IBM Research Data Science Course

srihari sridharan

Agenda

Neural Networks

- Weights and Bias
- Activation Functions (Non-Linearity)
- Data Space Transformation
- Keras Code

High School Mathematics

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \square & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}_{m \times n}$$

Self Aware Artificial Intelligence to destroy Humanity

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \square & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}_{m \times n}$$

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \square & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}_{m \times n}$$

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \square & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}_{m \times n}$$

Looking at Data

Sepal Length	Sepal Width	Petal Length	Petal Width	Flower
4.9	3.0	1.4	0.2	Iris-Setosa
6.3	2.5	5	1.9	Iris-Virginica
5.7	2.8	4.5	1.3	Iris-Versicolor
6.3	3.3	4.7	1.6	Iris-Versicolor
4.7	3.2	1.3	0.2	Iris-Setosa
6.5	3	5.2	2	Iris-Virginica

Looking at Data

4-dimension Vector Input

Sepal Length	Sepal Width	Petal Length	Petal Width	Flower
4.9	3.0	1.4	0.2	Iris-Setosa
6.3	2.5	5	1.9	Iris-Virginica
5.7	2.8	4.5	1.3	Iris-Versicolor
6.3	3.3	4.7	1.6	Iris-Versicolor
4.7	3.2	1.3	0.2	Iris-Setosa
6.5	3	5.2	2	Iris-Virginica

Looking at Data

4-dimension Vector Input

Sepal Length	Sepal Width	Petal Length	Petal Width	Flower	
4.9	3.0	1.4	0.2	Iris-Setosa	1
6.3	2.5	5	1.9	Iris-Virginica	3
5.7	2.8	4.5	1.3	Iris-Versicolor	2
6.3	3.3	4.7	1.6	Iris-Versicolor	2
4.7	3.2	1.3	0.2	Iris-Setosa	1
6.5	3	5.2	2	Iris-Virginica	3

1-dimension Vector Output

Looking at Data

Sepal Length	Sepal Width	Petal Length	Petal Width	Flower	
4.9	3.0	1.4	0.2	Iris-Setosa	1
6.3	2.5	5	1.9	Iris-Virginica	3
5.7	2.8	4.5	1.3	Iris-Versicolor	2
6.3	3.3	4.7	1.6	Iris-Versicolor	2
4.7	3.2	1.3	0.2	Iris-Setosa	1
6.5	3	5.2	2	Iris-Virginica	3

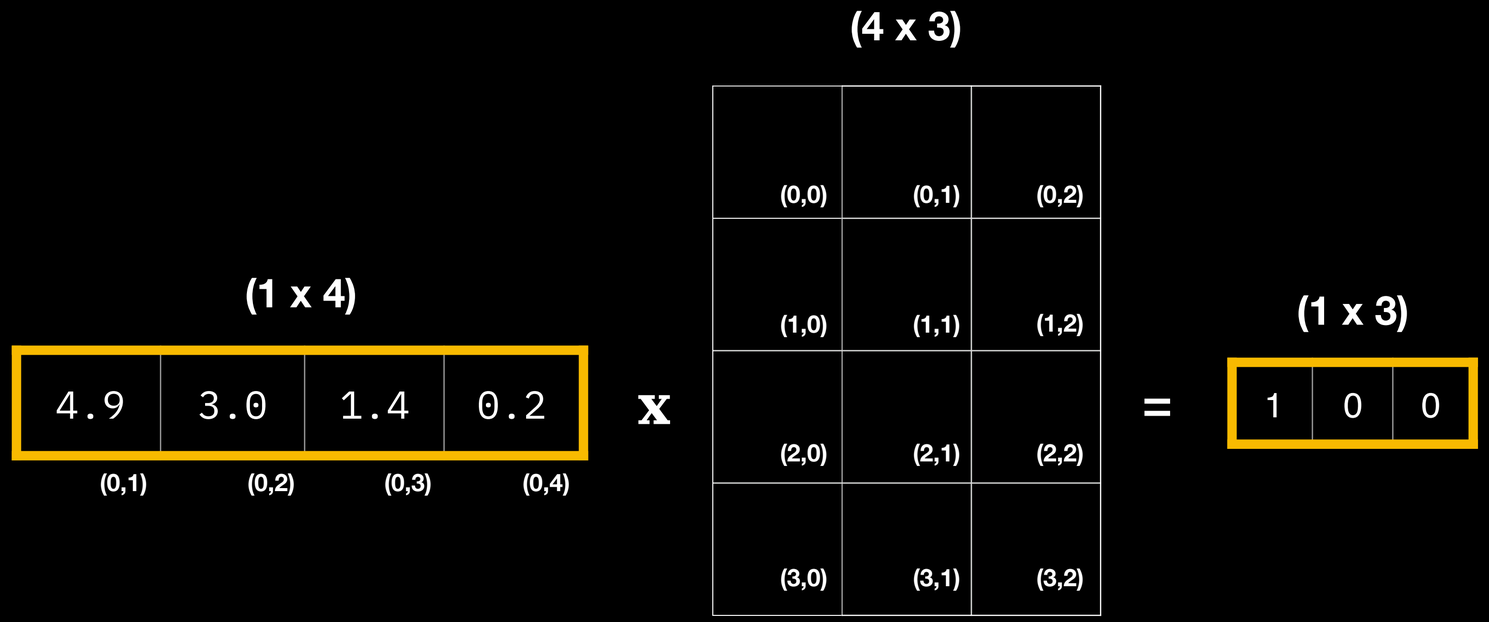
Another way to vectorise labels (One-Hot)

{ 1 , 2 , 3 }		
1	0	0
0	0	1
0	1	0
0	1	0
1	0	0
0	0	1

3-dimension Vector Output

How do we convert (transform) one vector to another?

Weights


(4 x 3)
(1 x 4)
4.9
3.0
1.4
0.2
(0,1)
(0,2)
(0,3)
(0,4)
x
(0,0)
(0,1)
(0,2)
(1,0)
(1,1)
(1,2)
(2,0)
(2,1)
(2,2)
(3,0)
(3,1)
(3,2)
=
(1 x 3)
1
0
0

Weights

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

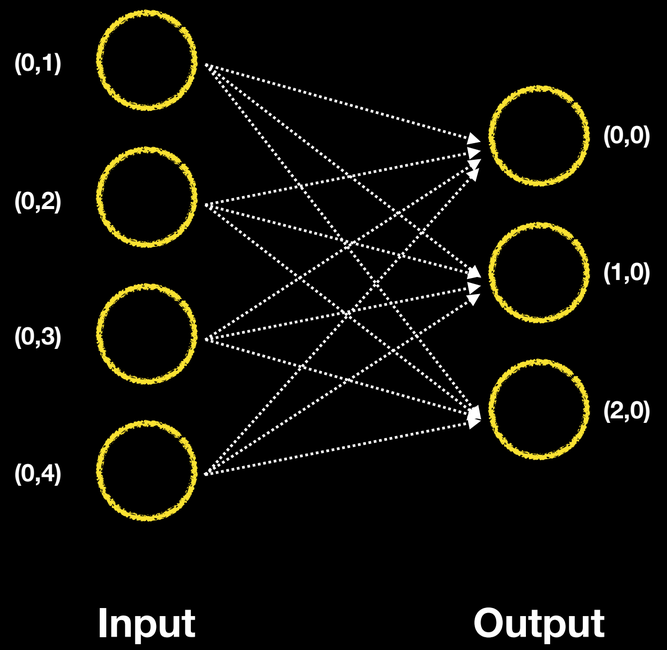
(4 x 3)

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

1	0	0



Weights

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

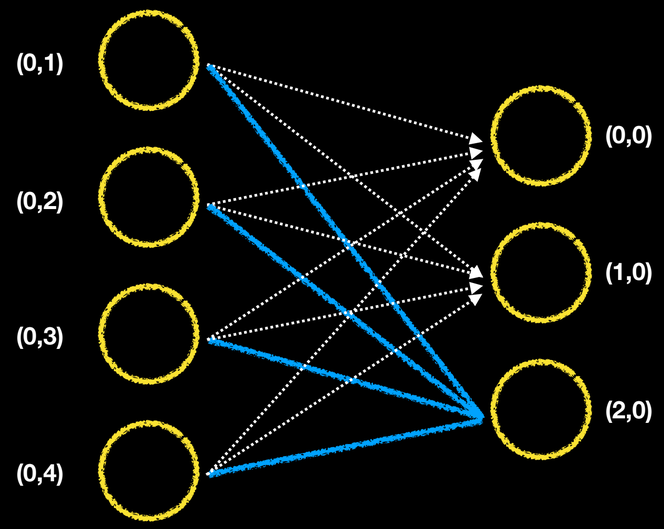
(4 x 3)

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

1	0	0



Weights

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

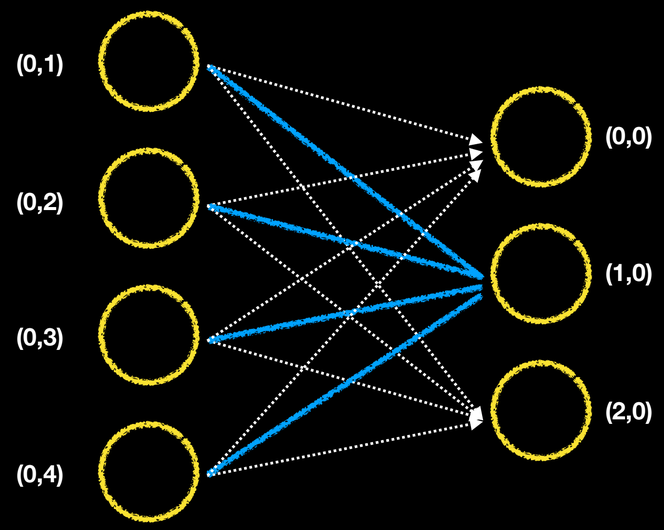
(4 x 3)

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

1	0	0



Weights

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

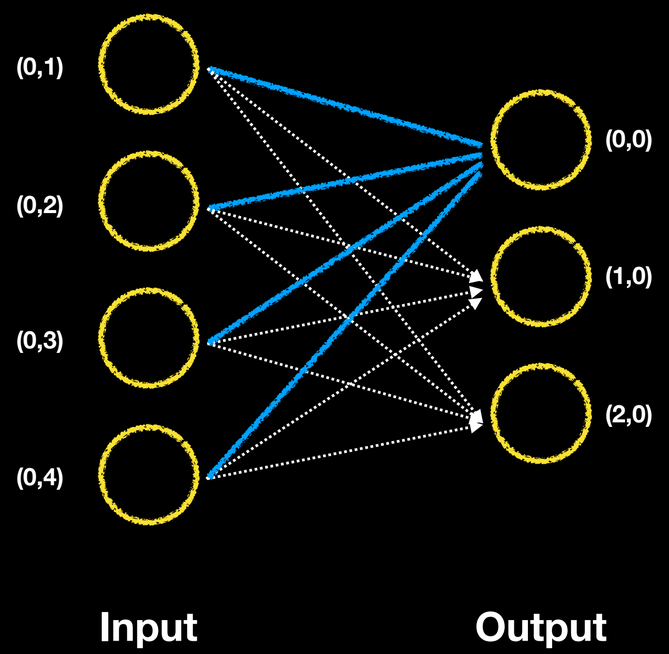
(4 x 3)

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

1	0	0



Weights

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

(4 x 3)

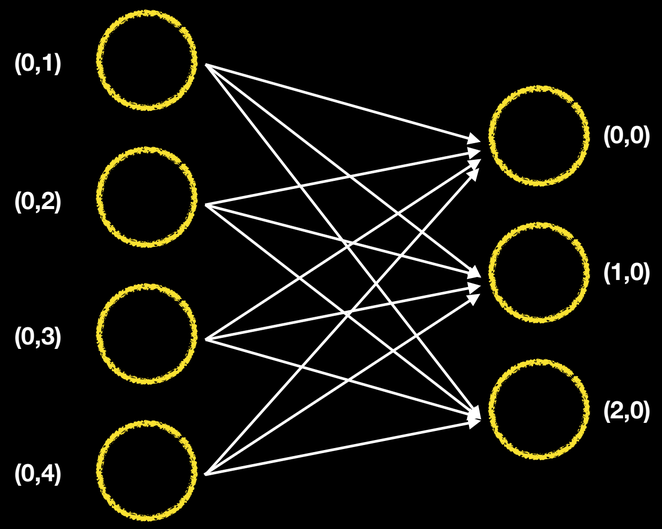
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

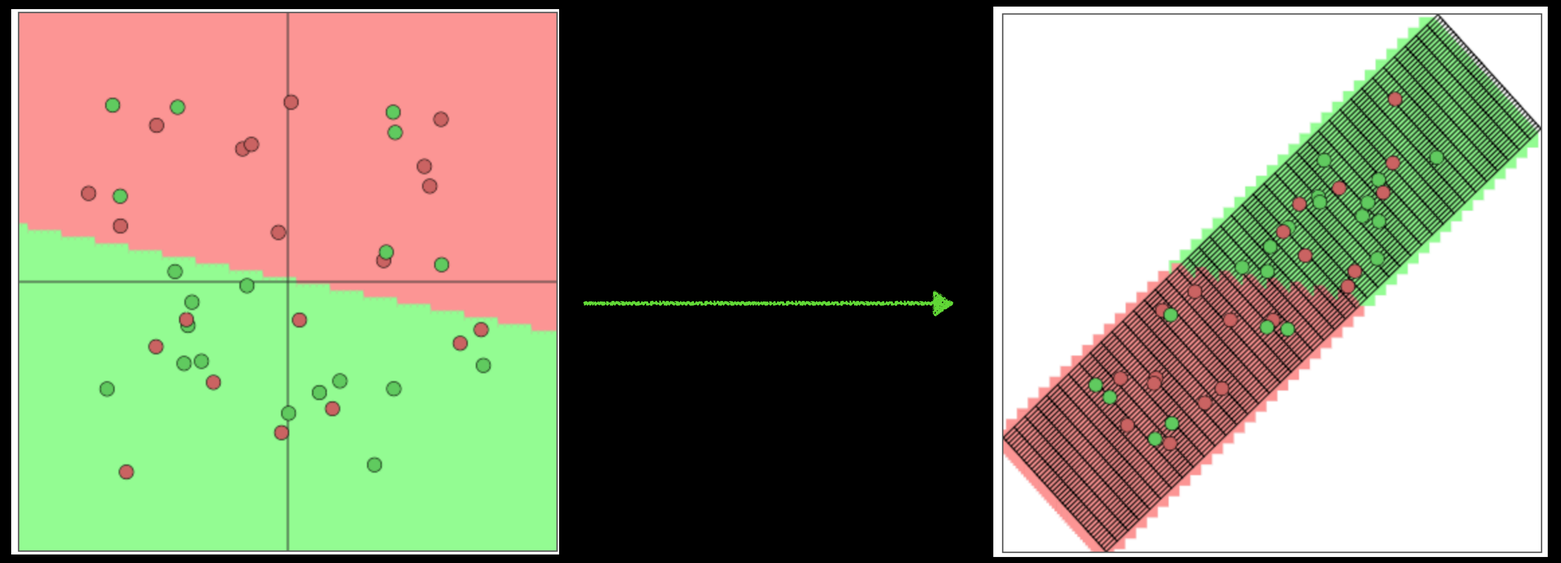
1	0	0

Input . Weights = Output



Weights

Another way to think about it : Which is the best transformation such that it is easy to draw a line between classes?
(Linearly Separable)



Bias

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

(4 x 3)

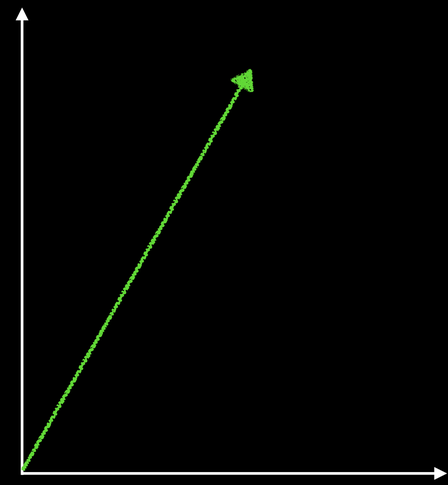
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

1	0	0

Input . Weights = Output



y = 2x

y = mx + c

Bias

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

(4 x 3)

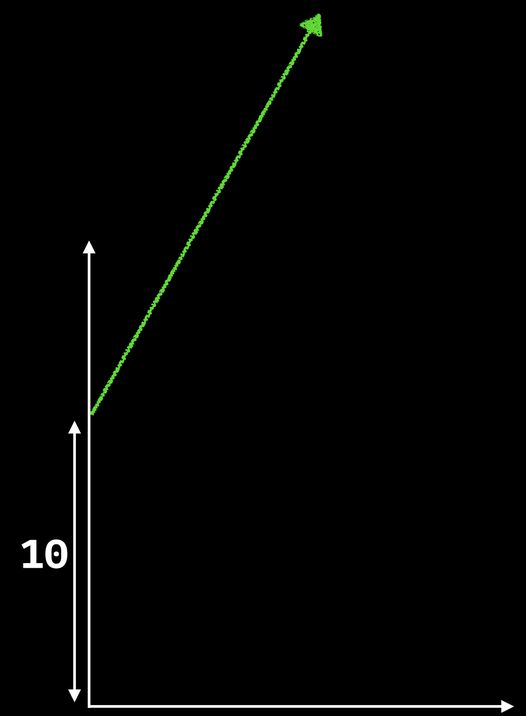
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

1	0	0

Input . Weights = Output



?

y = mx + c

Bias

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

(4 x 3)

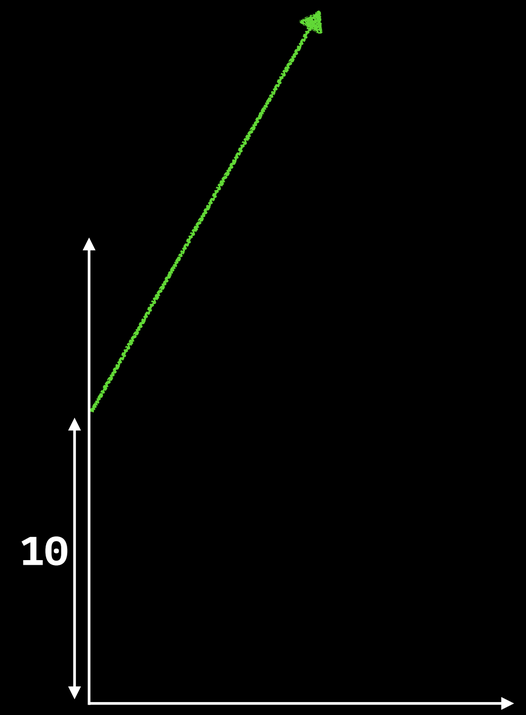
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

1	0	0

Input . Weights = Output



$y = 2x + 10$

$y = mx + c$

Bias

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

(4 x 3)

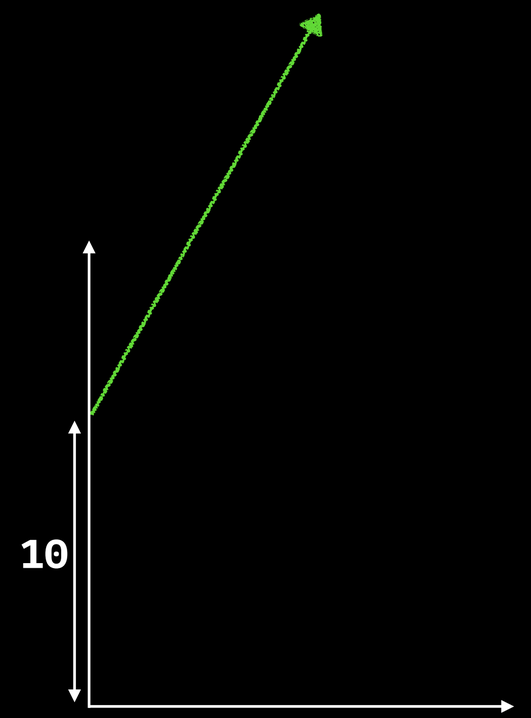
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

1	0	0

Input . Weights = Output



y = 2x + 10

y = mx + c Bias

Bias

(1 x 4)

4.9	3.0	1.4	0.2
(0,1)	(0,2)	(0,3)	(0,4)

x

(4 x 3)

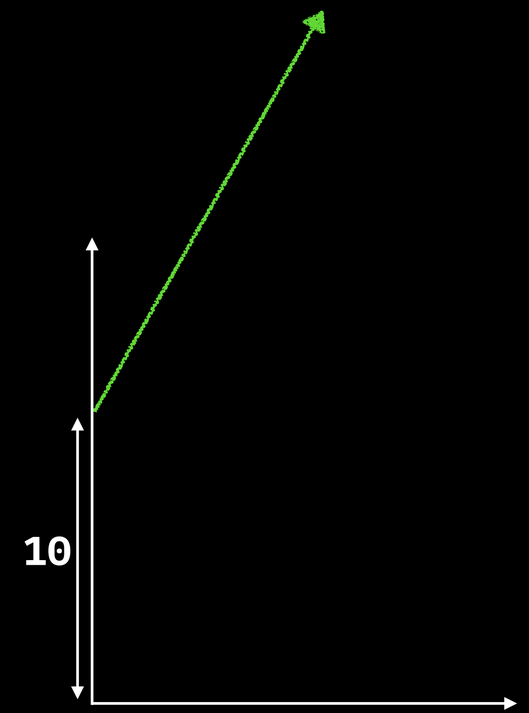
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

(1 x 3)

1	0	0

Input * Weights + Bias = Activations



$y = 2x + 10$

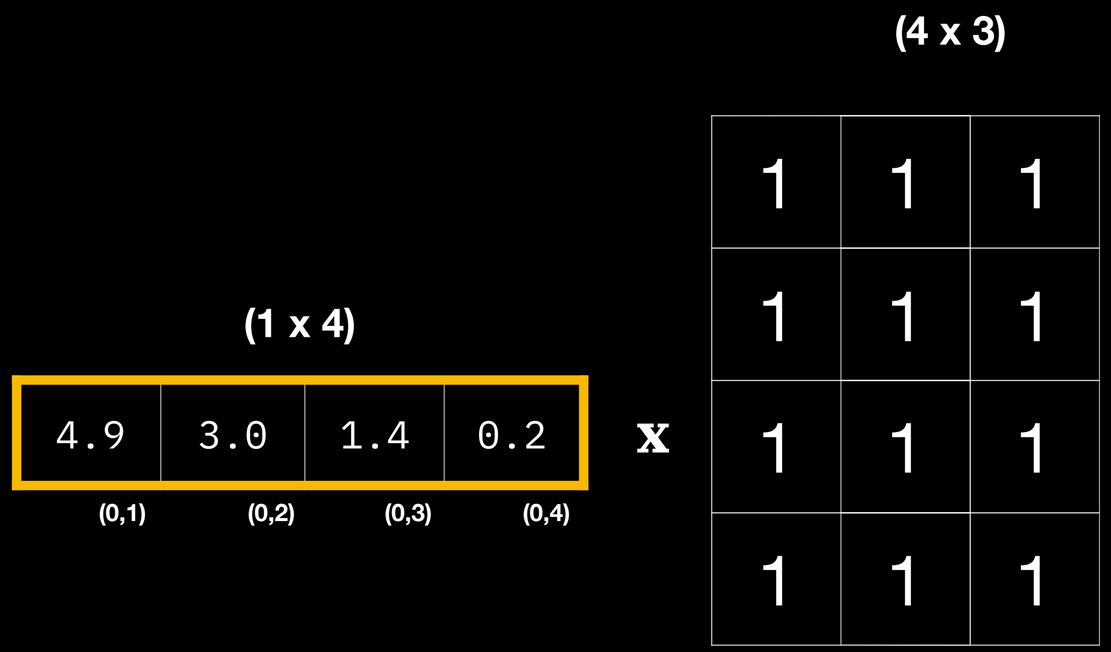
$y = mx + c$

Training Process (Backpropagation)

Step 1 : Initially weights and bias are randomly assigned.
Step 2 : Vector -> Matrix Multiplication -> Some Output (Vector)
Step 3 : Compare with the actual value (Label)
Step 4 : See the difference and pass it back to all the layers behind it.
Step 5 : Update the weights and repeat from Step 2

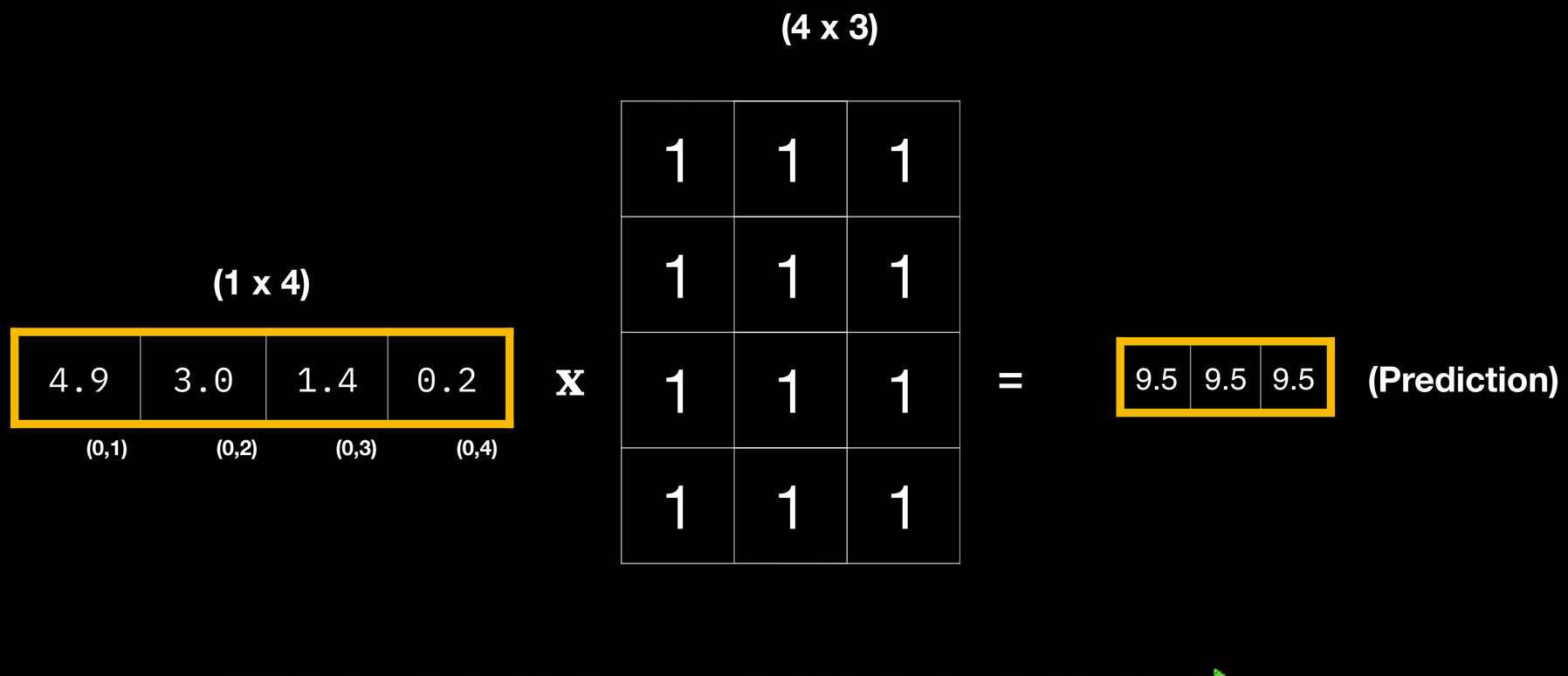
Training Process (Finding the values of W,b)

Step 1 : Initialise the weights to something



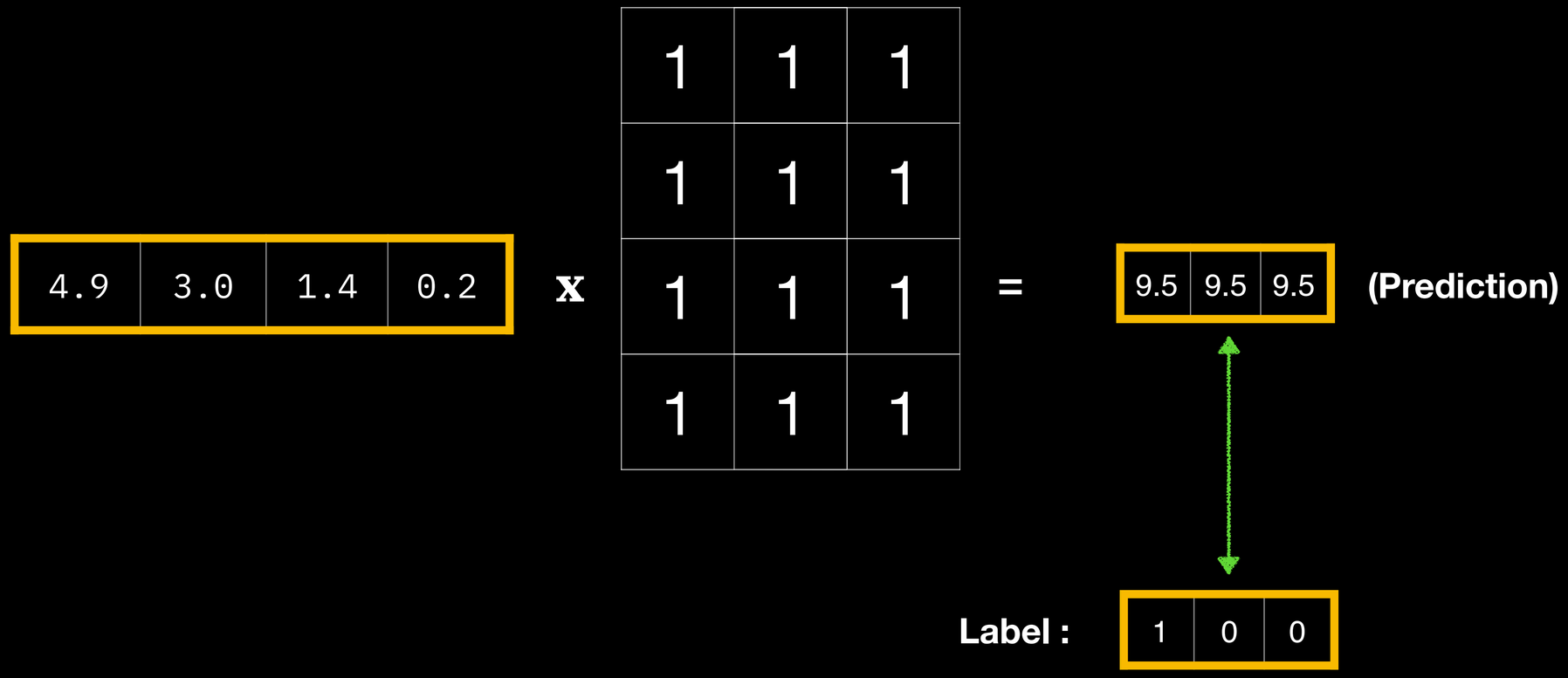
Training Process (Finding the values of W,b)

Step 2 : Do the matrix multiplication with initialised weights.



Training Process (Finding the values of W,b)

Step 3 : Compare with actual value (Label)



Training Process (Finding the values of W,b)

Step 4 : See the difference and pass back the changes (gradients) to previous layers

4.9	3.0	1.4	0.2

x

1	1	1
1	1	1
1	1	1
1	1	1

=

9.5	9.5	9.5

Square Error : Square of difference.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

= 84.25

Label :

1	0	0

Training Process (Finding the values of W,b)

Step 4 : See the difference and pass back the changes (gradients) to previous layers

4.9	3.0	1.4	0.2

x

1	1	1
1	1	1
1	1	1
1	1	1

=

9.5	9.5	9.5

Error = 84.25

Label :

1	0	0

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1})$$

Training Process (Finding the values of W,b)

Step 4 : See the difference and pass back the changes (gradients) to previous layers

4.9	3.0	1.4	0.2

x

1	1	1
1	1	1
1	1	1
1	1	1

=

9.5	9.5	9.5

Error = 84.25

Label :

1	0	0

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1})$$

Gradients (Derivatives)

Training Process (Finding the values of W,b)

Step 5 : Update the Weights

4.9	3.0	1.4	0.2

x

1-δ	1-δ	1-δ
1-δ	1-δ	1-δ
1-δ	1-δ	1-δ
1-δ	1-δ	1-δ

=

9.5	9.5	9.5

Error = 84.25

Label :

1	0	0

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

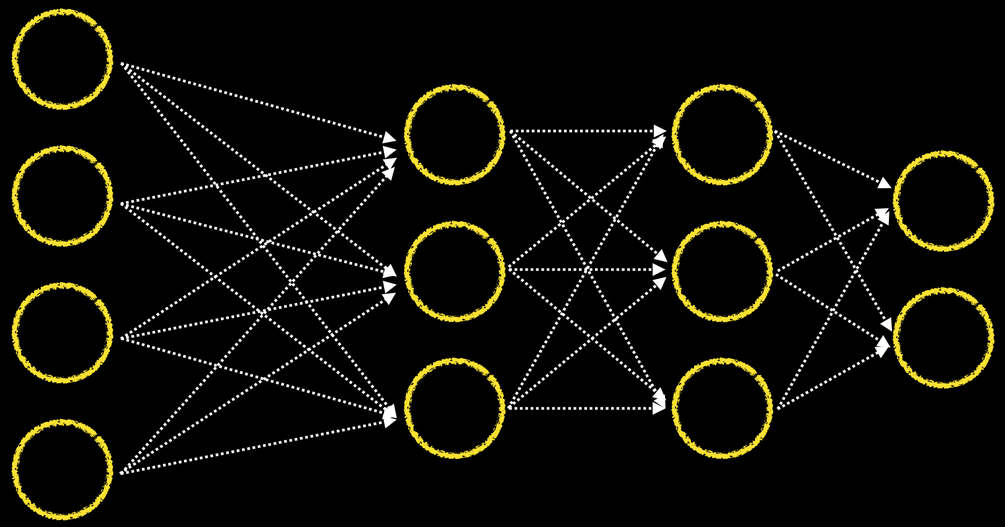
$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1})$$

Gradients (Derivatives)

Subtract from each weight a small fraction of its corresponding derivative (Delta Rule)

Fully Connected Layer



Activations

Input * Weights + Bias = Activations

Activations

Input * Weights + Bias = ~~Activations~~ Pre-Activations

Activations

Input * Weights + Bias = ~~Activations~~ Pre-Activations

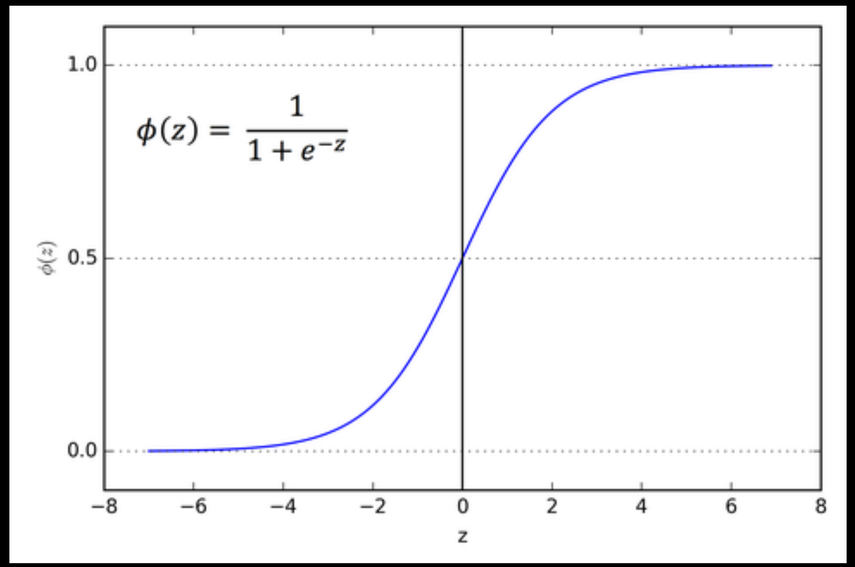
Non Linear (Pre-Activations) = Activations

Activations

Input * Weights + Bias = ~~Activations~~ Pre-Activations

Non Linear (Pre-Activations) = Activations

Most commonly Used :



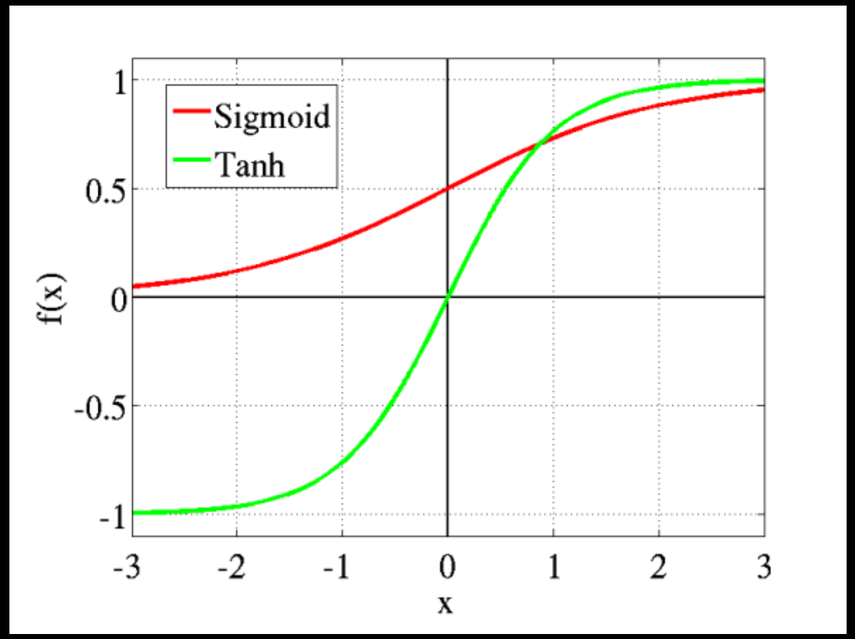
Sigmoid

Activations

Input * Weights + Bias = ~~Activations~~ Pre-Activations

Non Linear (Pre-Activations) = Activations

Most commonly Used :



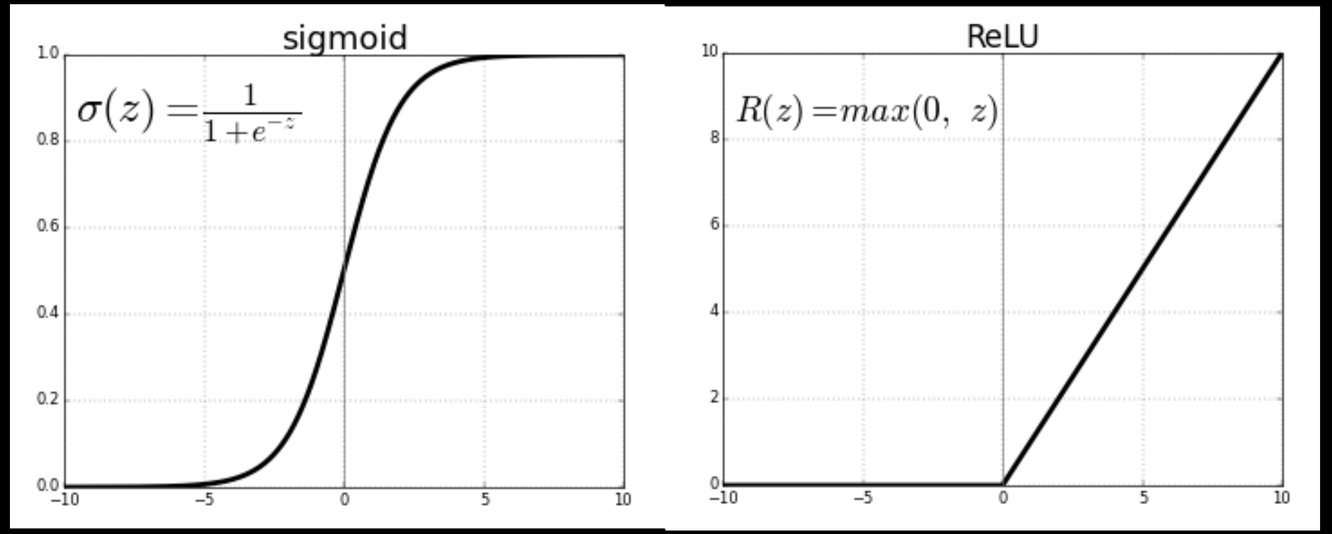
tanh

Activations

Input * Weights + Bias = ~~Activations~~ Pre-Activations

Non Linear (Pre-Activations) = Activations

Most commonly Used :



Rectified Linear Unit

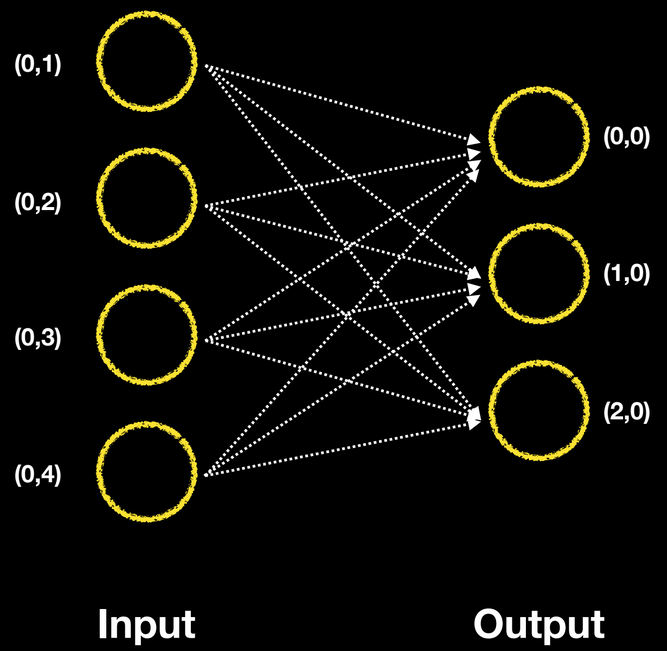
Activations

4.9		
3.0		
1.4		
0.2		

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

=

9.5	$\frac{1}{1+e^{-x}}$	0.9999251537724895
9.5	$\frac{1}{1+e^{-x}}$	0.9999251537724895
9.5	$\frac{1}{1+e^{-x}}$	0.9999251537724895



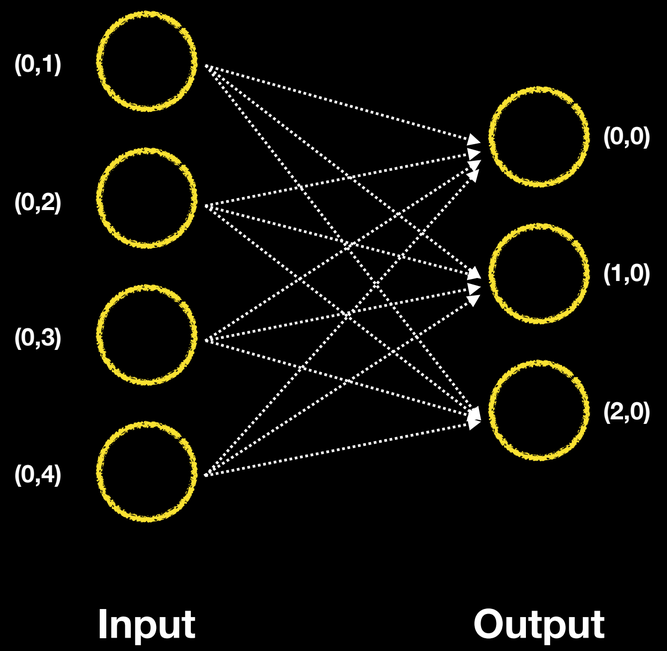
Activations

4.9		
3.0		
1.4		
0.2		

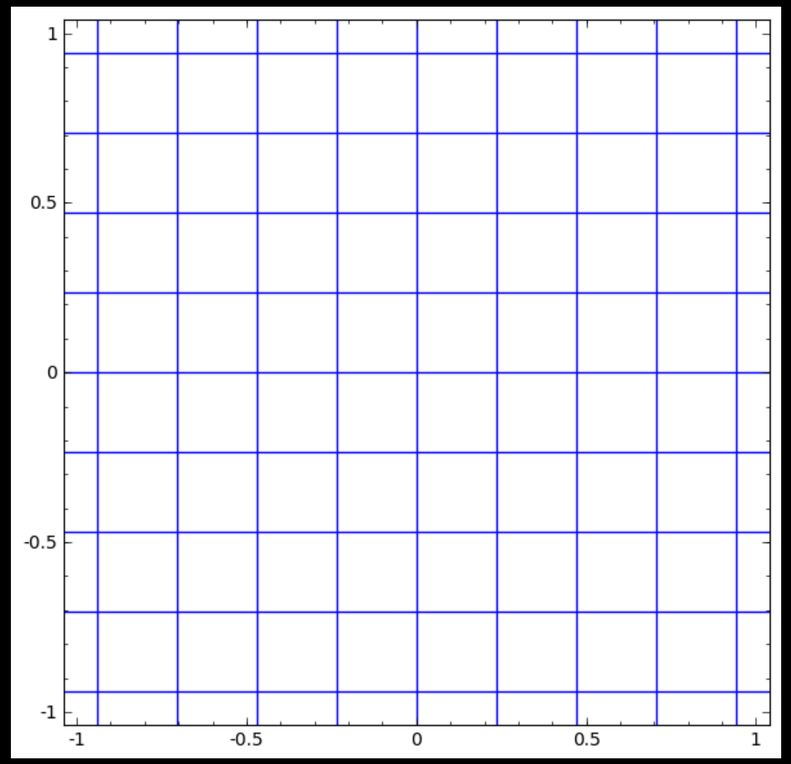
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

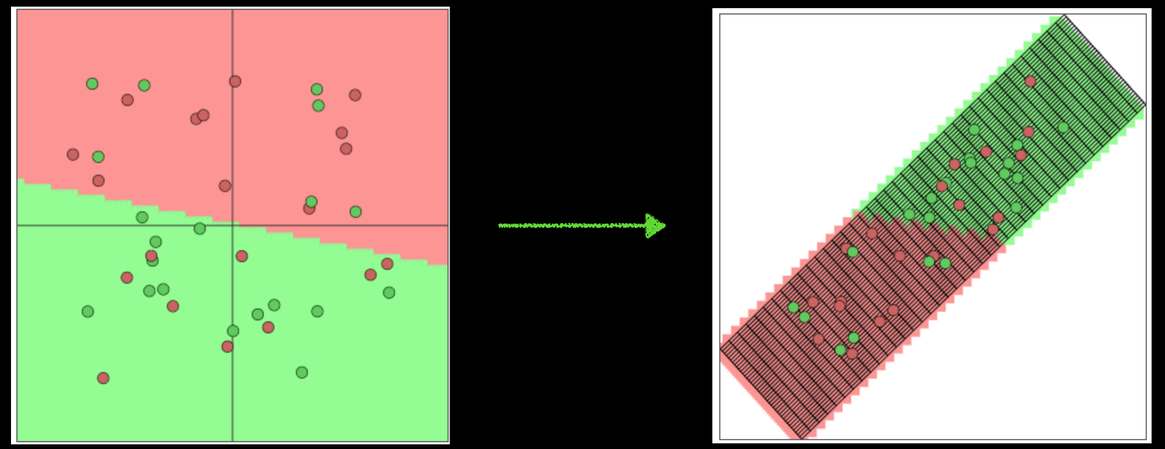
=

9.5	$\frac{1}{1+e^{-x}}$	0.9999251537724895
9.5	$\frac{1}{1+e^{-x}}$	0.9999251537724895
9.5	$\frac{1}{1+e^{-x}}$	0.9999251537724895



Data Space Transformation

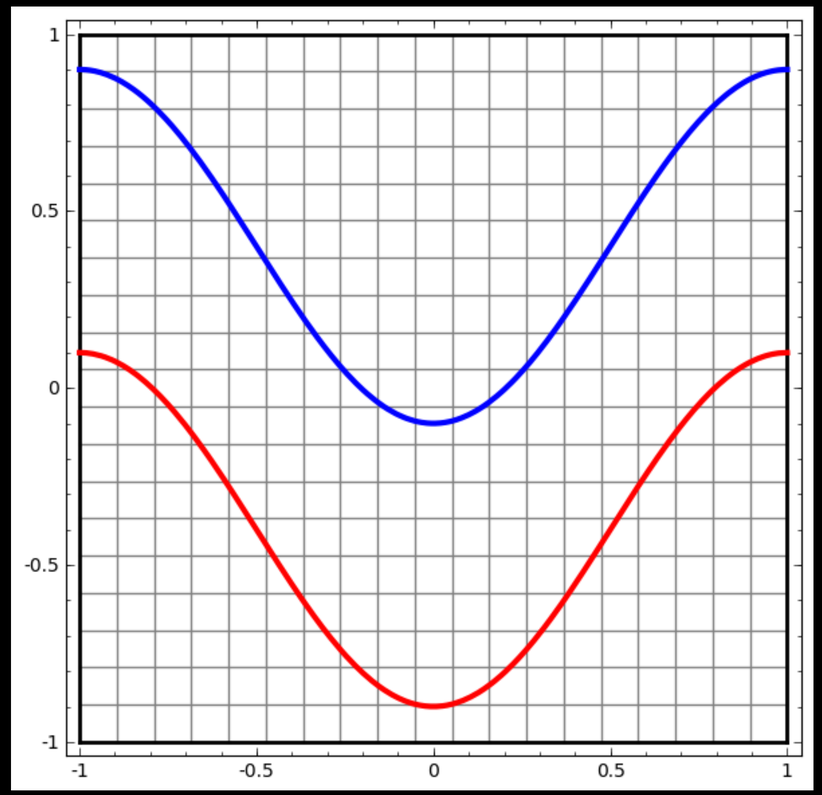


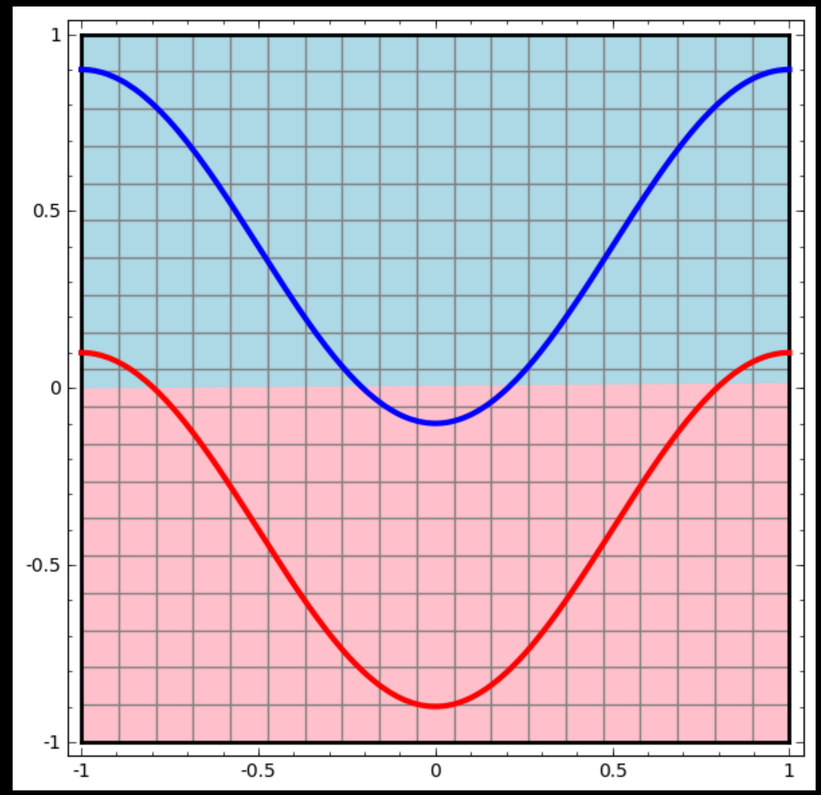


// Chris Olah Blog (Neural Networks, Manifolds and Topology)
// Karpathy ConvNetJS https://cs.stanford.edu/people/karpathy/convnetjs//demo/classify2d.html

Data Space Transformation

Without the non linear function : Not Linearly Separable

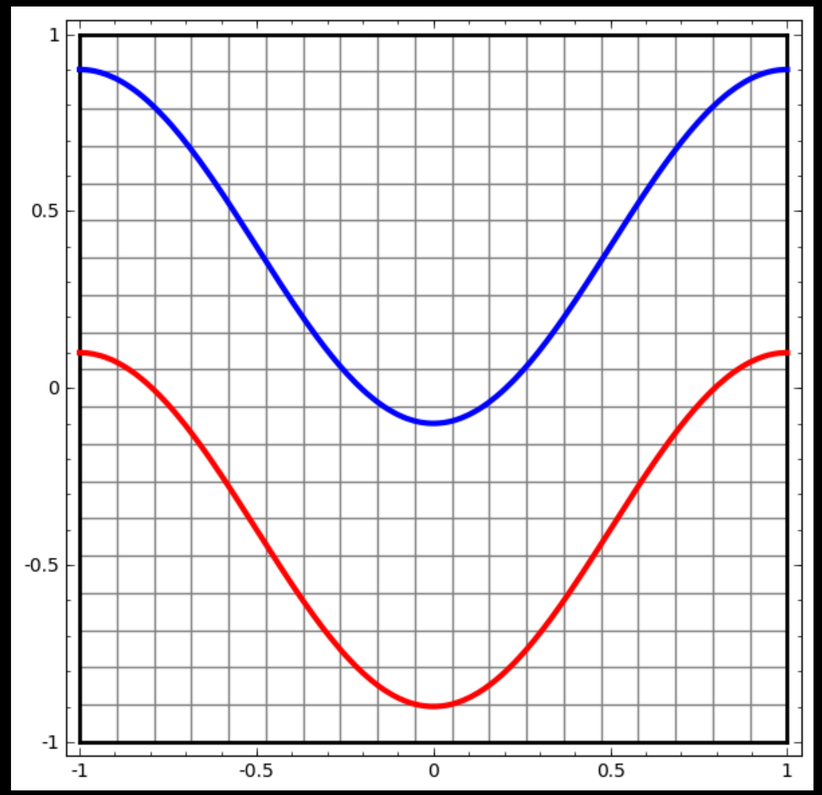


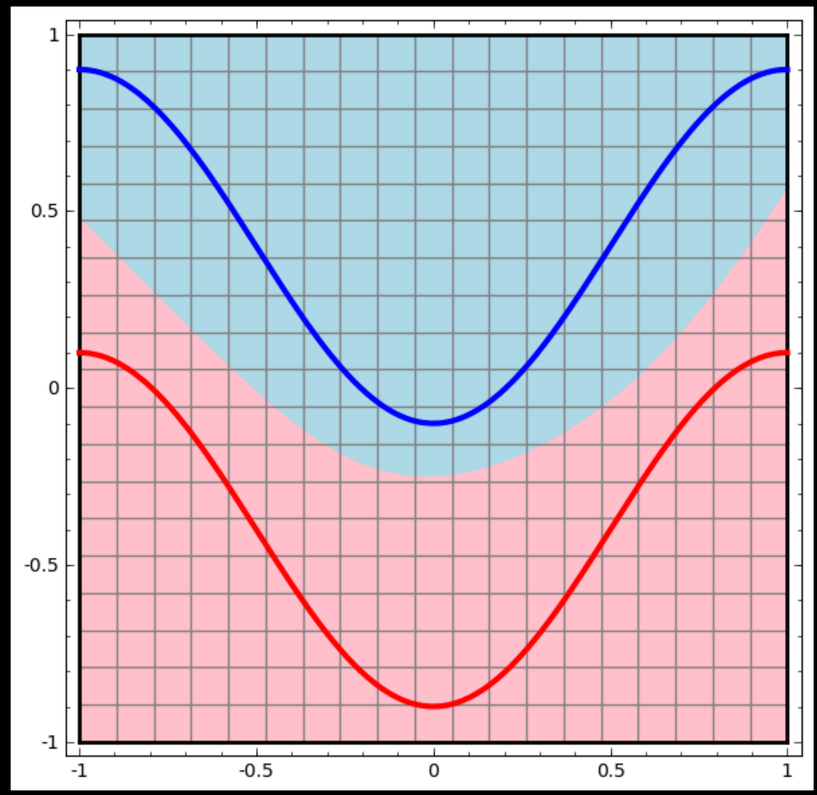


// Chris Olah Blog (Neural Networks, Manifolds and Topology)

Data Space Transformation

What we want

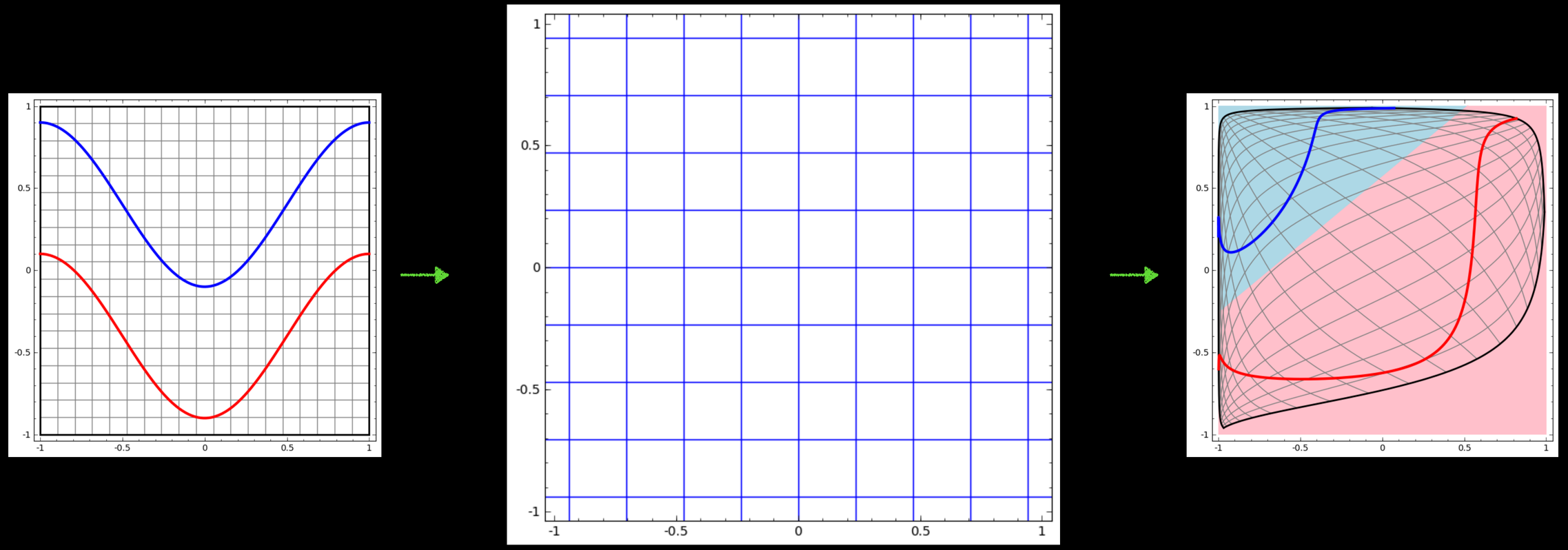




// Chris Olah Blog (Neural Networks, Manifolds and Topology)

Data Space Transformation

With the non linear function : **Linearly Separable but in a different space (After transformation)**



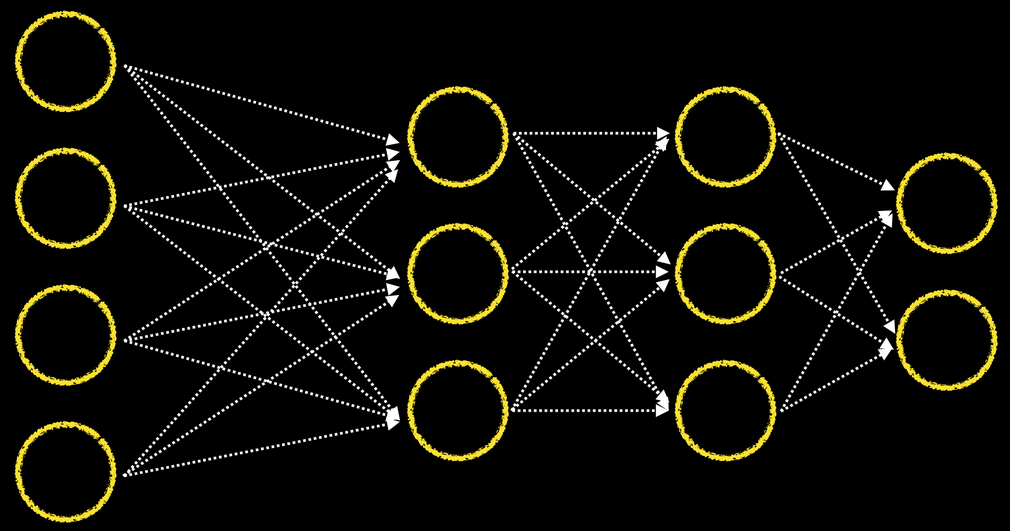
// Chris Olah Blog (Neural Networks, Manifolds and Topology)

Data Space Transformation

```
// https://cs.stanford.edu/people/karpathy/convnetjs//demo/classify2d.html
```

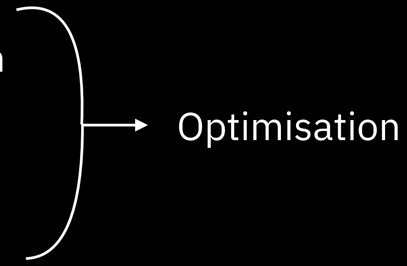
Summary

Step 1 : Initially weights and bias are randomly assigned.
Step 2 : Vector -> Matrix Multiplication -> Some Output (Vector) -> Non-Linear Activation
Step 3 : Calculate Loss (Mean Square Error)
Step 4 : Backpropagate (Pass the Gradients to prior weights)
Step 5 : Update the weights and repeat from Step 2



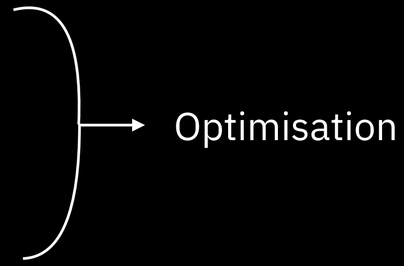
Keras Code

Step 1 : Initially weights and bias are randomly assigned.
Step 2 : Vector -> Matrix Multiplication -> Some Output (Vector) -> Non-Linear Activation
Step 3 : Calculate Loss (Mean Square Error)
Step 4 : Backpropagate (Pass the Gradients to prior weights)
Step 5 : Update the weights and repeat from Step 2



Keras Code

Step 1 : Initially weights and bias are randomly assigned.
Step 2 : Vector -> Matrix Multiplication -> Some Output (Vector) -> Non-Linear Activation
Step 3 : Calculate Loss (Mean Square Error)
Step 4 : Backpropagate (Pass the Gradients to prior weights)
Step 5 : Update the weights and repeat from Step 2



- Packages like Tensorflow, Pytorch, Keras are all equipped with optimisers.
- There are several advanced optimisers besides what we just learnt.

```
// coding time
// use pip or conda (if using anaconda) to install the packages
// avoid using gpu for tutorials if you don't want to spend eternity configuring cuda.
```

Any Questions?

email id : harisris@gmail.com