

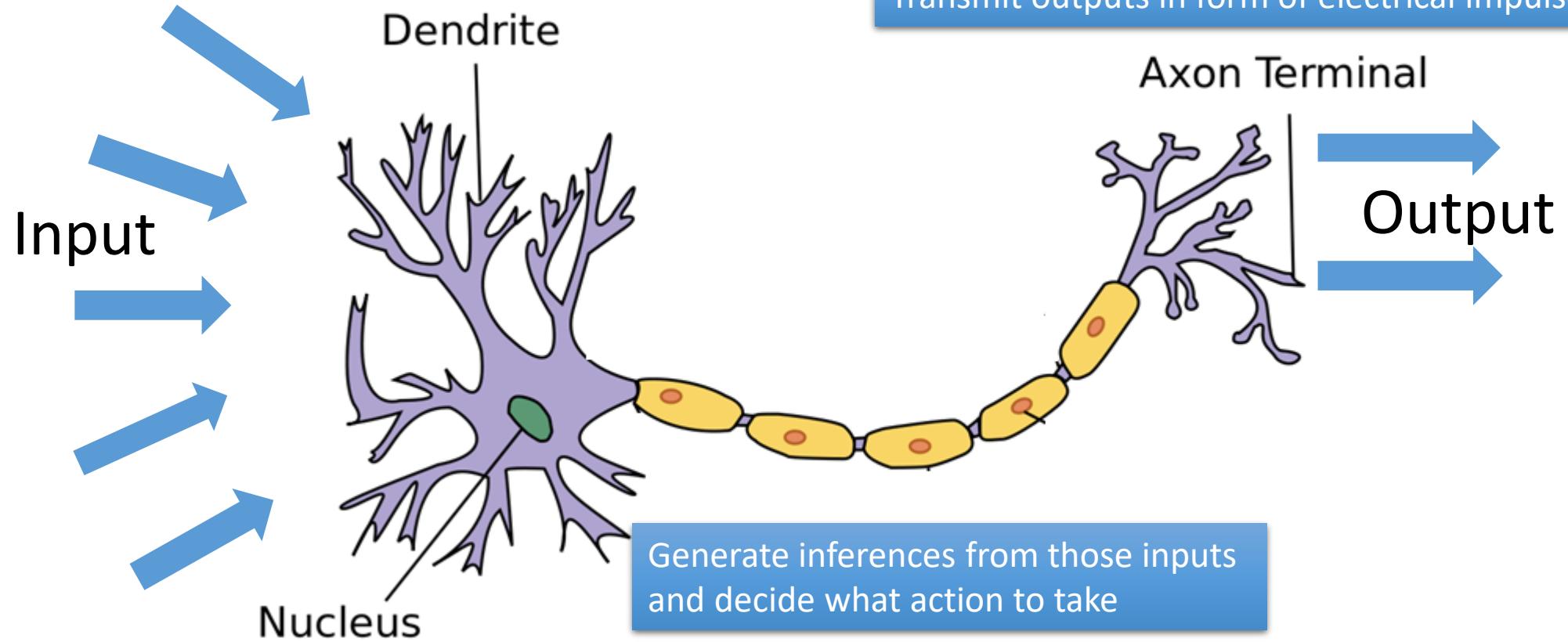
Complete Data Science and Machine Learning Using Python

By
Jitesh Khurkhuriya

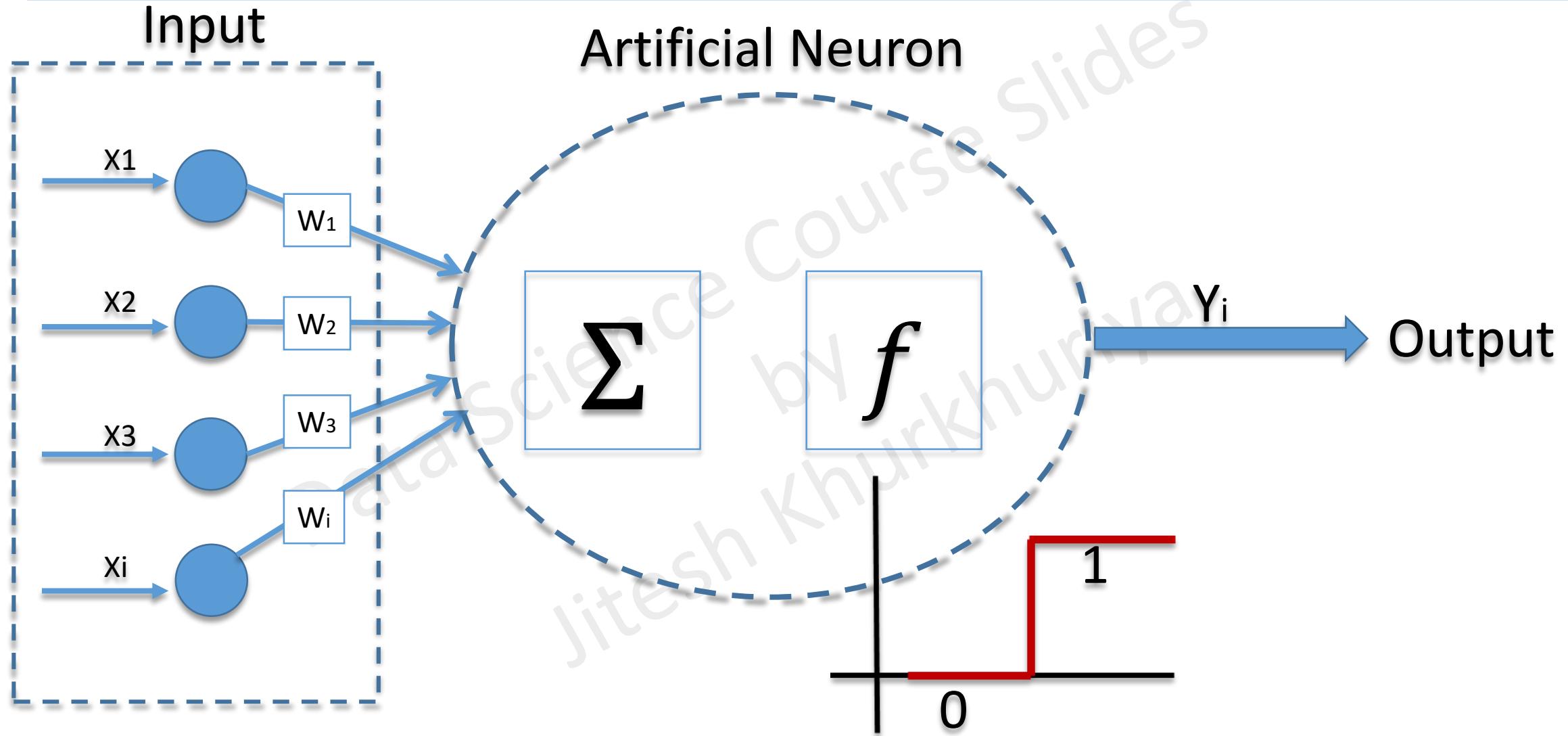
Neural Network

How Neuron Works?

It takes input from other neurons in the form of an electrical impulse.



Artificial Neuron

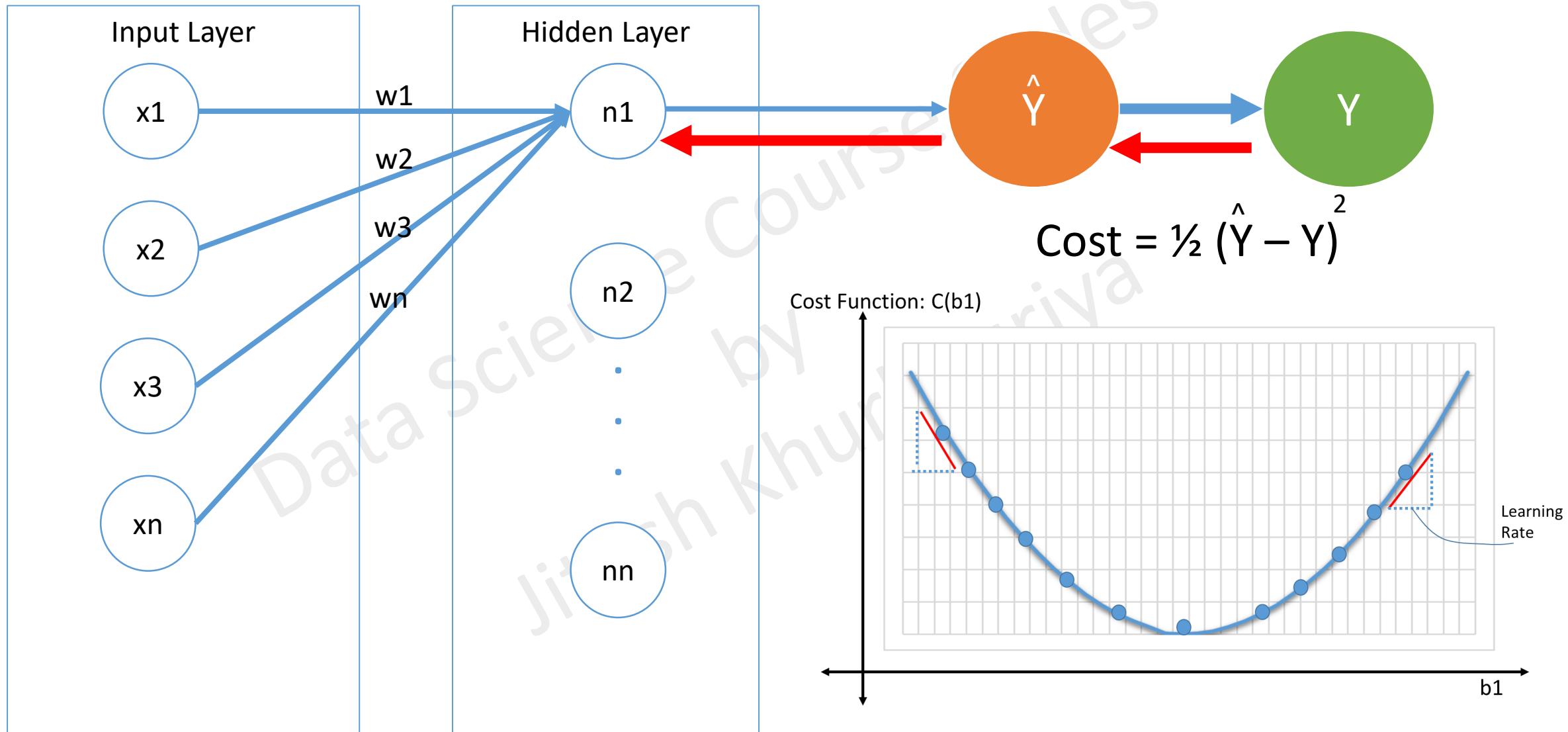


ANN Capabilities

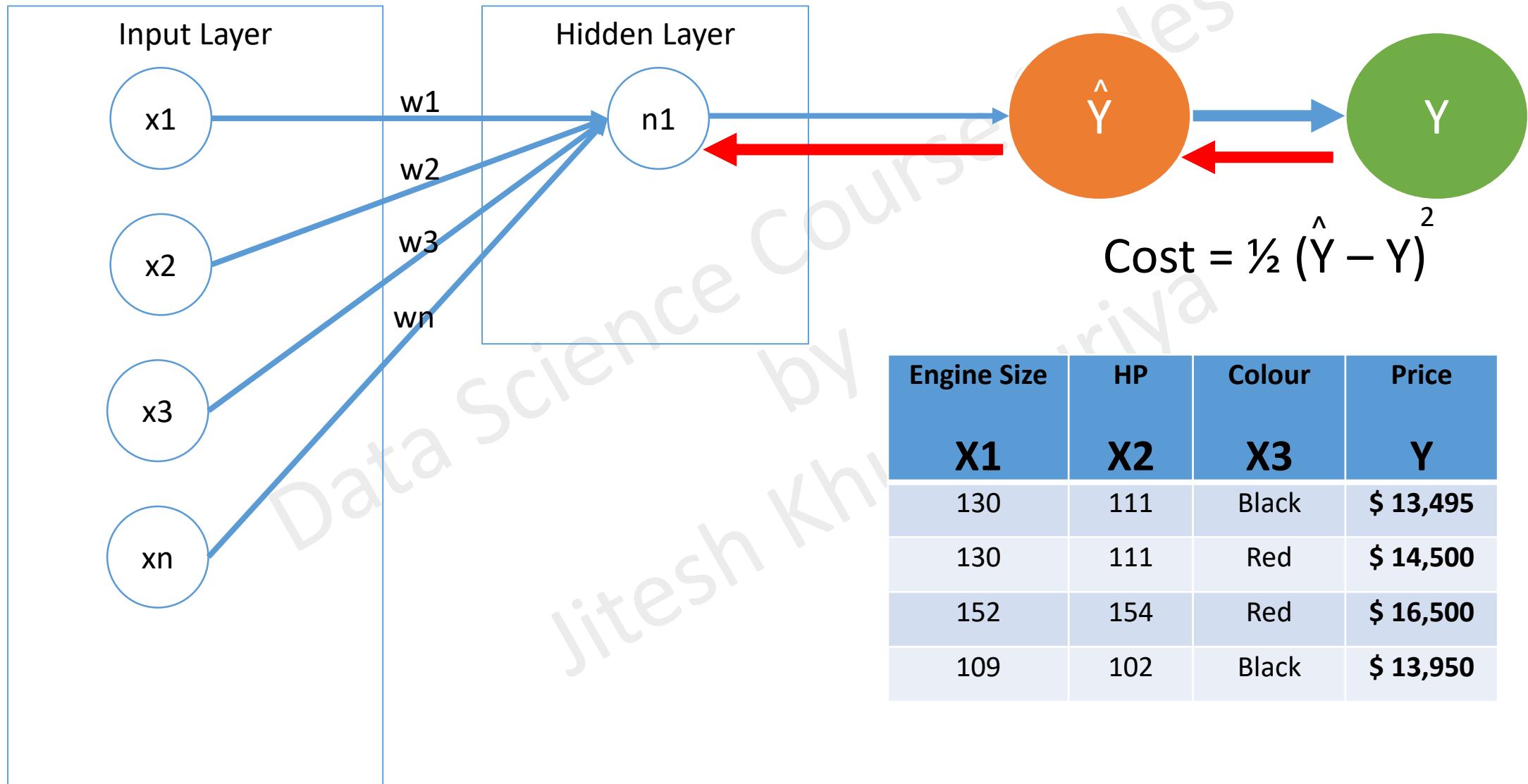
- Exploits the Non-Linear behaviour
- Guided Learning
- Can change with the surrounding environment
- Confidence Measure
- Large Scale Integrations are possible
- Massively Parallel processing ensures Fault Tolerance

How ANN Works?

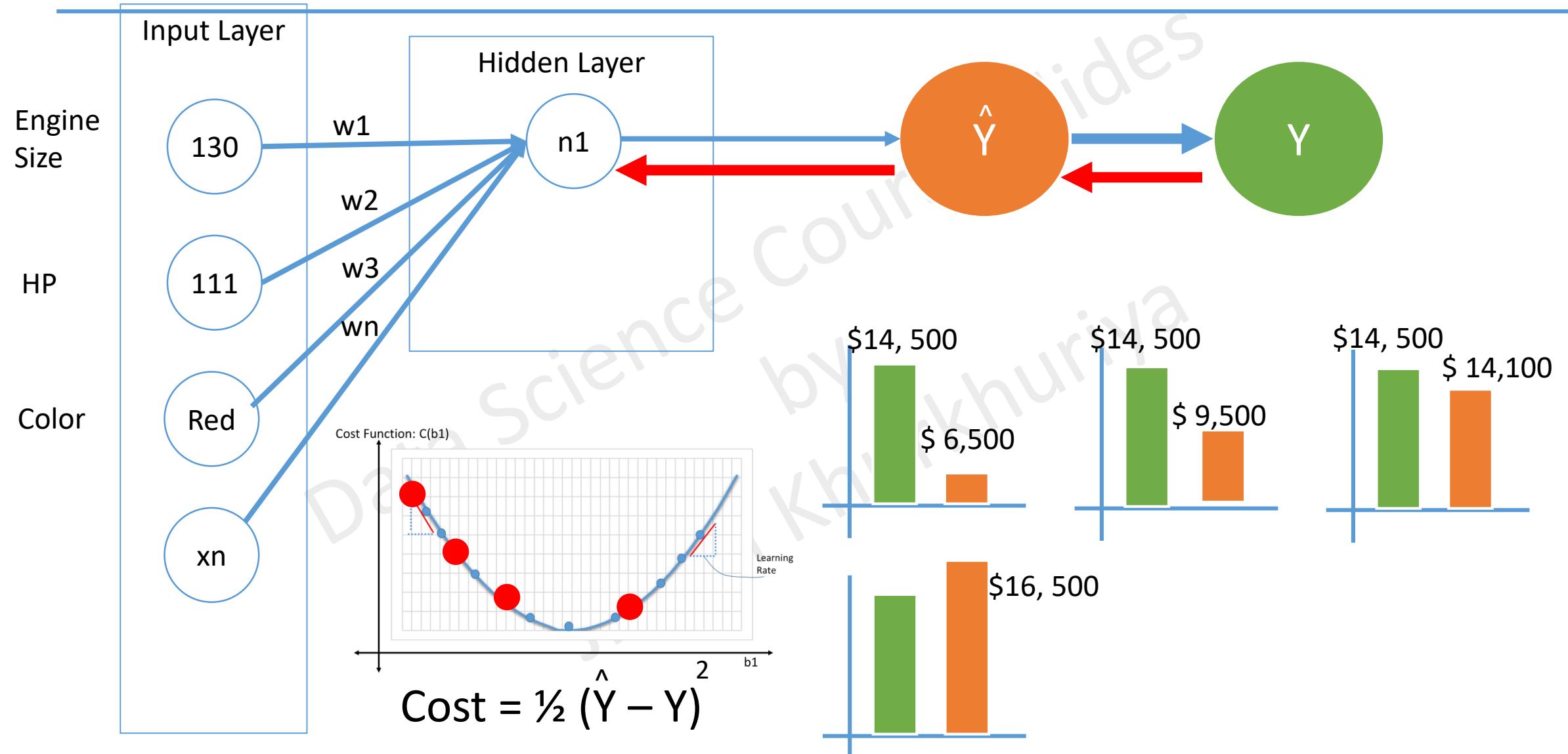
How ANN Works?



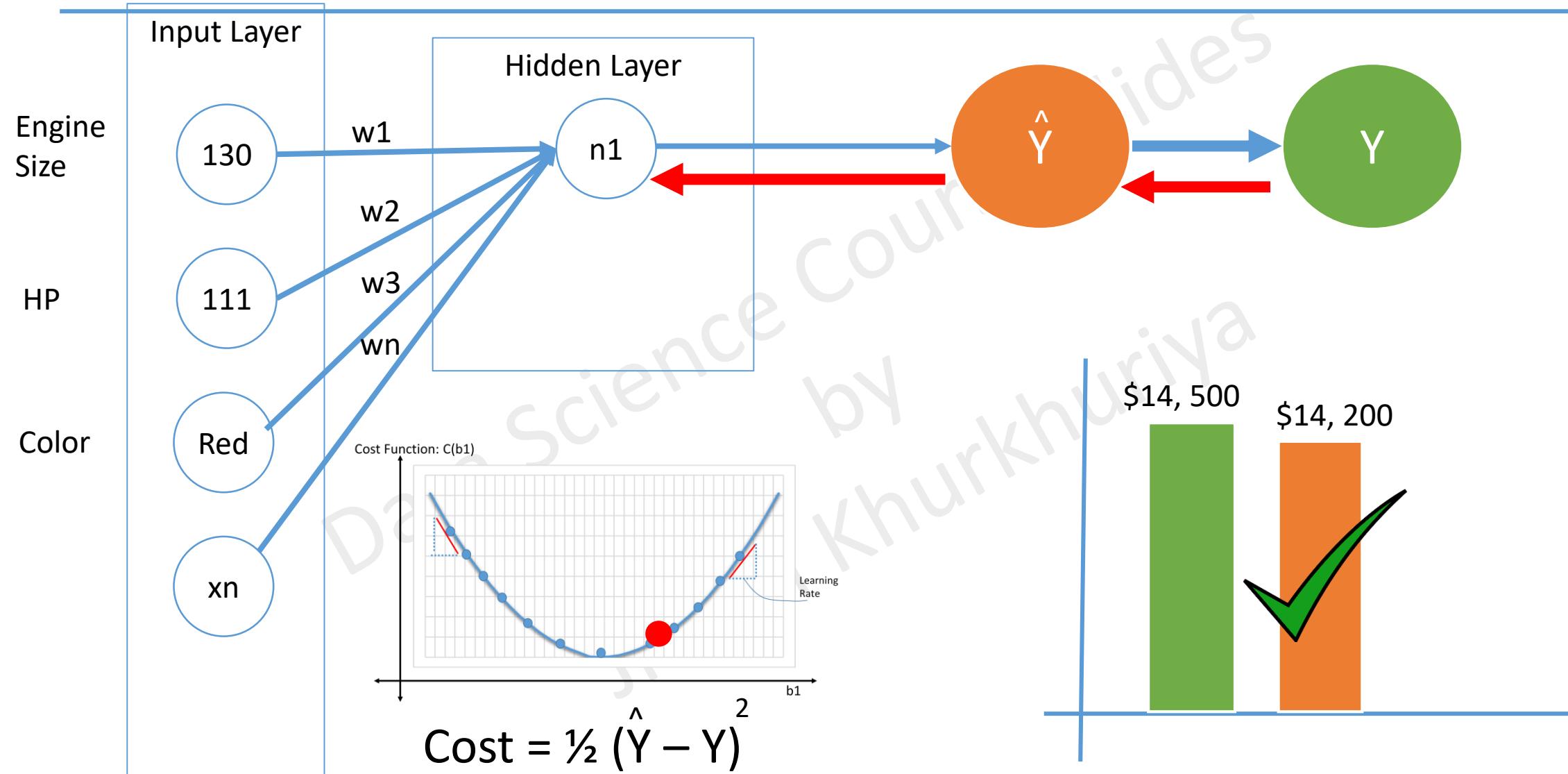
How ANN Works?



How ANN Works?



How ANN Works?

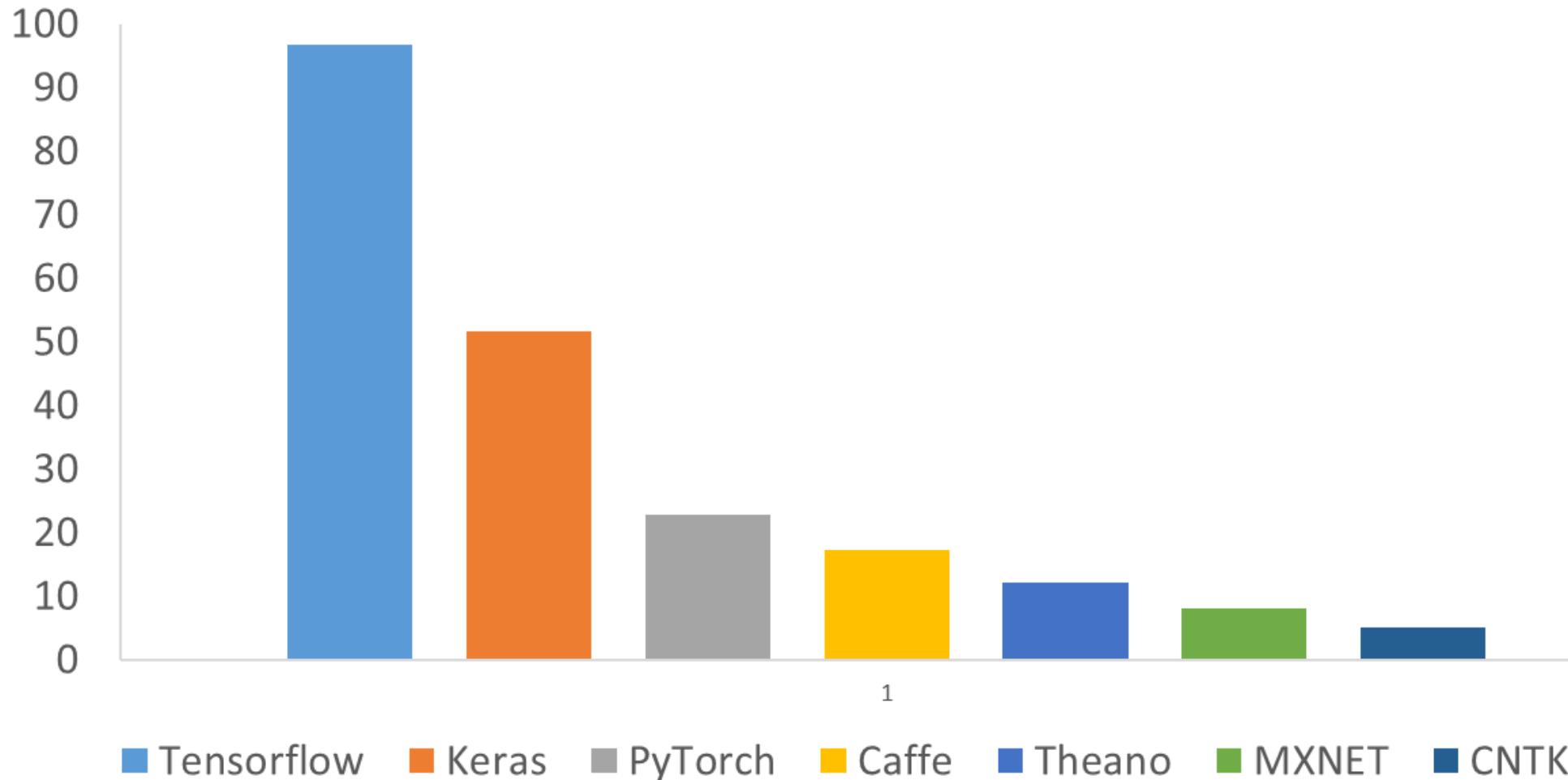


Deep Learning Frameworks

Deep Learning Frameworks



Deep Learning Framework Ranking



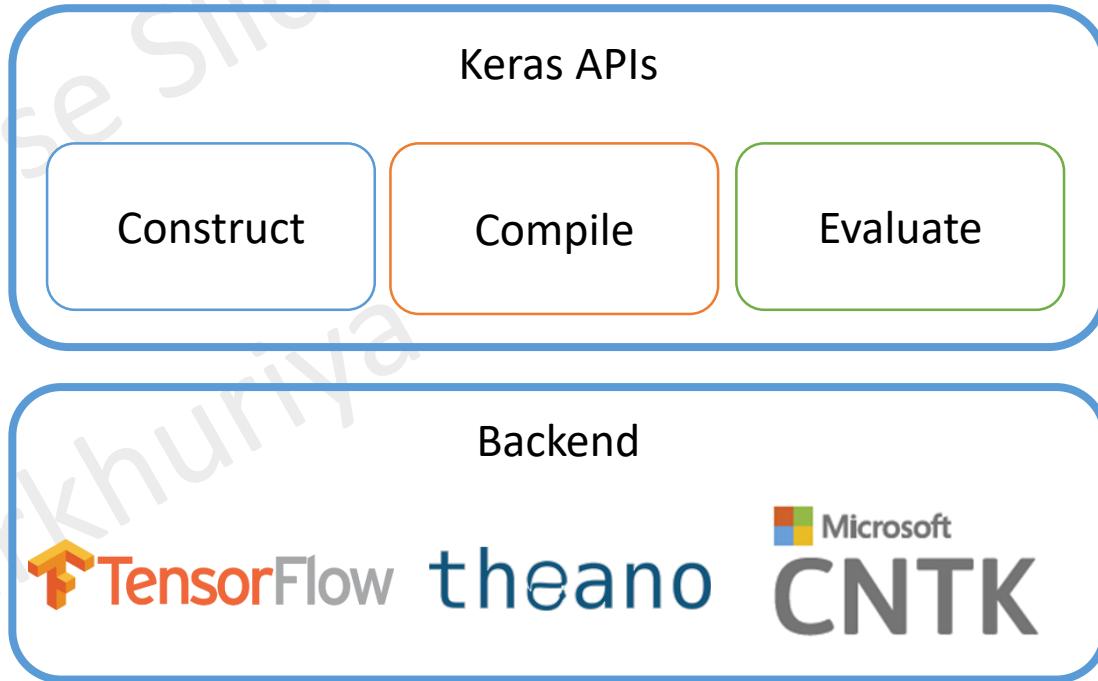
What is TensorFlow?

<https://www.tensorflow.org/>

- Machine Learning Library for neural networks
- Developed by Google
- Open Sourced in 2015 with release 1.0.0 in 2017
- Supports both the CPU as well as GPU processing
- Multi-platform
- Best supported language is Python
- Uses Tensor as the data structure

What is Keras? <https://keras.io/>

- High Level neural network APIs
- Can run on top of TensorFlow, Theano and CNTK
- Compatible for Python 2.7 to 3.6
- Built for User Friendliness, Modularity, Easy Extensibility



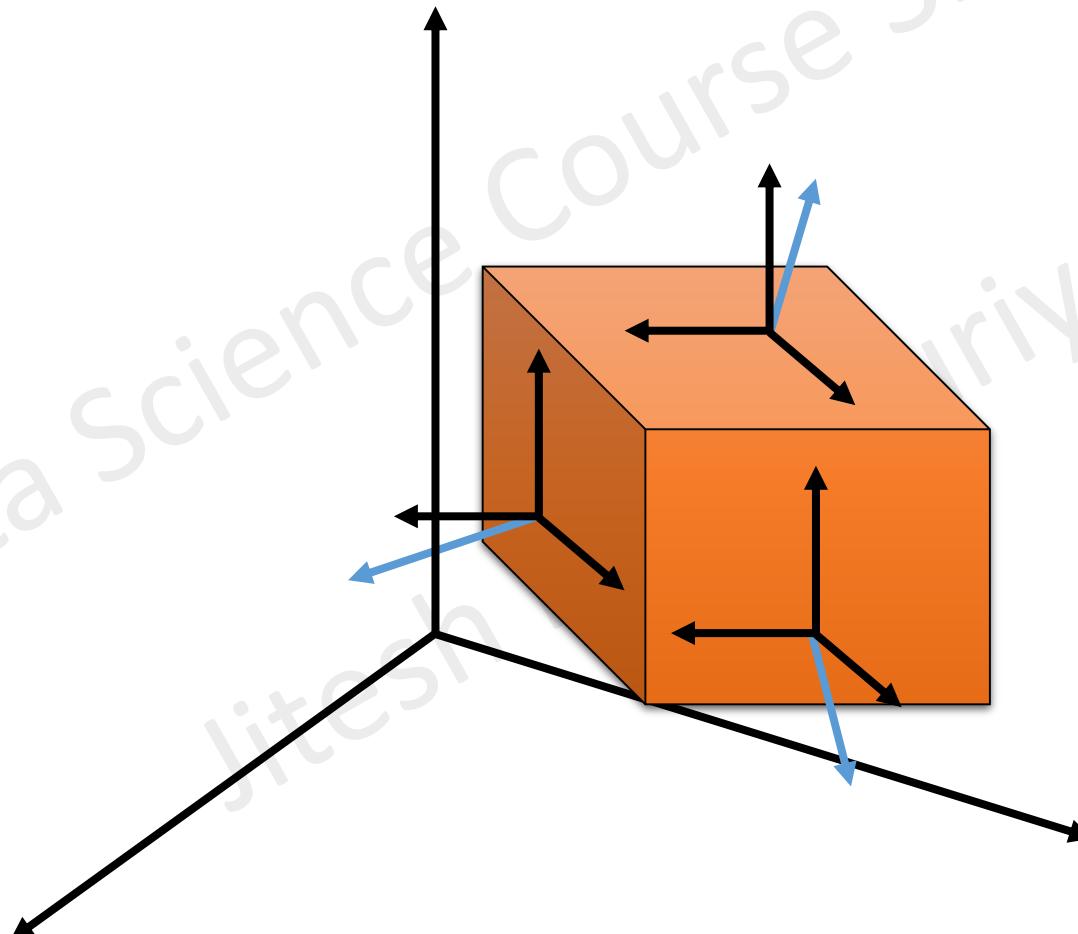
What is a Tensor?

What is a Tensor – Various Definitions

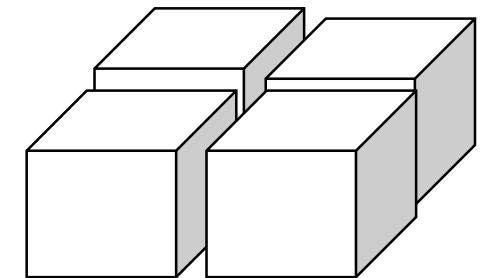
Geometric Object

Stress Tensor

Generalization of Matrix

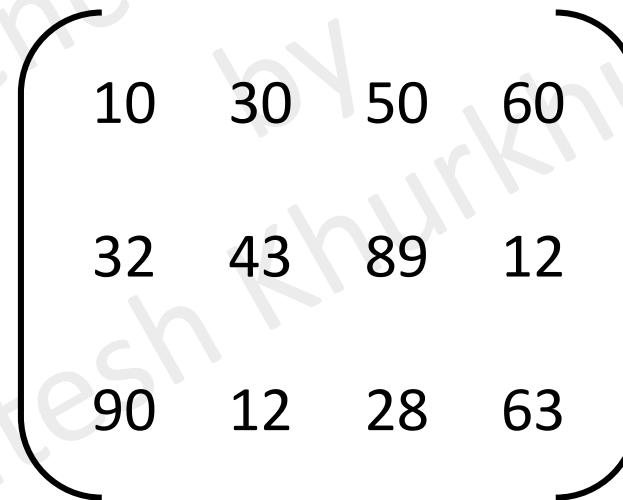
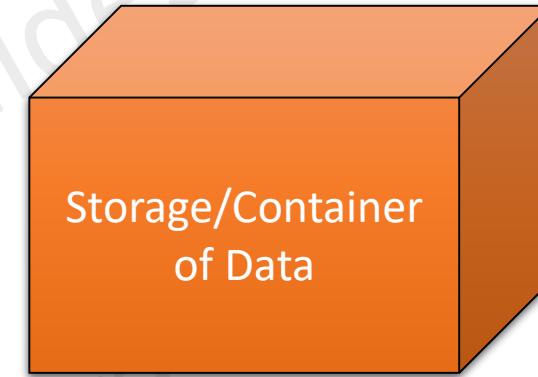


$$\begin{pmatrix} 10 & 30 & 50 & 60 \\ 32 & 43 & 89 & 12 \\ 90 & 12 & 28 & 63 \end{pmatrix}$$



What is a Tensor in TensorFlow?

- n-dimensional array of base datatype
- Number of dimensions define the rank of the Tensor
- Shape



Tensor – Dimensions

Rank	Mathematical Entity
0	Scalar

Rank	Mathematical Entity
1	Vector

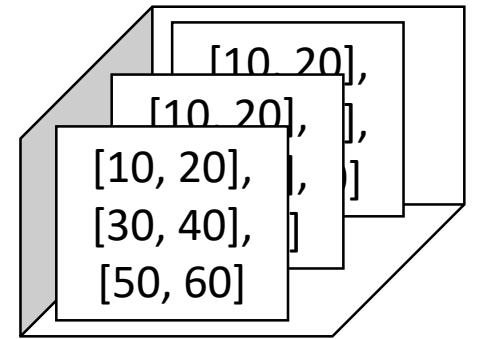
Rank	Mathematical Entity
2	Matrix

Rank	Mathematical Entity
3	Tensor

10

[10, 20]

[
[10, 20],
[30, 40],
[50, 60]
]



Tensor – Dimensions

Rank	Mathematical Entity
0	Scalar

Rank	Mathematical Entity
1	Vector

Rank	Mathematical Entity
2	Matrix

Rank	Mathematical Entity
3	Tensor

10

[10, 20]

[
[10, 20],
[30, 40],
[50, 60]
]

[
[
[10, 20],
[30, 40],
[50, 60]
,
[
[],
[],
[]
]
]

Tensor – Shapes

Rank	Mathematical Entity
0	Scalar

Rank	Mathematical Entity
1	Vector

Rank	Mathematical Entity
2	Matrix

Rank	Mathematical Entity
3	Tensor

10

[10, 20]

()

[
[10, 20],
[30, 40],
[50, 60]
]
2

(2,)

(3, 2)

3
[
[10, 20],
[30, 40],
[50, 60]
]
3
2

(3, 3, 2)

Tensor – Examples

wheelbase	length	enginesize	horsepower	price
88.6	168.8	130	111	13495
88.6	168.8	130	111	16500
94.5	171.2	152	154	16500
99.8	176.6	109	102	13950
99.4	176.6	136	115	17450
99.8	177.3	136	110	15250
105.8	192.7	136	110	17710
105.8	192.7	136	110	18920
105.8	192.7	131	140	23875

5

9

Rank of this Tensor → 2

Shape of this Tensor → (9,5)

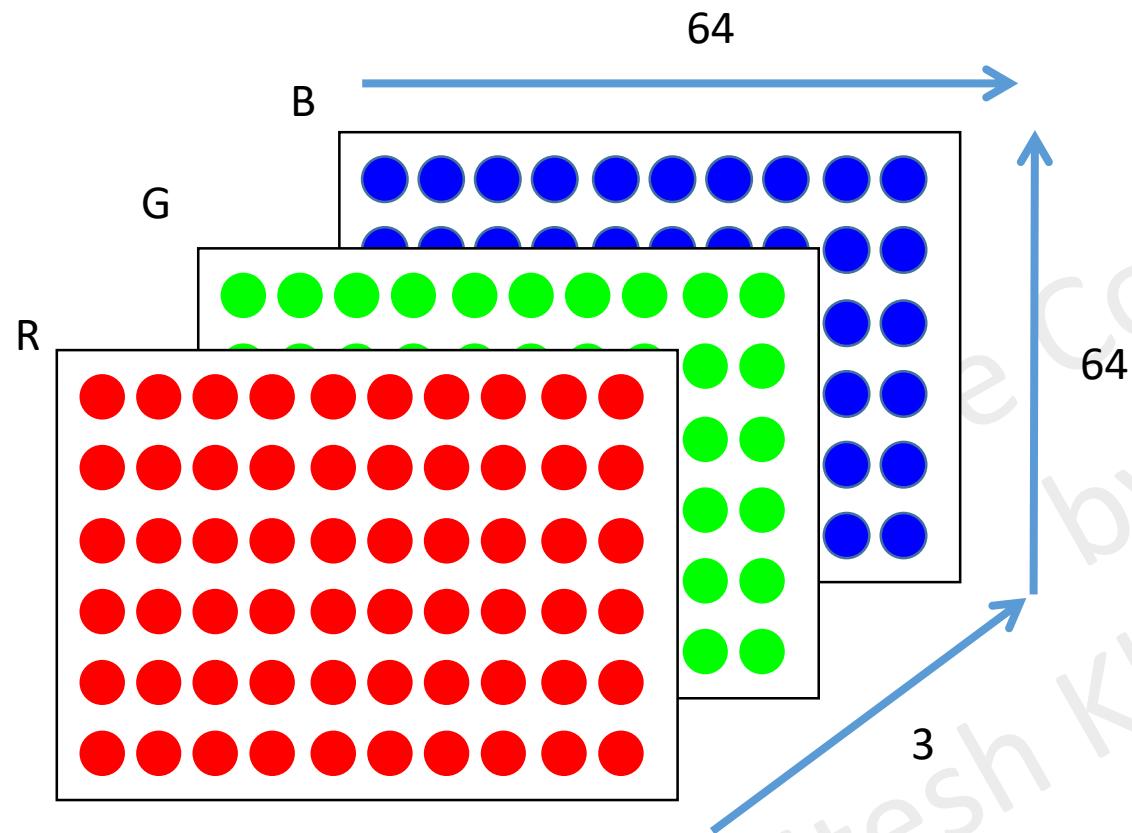
Tensor – Examples

						6
hour	holiday	weekday	humidity	windspeed	demand	
0	0	6	0.81	0	16	
1	0	6	0.8	0	40	40
2	0	6	0.8	0	32	32
3	0	6	0.75	0	13	13
.....
.....	2
22	0	6	0.8	0	2	2
23	0	6	0.86	0	3	3

Rank of this Tensor → 3

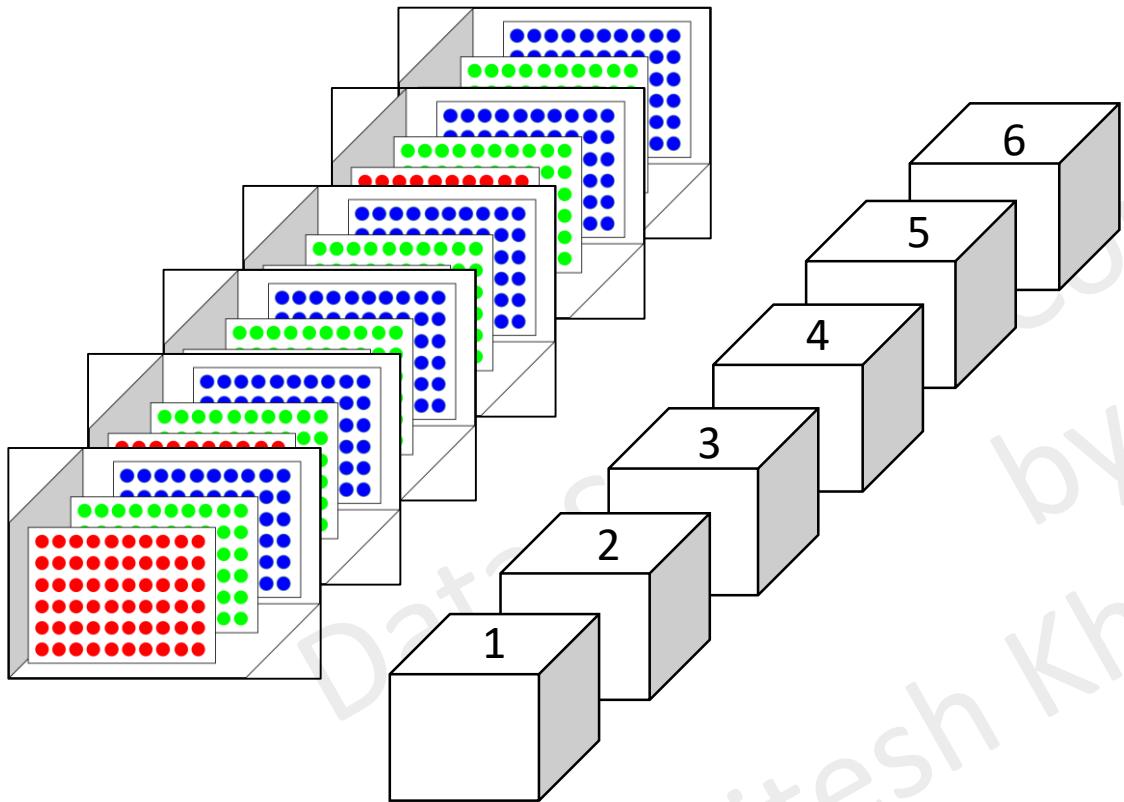
Shape of this Tensor → (4, 24, 6)

Tensor – Examples



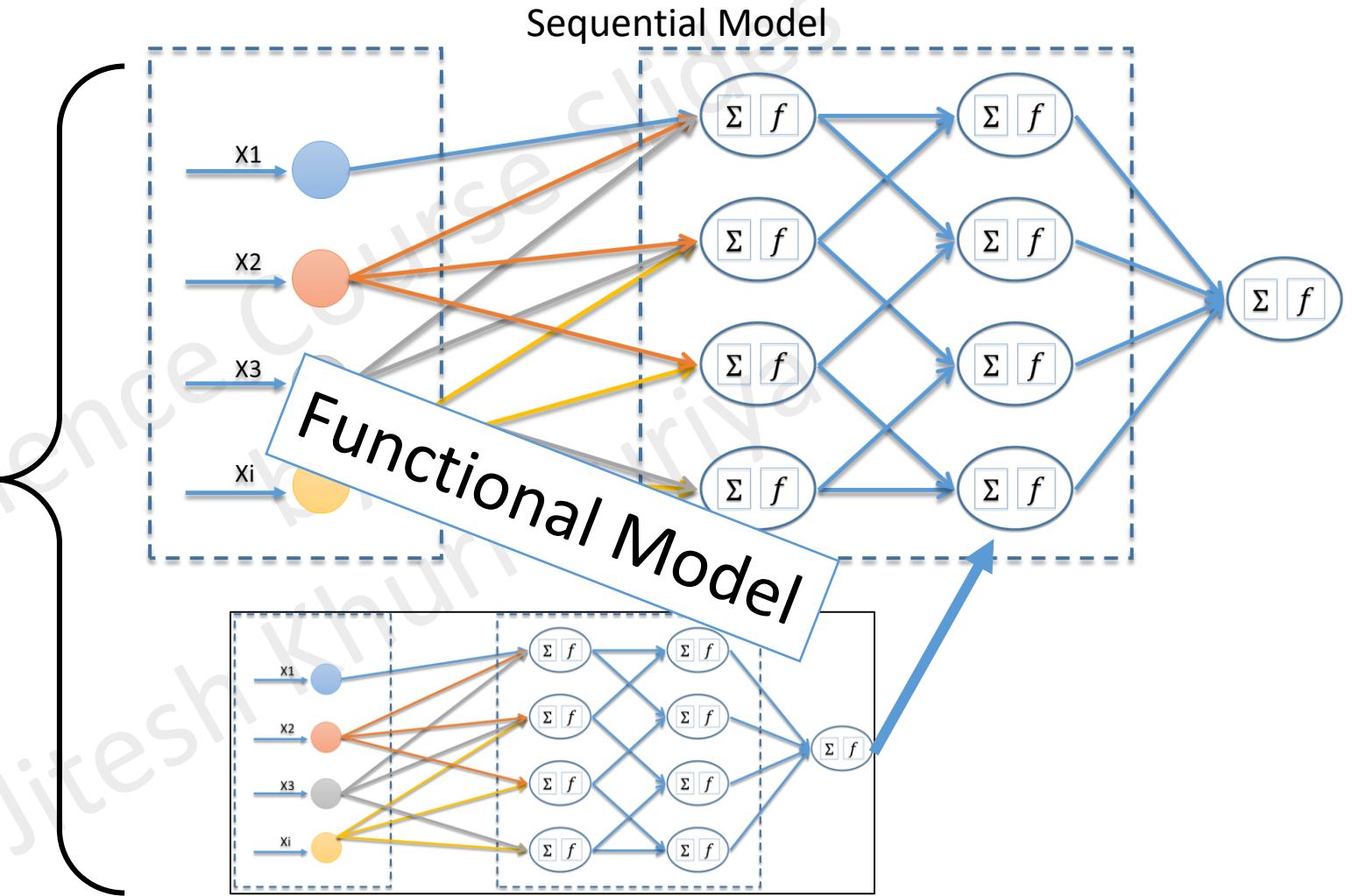
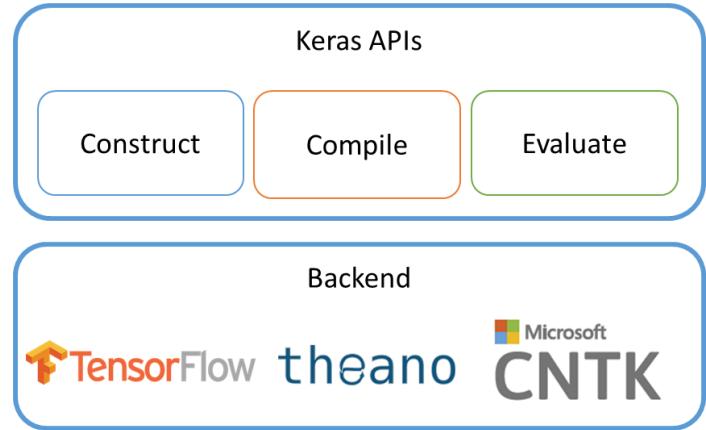
Shape of this Tensor $\rightarrow (3, 64, 64)$

Tensor – Examples

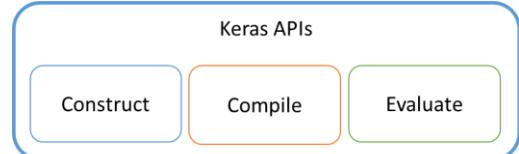


Shape of this Tensor → (6, 3, 64, 64)

Model Types

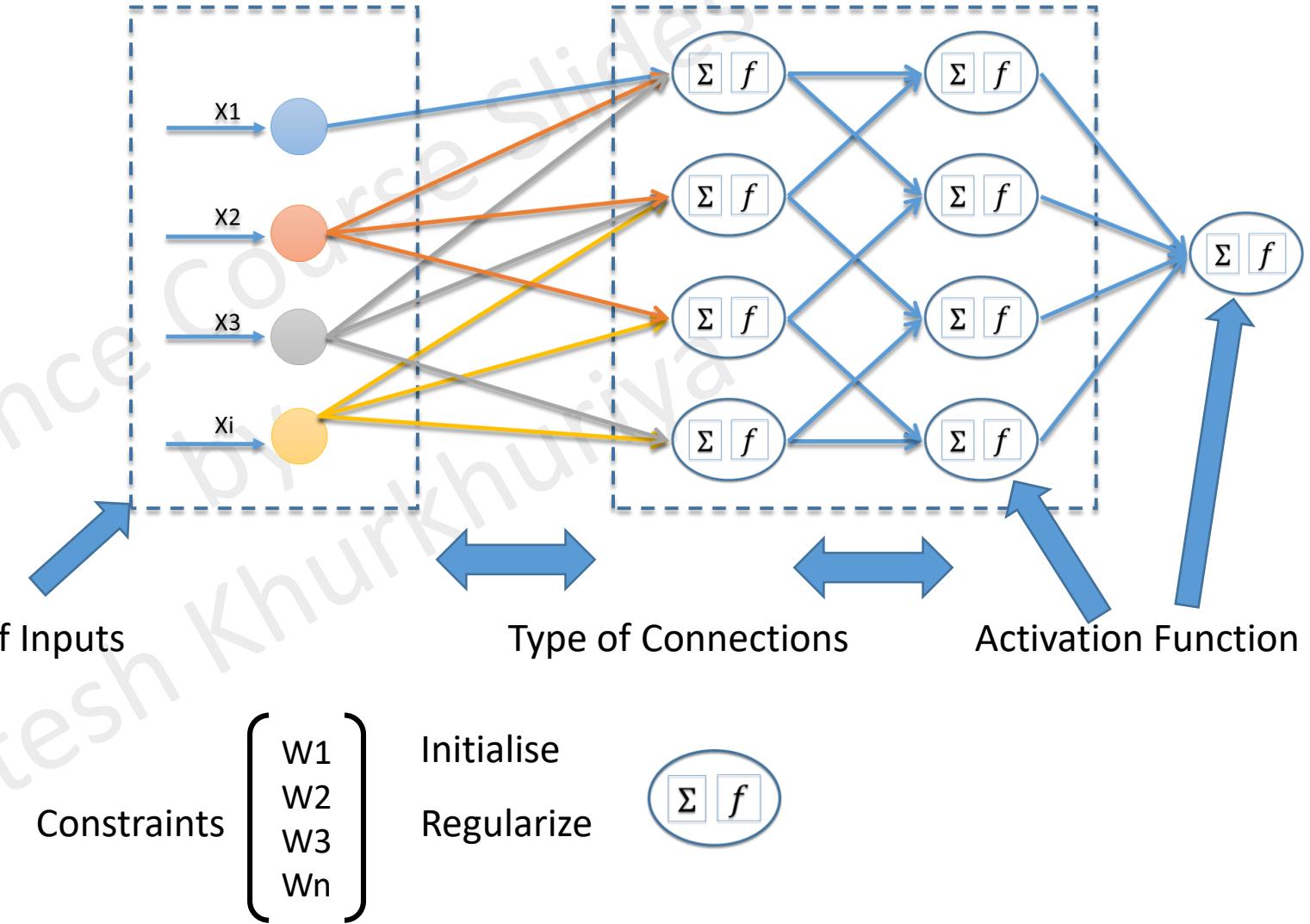


Model Building Steps

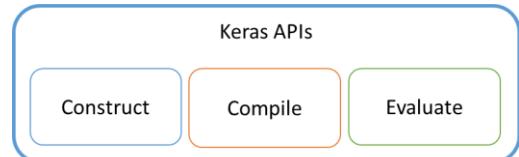


Step 1 – Specify/Add Layers

Step 2 – Define Layer Characteristics



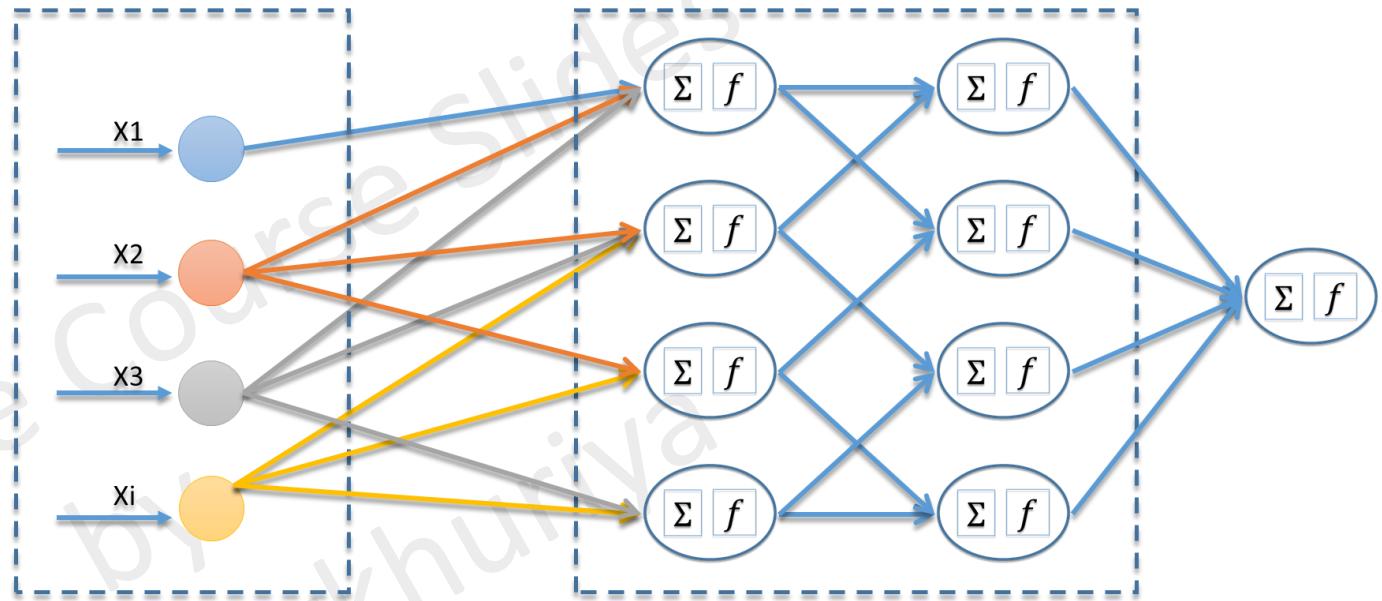
Model Building Steps



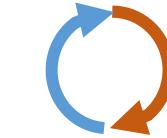
Step 1 – Specify/Add Layers

Step 2 – Define Layer Characteristics

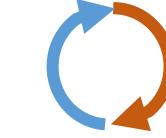
Step 3 – Compile the Model



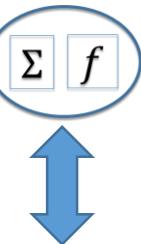
$$\begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_n \end{bmatrix}$$



Regularize



Optimize



Σf

Measure Loss

Model Building Steps

Step 1 – Specify/Add Layers

Number of Layers

Types of Layers

Step 2 – Define Layer Characteristics

Number of Inputs

Constraints

Type of Connections

Initialise

Activation Function

Regularize

Step 3 – Compile the Model

Optimize

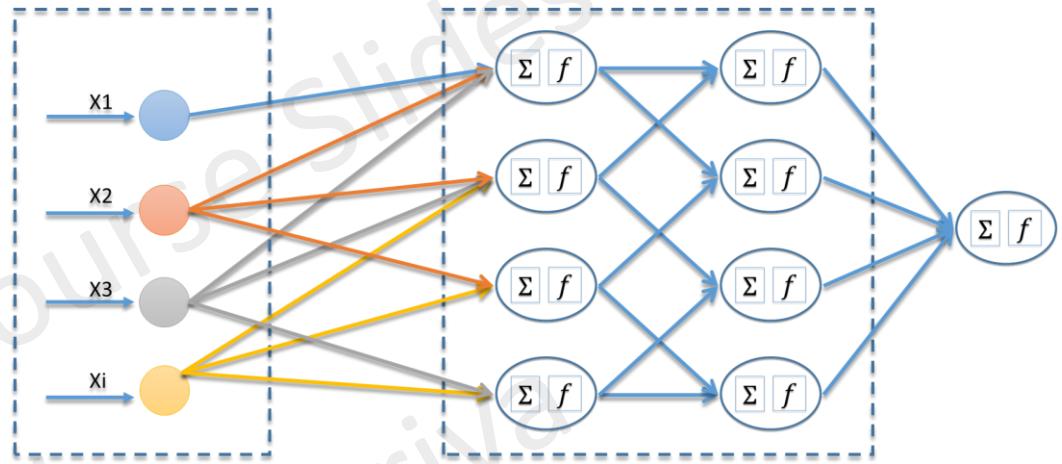
Measure Loss

Step 4 – Evaluate the Model

Metrics to measure performance

Types of Layers

- **Core or Dense Layer**
- Convolutional Layers
- Recurrent Layers
- Pooling Layers
- Locally connected Layers



Dense layer parameters

- units

- kernel_initializer

- kernel_regularizer

- kernel_constraint

- use_bias

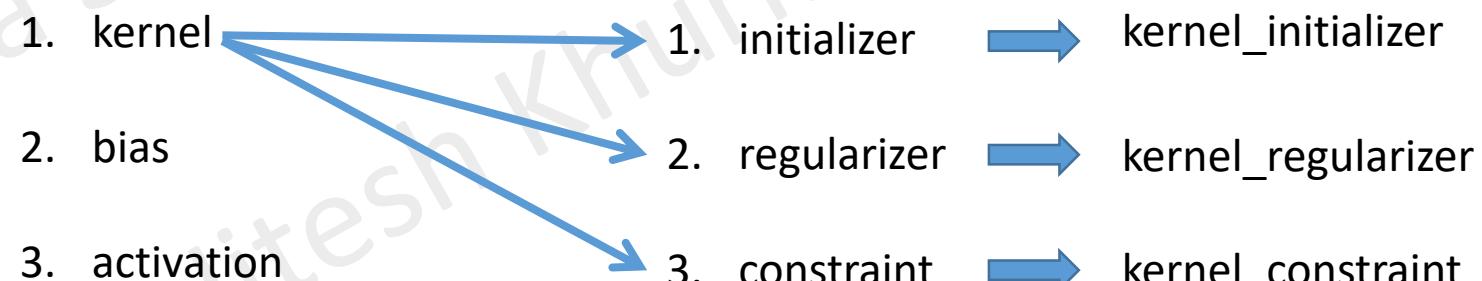
- bias_initializer

- bias_regularizer

- bias_constraint

- activation

- activity_regularizer



Important Terms

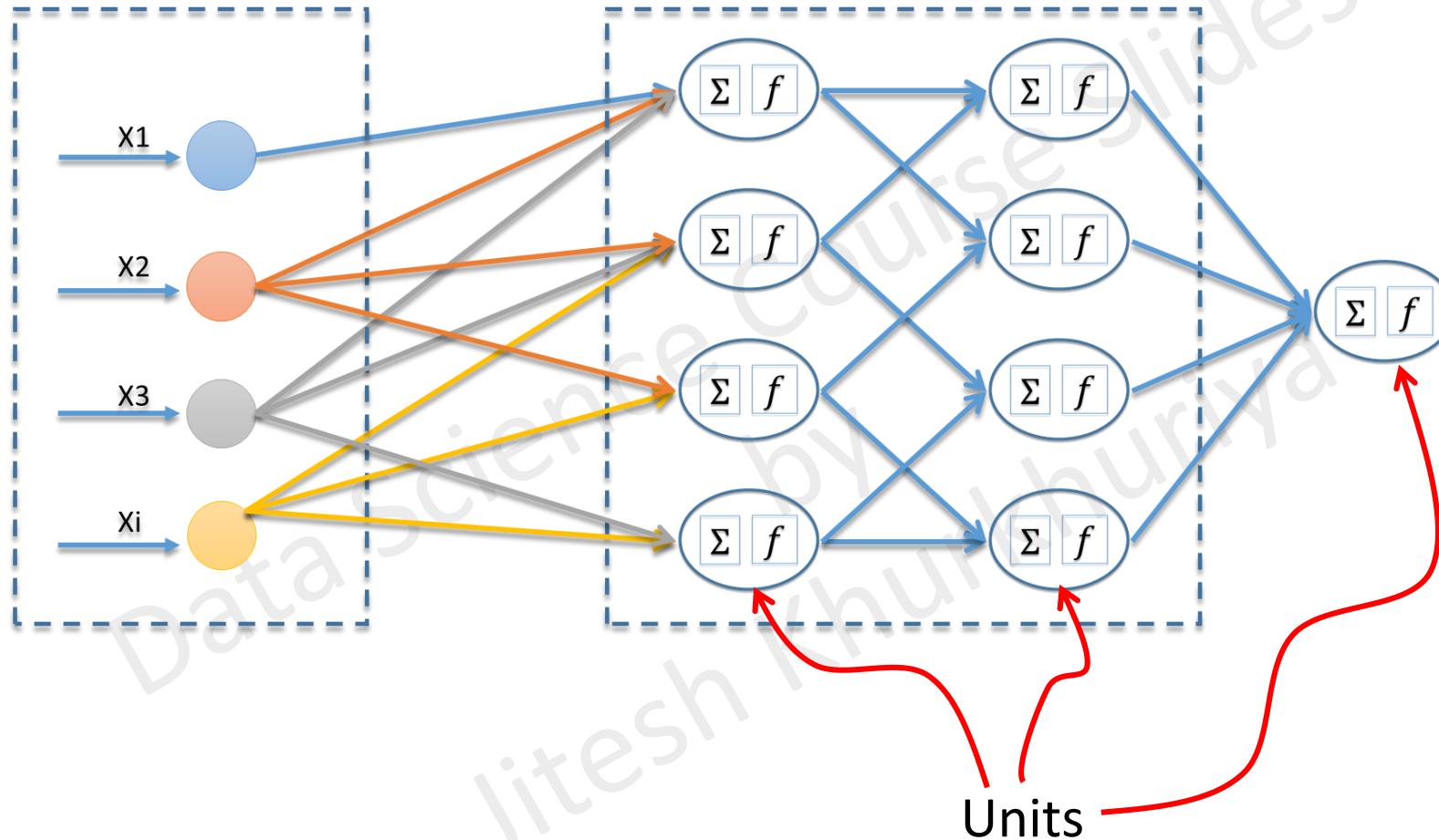
Units

Kernel

Bias

Activation Function

Units



Important Terms

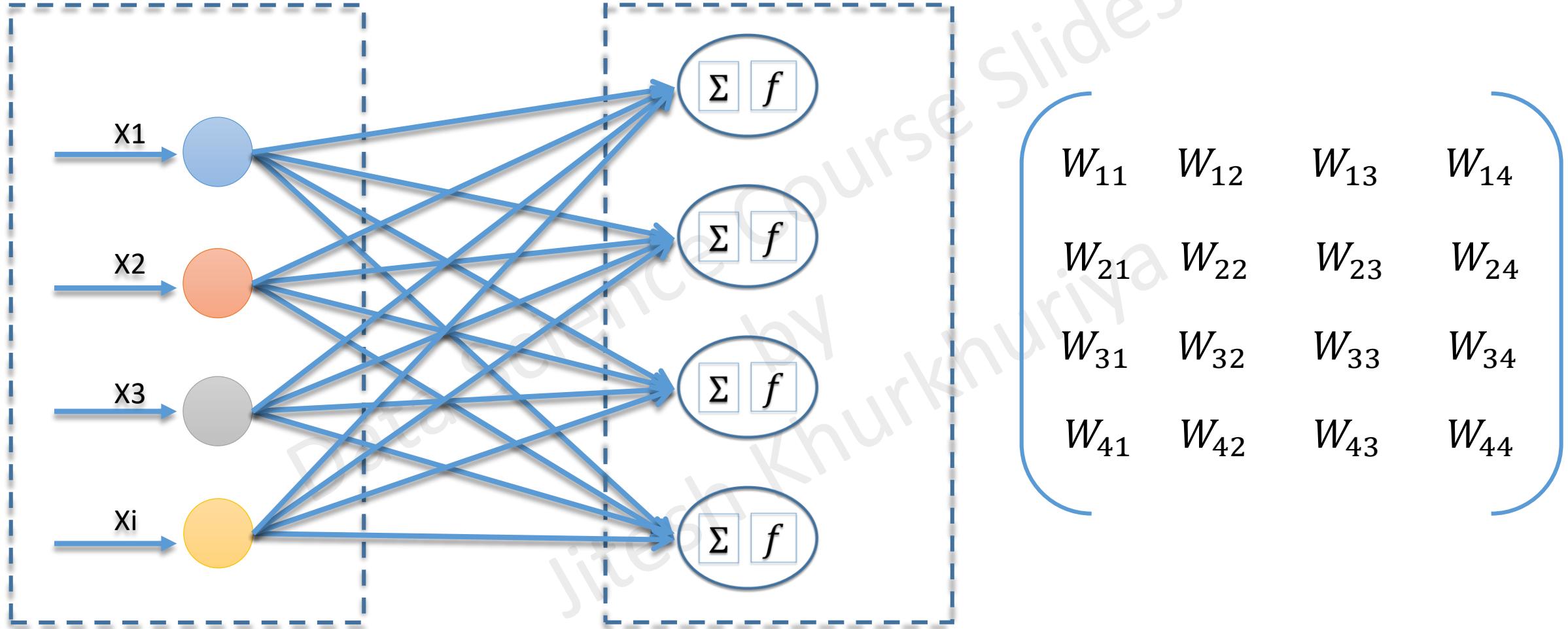
Units

Kernel

Bias

Activation Function

Kernel



Important Terms

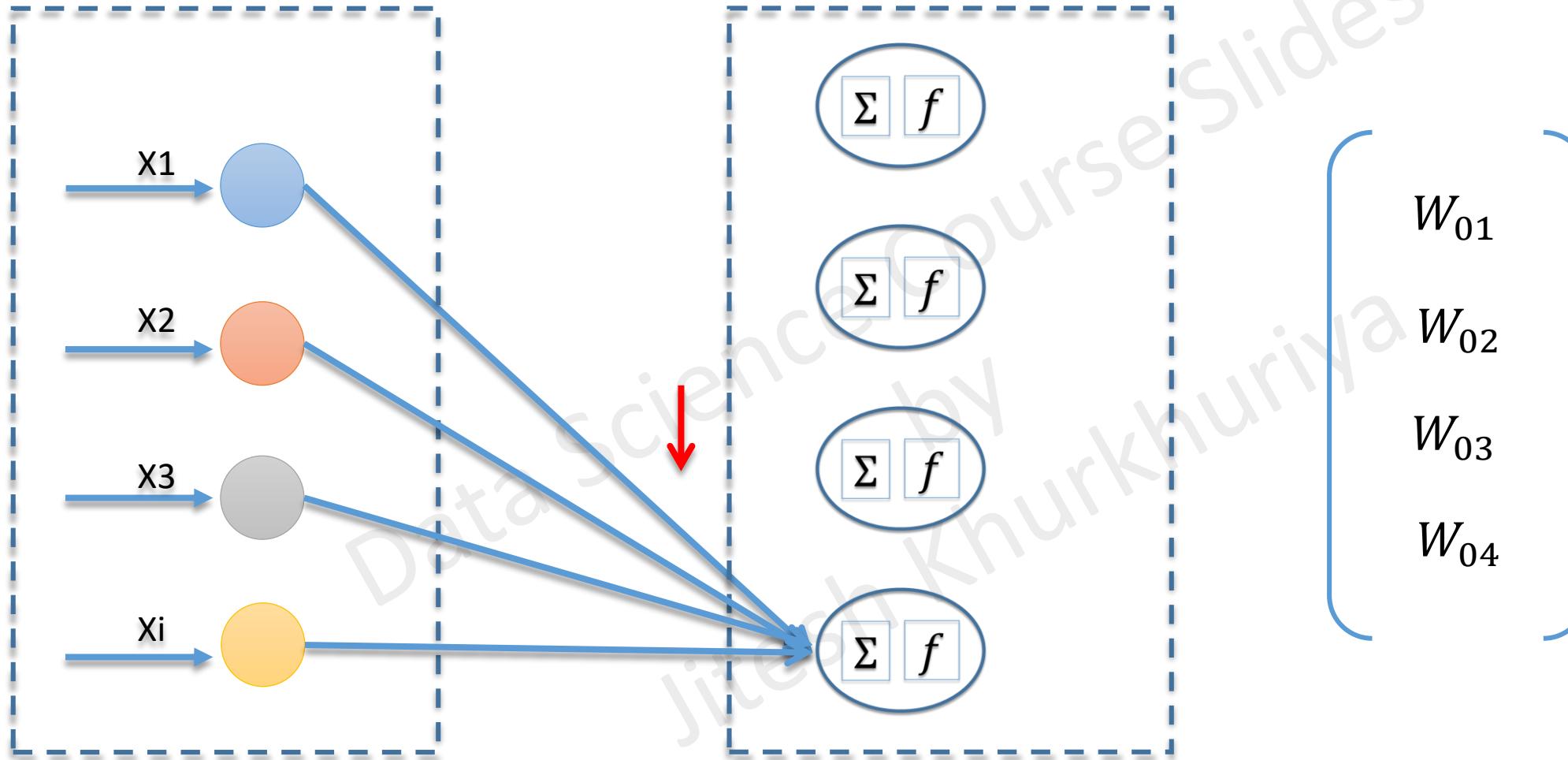
Units

Kernel

Bias

Activation Function

Bias



Important Terms

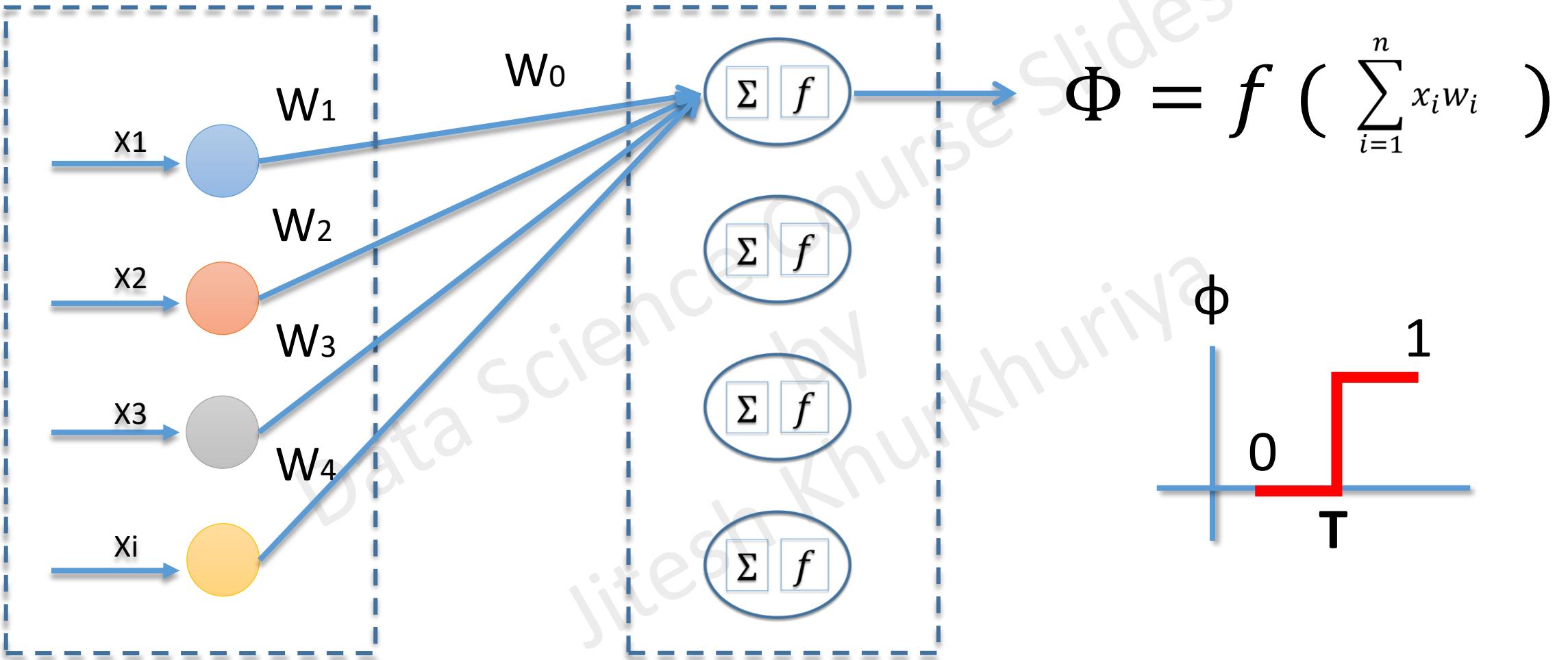
Units

Kernel

Bias

Activation Function

Activation Function

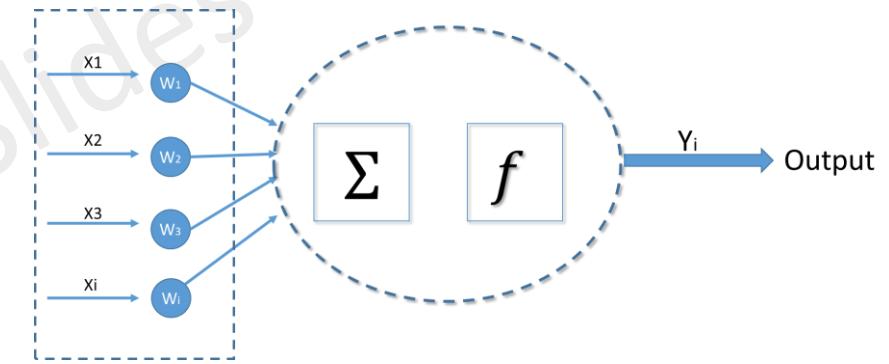
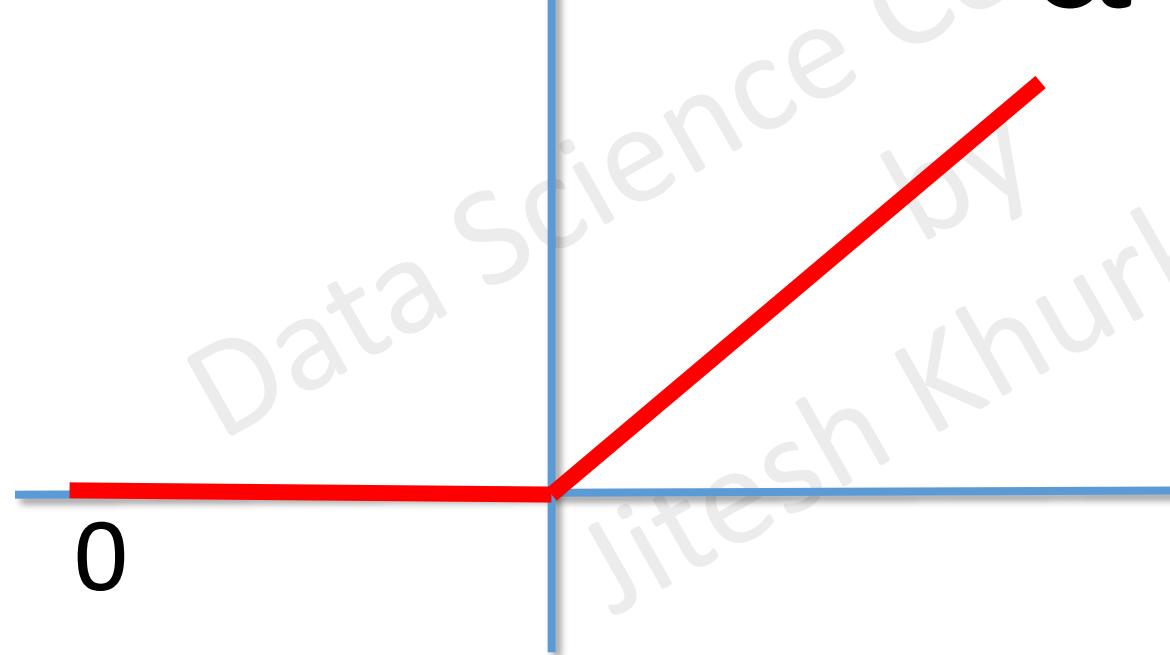


Types of Activation Functions

- relu
- tanh
- sigmoid
- softmax
- elu
- selu
- hard_sigmoid
- softplus
- softsign
- exponential
- linear

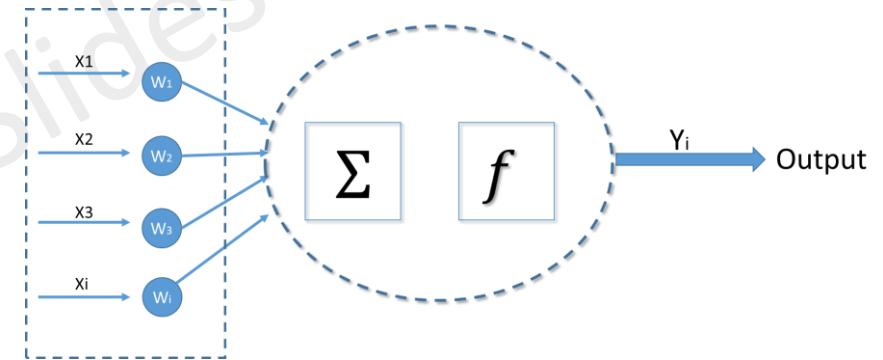
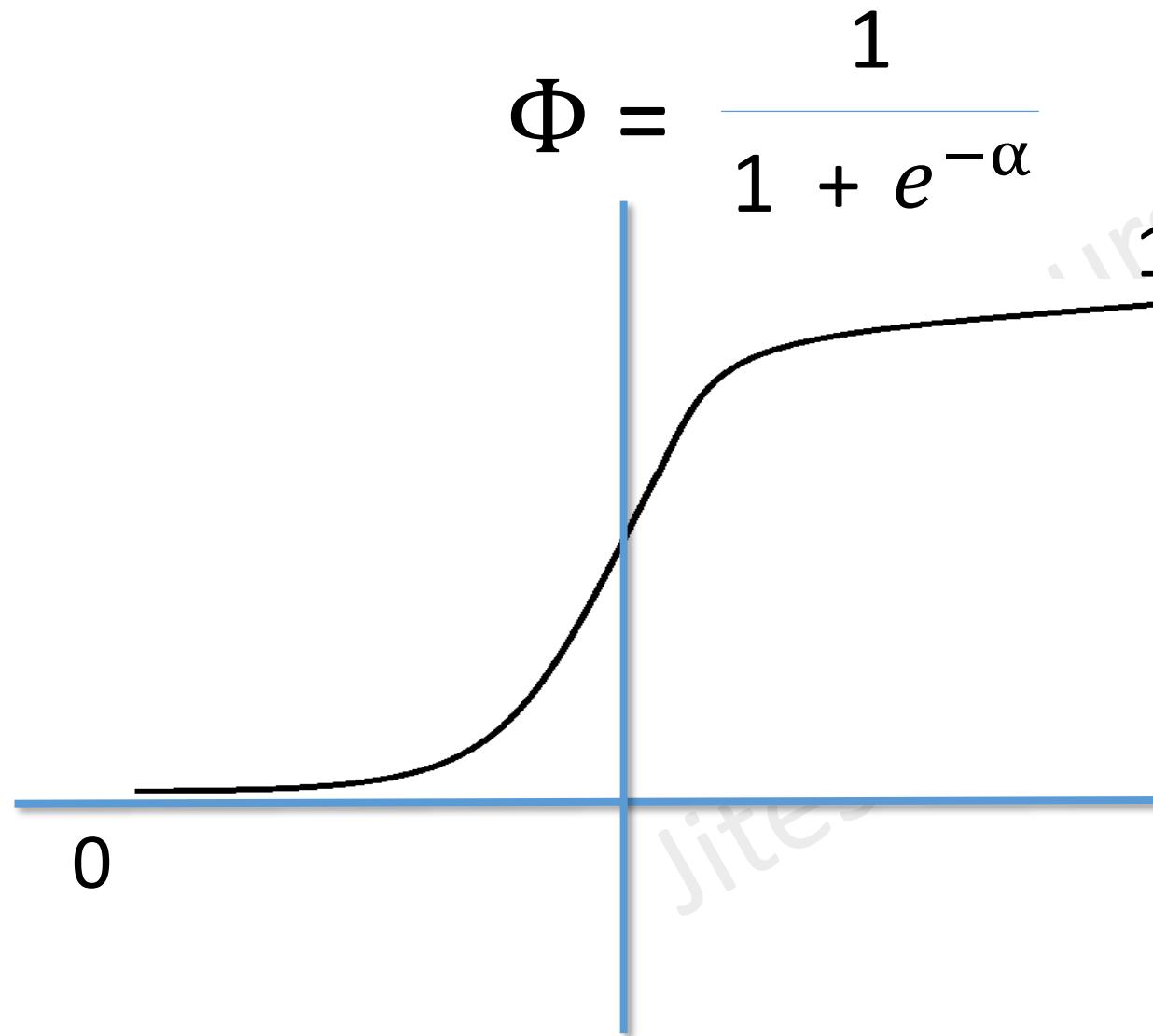
Rectified Linear Unit (ReLU)

$$\Phi = (0, \alpha)$$



$$\alpha = \sum_{i=1}^n x_i w_i$$

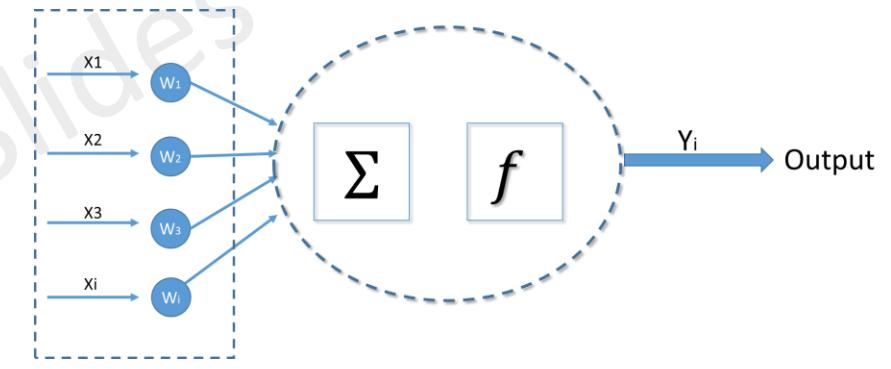
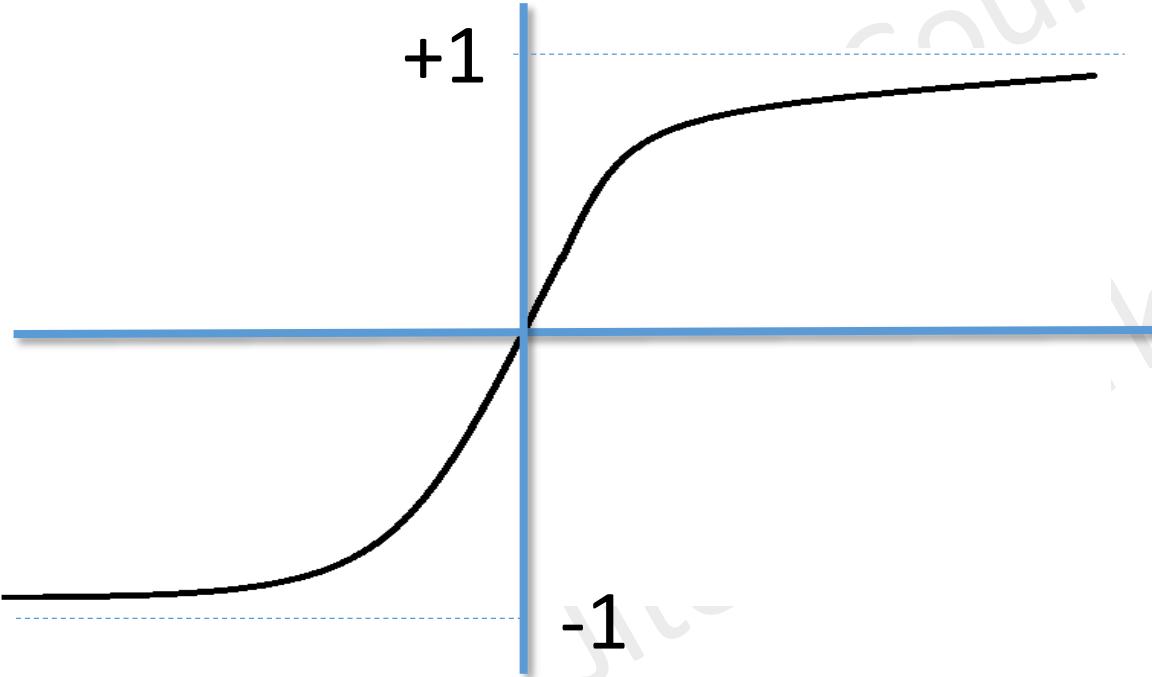
Sigmoid Function



$$\alpha = \sum_{i=1}^n x_i w_i$$

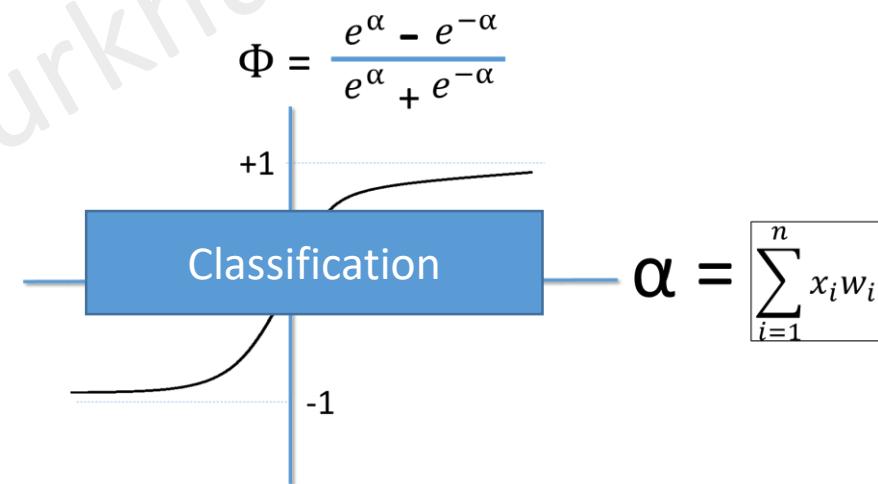
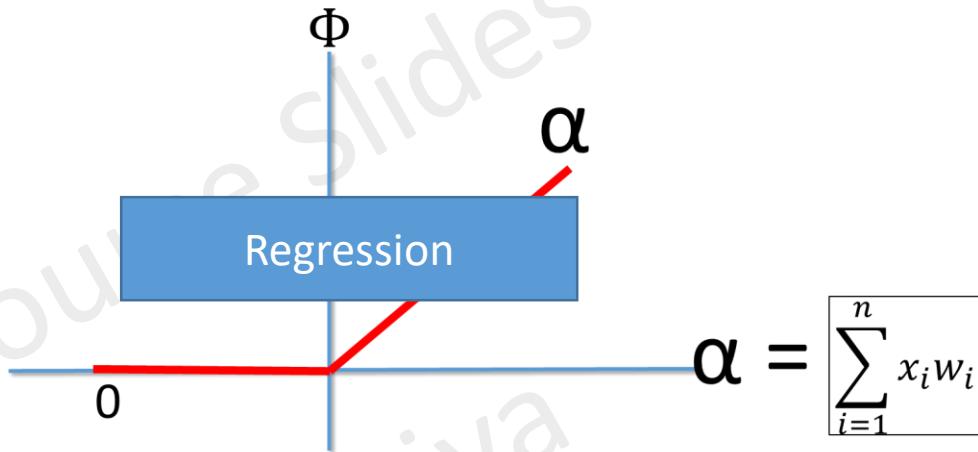
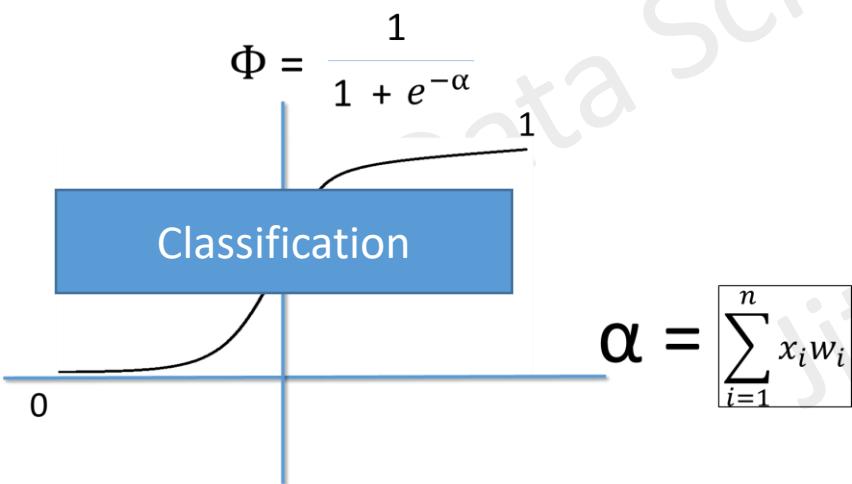
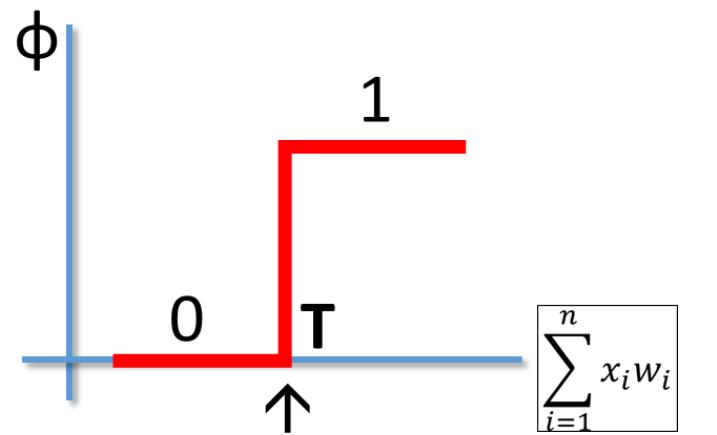
TanH Function

$$\Phi = \frac{e^\alpha - e^{-\alpha}}{e^\alpha + e^{-\alpha}}$$

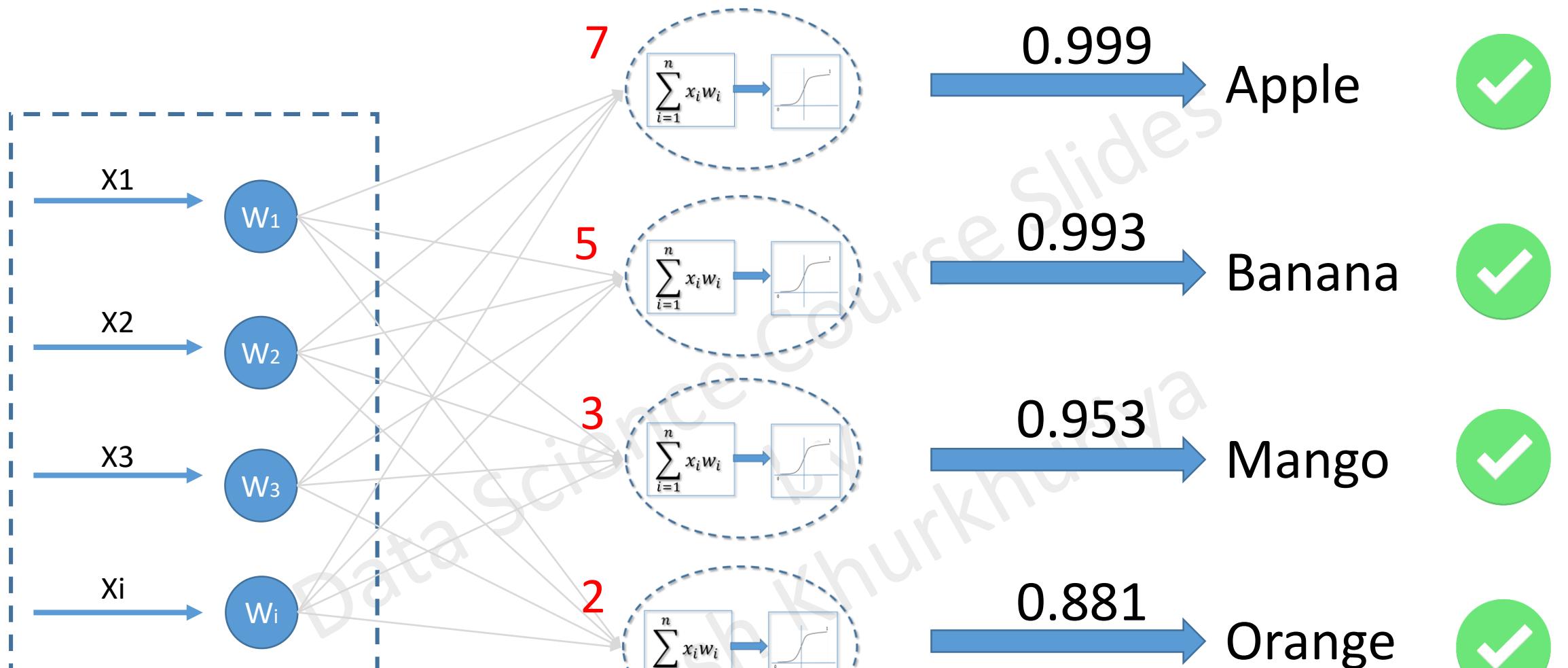


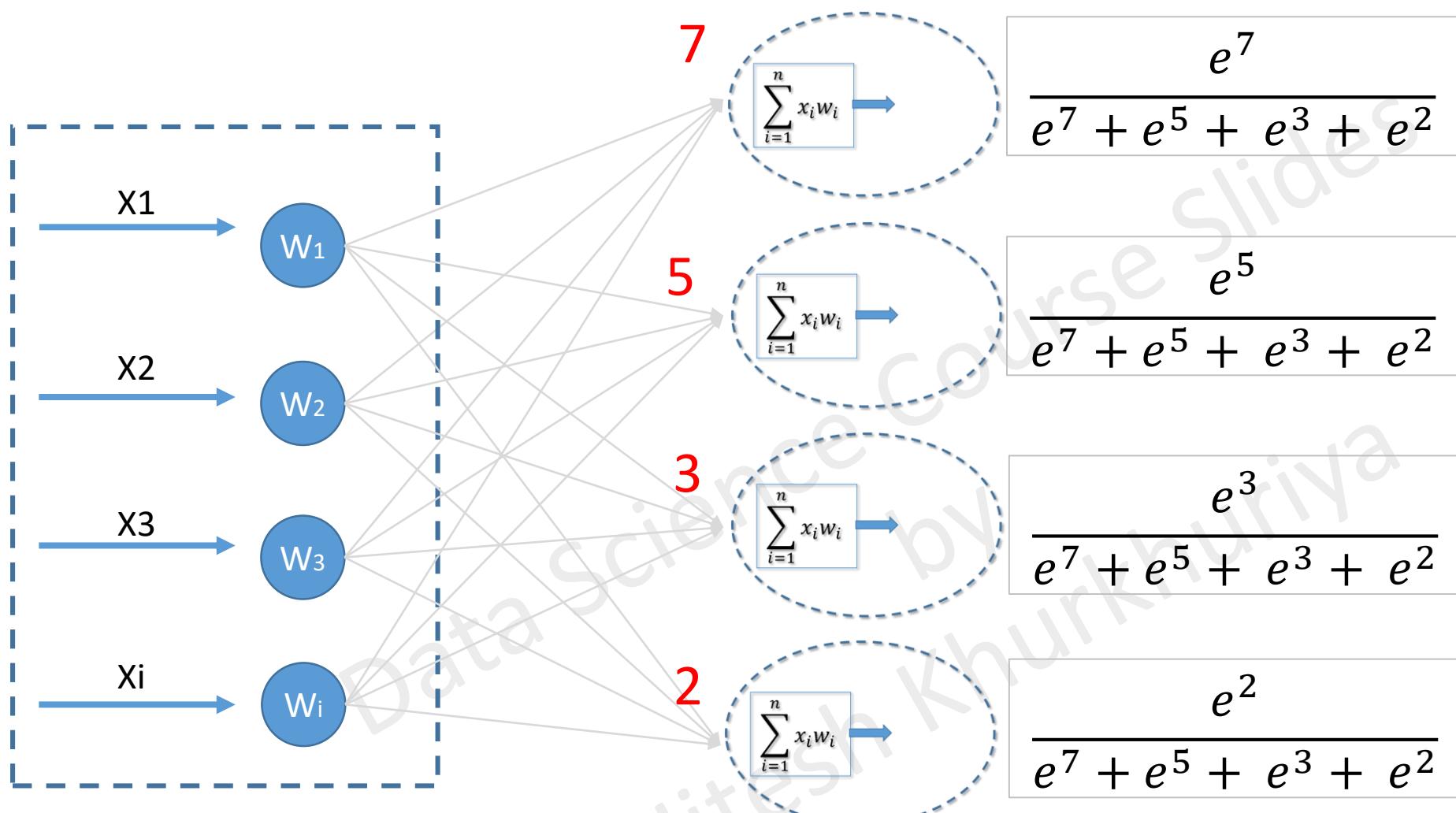
$$\alpha = \sum_{i=1}^n x_i w_i$$

Activation Functions



Softmax Activation



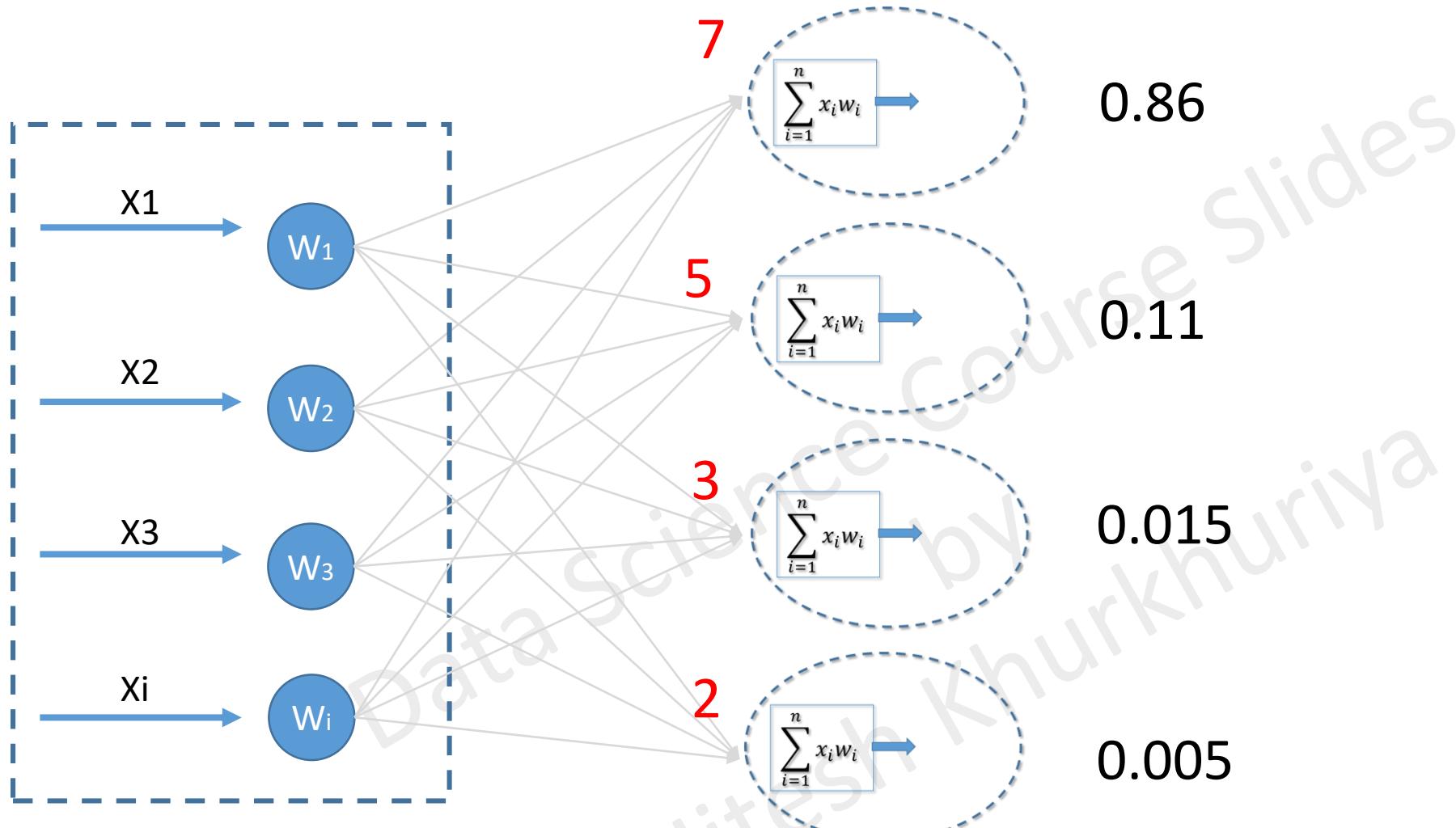


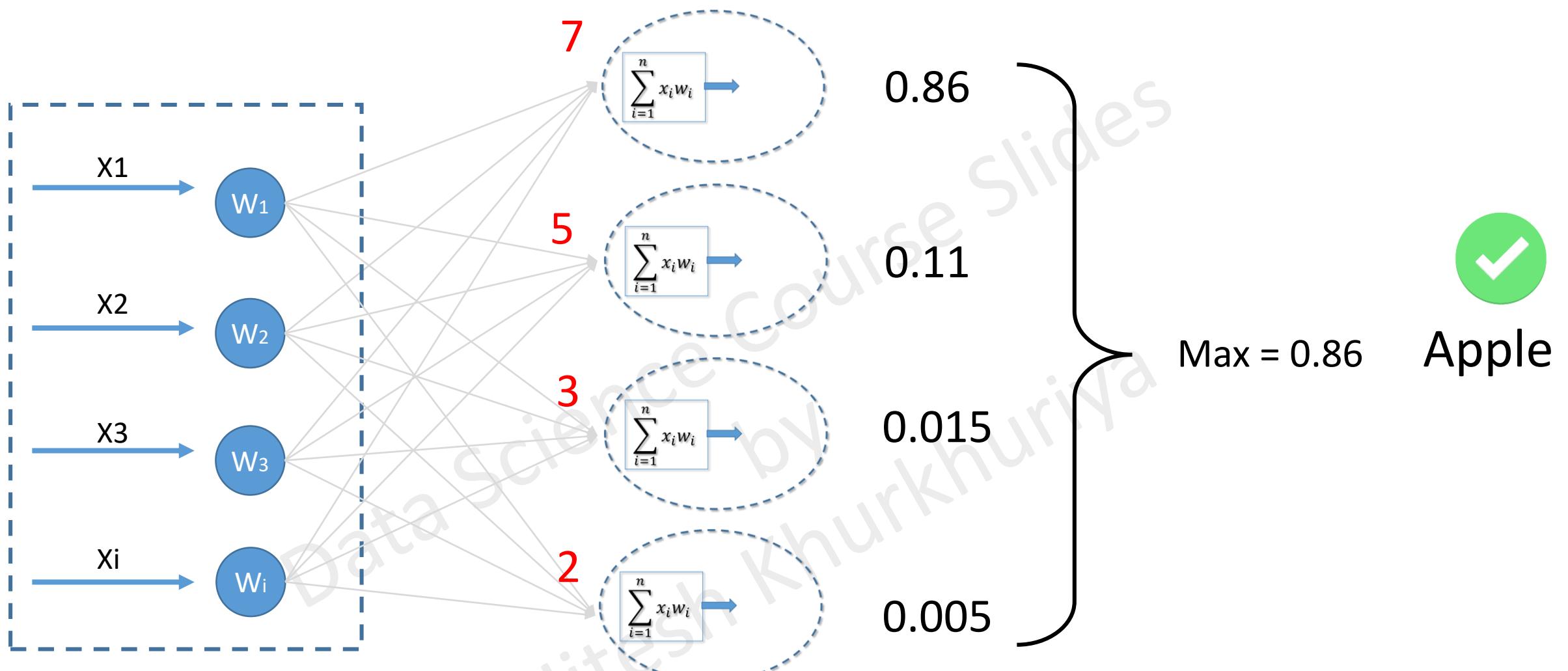
$$\frac{1097}{1097 + 148 + 20 + 7}$$

$$\frac{148}{1097 + 148 + 20 + 7}$$

$$\frac{20}{1097 + 148 + 20 + 7}$$

$$\frac{7}{1097 + 148 + 20 + 7}$$





Softmax Function

$$\Phi_{\alpha_i} = \frac{e^{\alpha_i}}{\sum_{j=0}^k e^{\alpha_j}}$$

Loss Functions

Loss Functions in Keras

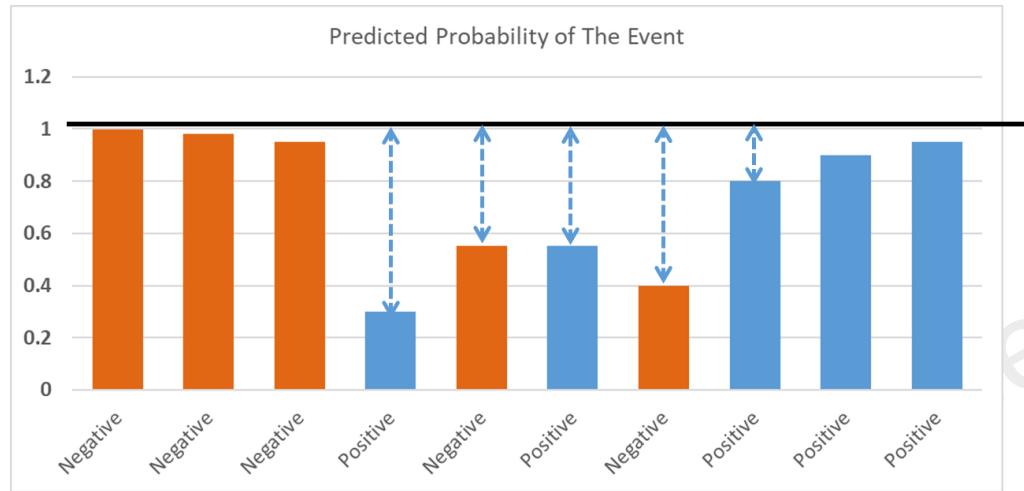
- **Regression**
 - Mean Squared Error
 - Mean Absolute Error
 - Mean Absolute Percentage Error
 - Mean Squared Logarithmic Error
 - L1-Norm
 - L2-Norm
- **Classification**
 - Categorical_crossentropy
 - Binary_crossentropy
 - Sparse_categorical_crossentropy
 - Hinge
 - Squared_hinge
 - Categorical_hinge

Regression Loss Functions

Name	Formula	Usage/Characteristics
Mean Squared Error or MSE	$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$	<ul style="list-style-type: none">1. One of the most popular2. Used when predicted values are small
Mean Squared Logarithmic Error	$\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i + 1))^2$	<ul style="list-style-type: none">1. Used when predicted values are large and we do not want to penalise the large error
L2-Norm	$\sum_{i=1}^n (y_i - \hat{y}_i)^2$	Same as MSE without the mean.
Mean Absolute Error	$\frac{1}{n} \sum_{i=1}^n y_i - \hat{y}_i $	<ul style="list-style-type: none">1. Robust but unstable solution2. Gradient is difficult to calculate
Mean Absolute Percentage Error	$\frac{100\%}{n} \sum_{i=1}^n \left \frac{y_i - \hat{y}_i}{y_i} \right $	<ul style="list-style-type: none">1. For zero values could pose a challenge2. Does not always remain below 100%
L1-Norm	$\sum_{i=0}^n y_i - \hat{y}_i $	Same as MAE without the mean calculation

Classification Loss Functions

Cross Entropy – Measure of Impurity/Uncertainty



$$H(p, q) = - \sum_{i=1}^x p(x) \log q(x)$$

Binary_crossentropy

Two-Class Classification

1

0

Categorical_crossentropy

0

1

2

3

Sparse_categorical_crossentropy

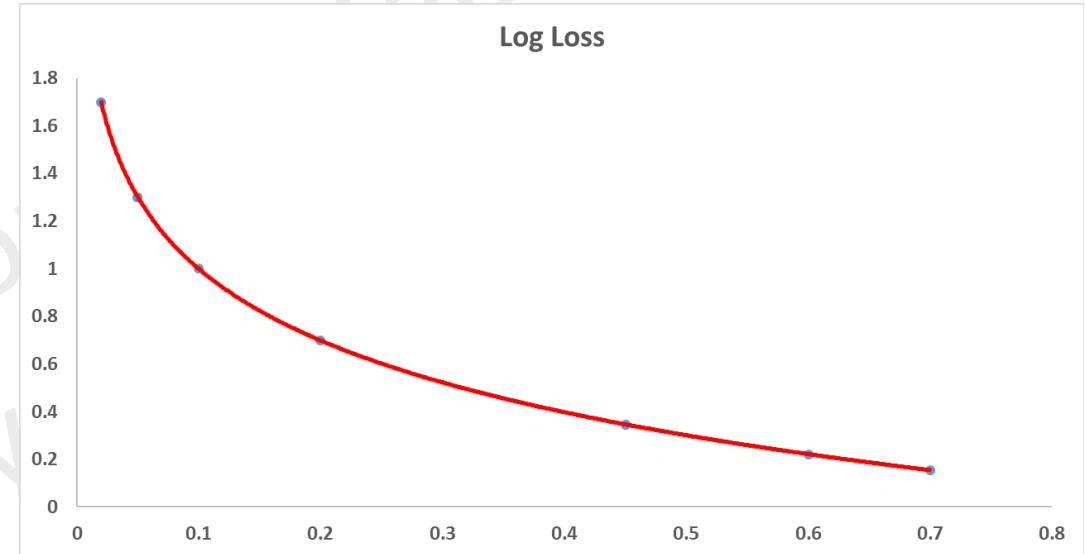
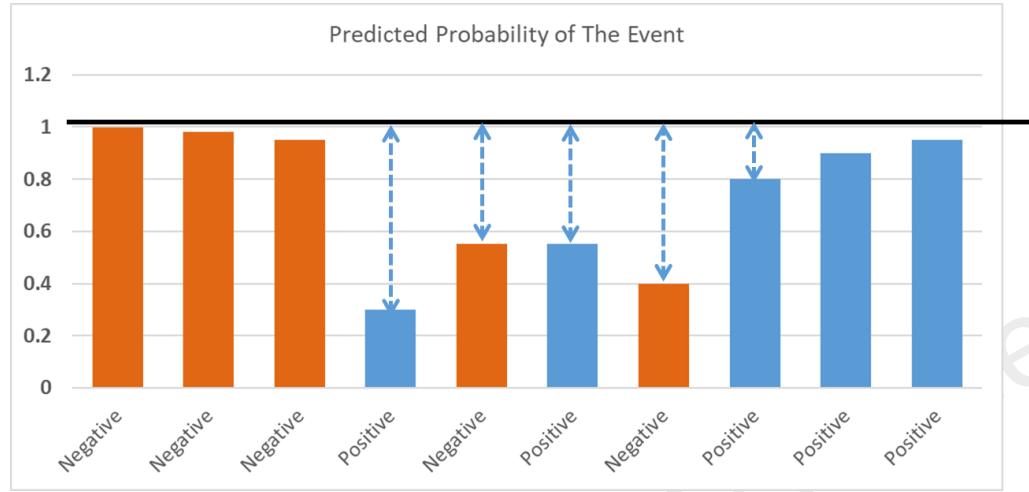
Hot-encoded Multi Classification

0 1 0

1 0 0

0 0 1

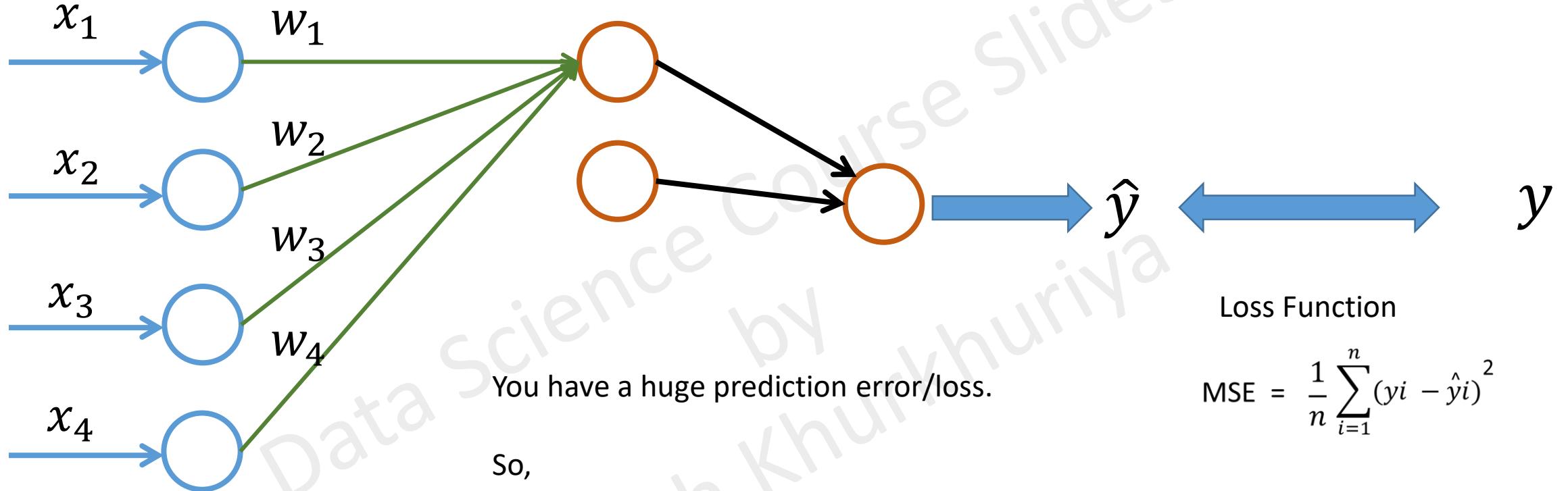
Cross Entropy – Measure of Impurity/Uncertainty



$$H(p, q) = - \sum_{i=1}^x p(x) \log q(x)$$

What is Optimization?

What is Optimization?



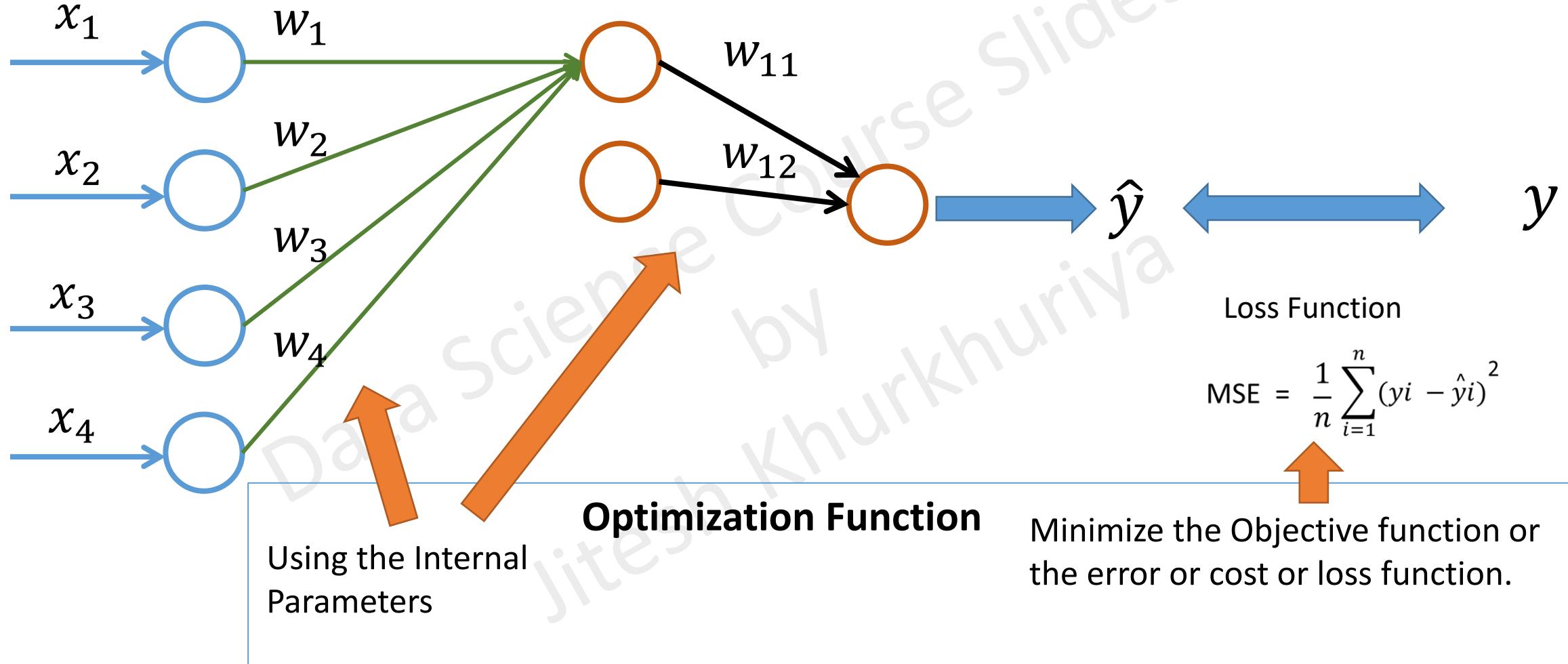
So,

1. What do we do?
2. How do we minimize the loss?
3. How should we change the weights?
4. How much should we change the weights?

Loss Function

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

What is Optimization?



Optimization using First Order Derivatives

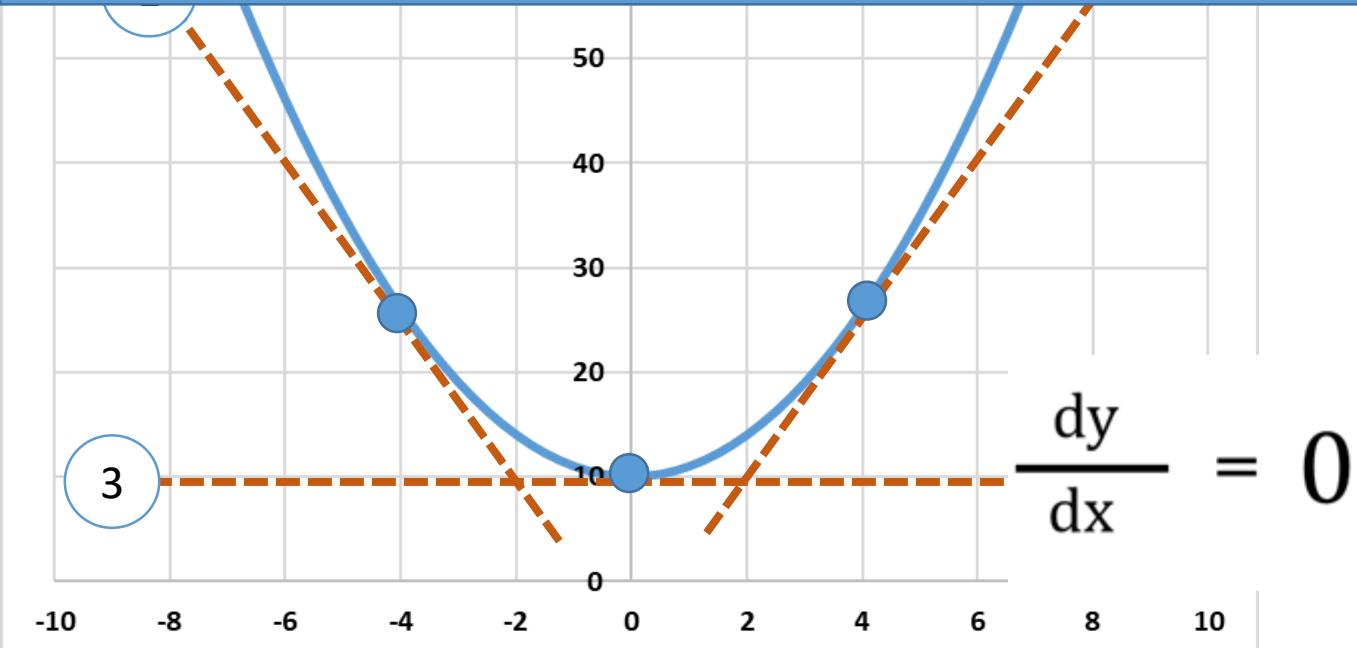
$$y = f(x) = x^2 + 10$$

$$\frac{dy}{dx} = 2x$$

1 $\frac{dy}{dx} = -8$

2 $\frac{dy}{dx} = +8$

1. Easier to compute
2. Faster to converge and get the function minima

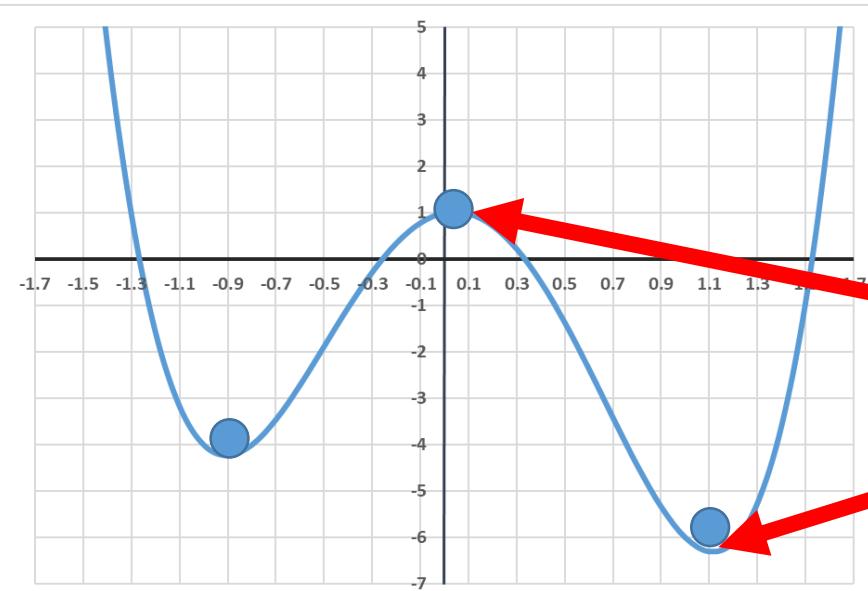


Optimization using Second Order Derivative

$$y = 6x^4 - 2x^3 - 12x^2 + x + 1$$

$$\frac{dy}{dx} = 24x^3 - 6x^2 - 24x + 1$$

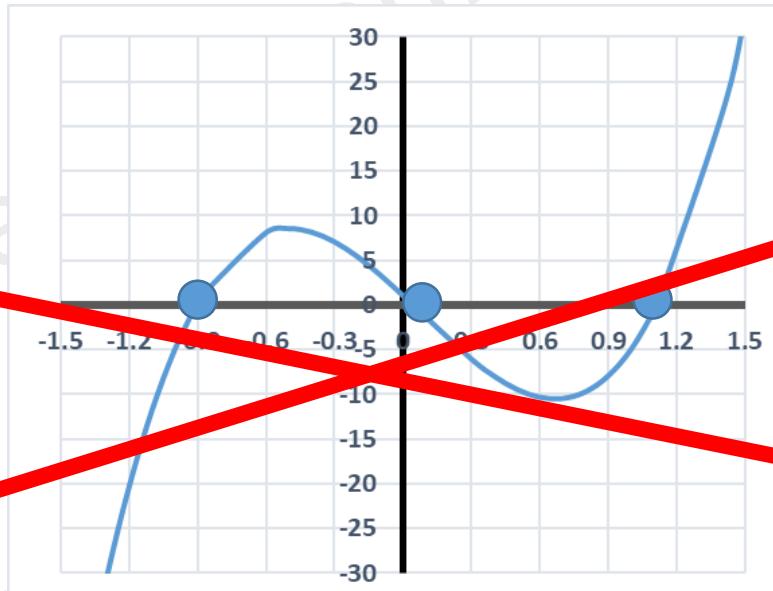
$$\frac{d^2y}{dx^2} = 72x^2 - 12x - 24$$



-0.9054

0.04131

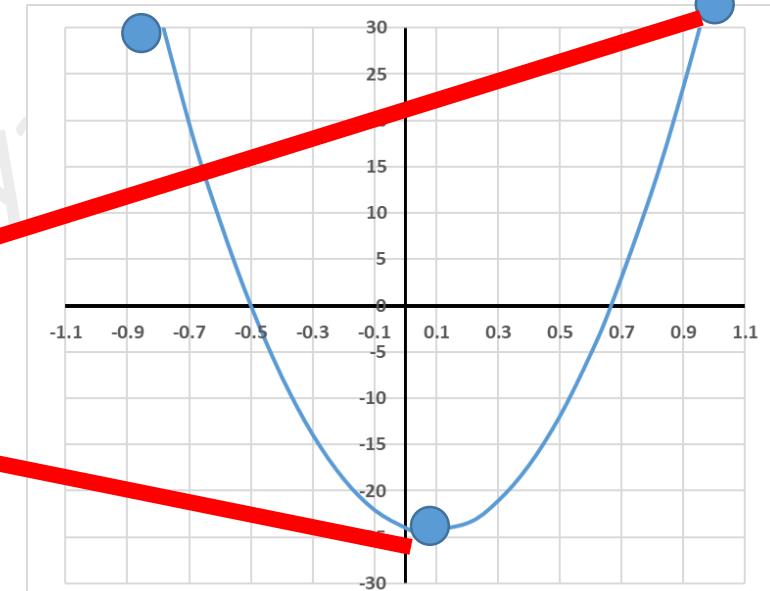
1.1141



-0.9054

0.04131

1.1141



-0.9054

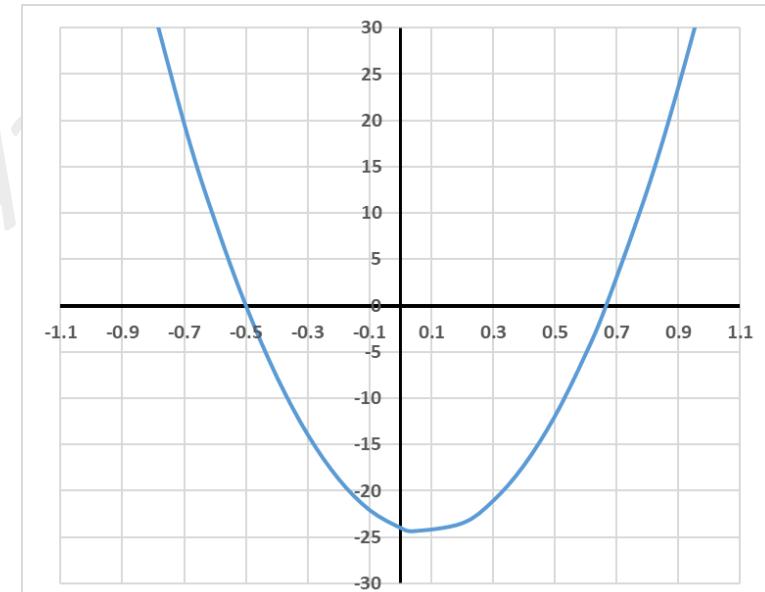
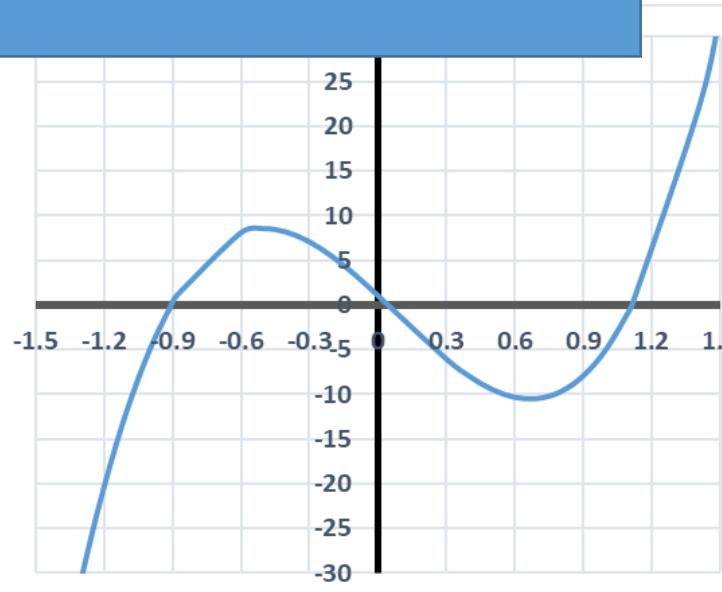
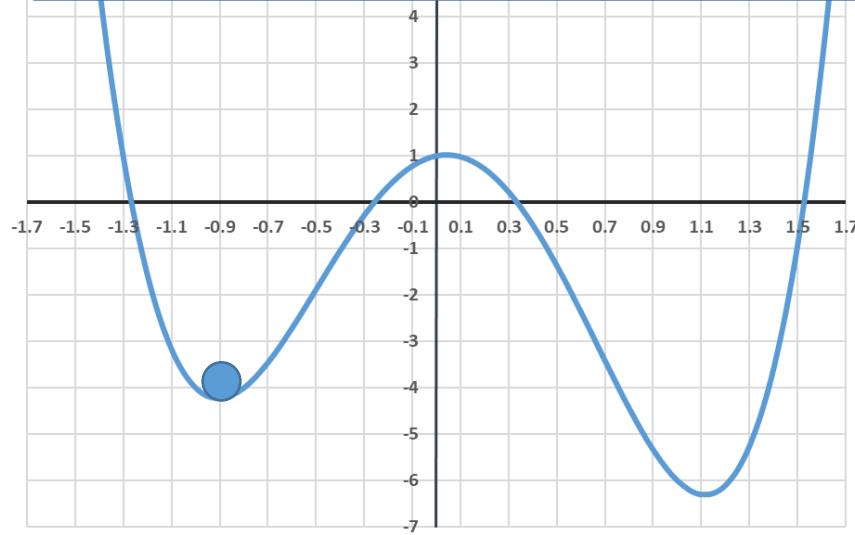
0.04131

1.1141

Optimization using Second Order Derivative

1. Slower and costly to compute
2. Does not get stuck in local minima.

$$\frac{d^2y}{dx^2} = 72x^2 - 12x - 24$$



Gradient Descent

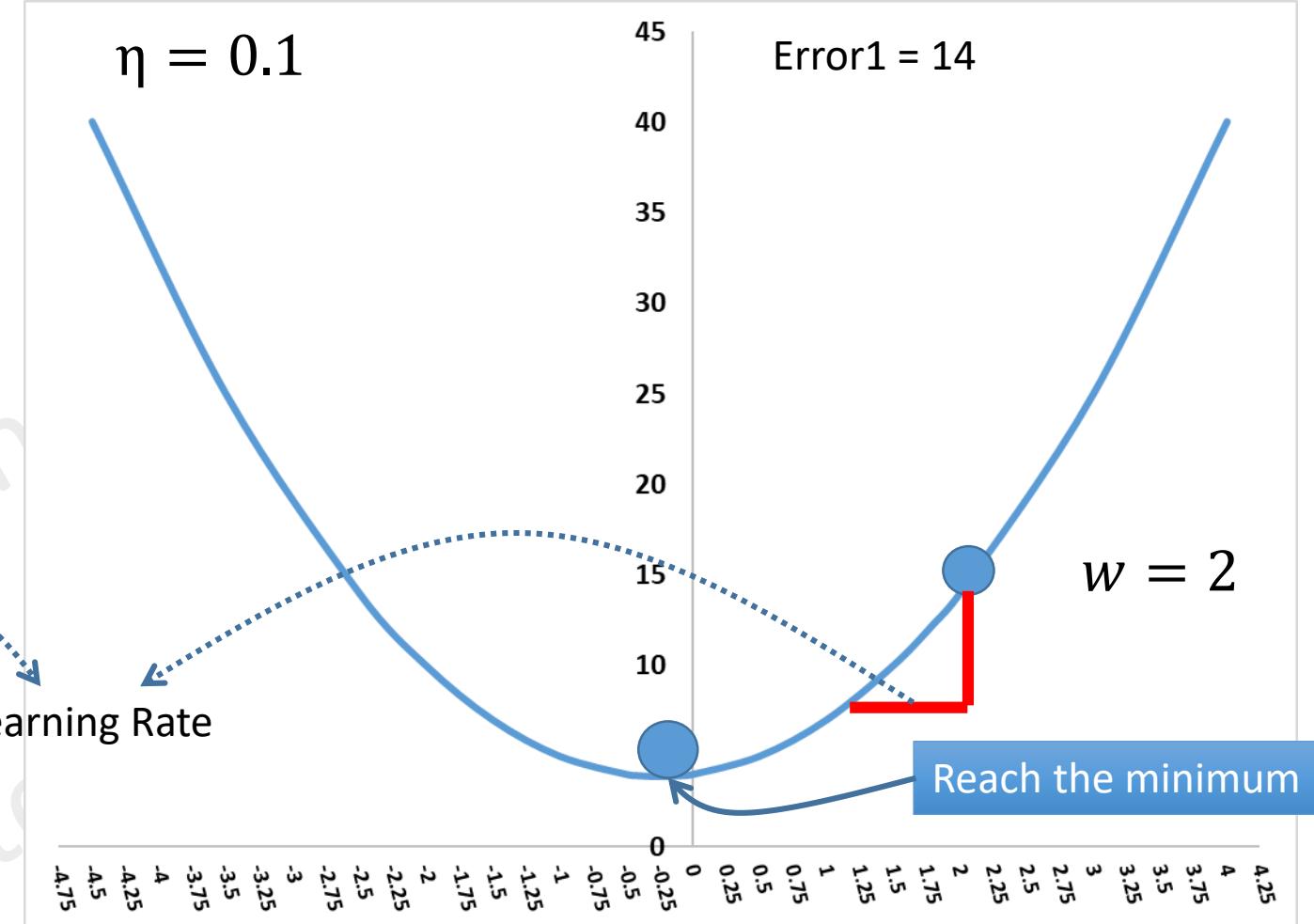
Let's Solve the gradient descent

$$f(w) = 2w^2 + w + 4$$

$$f'(w) = 4w + 1$$

$$\text{Loss} = 2 * 2^2 + 2 + 4 = 14$$

$$w_{i+1} = w_i - \eta * f'(w)$$



Let's Solve the gradient descent

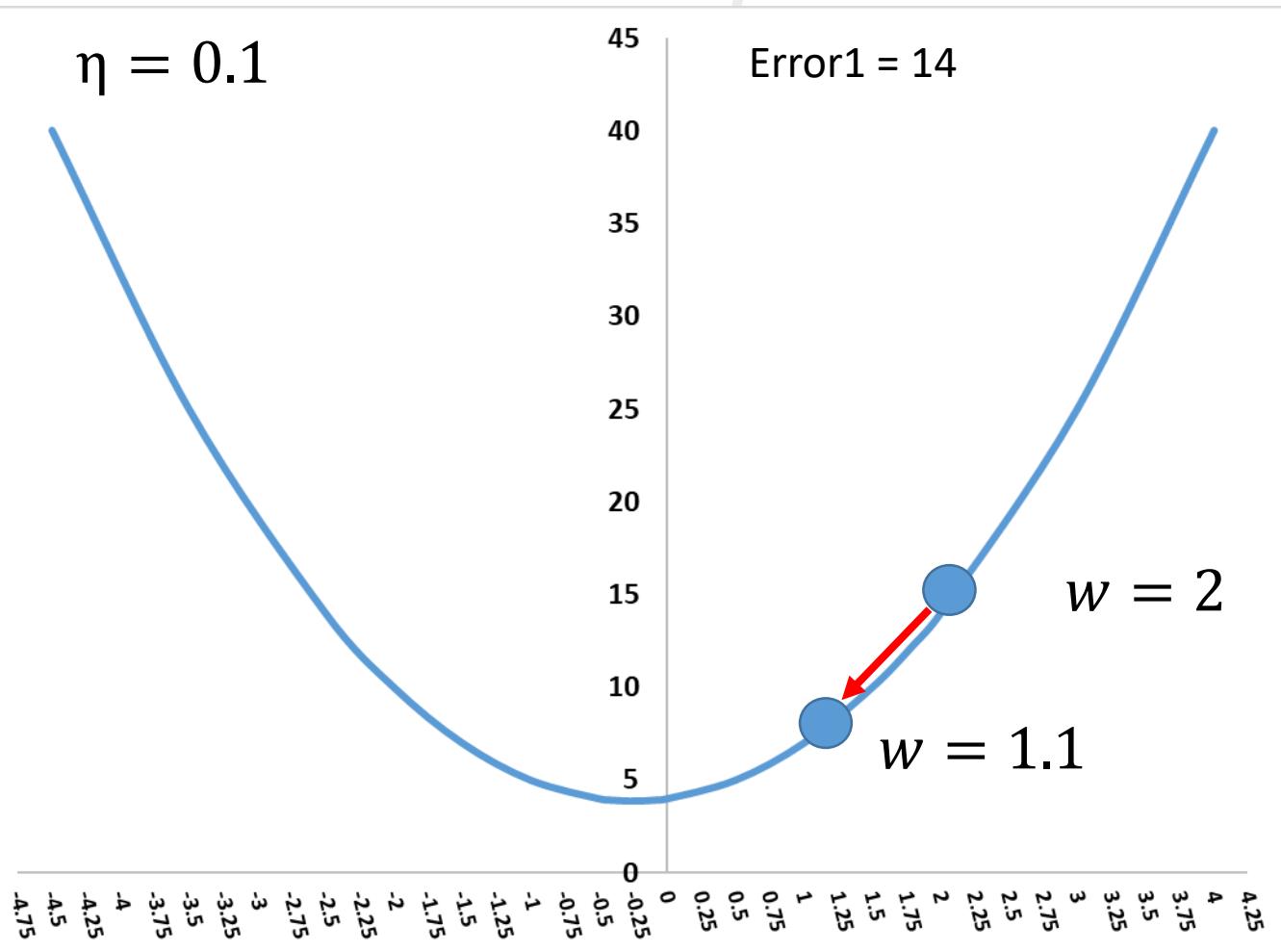
$$f(w) = 2w^2 + w + 4$$

$$f'(w) = 4w + 1$$

$$\text{Loss} = 2 * 2^2 + 2 + 4 = 14$$

$$w_{i+1} = w_i - \eta * f'(w)$$

$$w_{i+1} = 1.1$$



Let's Solve the gradient descent

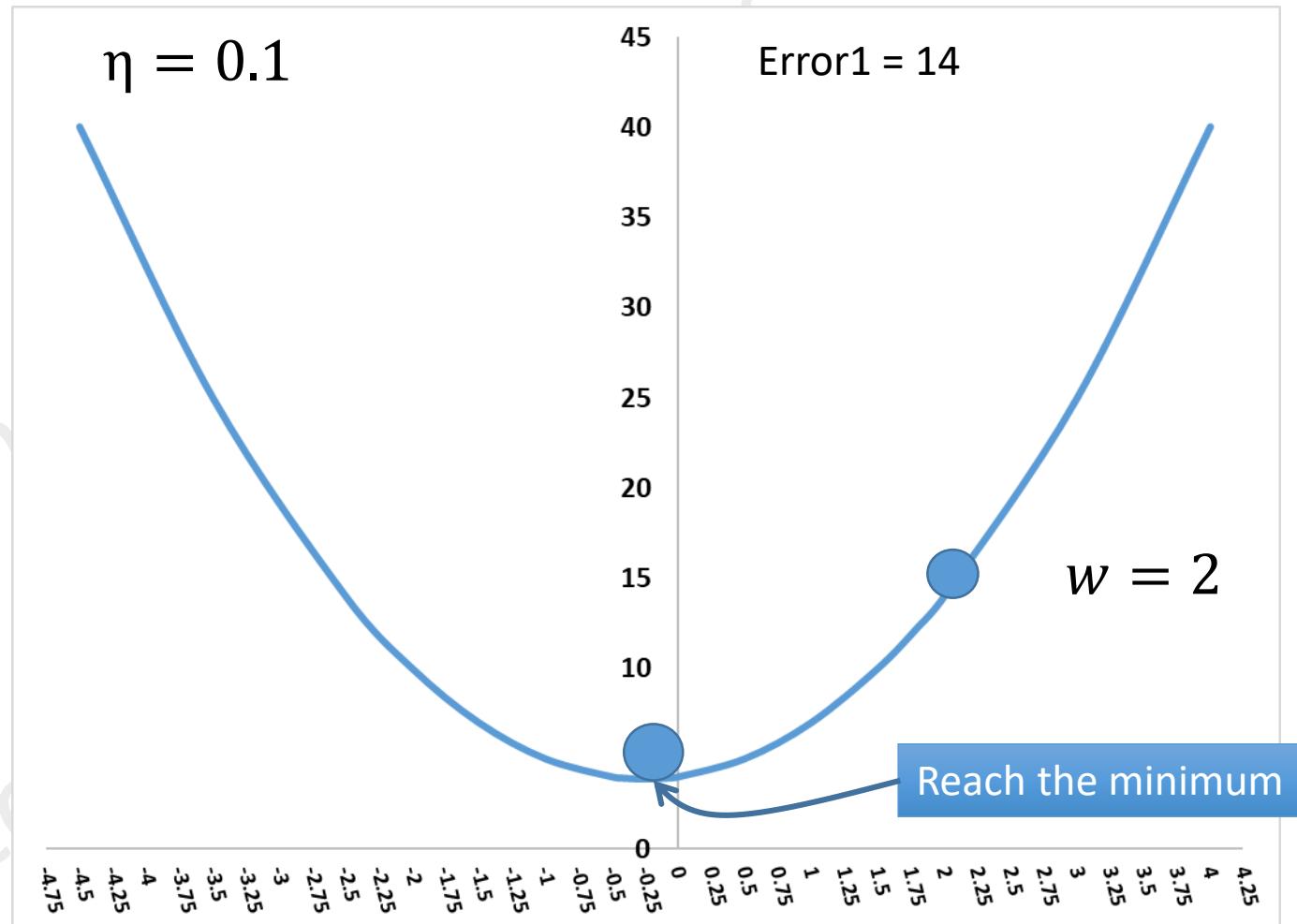
$$f(w) = 2w^2 + w + 4$$

$$f'(w) = 4w + 1$$

$$\text{Loss} = 2 * 2^2 + 2 + 4 = 14$$

$$w_{i+1} = w_i - \eta * f'(w)$$

$$\begin{aligned}w_{i+1} &= 2 - 0.1(4 * 2 + 1) \\&= 2 - 0.9\end{aligned}$$



Let's Solve the gradient descent

$$f(w) = 2w^2 + w + 4$$

$$f'(w) = 4w + 1$$

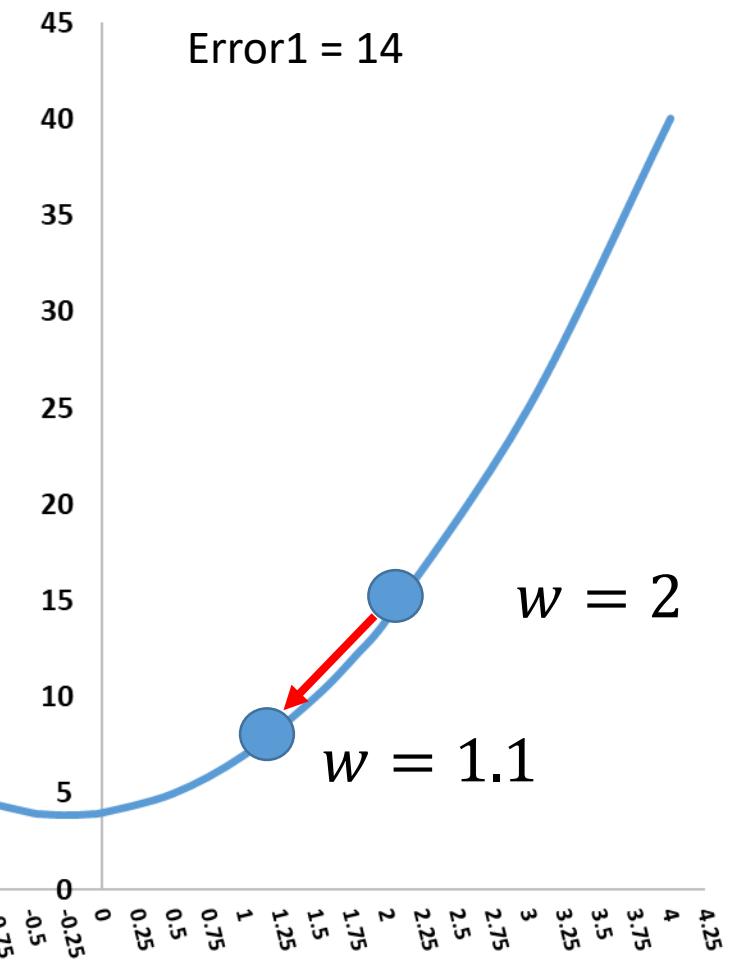
Iteration 1

$$\text{Loss} = 2 * 2^2 + 2 + 4 = 14$$

$$w_{i+1} = w_i - \eta * f'(w)$$

$$w_{i+1} = 1.1$$

$$\eta = 0.1$$



Let's Solve the gradient descent

$$f(w) = 2w^2 + w + 4$$

$$f'(w) = 4w + 1$$

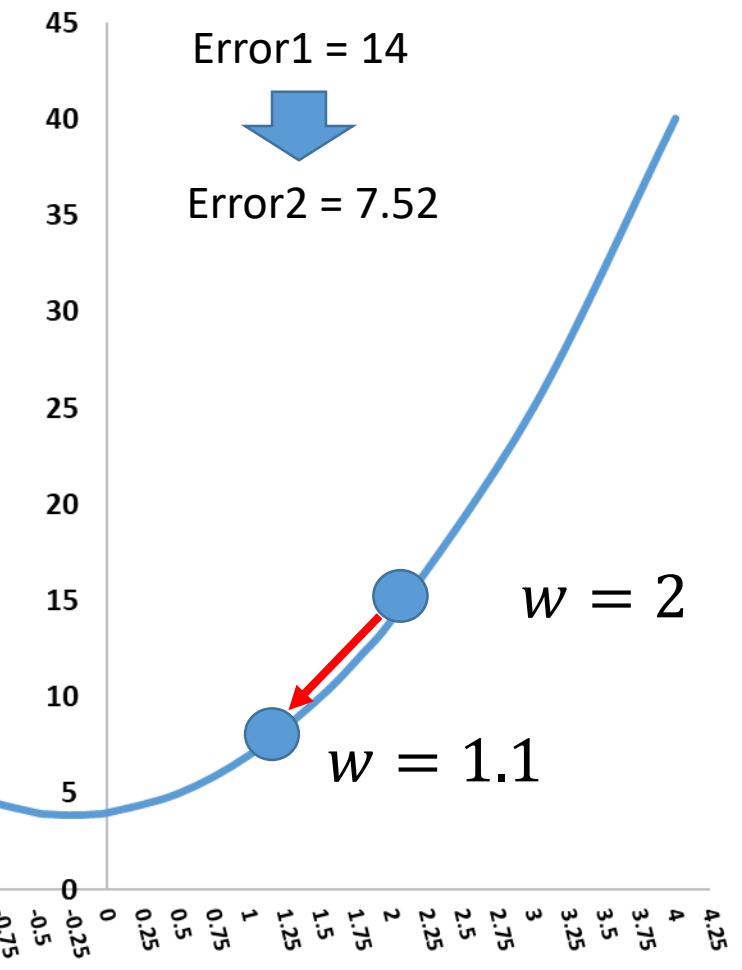
Iteration 2

$$\text{Loss} = 2 * 1.1^2 + 1.1 + 4 = 7.52$$

$$w_{i+1} = w_i - \eta * f'(w)$$

$$w_{i+1} = 0.56$$

$$\eta = 0.1$$



Let's Solve the gradient descent

$$f(w) = 2w^2 + w + 4$$

$$f'(w) = 4w + 1$$

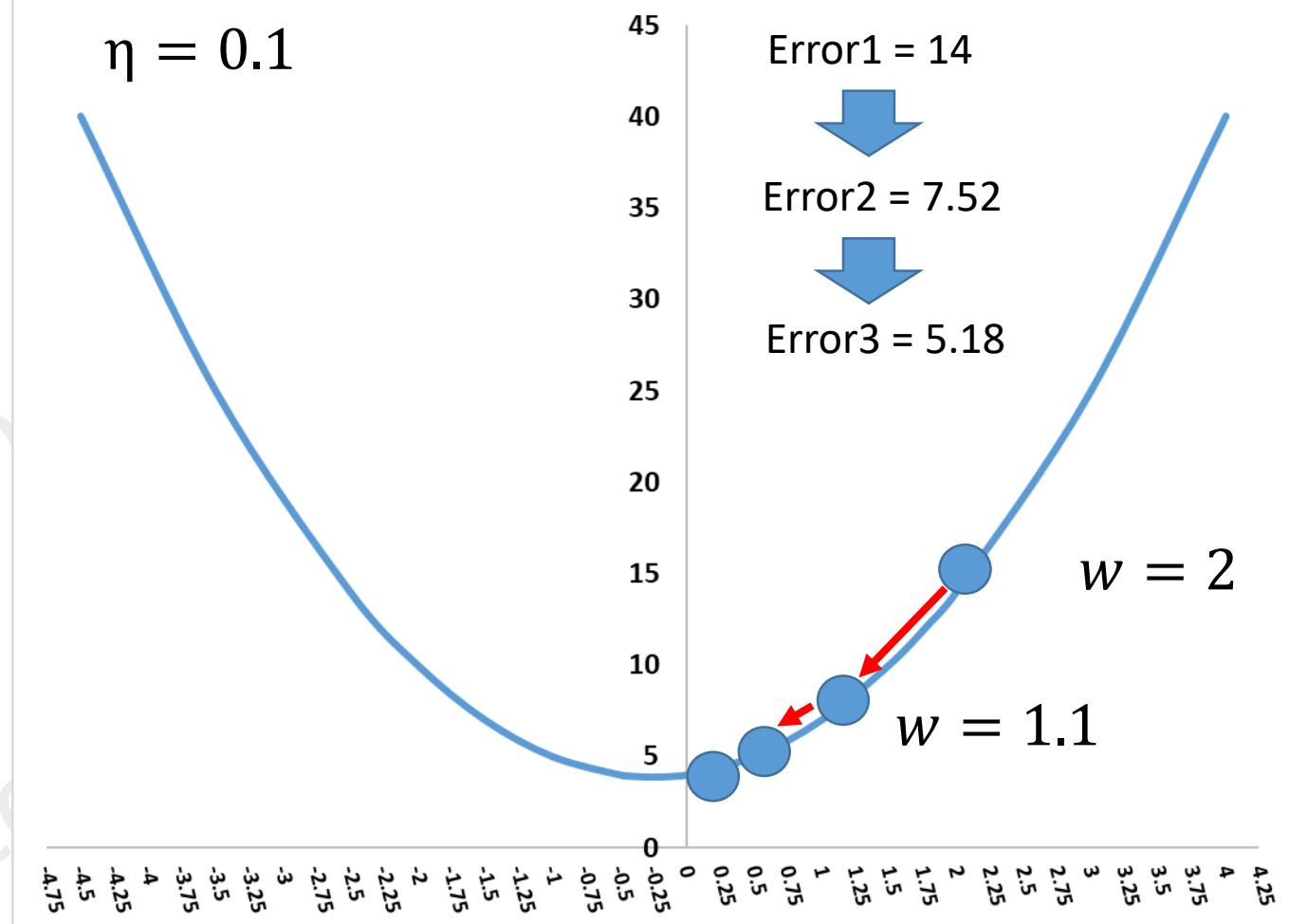
Iteration 3

$$\text{Loss} = 2 * 0.56^2 + 0.56 + 4 = 5.18$$

$$w_{i+1} = w_i - \eta * f'(w)$$

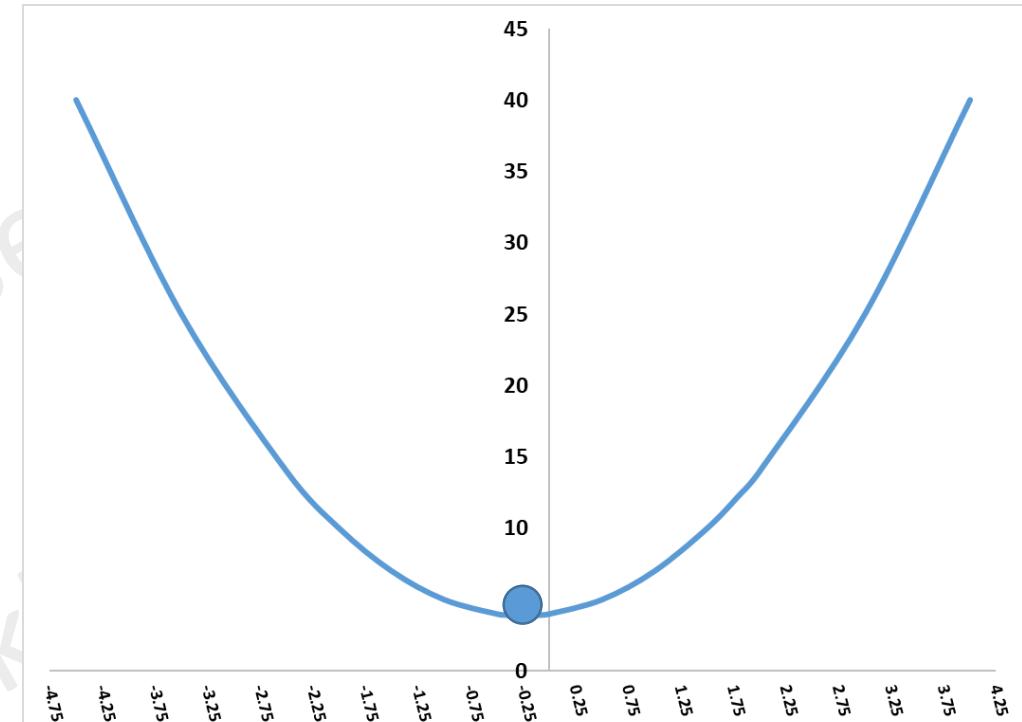
$$w_{i+1} = 0.236$$

$$\eta = 0.1$$



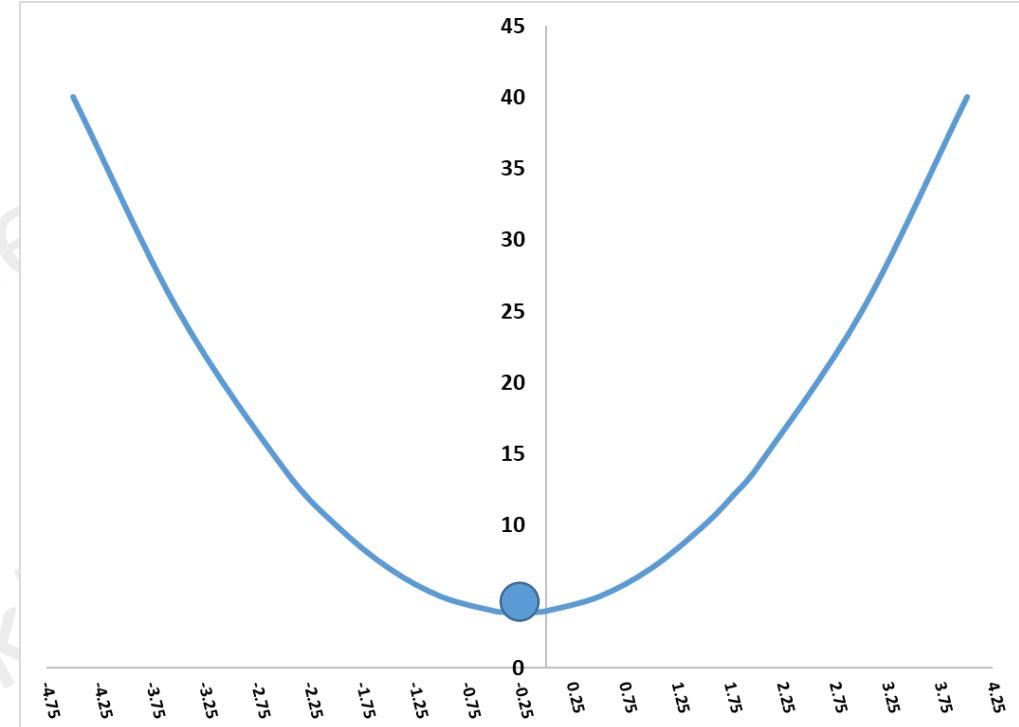
Iteration	W	f (W)	f'(W)	$w_{i+1} = w_i - \eta * f'(w)$
1	2.00000	14.00000	9.00000	1.10000
2	1.10000	7.52000	5.40000	0.56000
3	0.56000	5.18720	3.24000	0.23600
4	0.23600	4.34739	1.94400	0.04160
5	0.04160	4.04506	1.16640	-0.07504
6	-0.07504	3.93622	0.69984	-0.14502
7	-0.14502	3.89704	0.41990	-0.18701
8	-0.18701	3.88293	0.25194	-0.21221
9	-0.21221	3.87786	0.15117	-0.22733
10	-0.22733	3.87603	0.09070	-0.23640
11	-0.23640	3.87537	0.05442	-0.24184
12	-0.24184	3.87513	0.03265	-0.24510
13	-0.24510	3.87505	0.01959	-0.24706
14	-0.24706	3.87502	0.01175	-0.24824
15	-0.24824	3.87501	0.00705	-0.24894
16	-0.24894	3.87500	0.00423	-0.24937
17	-0.24937	3.87500	0.00254	-0.24962
18	-0.24962	3.87500	0.00152	-0.24977
19	-0.24977	3.87500	0.00091	-0.24986
20	-0.24986	3.87500	0.00055	-0.24992
21	-0.24992	3.87500	0.00033	-0.24995

$\eta = 0.1$



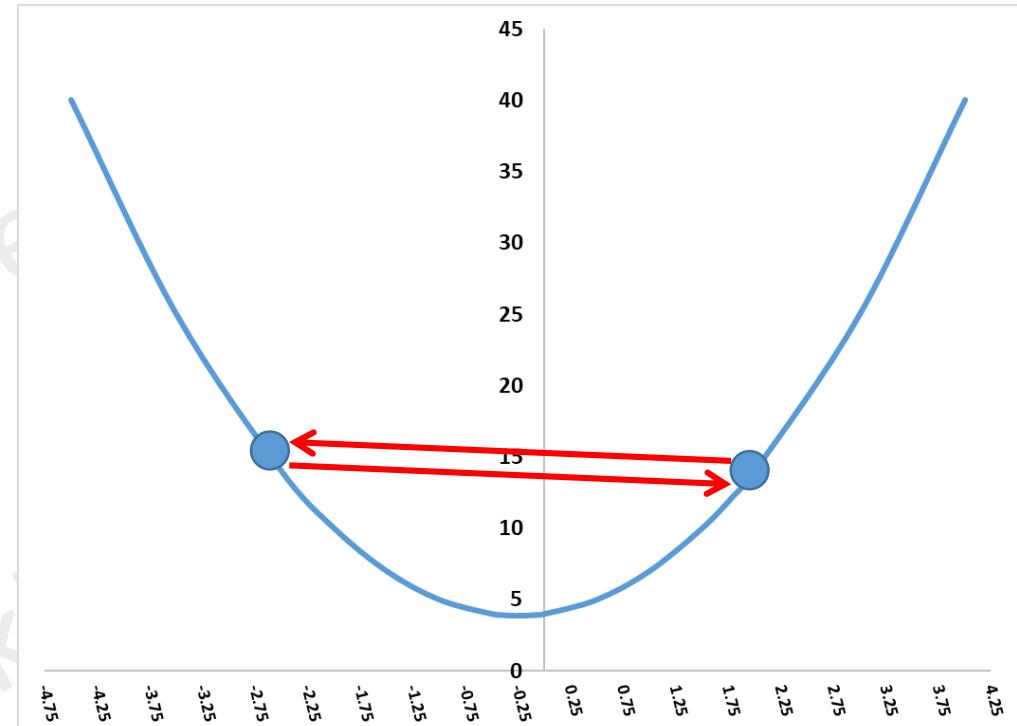
Iteration	W	f (W)	f'(W)	$w_{i+1} = w_i - \eta * f'(w)$
1	2.00000	14.00000	9.00000	0.20000
2	0.20000	4.28000	1.80000	-0.16000
3	-0.16000	3.89120	0.36000	-0.23200
4	-0.23200	3.87565	0.07200	-0.24640
5	-0.24640	3.87503	0.01440	-0.24928
6	-0.24928	3.87500	0.00288	-0.24986
7	-0.24986	3.87500	0.00058	-0.24997
8	-0.24997	3.87500	0.00012	-0.24999
9	-0.24999	3.87500	0.00002	-0.25000
10	-0.25000	3.87500	0.00000	-0.25000

$\eta = 0.2$



Iteration	W	f (W)	f'(W)	$w_{i+1} = w_i - \eta * f'(w)$
1	2.00000	14.00000	9.00000	-2.50000
2	-2.50000	14.00000	-9.00000	2.00000
3	2.00000	14.00000	9.00000	-2.50000
4	-2.50000	14.00000	-9.00000	2.00000
5	2.00000	14.00000	9.00000	-2.50000
6	-2.50000	14.00000	-9.00000	2.00000
7	2.00000	14.00000	9.00000	-2.50000
8	-2.50000	14.00000	-9.00000	2.00000
9	2.00000	14.00000	9.00000	-2.50000
10	-2.50000	14.00000	-9.00000	2.00000

$$\eta = 0.5$$



What is an Error?

What is an Error?

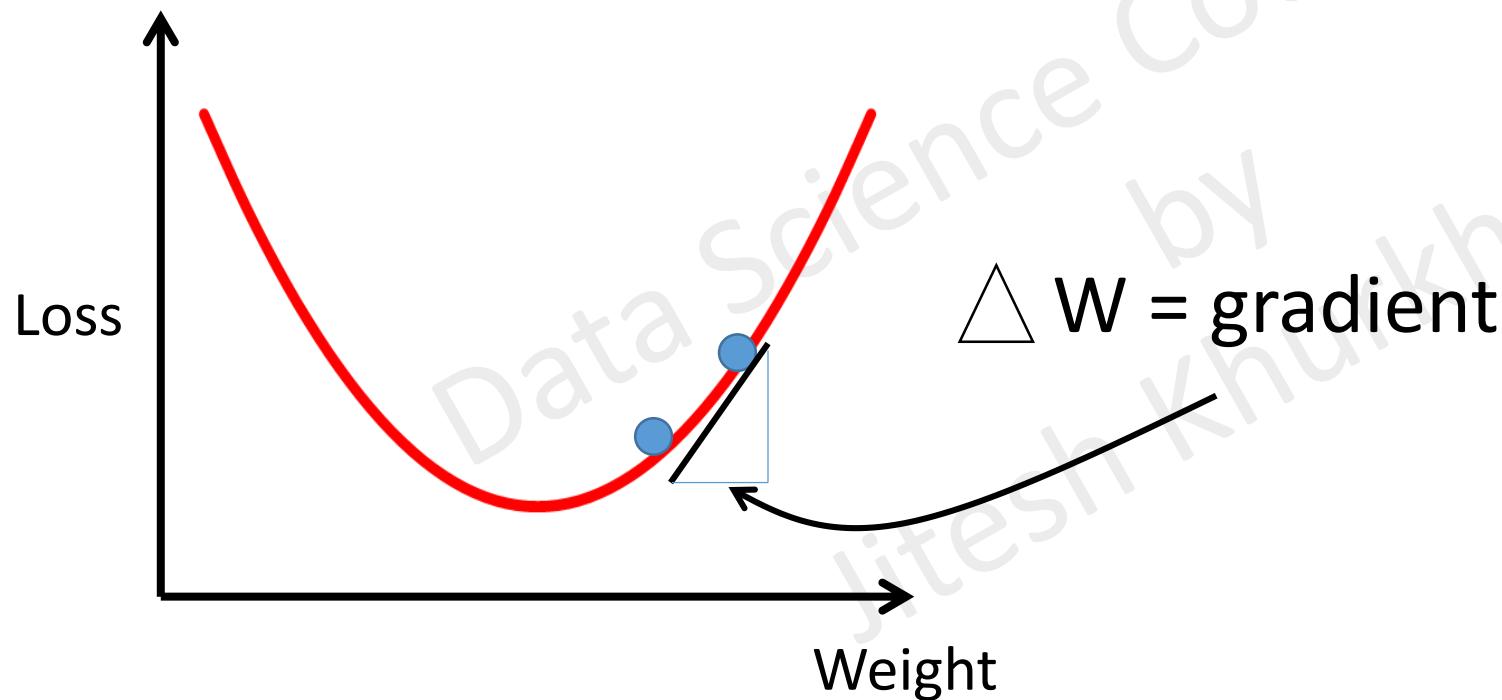
$$\text{error or loss} = \frac{1}{2}(y - \hat{y})^2$$

$$\text{error or loss} = \frac{1}{2}(y - f(\text{weighted sum}))^2$$

$$\text{error or loss} = \frac{1}{2}(y - f(\sum_{i=1}^n w_i x_i))^2$$

Gradient Descent

$$\text{error or loss} = \frac{1}{2} (y - f(\sum_{i=1}^n w_i x_i))^2$$



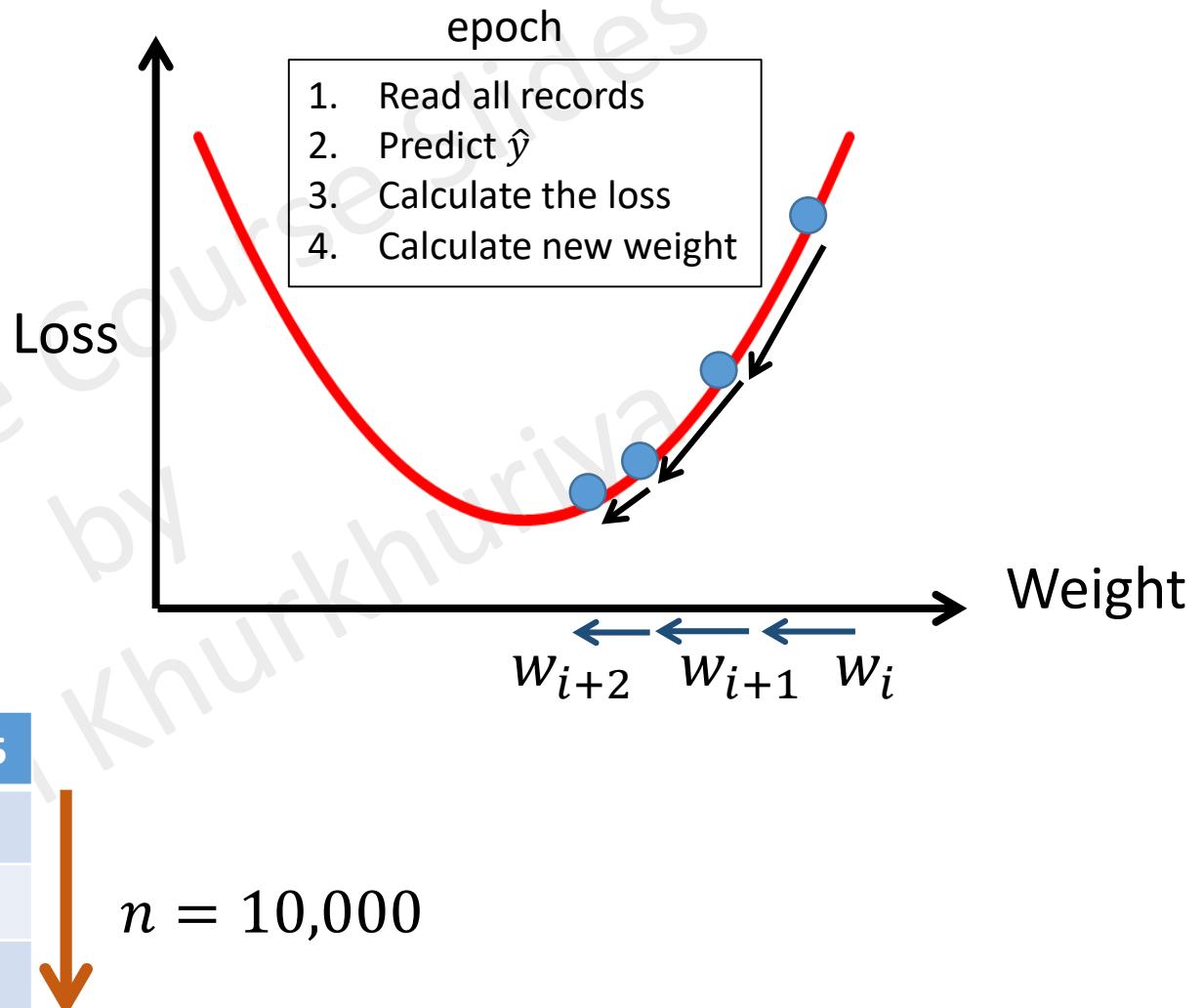
$$w_i = w_j - \eta \Delta w$$

η = Learning Rate

Gradient Descent

$$w_{i+1} = w_i - \eta * f'(w)$$

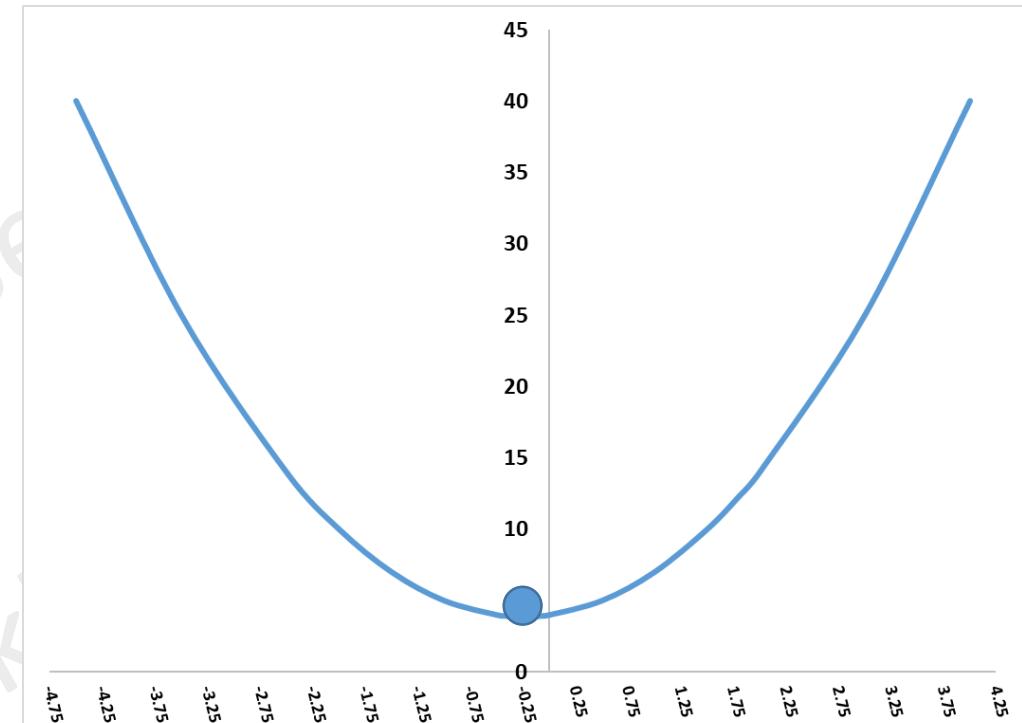
$$L2Norm = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$



Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
...
...
...

Iteration	W	f (W)	f'(W)	$w_{i+1} = w_i - \eta * f'(w)$
1	2.00000	14.00000	9.00000	1.10000
2	1.10000	7.52000	5.40000	0.56000
3	0.56000	5.18720	3.24000	0.23600
4	0.23600	4.34739	1.94400	0.04160
5	0.04160	4.04506	1.16640	-0.07504
6	-0.07504	3.93622	0.69984	-0.14502
7	-0.14502	3.89704	0.41990	-0.18701
8	-0.18701	3.88293	0.25194	-0.21221
9	-0.21221	3.87786	0.15117	-0.22733
10	-0.22733	3.87603	0.09070	-0.23640
11	-0.23640	3.87537	0.05442	-0.24184
12	-0.24184	3.87513	0.03265	-0.24510
13	-0.24510	3.87505	0.01959	-0.24706
14	-0.24706	3.87502	0.01175	-0.24824
15	-0.24824	3.87501	0.00705	-0.24894
16	-0.24894	3.87500	0.00423	-0.24937
17	-0.24937	3.87500	0.00254	-0.24962
18	-0.24962	3.87500	0.00152	-0.24977
19	-0.24977	3.87500	0.00091	-0.24986
20	-0.24986	3.87500	0.00055	-0.24992
21	-0.24992	3.87500	0.00033	-0.24995

$\eta = 0.1$



$n = 10,000$

21 epochs

210,000 read operations

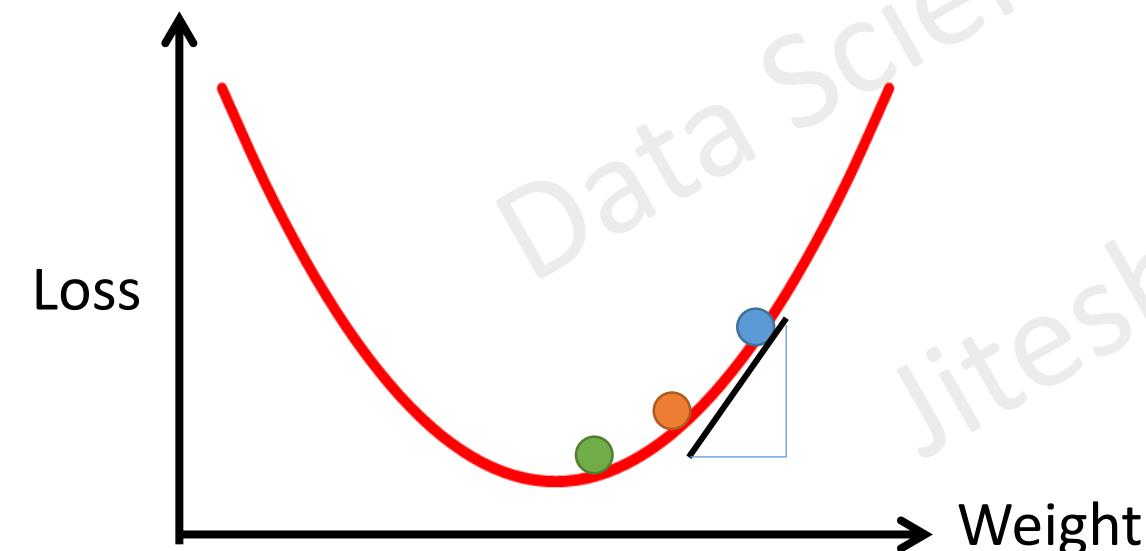
Batch Gradient Descent

x ₁	x ₂	x _n

Sum of All before taking one step (epoch)

Long time to reach the bottom

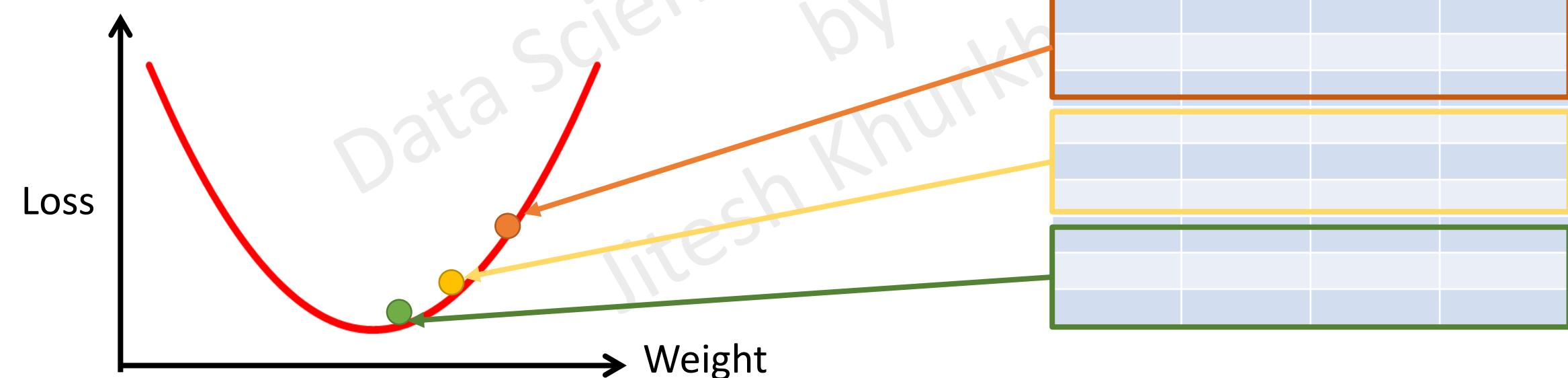
$$w_{i+1} = w_i - \eta * f'(w)$$



Mini Batch Gradient Descent

x1	x2	xn

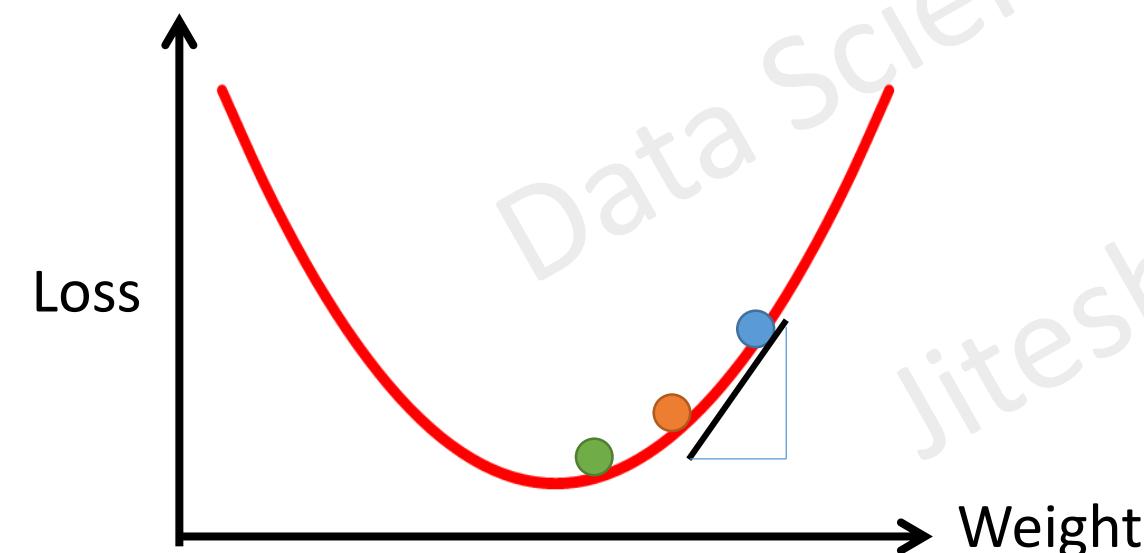
Mini-Batch Gradient Descent



Online or Stochastic Gradient Descent

x1	x2	Xn

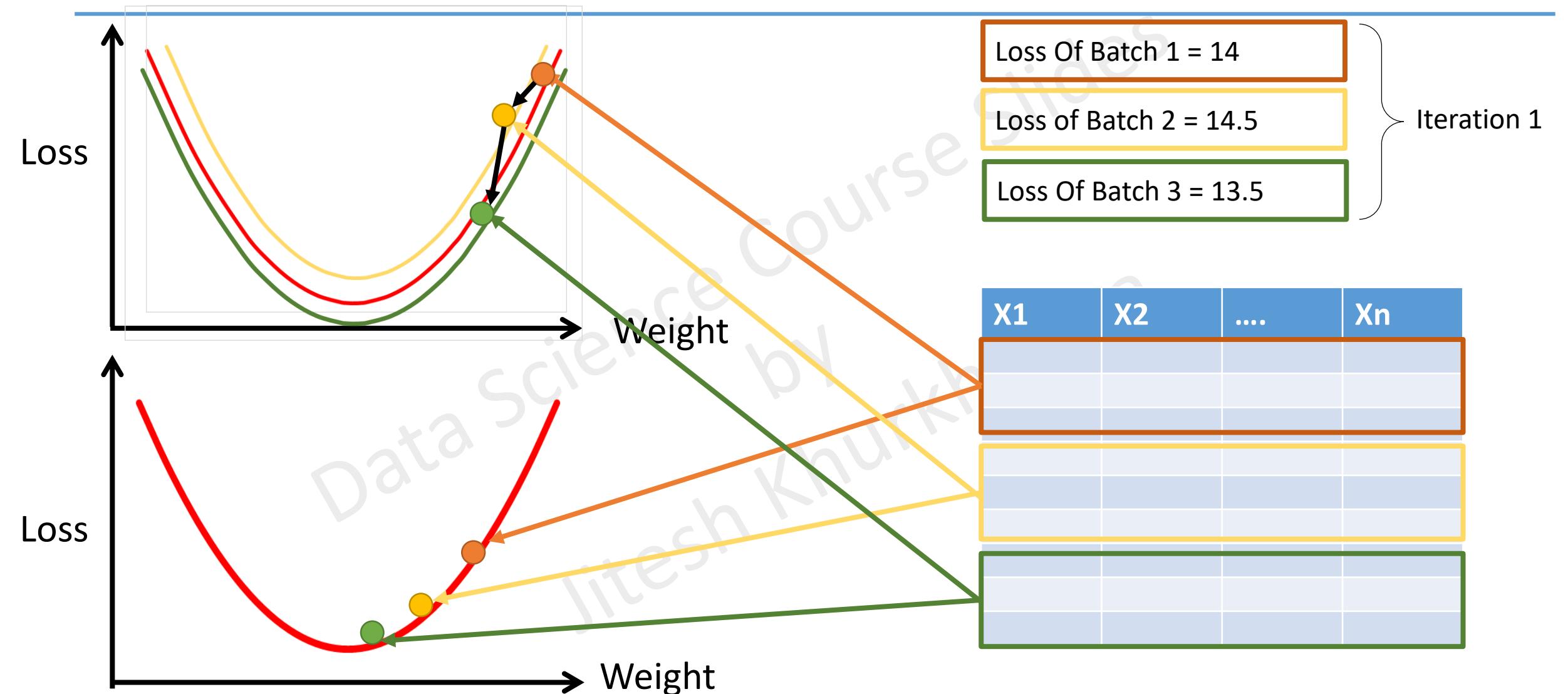
One observation at a time



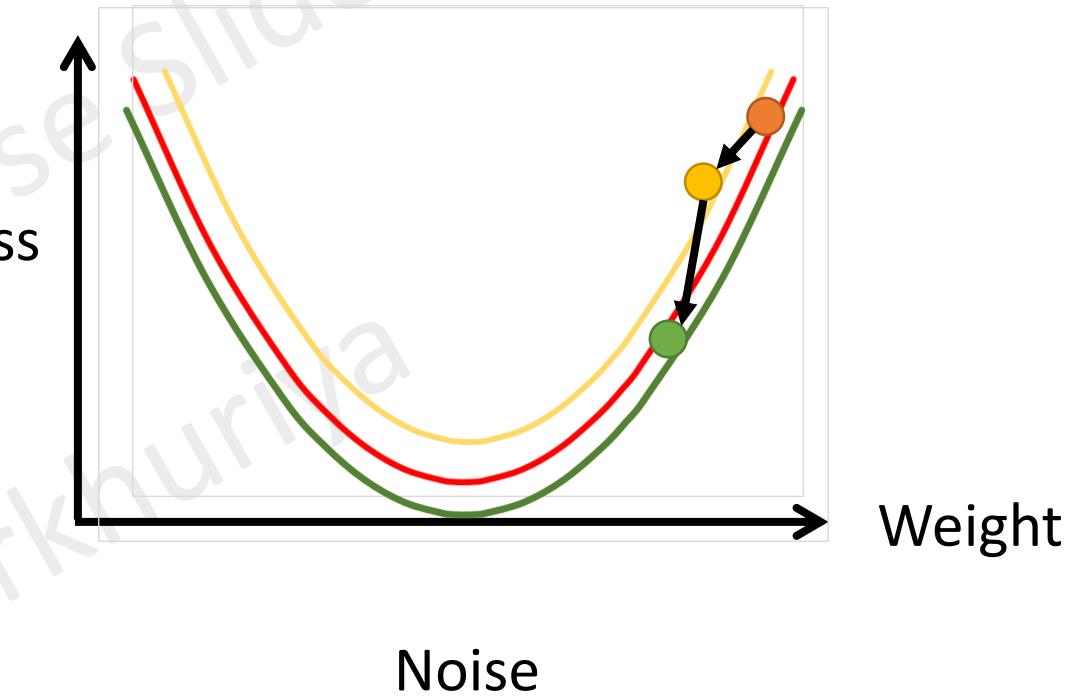
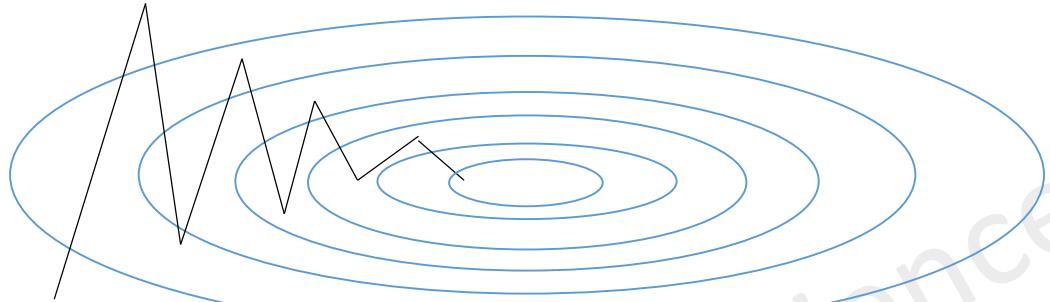
$$w_{i+1} = w_i - \eta * f'(w)$$

SGD With Momentum

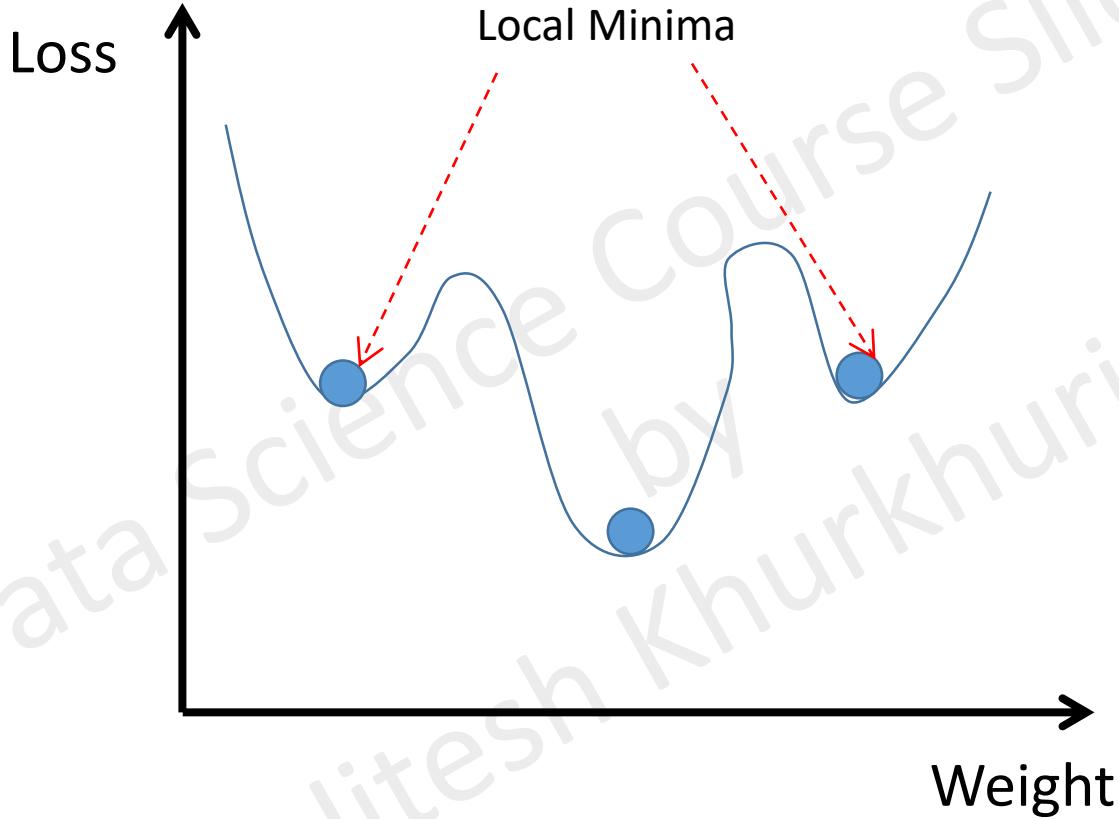
Mini Batch Gradient Descent



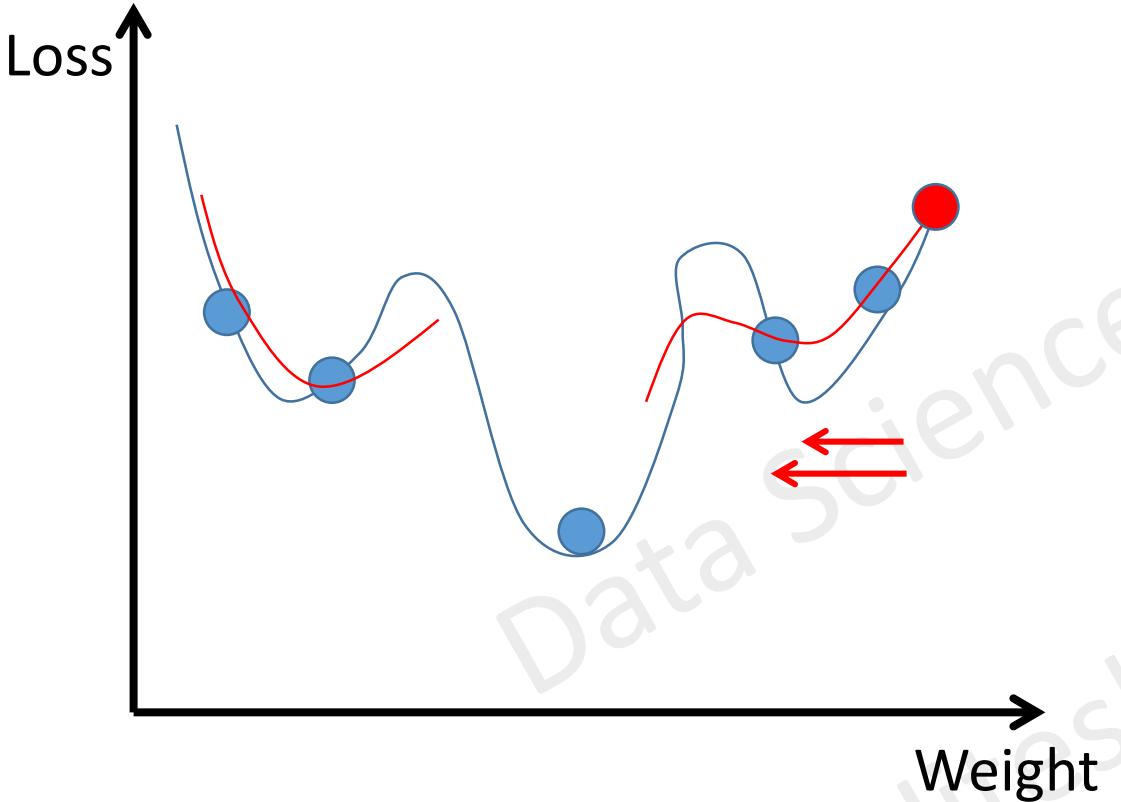
Mini Batch Gradient Descent



Problem of Local Minima



Add Momentum to the Weight



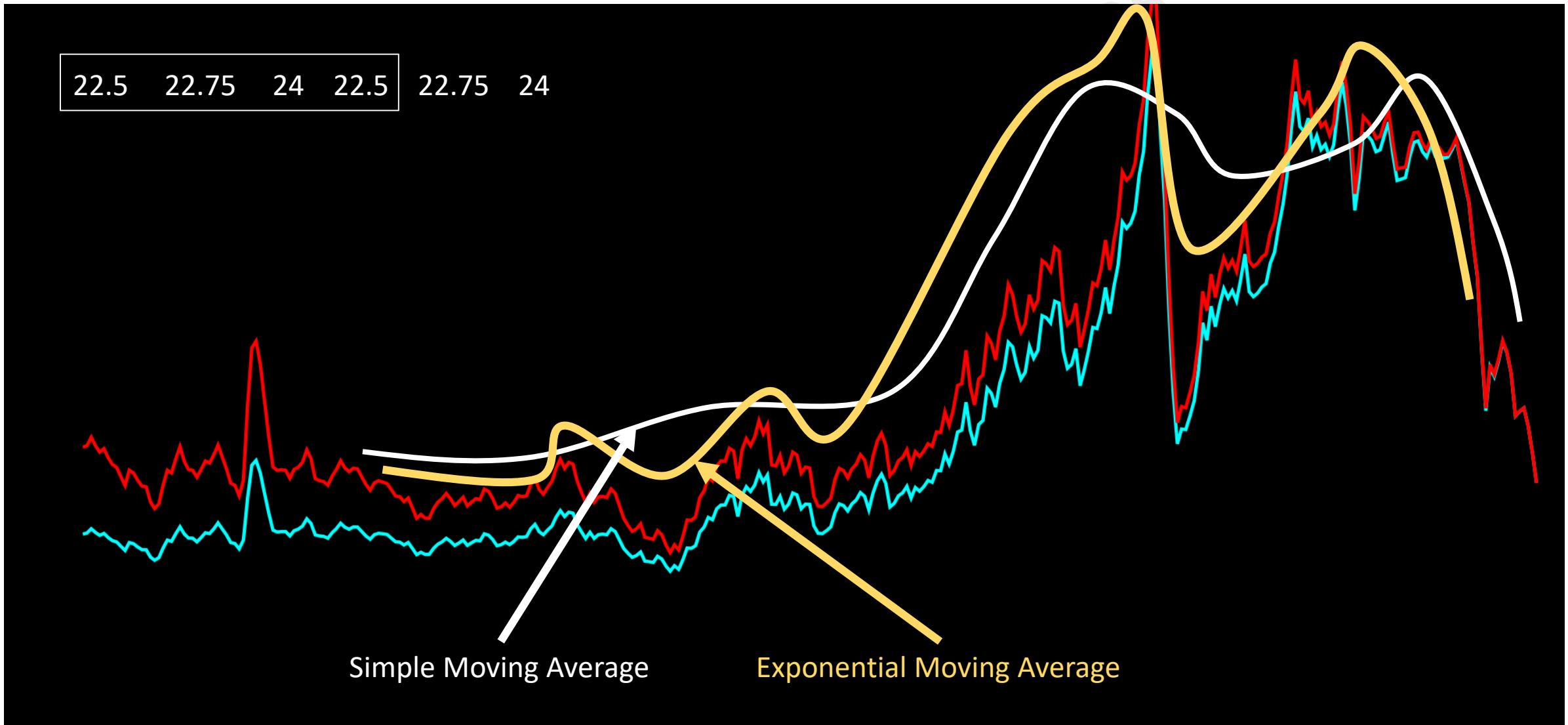
$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w_i}$$

$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w_i} - \gamma * \eta * \frac{\partial L}{\partial w_{i-1}}$$

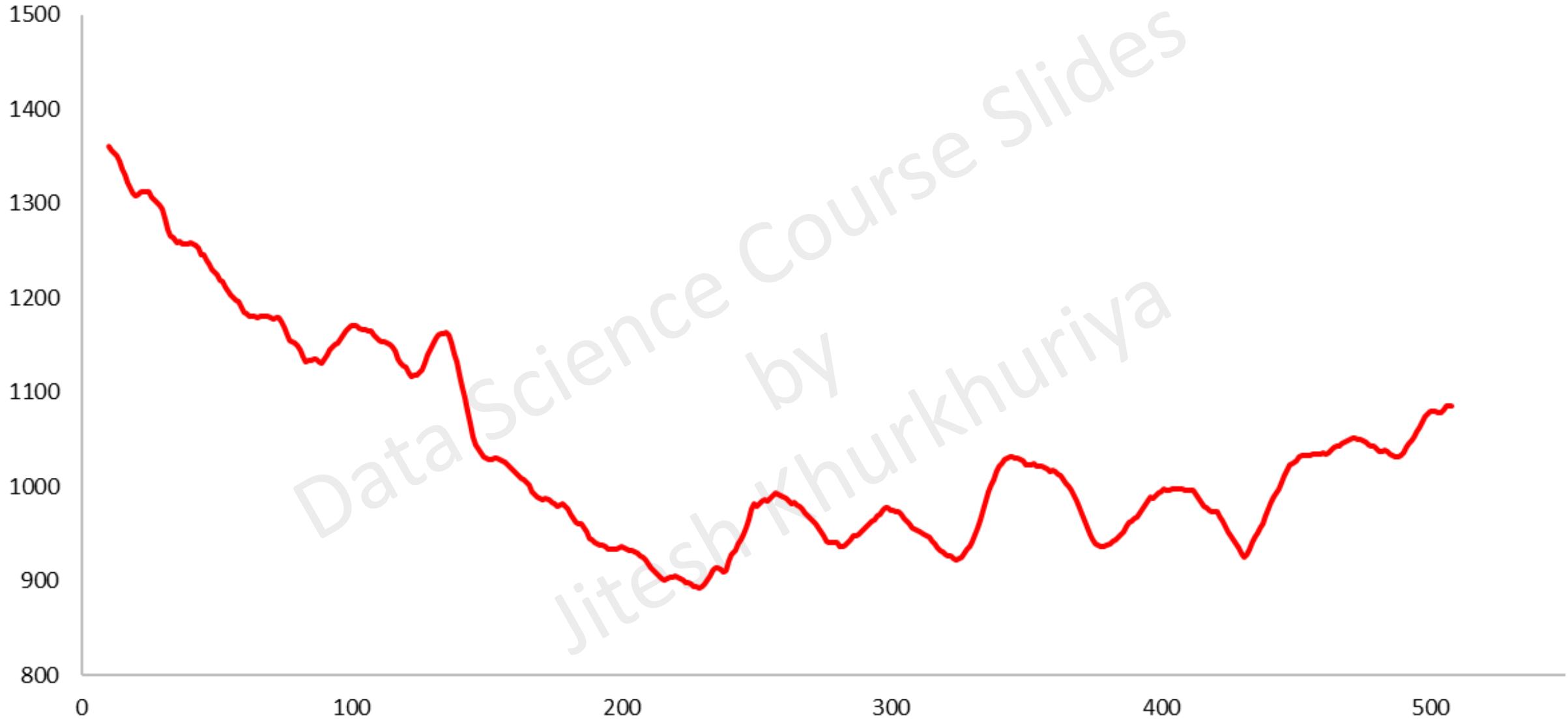
Momentum

Understand the Exponential Moving Average

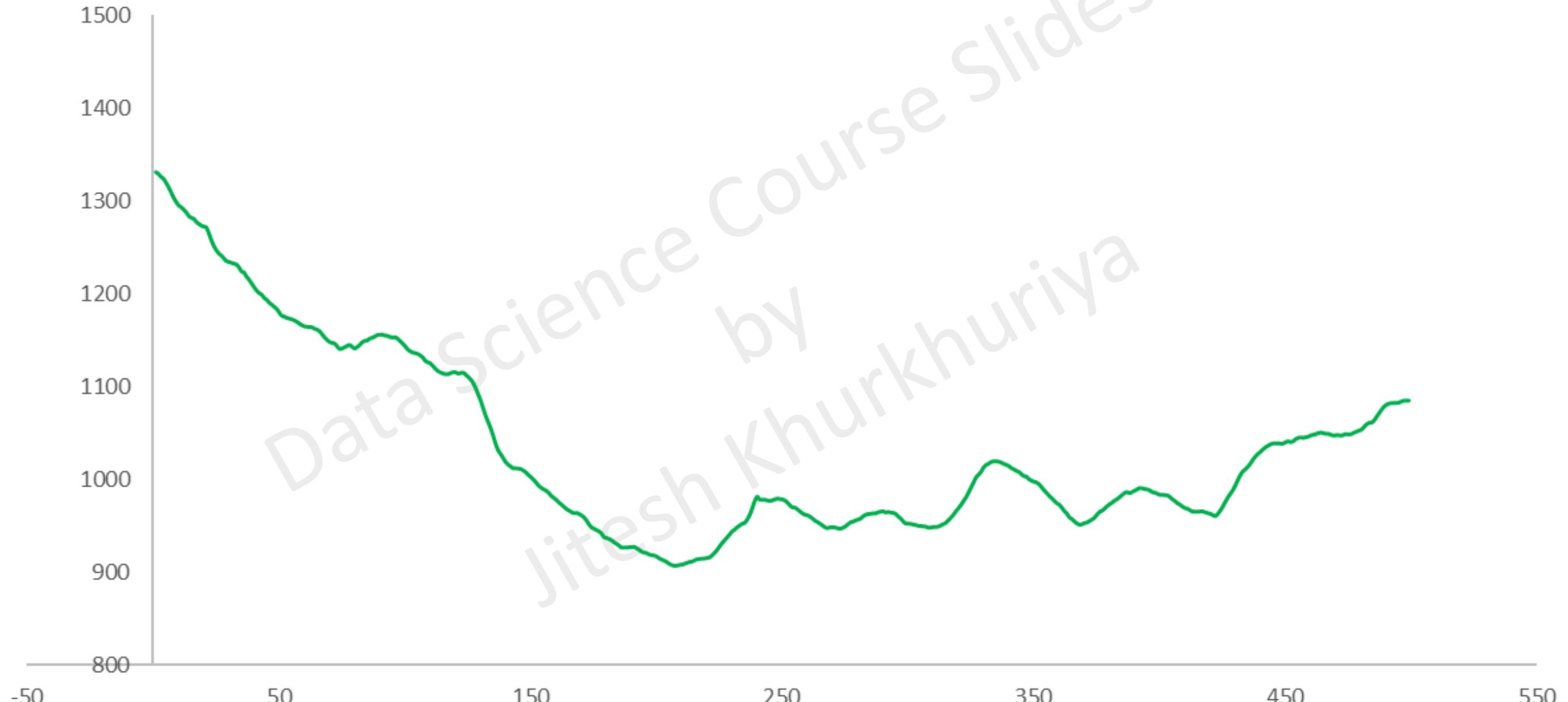
Moving Average



Close Price

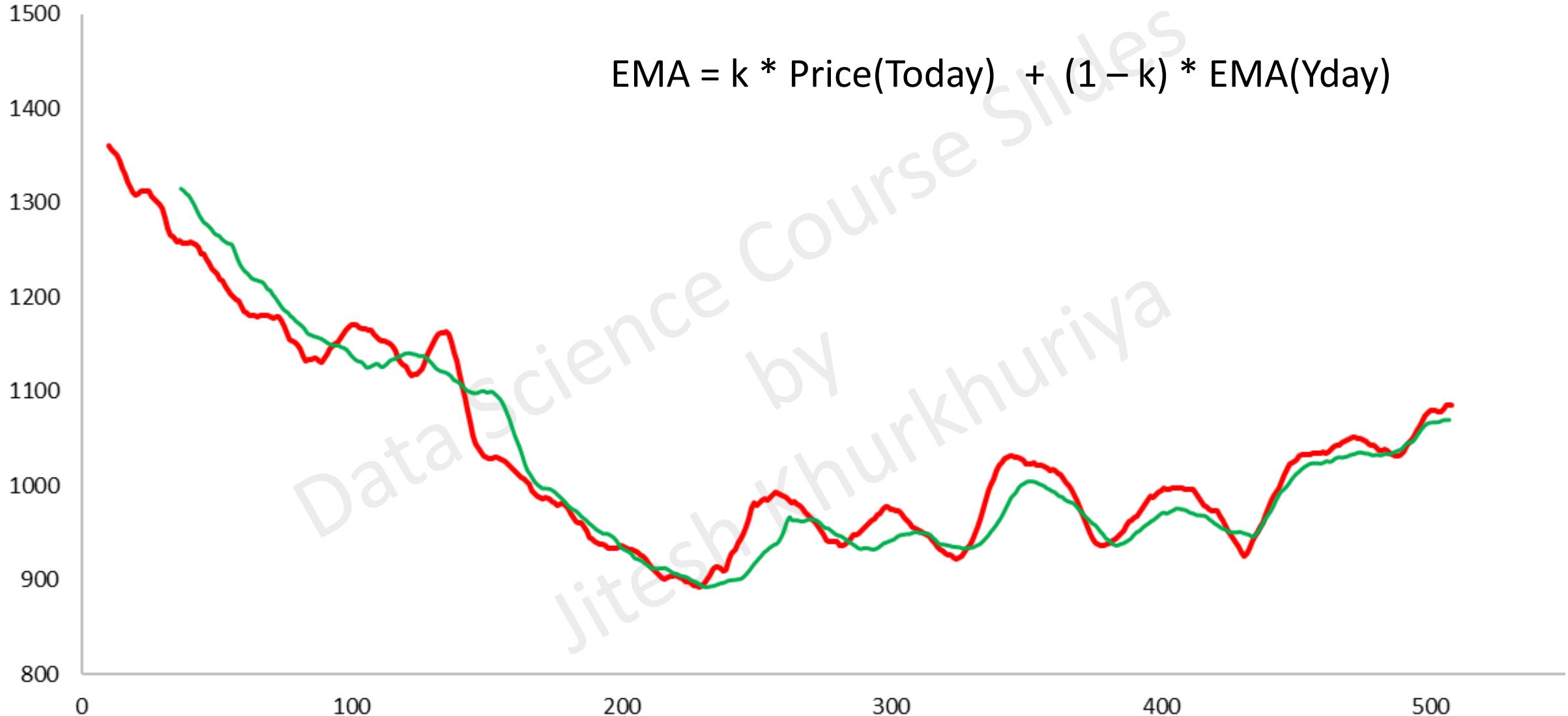


EMA



Close Price

$$\text{EMA} = k * \text{Price(Today)} + (1 - k) * \text{EMA(Yday)}$$



SGD with Momentum

$$\text{EMA} = k * \text{Price(Today)} + (1 - k) * \text{EMA(Yday)}$$

Velocity (V) Prev Velocity Momentum β Gradient

$$\frac{\partial L}{\partial w_i}$$
$$\frac{\partial L}{\partial w_{i-1}}$$

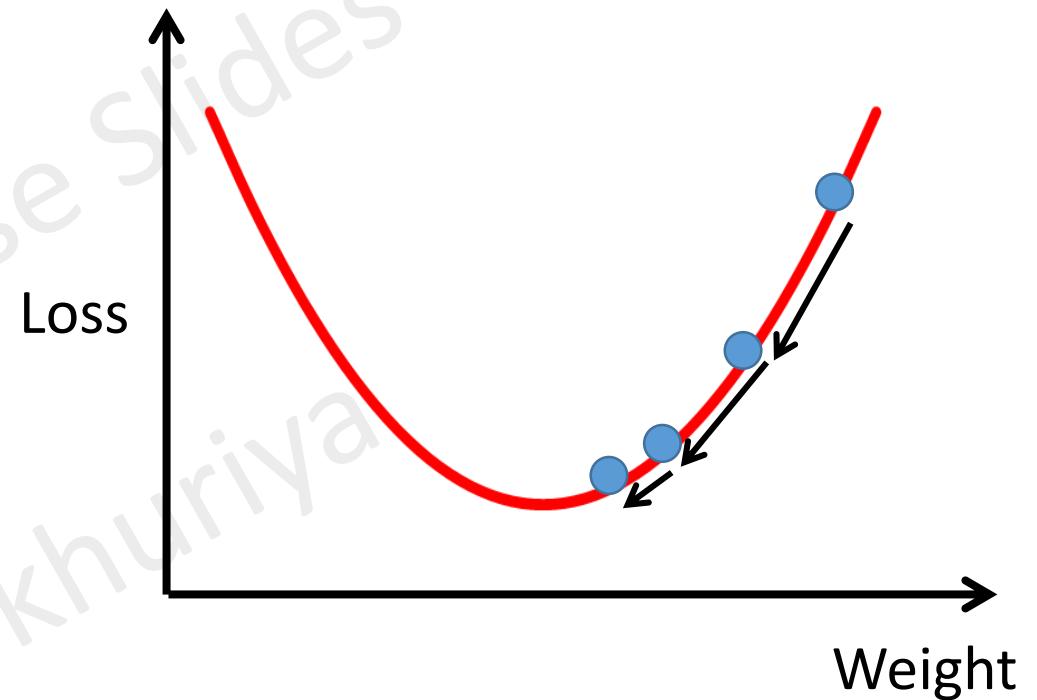
SGD with Momentum

$$\text{EMA} = k * \text{Price(Today)} + (1 - k) * \text{EMA(Yday)}$$

Velocity (V) Prev Velocity Momentum β Gradient $\frac{\partial L}{\partial w_i}$

$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w_i}$$

$$w_{i+1} = w_i - \eta * V_i$$

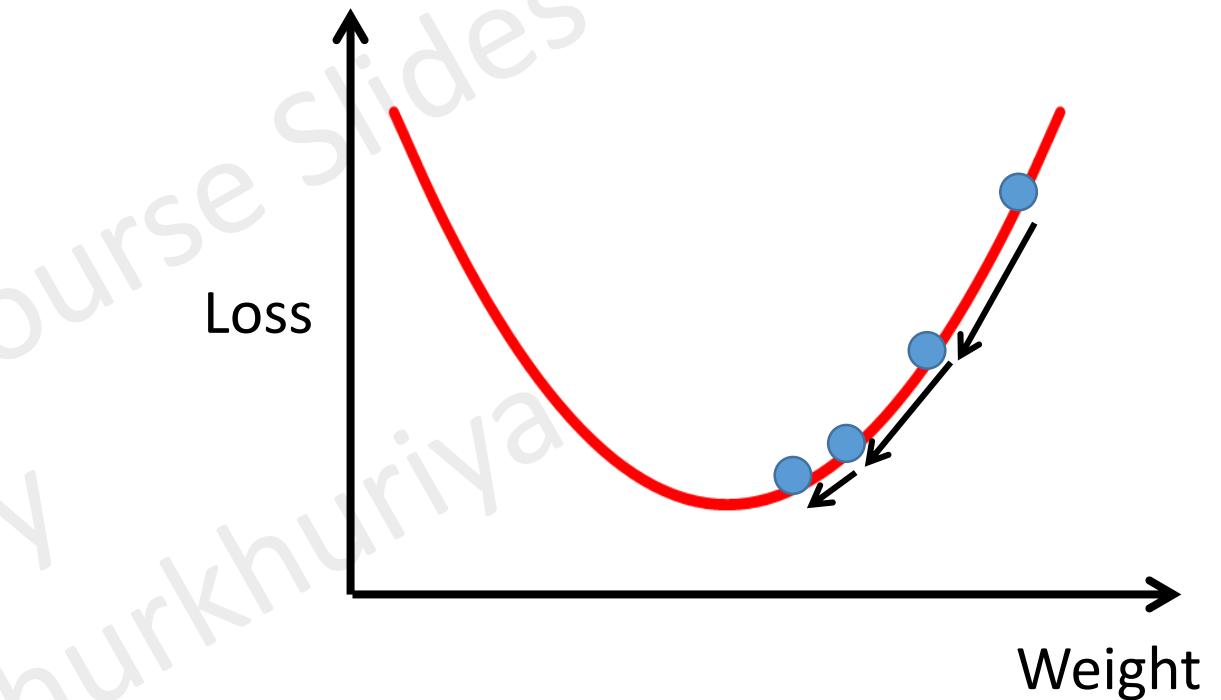


SGD with Momentum

$$V_i = \beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w_i}$$

$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w_i}$$

$$w_{i+1} = w_i - \eta * V_i$$



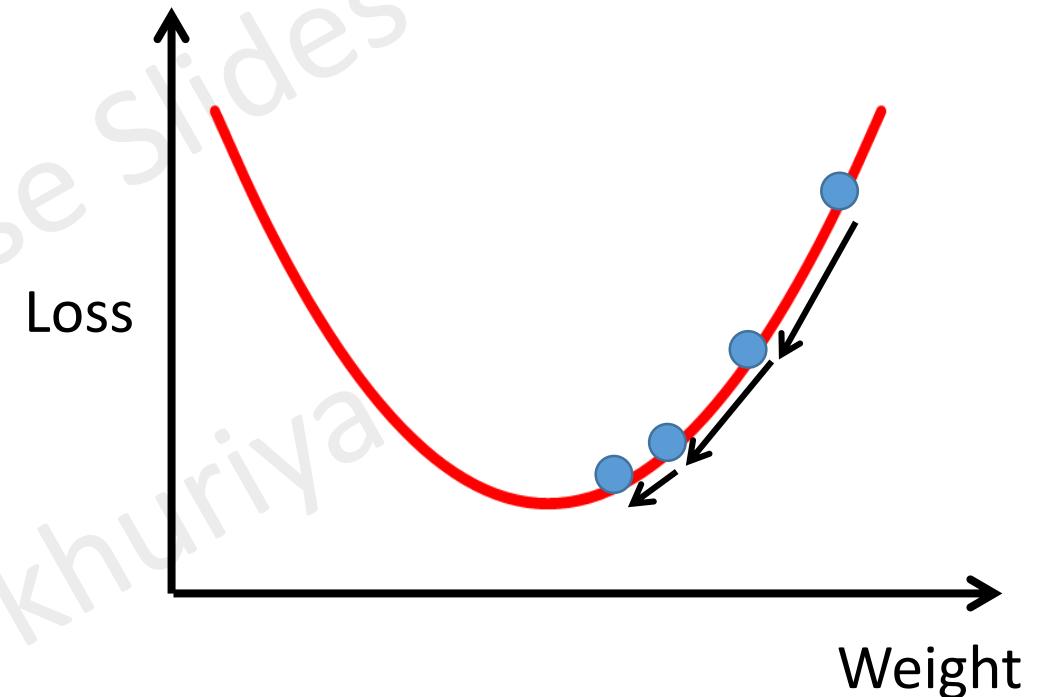
$$V_i = \beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w_i}$$

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$V_i = \beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w_i}$$

$$V_{i-1} = \beta * V_{i-2} + (1 - \beta) * \frac{\partial L}{\partial w_{i-1}}$$



SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$V_i = \beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w_i}$$

$$w_{i+1} = w_i - \eta * (\beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w_i})$$

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$V_i = \beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w_i}$$

$$w_{i+1} = w_i - \eta * (\beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w_i})$$


$$w_{i+1} = w_i - \eta * (\beta * (\beta * V_{i-2} + (1 - \beta) * \frac{\partial L}{\partial w_{i-1}}) + (1 - \beta) * \frac{\partial L}{\partial w_i})$$

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$w_{i+1} = w_i - \eta * (\beta^2 * V_{i-2} + \beta * (1 - \beta) * \frac{\partial L}{\partial w_{i-1}} + (1 - \beta) * \frac{\partial L}{\partial w_i})$$

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$w_{i+1} = w_i - \eta * (\beta^2 * V_{i-2} + \beta * (1 - \beta) * \frac{\partial L}{\partial w_{i-1}} + (1 - \beta) * \frac{\partial L}{\partial w_i})$$

$$w_{i+1} = w_i - \eta * (0.9^2 * V_{i-2} + 0.9(0.1) * \frac{\partial L}{\partial w_{i-1}} + 0.1 * \frac{\partial L}{\partial w_i})$$

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$w_{i+1} = w_i - \eta * (0.9^2 * V_{i-2} + 0.9(0.1) * \frac{\partial L}{\partial w_{i-1}} + 0.1 * \frac{\partial L}{\partial w_i})$$

$$w_{i+1} = w_i - \eta * (0.81 * V_{i-2} + 0.09 * \frac{\partial L}{\partial w_{i-1}} + 0.1 * \frac{\partial L}{\partial w_i})$$

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$w_{i+1} = w_i - \eta * (0.9^2 * V_{i-2} + 0.9(0.1) * \frac{\partial L}{\partial w_{i-1}} + 0.1 * \frac{\partial L}{\partial w_i})$$

$$w_{i+1} = w_i - \eta * (0.81 * V_{i-2} + 0.09 * \frac{\partial L}{\partial w_{i-1}} + 0.1 * \frac{\partial L}{\partial w_i})$$

$$w_{i+1} = w_i - \eta * (0.729 * V_{i-3} + 0.081 * \frac{\partial L}{\partial w_{i-2}} + 0.09 * \frac{\partial L}{\partial w_{i-1}} + 0.1 * \frac{\partial L}{\partial w_i})$$



Problem With Constant Learning Rate

- Oscillations around the centre
- Higher learning rates would never converge
- Same learning rate for all weights

Optimization Functions With Learning Rate Decay

- Adagrad
- RMSProp

Adagrad

$$w_{i+1} = w_i - \eta_i * \frac{\partial L}{\partial w_i}$$

$$\eta_i = \frac{\eta}{\epsilon + \sqrt{\sum_{i=1}^t \left(\frac{\partial L}{\partial w_i} \right)^2}}$$

$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w_i}$$

- ✓ Higher Learning Rate at the beginning
- ✓ Different learning rates for every iteration
- ✓ Different learning rate for every weight

RMSProp

$$w_{i+1} = w_i - \eta_i * \frac{\partial L}{\partial w_i}$$

$$\eta_i = \frac{\eta}{\epsilon + \sqrt{w_{ave}}}$$

$$w_{ave(i)} = \beta * w_{ave(i-1)} + (1 - \beta) * \left(\frac{\partial L}{\partial w_i} \right)^2$$

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$V_i = \beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w_i}$$

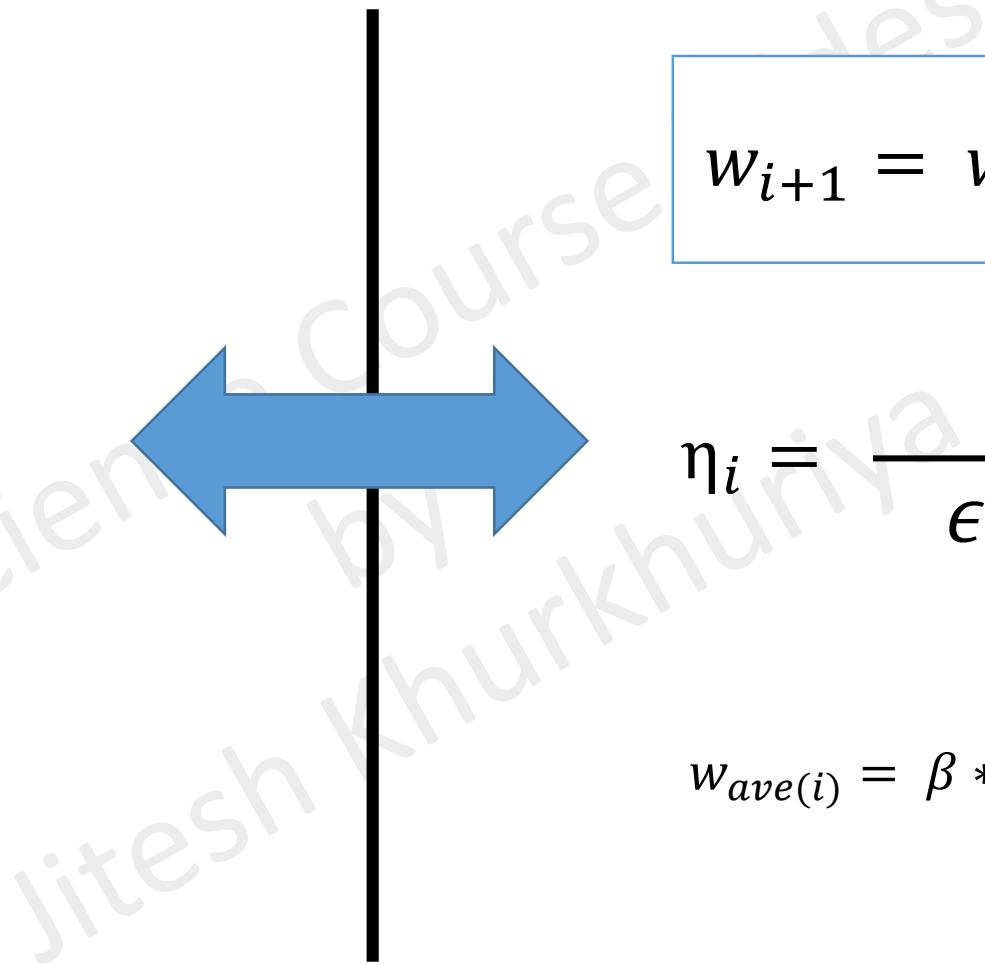
$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w_i}$$

RMSProp

$$w_{i+1} = w_i - \eta_i * \frac{\partial L}{\partial w_i}$$

$$\eta_i = \frac{\eta}{\epsilon + \sqrt{w_{ave}}}$$

$$w_{ave(i)} = \beta * w_{a(i-1)} + (1 - \beta) * \left(\frac{\partial L}{\partial w_i} \right)^2$$



Adam Optimization

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w_i}$$

RMSProp

$$V_i = \beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w_i}$$

$$w_{i+1} = w_i - \frac{\eta}{\epsilon + \sqrt{w_{ave}}} * \frac{\partial L}{\partial w_i}$$

$$w_{i+1} = w_i - \frac{\eta * V_i}{\epsilon + \sqrt{w_{ave}}}$$

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w_i}$$

RMSProp

$$w_{i+1} = w_i - \frac{\eta}{\epsilon + \sqrt{w_{ave}}} * \frac{\partial L}{\partial w_i}$$

$$V_i = \beta_1 * V_{i-1} + (1 - \beta_1) * \frac{\partial L}{\partial w_i}$$

$$\hat{v}_i = \frac{v_i}{1 - \beta_1}$$

Bias Correction

$$w_{i+1} = w_i - \frac{\eta * V_i}{\epsilon + \sqrt{w_{ave}}}$$

$$w_{ave(i)} = \beta_2 * w_{ave(i-1)} + (1 - \beta_2) * \left(\frac{\partial L}{\partial w_i} \right)^2$$

$$\hat{w}_{ave} = \frac{w_{ave}}{1 - \beta_2}$$

SGD with Momentum

$$w_{i+1} = w_i - \eta * V_i$$

$$V_i = \beta_1 * V_{i-1} + (1 - \beta_1) * \frac{\partial L}{\partial w_i}$$

$$\hat{v}_i = \frac{v_i}{1 - \beta_1}$$

$$w_{i+1} = w_i - \frac{\eta * \hat{v}_i}{\epsilon + \sqrt{\hat{w}_{ave}}}$$

$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w_i}$$

RMSProp

$$w_{i+1} = w_i - \frac{\eta}{\epsilon + \sqrt{w_{ave}}} * \frac{\partial L}{\partial w_i}$$

$$w_{ave(i)} = \beta_2 * w_{ave(i-1)} + (1 - \beta_2) * \left(\frac{\partial L}{\partial w_i} \right)^2$$

$$\hat{w}_{ave} = \frac{w_{ave}}{1 - \beta_2}$$

Adam

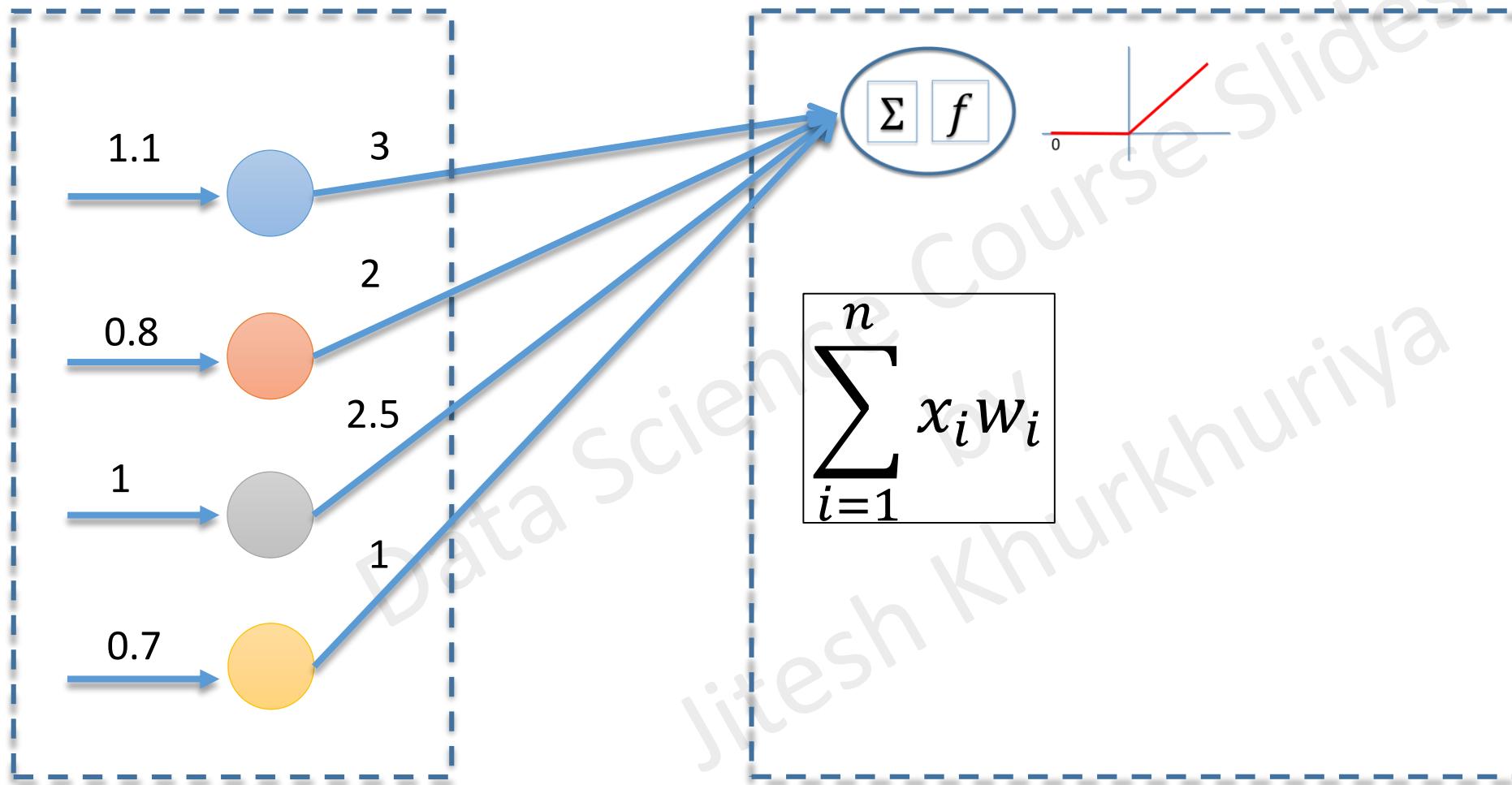
$$w_{i+1} = w_i - \frac{\eta * \hat{v}_i}{\epsilon + \sqrt{\hat{w}_{ave}}}$$

$$\hat{v}_i = \frac{v_i}{1 - \beta_1} \quad \xrightarrow{\text{blue arrow}} \quad V_i = \beta_1 * V_{i-1} + (1 - \beta_1) * \frac{\partial L}{\partial w_i}$$

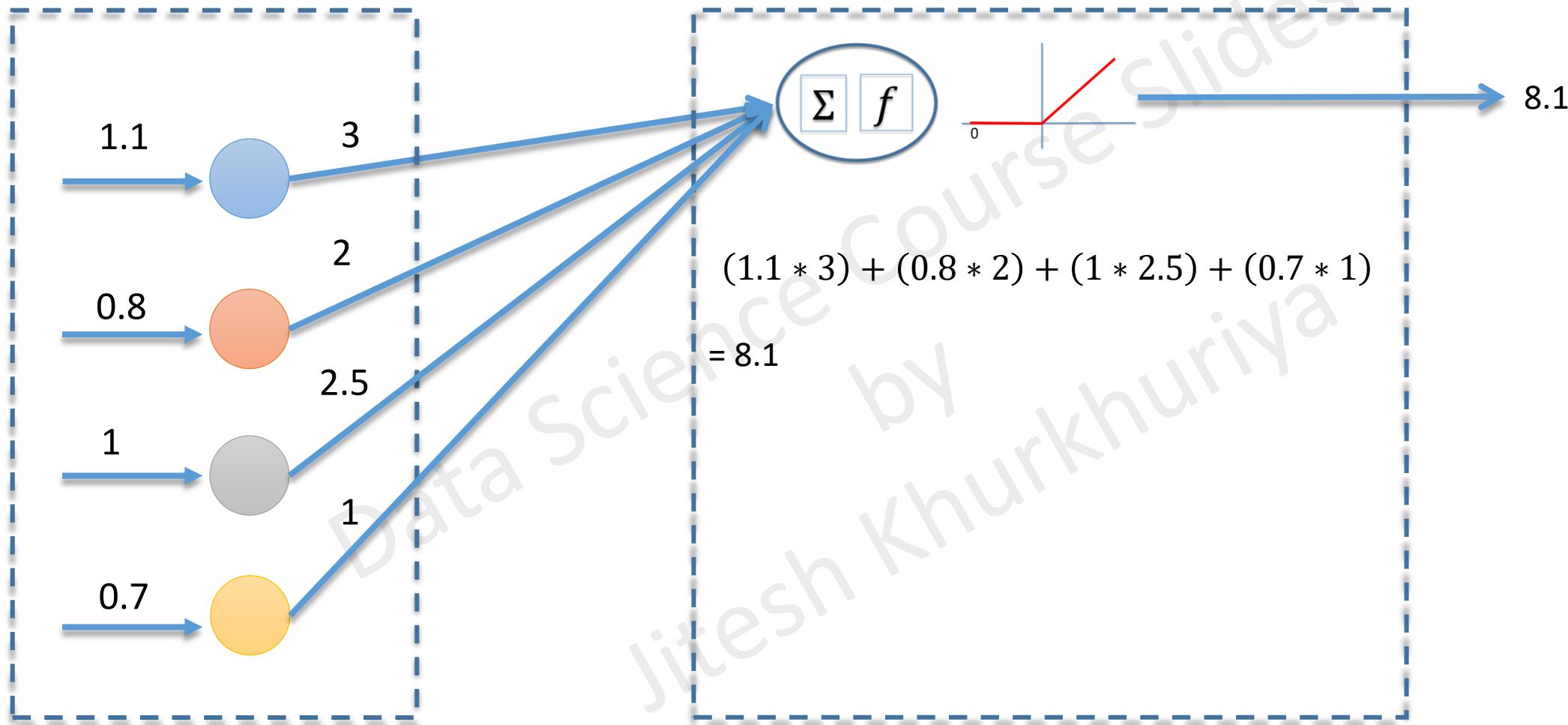
$$\hat{w}_{ave} = \frac{w_{ave}}{1 - \beta_2} \quad \xrightarrow{\text{blue arrow}} \quad w_{ave(i)} = \beta_2 * w_{ave(i-1)} + (1 - \beta_2) * \left(\frac{\partial L}{\partial w_i} \right)^2$$

Vanishing/Exploding Gradients

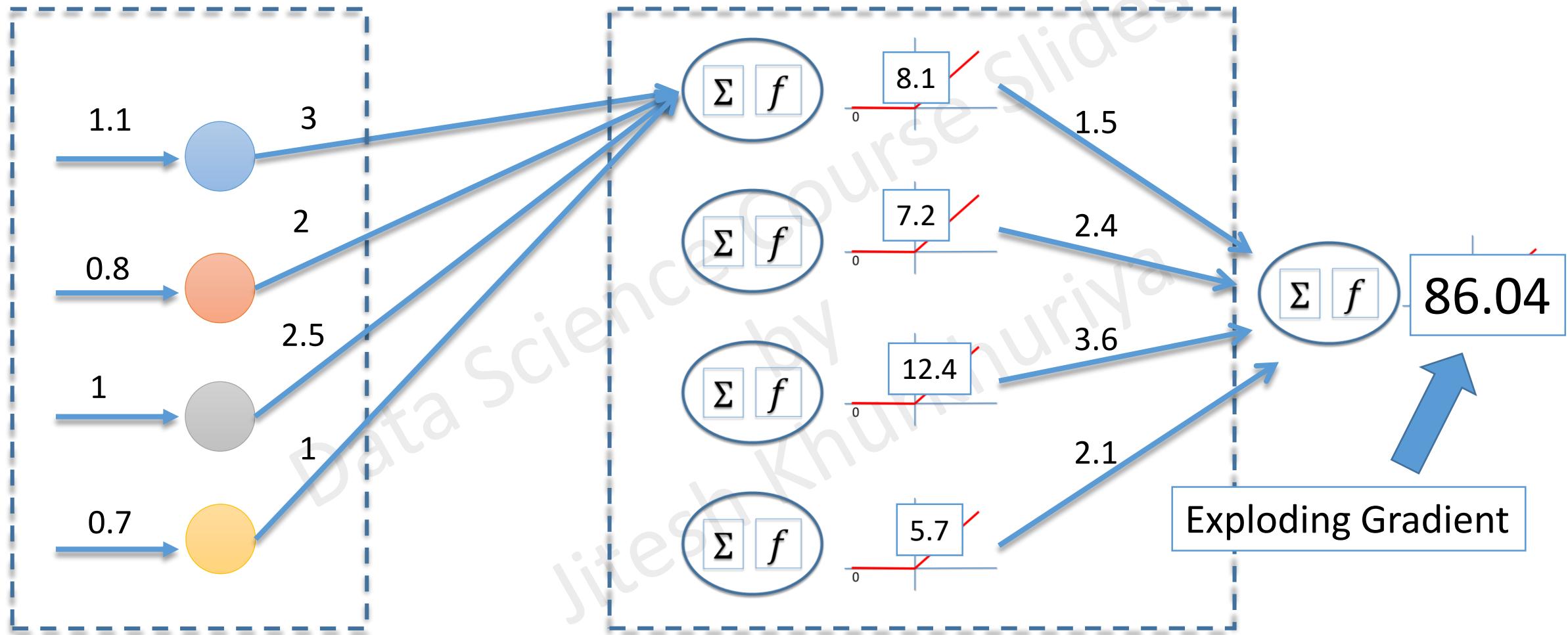
High Weights for all



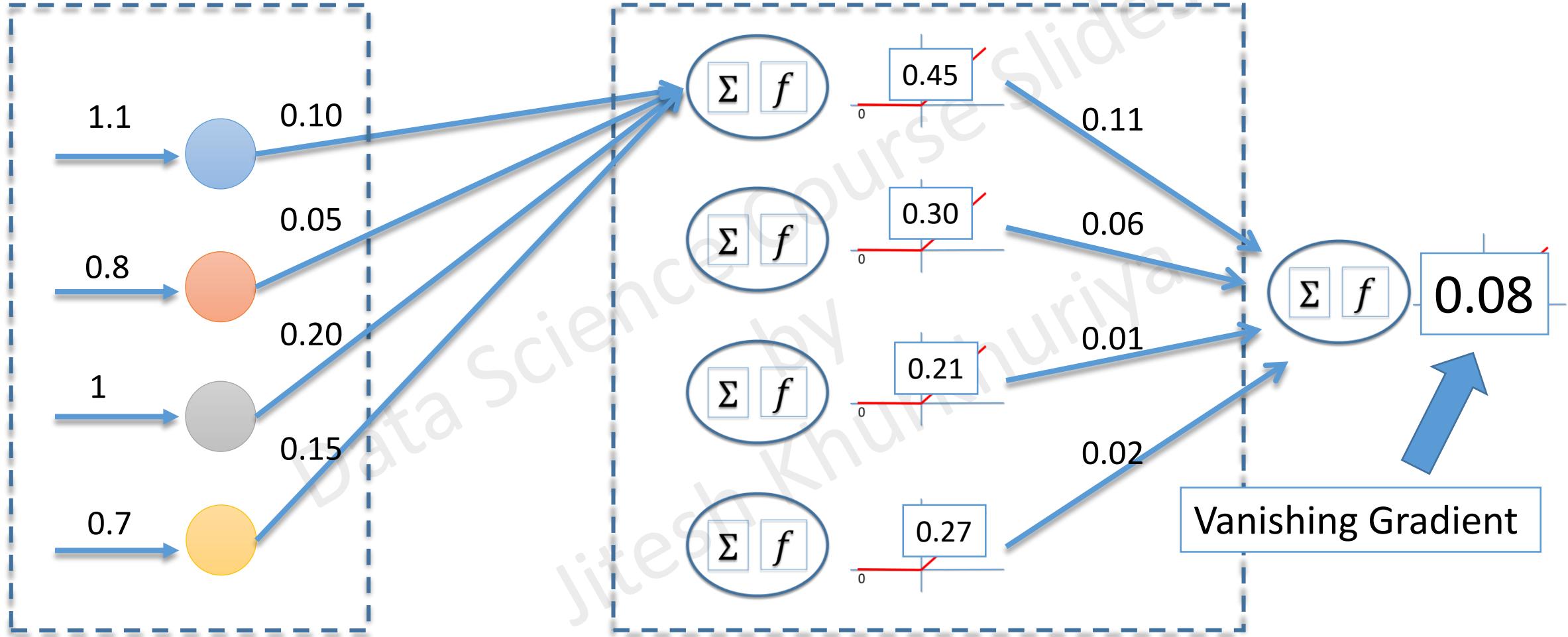
High Weights for all



High Weights for all



Too Low Weights for all



Initializers

Types of Initializers

Constants

- Zeros
- Ones
- Constant

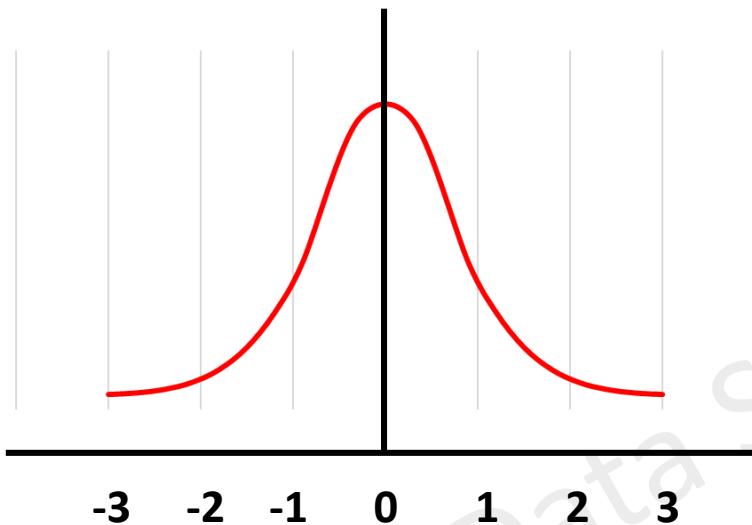
Normal

- RandomNormal
- glorot_normal
- he_normal
- lecun_normal
- TruncatedNormal

Uniform

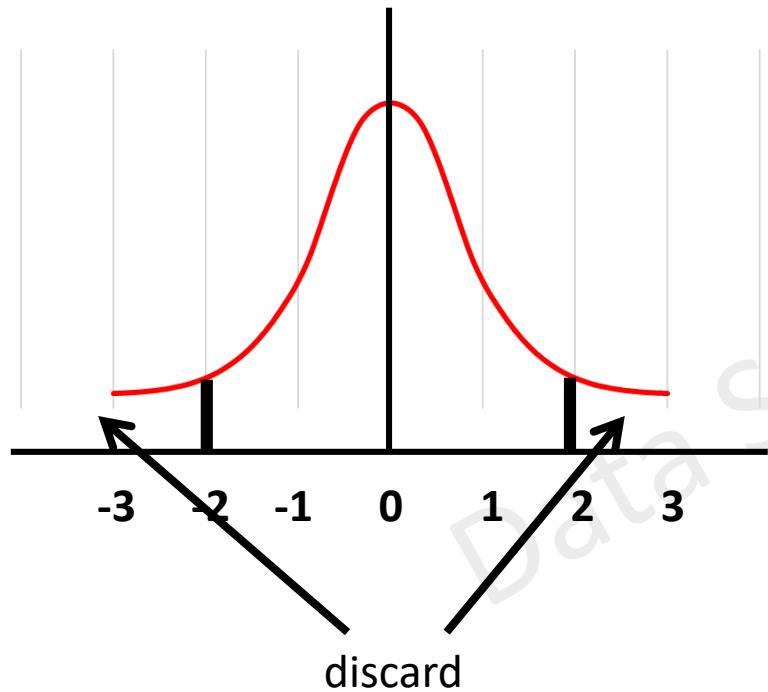
- RandomUniform
- glorot_uniform
- he_uniform
- lecun_uniform

RandomNormal



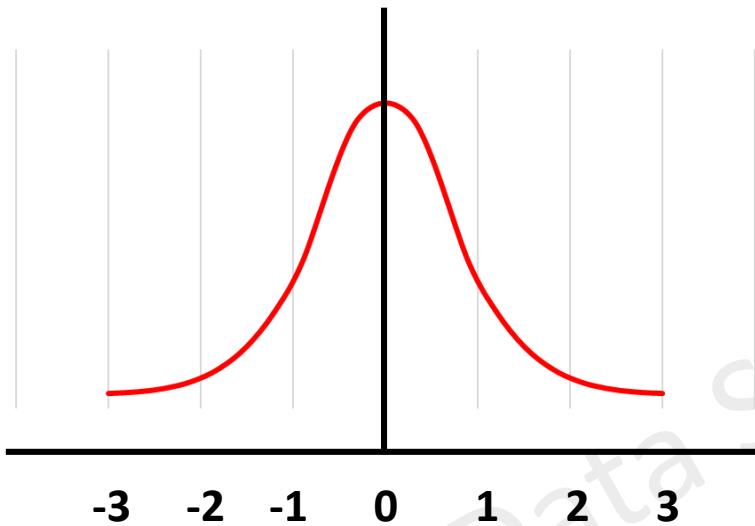
`init = keras.initializers.RandomNormal(mean=0.0, stddev=0.05, seed=1234)`

TruncatedNormal



`init = keras.initializers.TruncatedNormal(mean=0.0, stddev=0.05, seed=1234)`

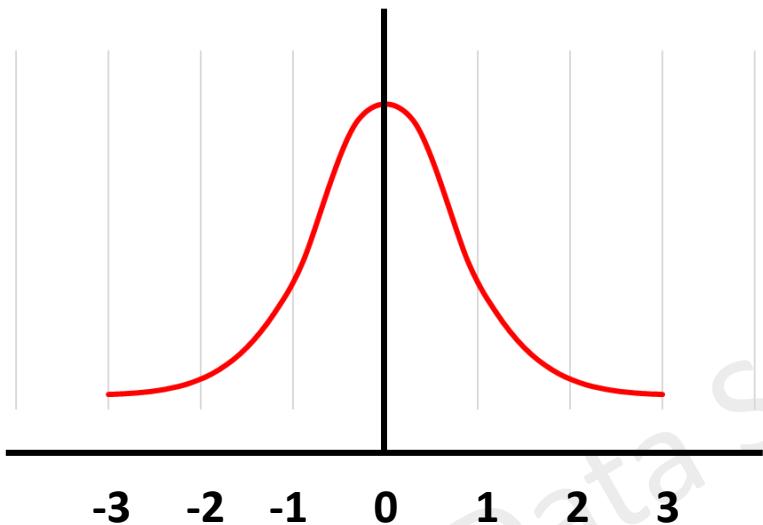
glorot_normal or Xavier Normal Initializer



`init = keras.initializers.glorot_normal(seed=1234)`

$$\sigma = \sqrt{\frac{2}{fan_in + fan_out}}$$

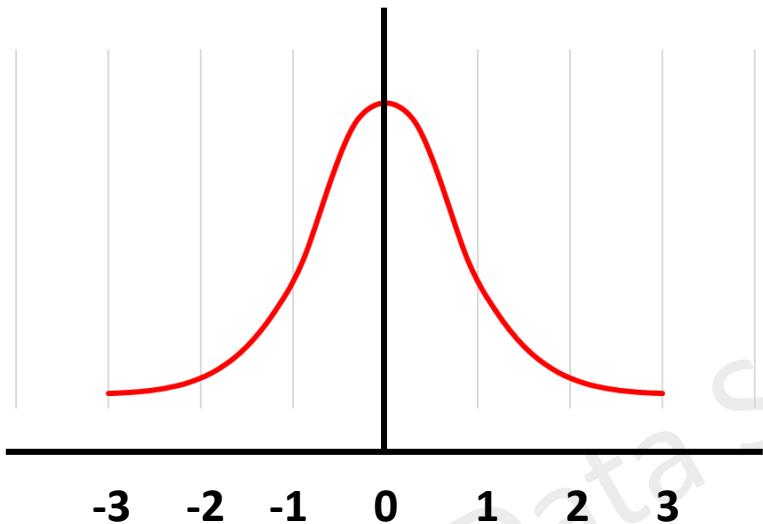
he_normal



`init = keras.initializers.he_normal(seed=1234)`

$$\sigma = \sqrt{\frac{2}{fan_in}}$$

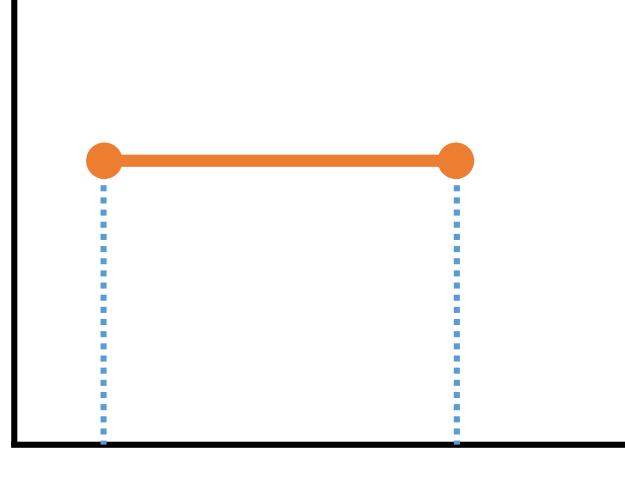
lecun_normal



`init = keras.initializers.lecun_normal(seed=1234)`

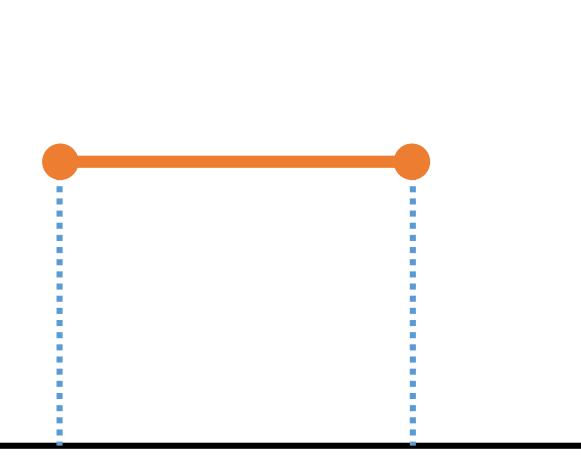
$$\sigma = \sqrt{\frac{1}{fan_in}}$$

RandomUniform



`init = keras.initializers.RandomUniform(minval=-0.5, maxval=0.05, seed=1234)`

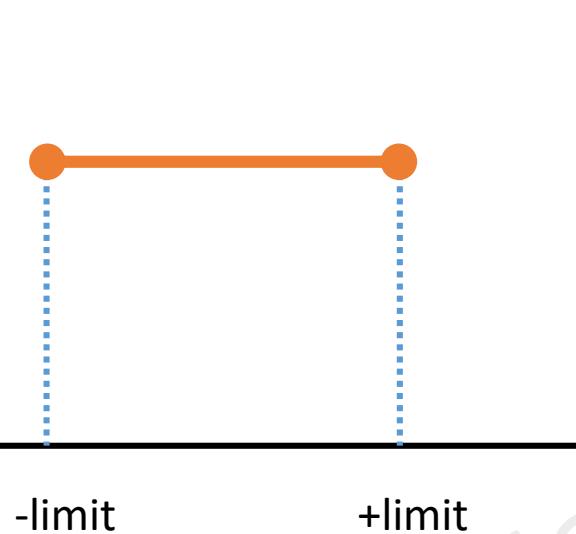
glorot_uniform



`init = keras.initializers.glorot_uniform(seed=1234)`

$$limit = \sqrt{\frac{6}{fan_in + fan_out}}$$

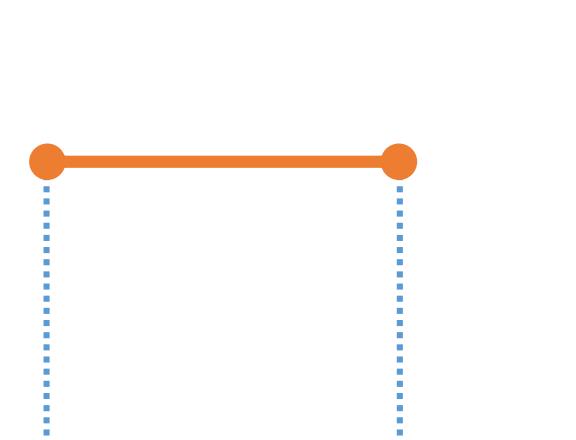
he_uniform



`init = keras.initializers.he_uniform(seed=1234)`

$$limit = \sqrt{\frac{6}{fan_in}}$$

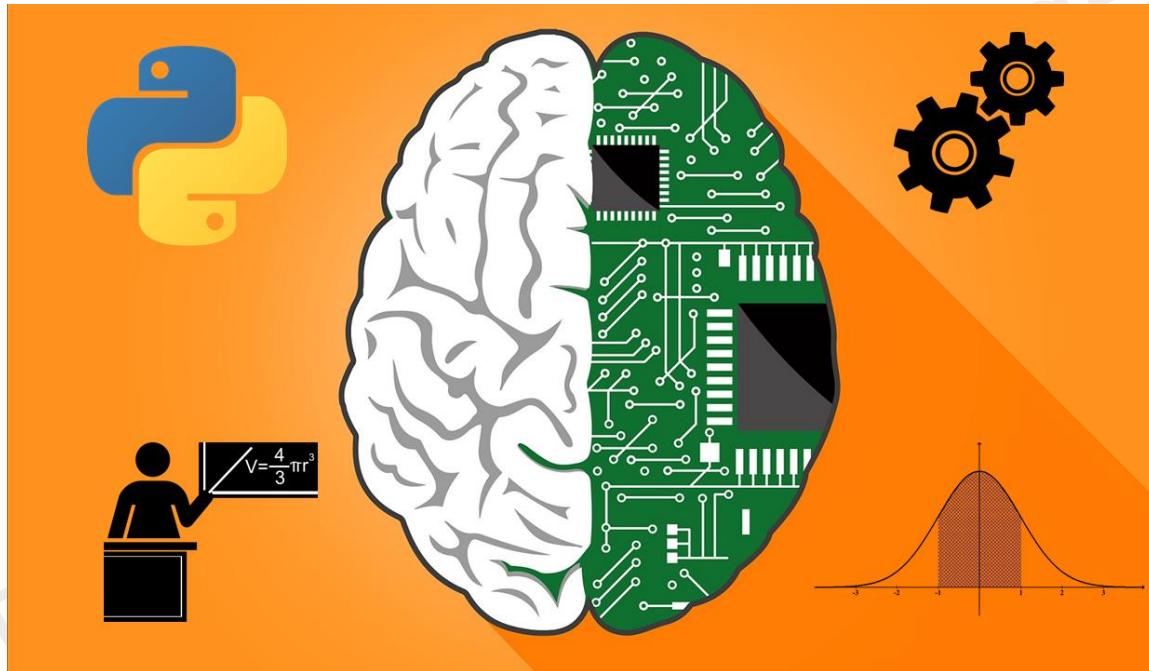
lecun_uniform



`init = keras.initializers.lecun_uniform(seed=1234)`

$$limit = \sqrt{\frac{3}{fan_in}}$$

Complete Data Science and Machine Learning Using Python



Thank You!