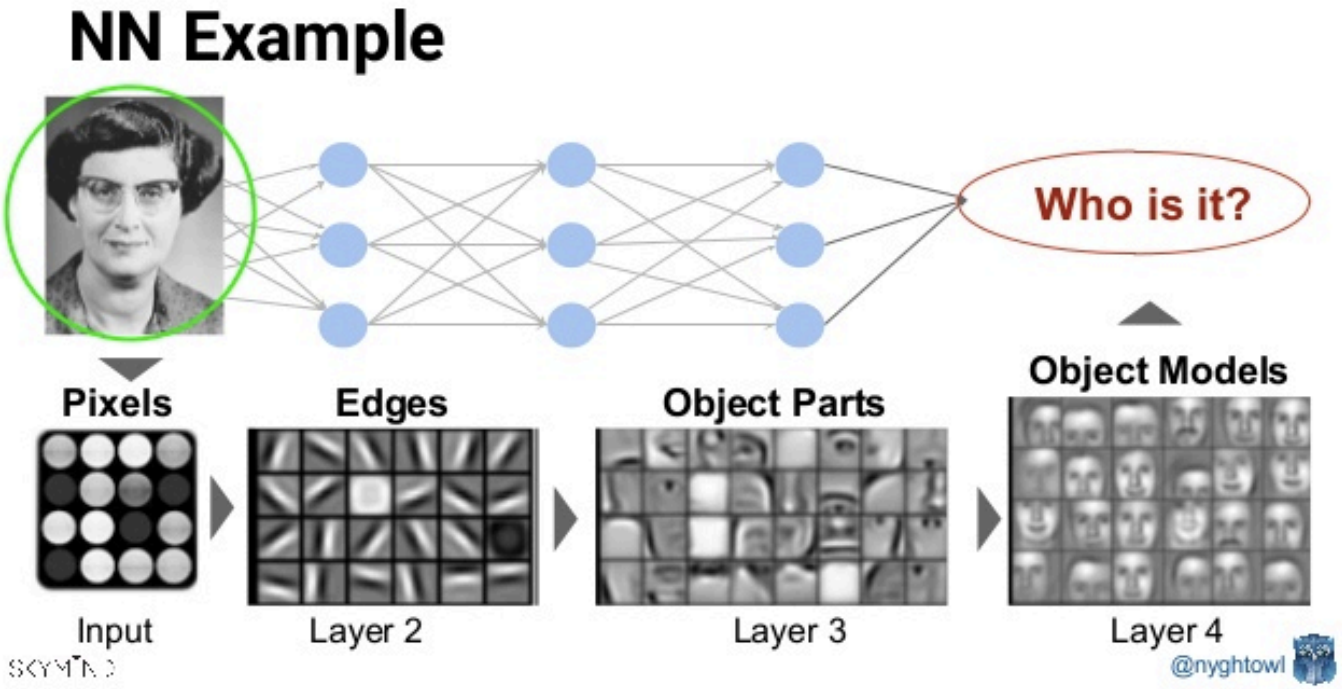
- One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

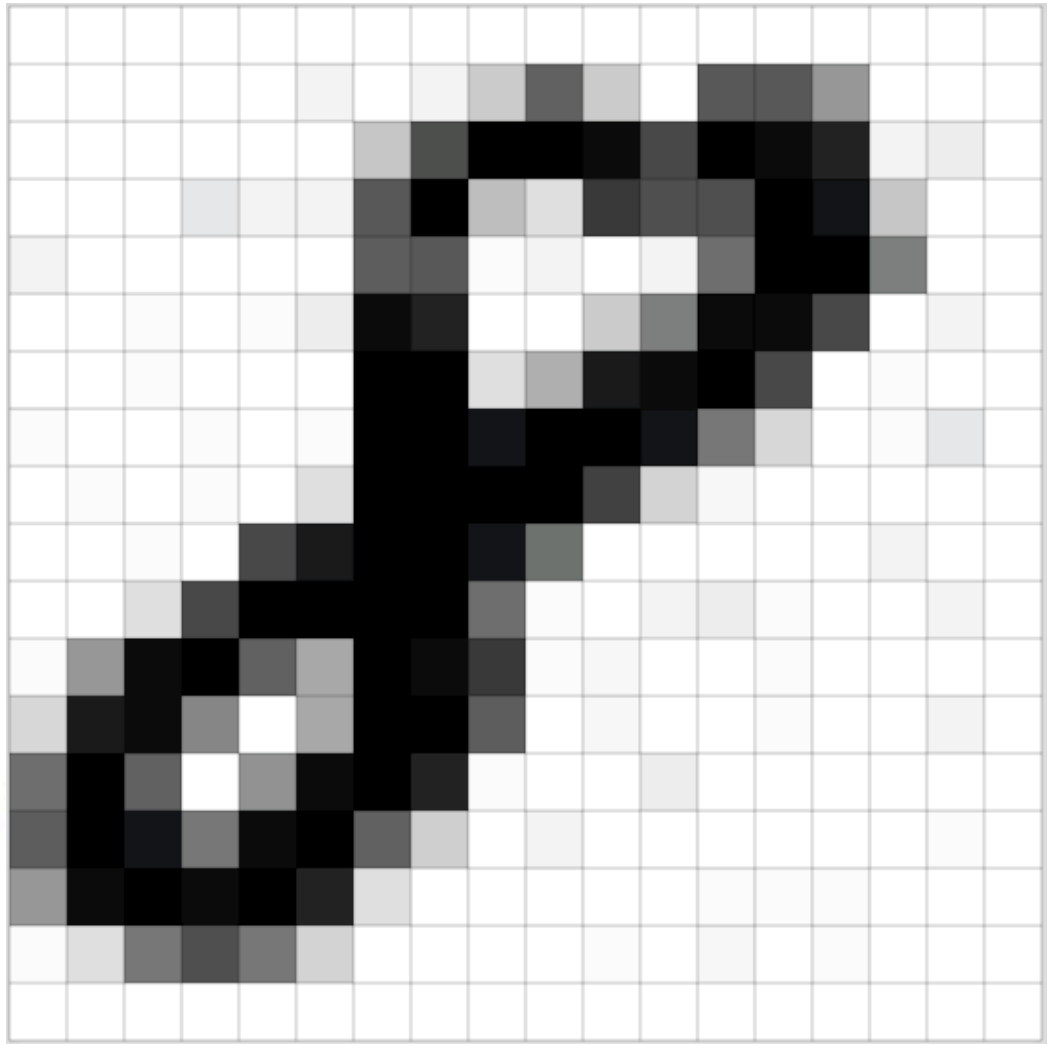
➢ Batch:

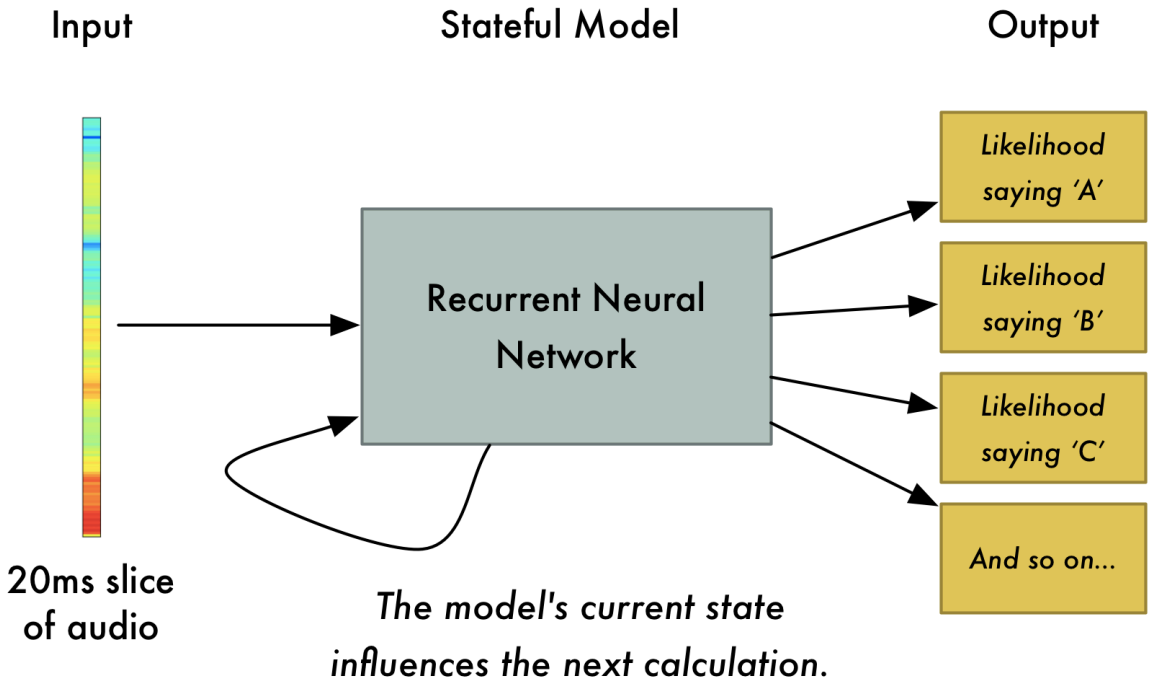
- Total number of training examples present in a single batch.
- If you can't pass the entire dataset into the neural net at once, divide dataset into Number of Batches or sets or parts.

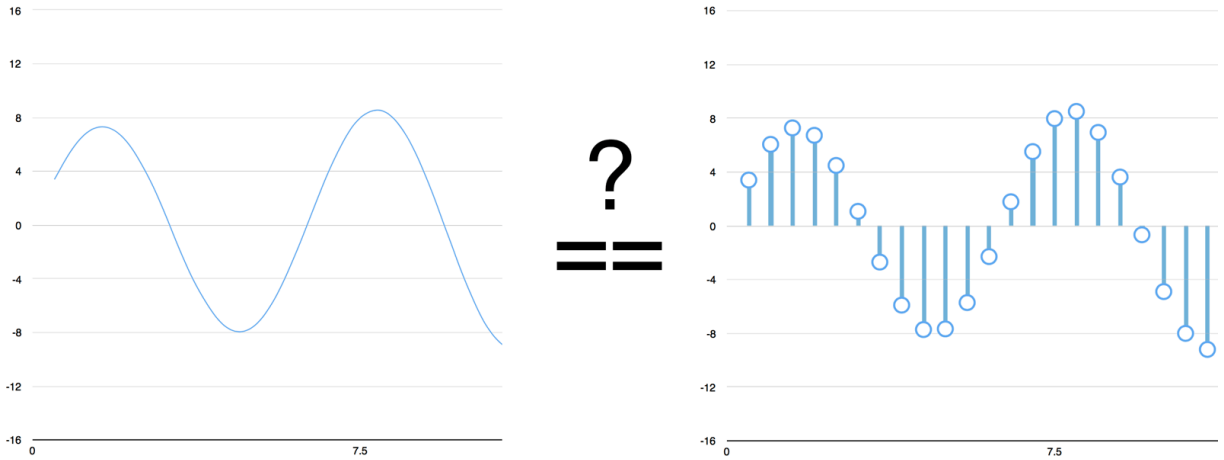
- **Batch Gradient Descent**. Batch Size = Size of Training Set
- **Stochastic Gradient Descent**. Batch Size = 1
- **Mini-Batch Gradient Descent**. 1 < Batch Size < Size of Training Set

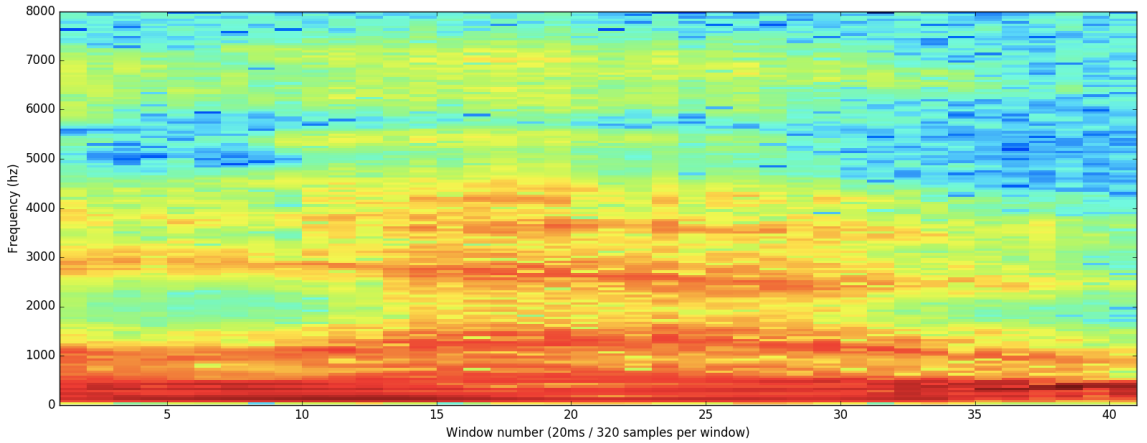
Use Cases


NN Example
Who is it?
Pixels
Edges
Object Parts
Object Models
Input
Layer 2
Layer 3
Layer 4
@nyghtowl



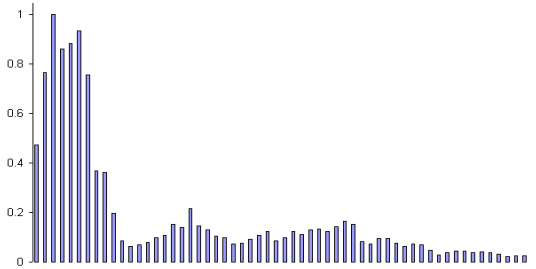

Input
Stateful Model
Output
Recurrent Neural Network
Likelihood saying 'A'
Likelihood saying 'B'
Likelihood saying 'C'
And so on...
20ms slice of audio
The model's current state influences the next calculation.


16
12
8
4
0
-4
-8
-12
-16
0
7.5
?
==
Original Analog Signal
Sampled Digital Signal


Frequency (hz)
8000
7000
6000
5000
4000
3000
2000
1000
0
5
10
15
20
25
30
35
40
Window number (20ms / 320 samples per window)

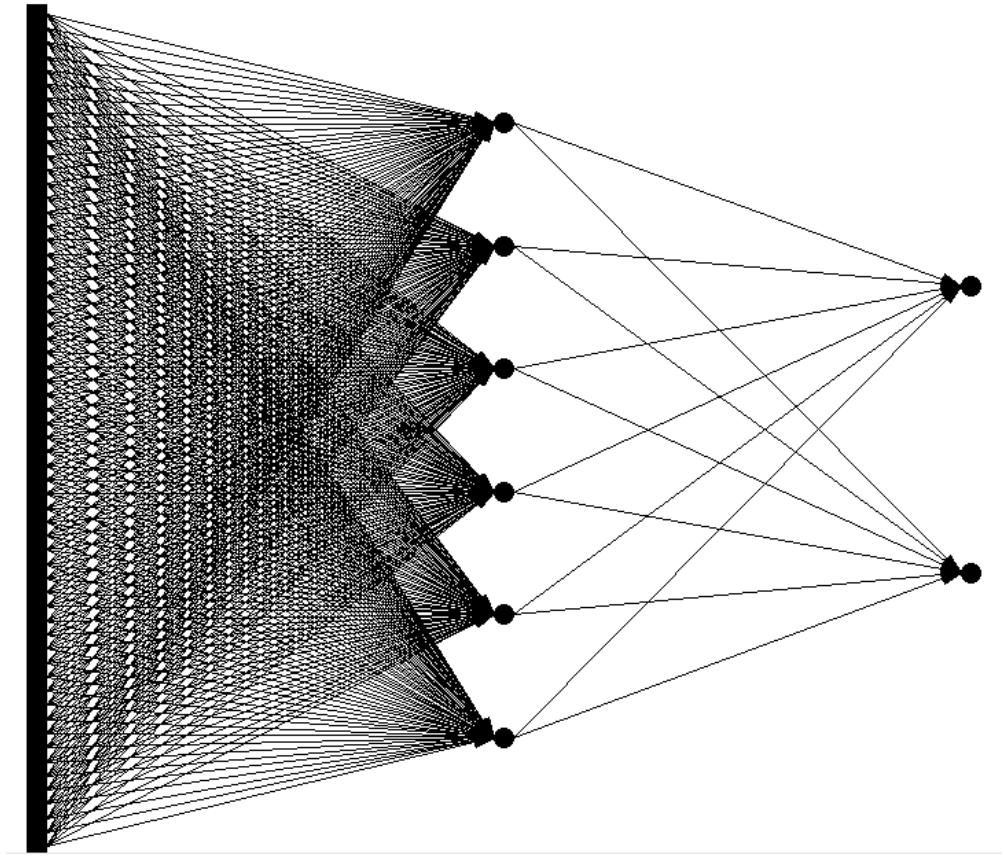
- Task: Learn to discriminate between two different voices saying "Hello"

- Data
  - Sources
    - Steve Simpson
    - David Raubenheimer
  - Format
    - Frequency distribution (60 bins)
    - Analogy: cochlea1sw 1sw

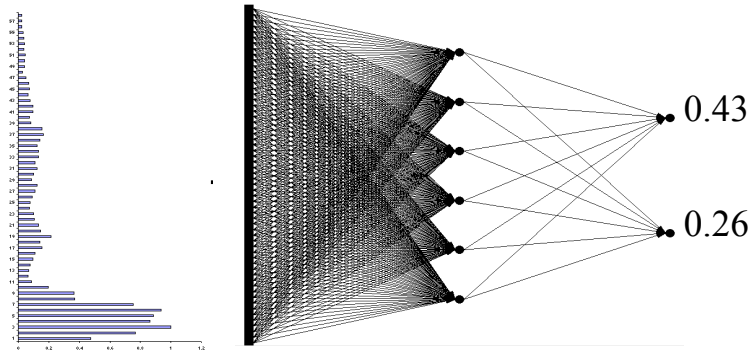


Network architecture

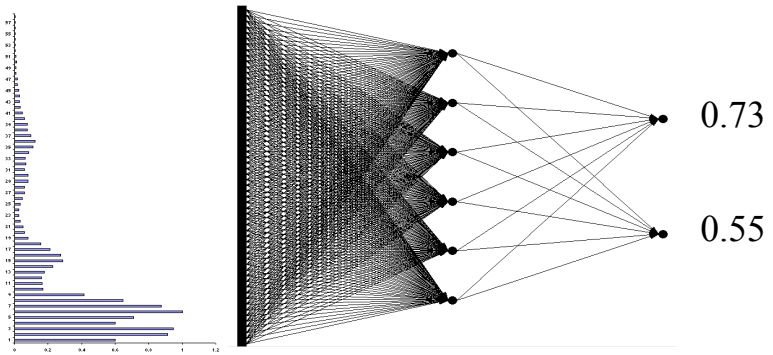
- Feed forward network
  - 60 input (one for each frequency bin)
  - 6 hidden
  - 2 output (0-1 for “Steve”, 1-0 for “David”)



Steve

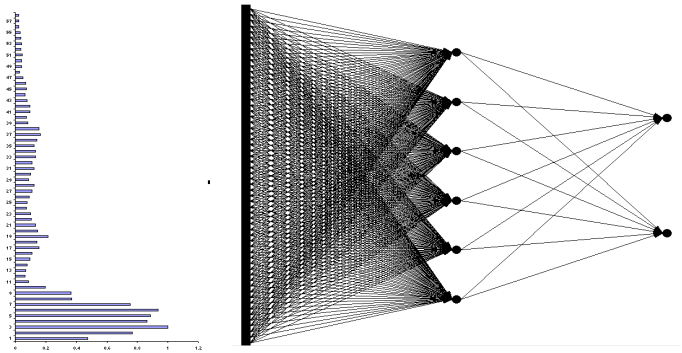


David

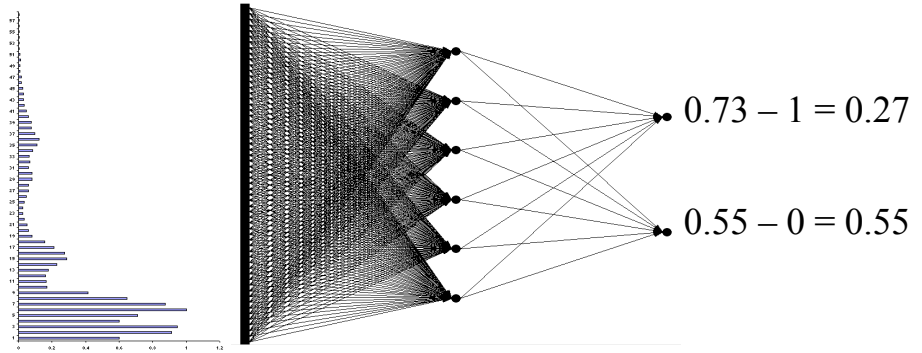


Calculate error

Steve



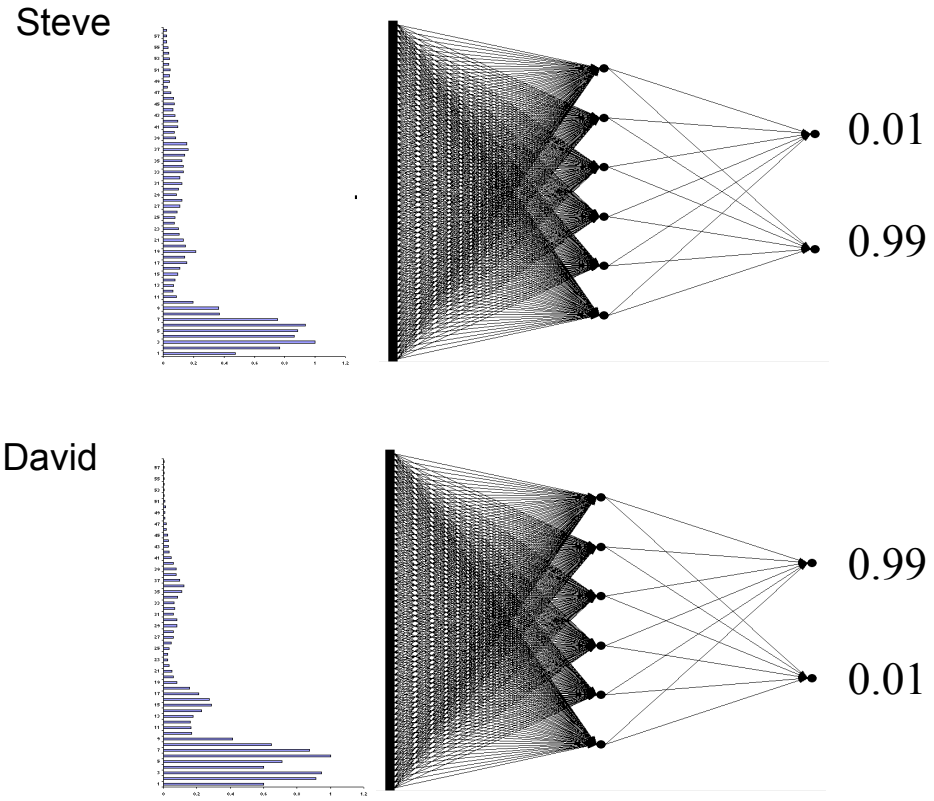
David



Performance of trained network

Discrimination accuracy between known “Hello”s
100%

Discrimination accuracy between new “Hello”s
100%



Back Up