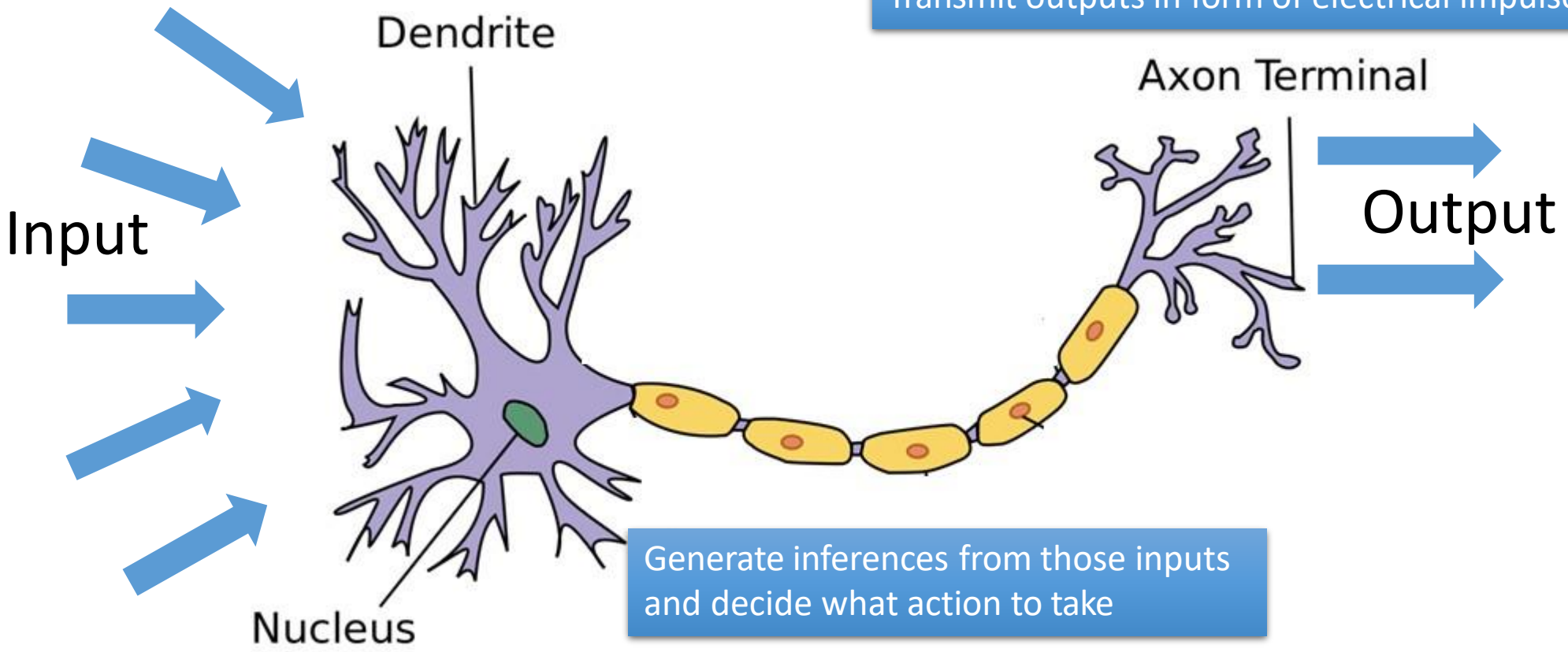
Deep Learning

How Neuron Works?

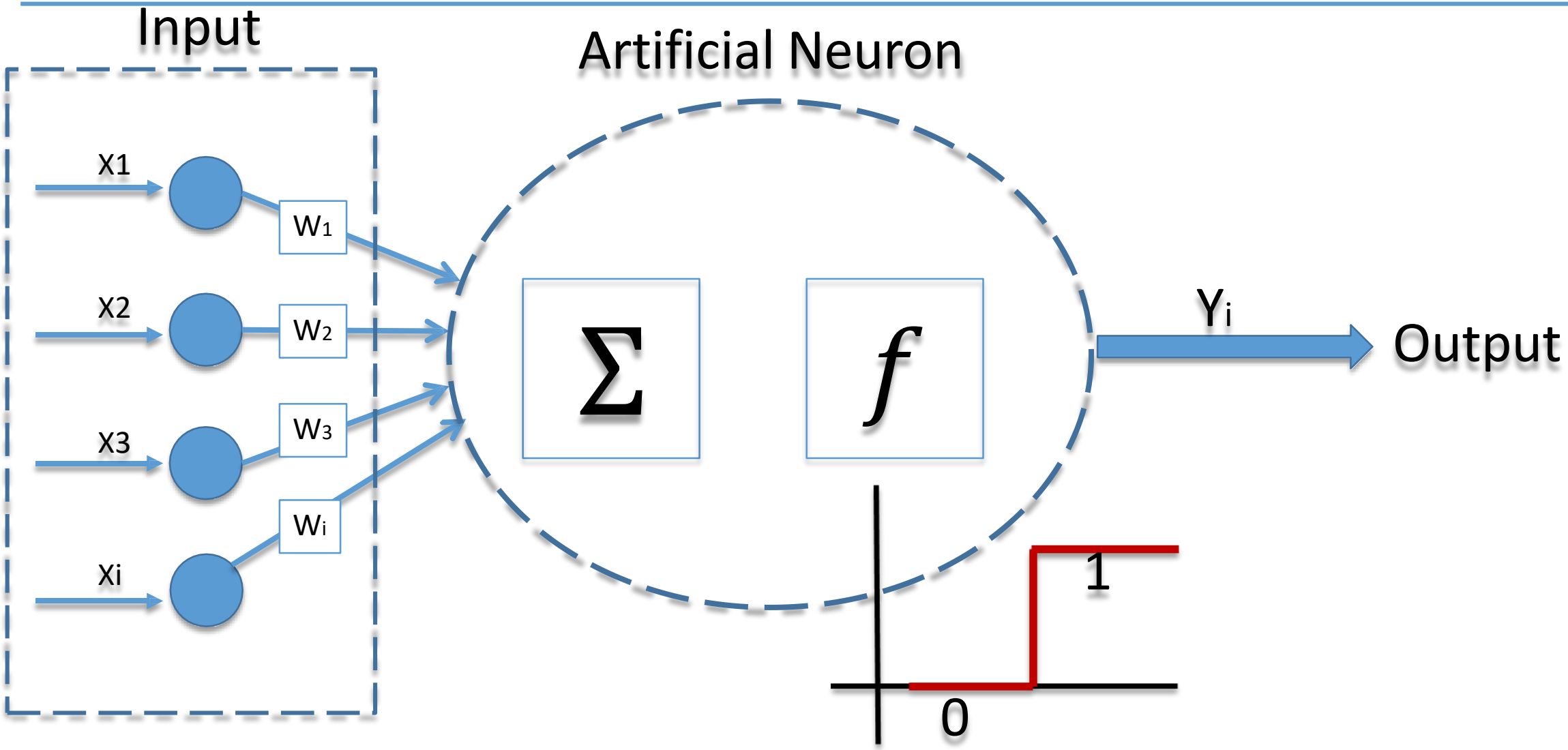
It takes input from other neurons in the form of an electrical impulse.

Transmit outputs in form of electrical impulse



Generate inferences from those inputs and decide what action to take

Artificial Neuron

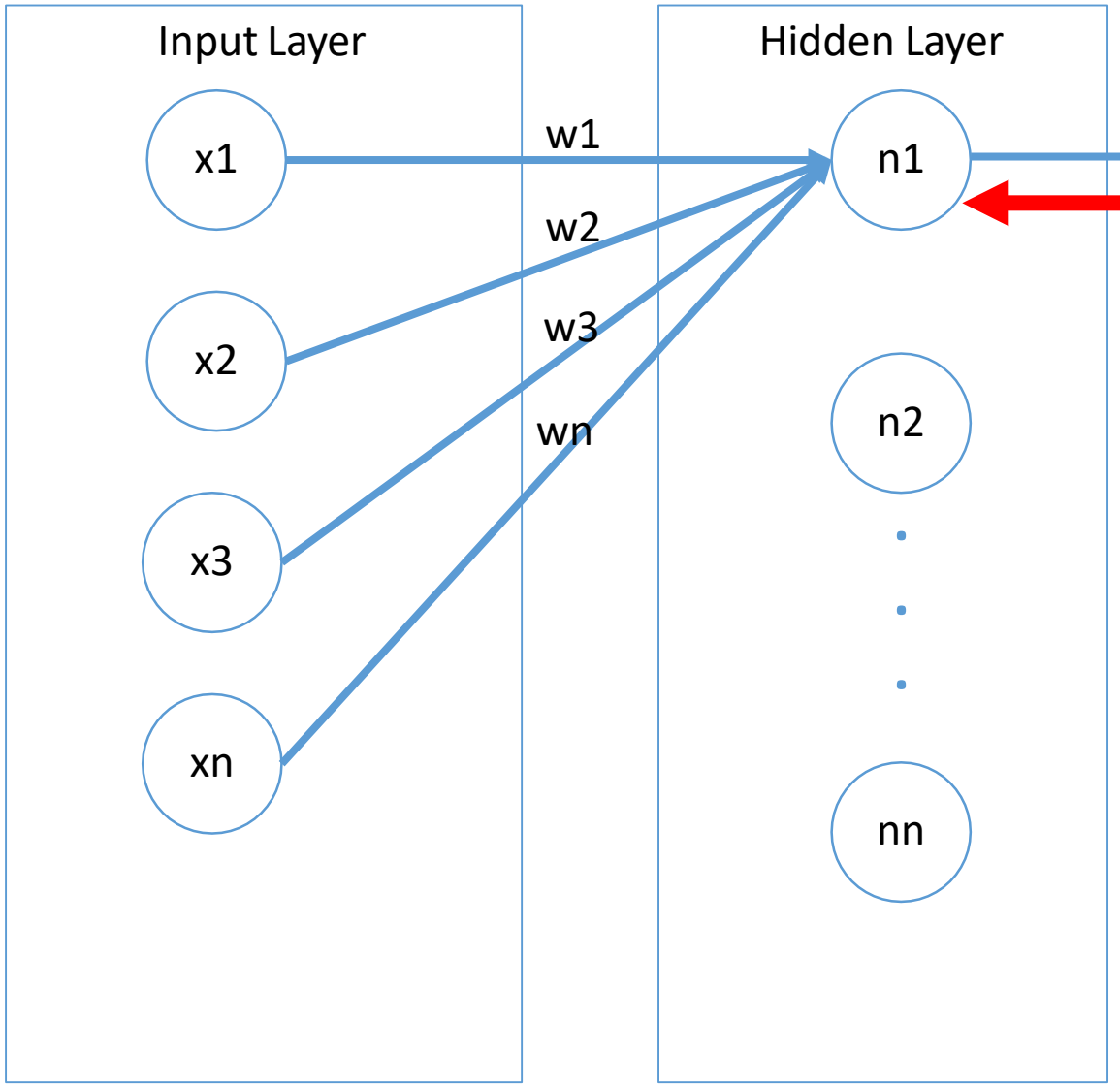


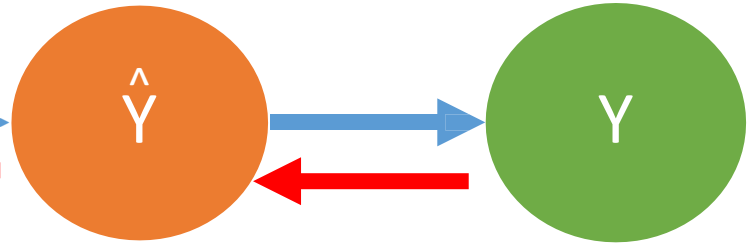
ANN Capabilities

- 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

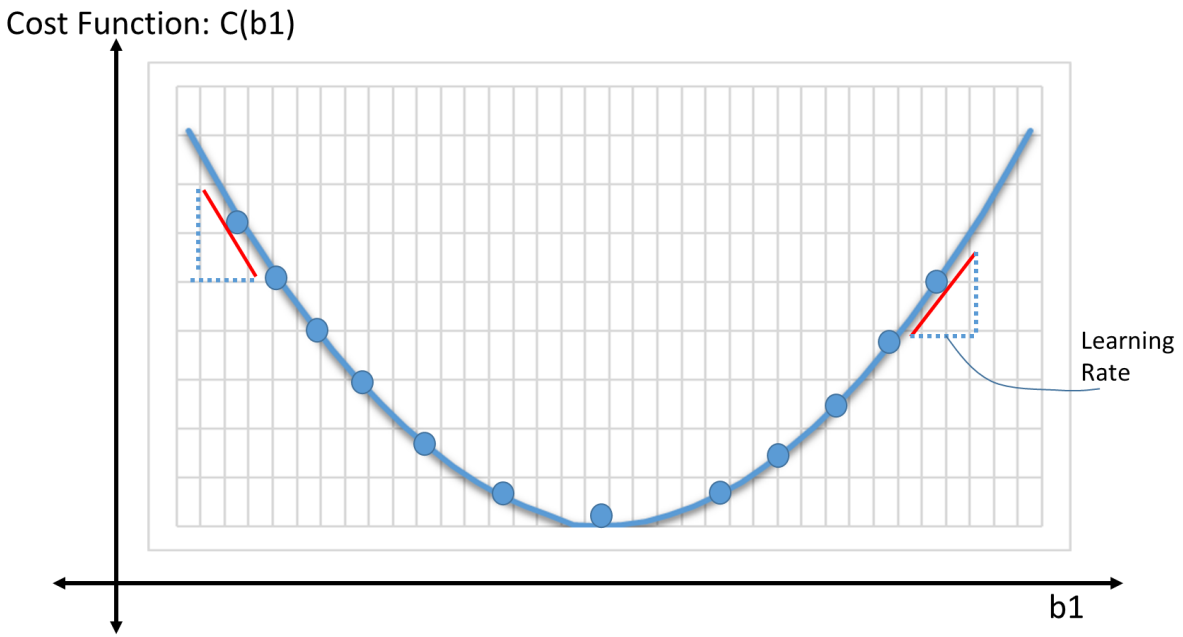
How ANN Works?

How ANN Works?

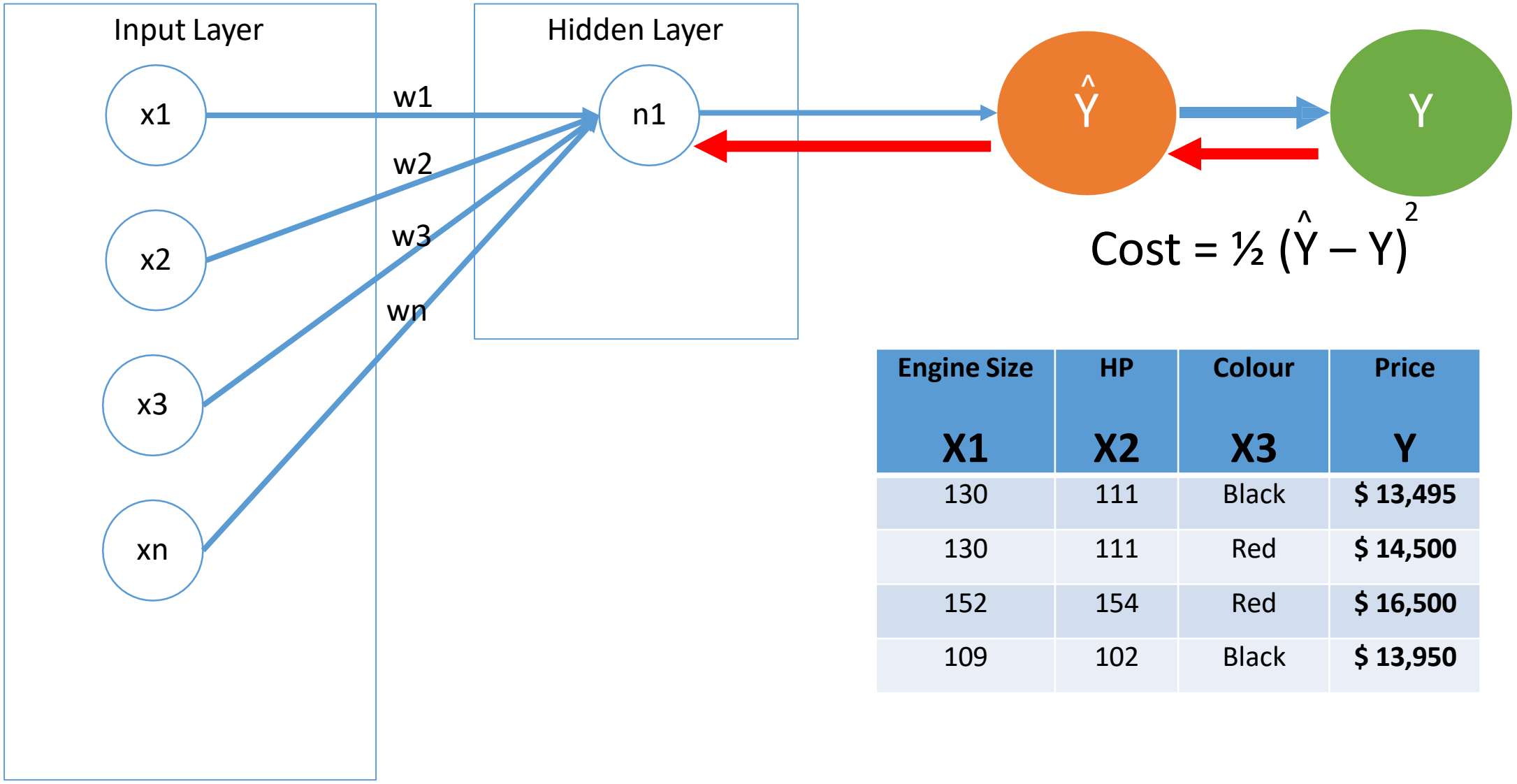




$$Cost = \frac{1}{2}(\hat{Y} - Y)^2$$

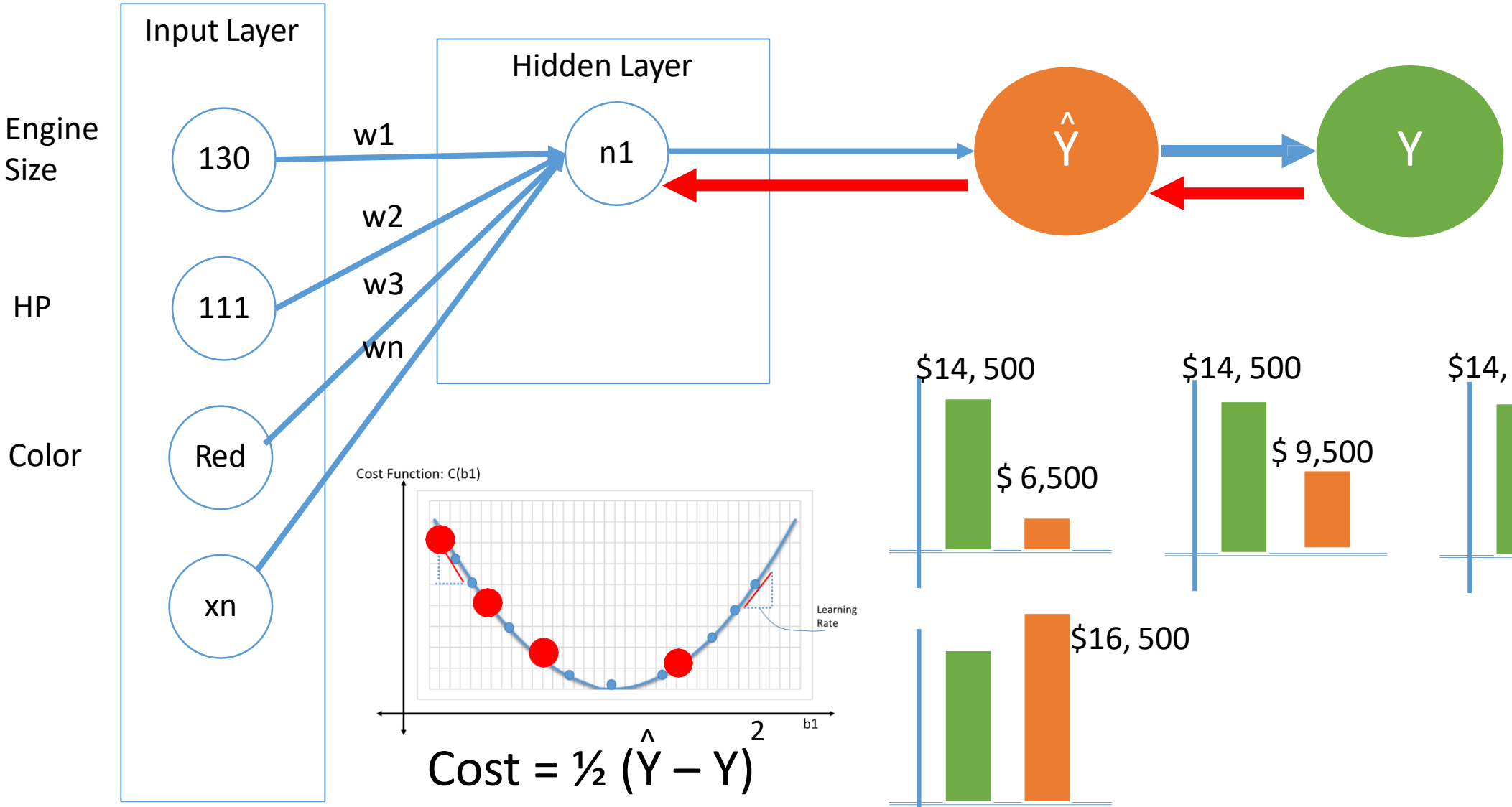


How ANN Works?

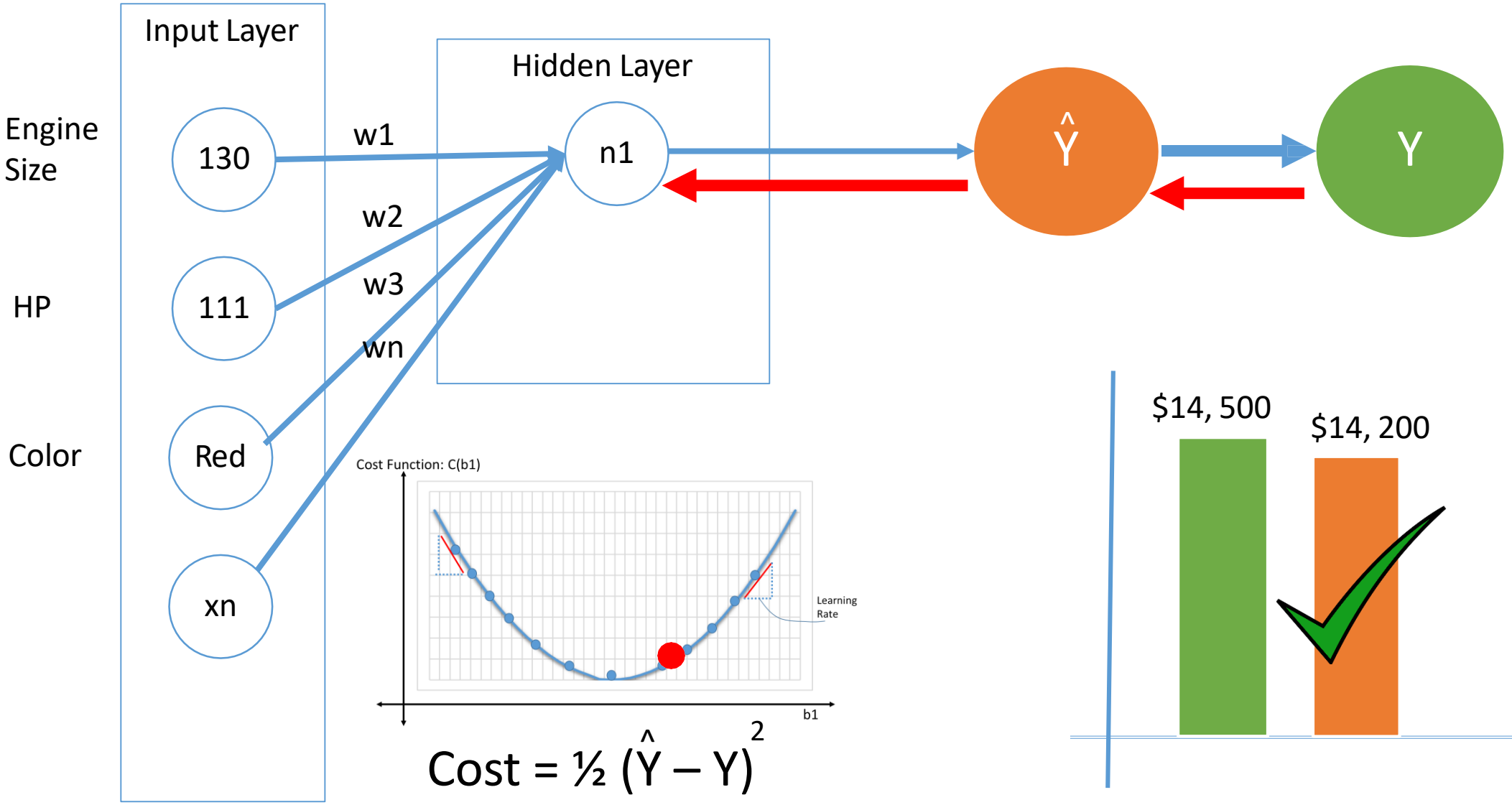


Engine Size X1	HP X2	Colour X3	Price Y
130	111	Black	**$ 13,495**
130	111	Red	**$ 14,500**
152	154	Red	**$ 16,500**
109	102	Black	**$ 13,950**

How ANN Works?



How ANN Works?



Deep Learning Frameworks

Deep Learning Frameworks

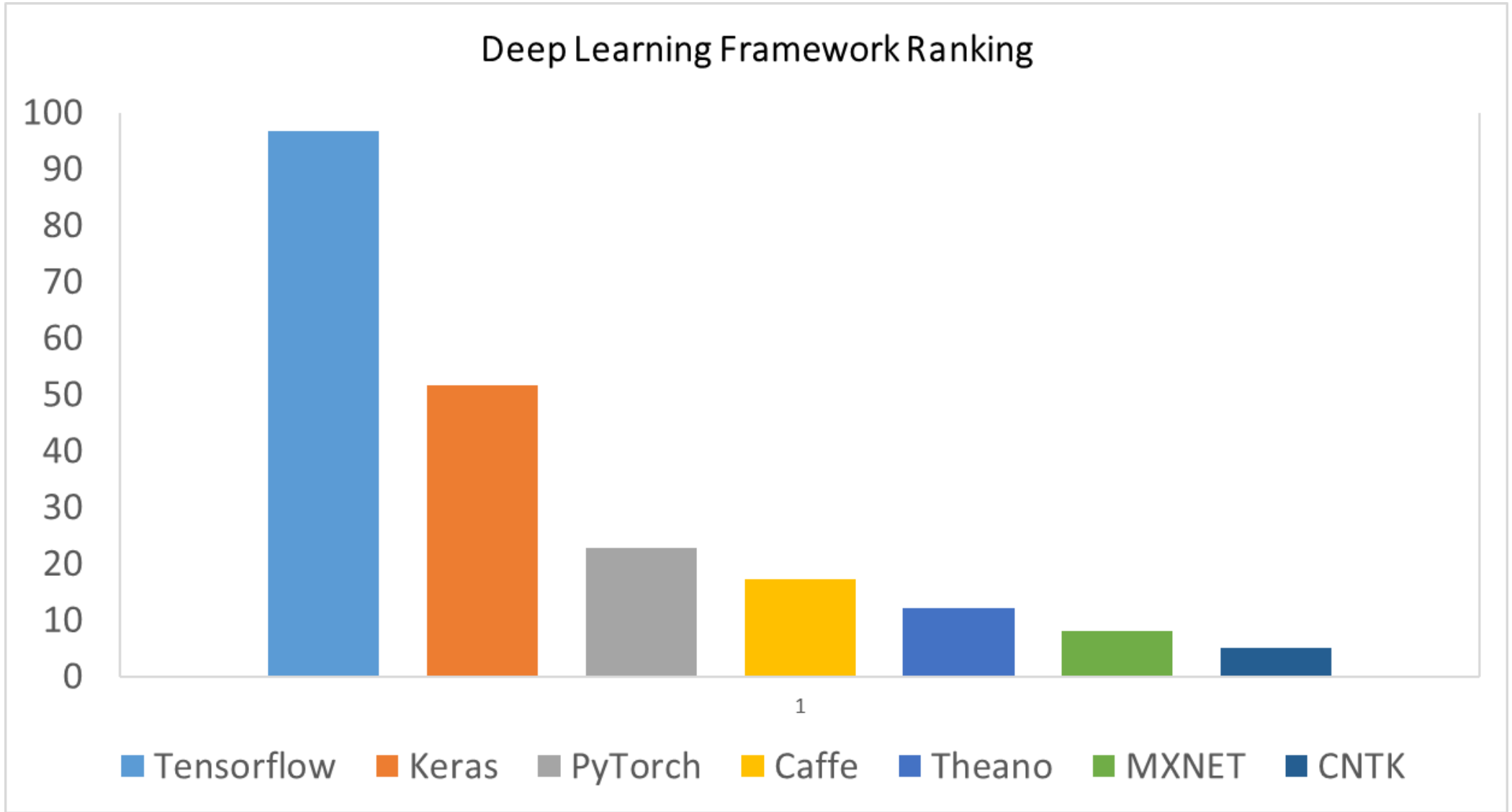
Caffe

PYTORCH

theano

Microsoft CNTK

mxnet


Deep Learning Framework Ranking
100
90
80
70
60
50
40
30
20
10
0
1
Tensorflow
Keras
PyTorch
Caffe
Theano
MXNET
CNTK

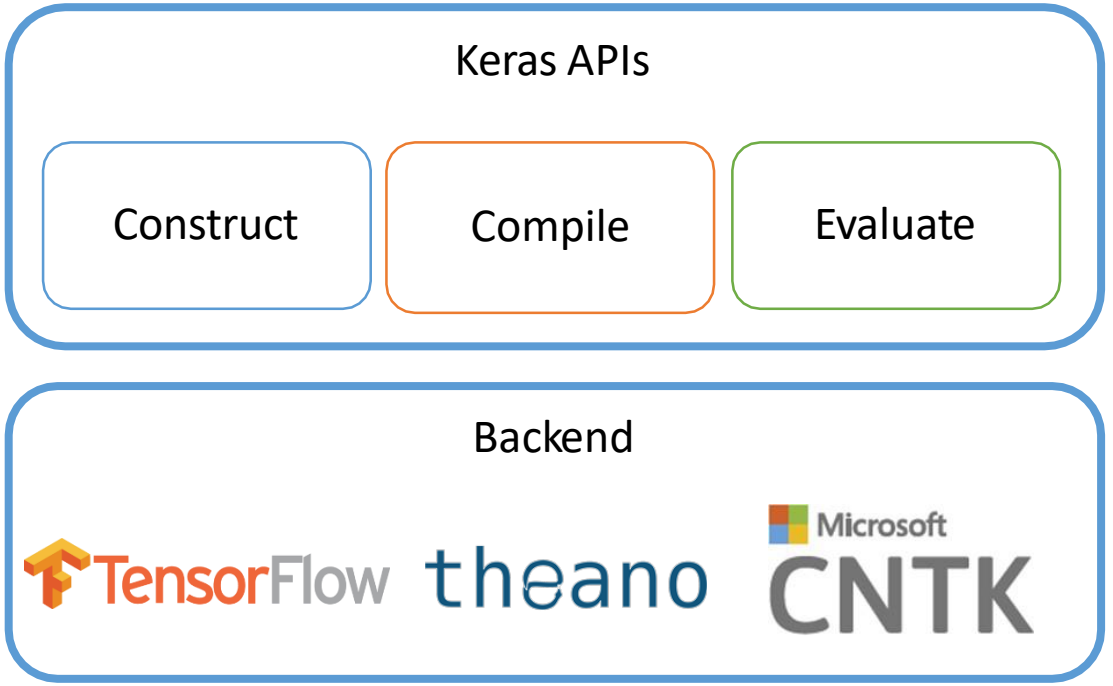
What is TensorFlow?

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

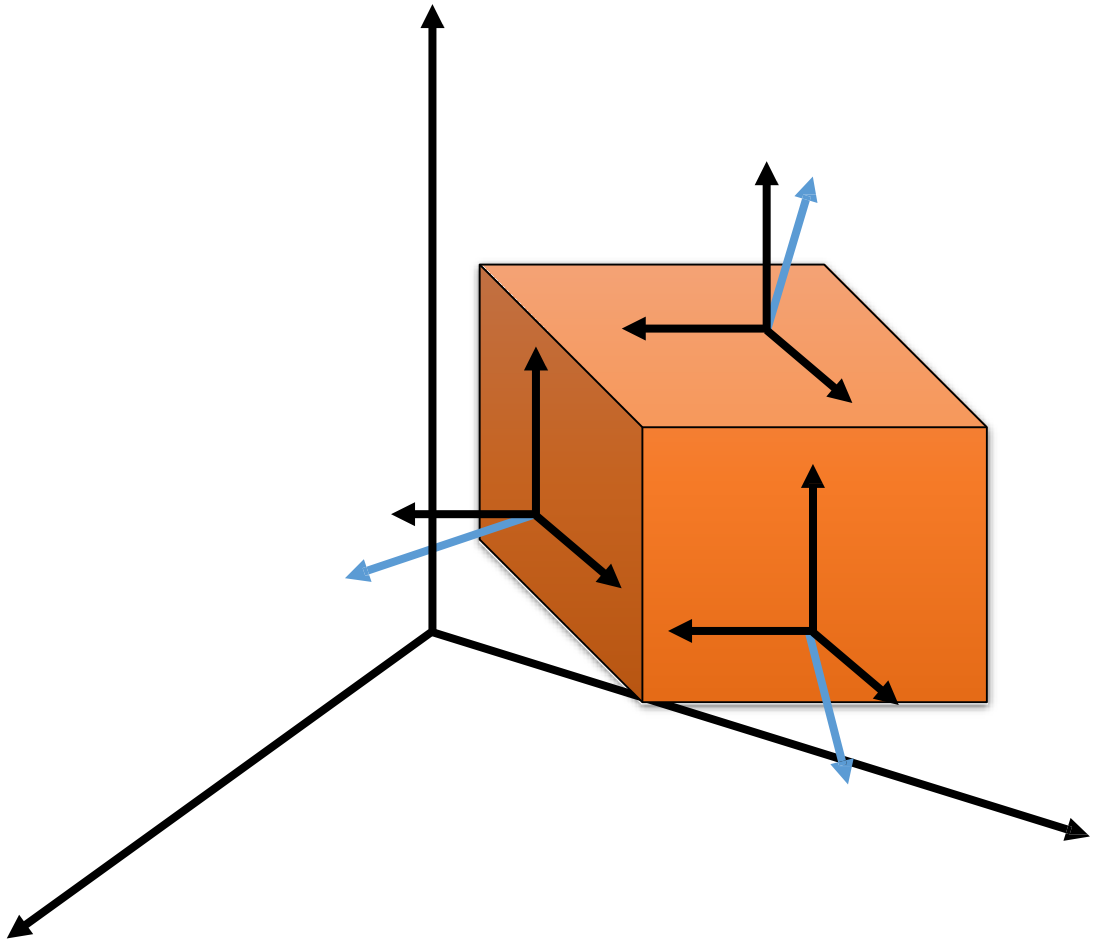


What is a Tensor?

What is a Tensor – Various Definitions

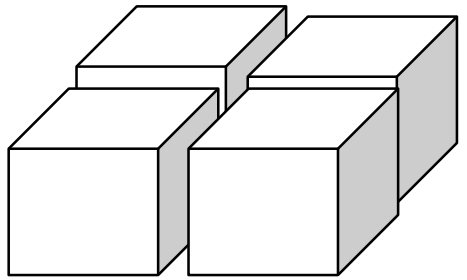
Geometric Object

Stress Tensor



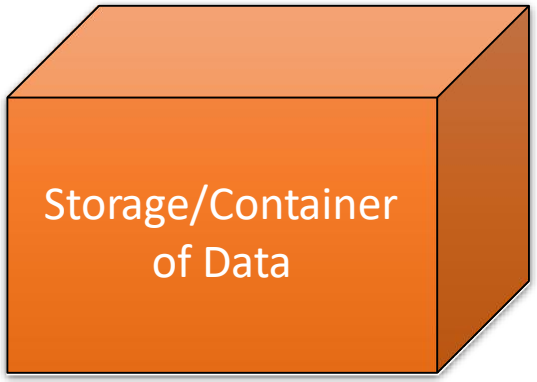
Generalization of Matrix

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



What is a Tensor in TensorFlow?

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



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

Tensor – Dimensions

Rank	Mathematical Entity
0	Scalar

10

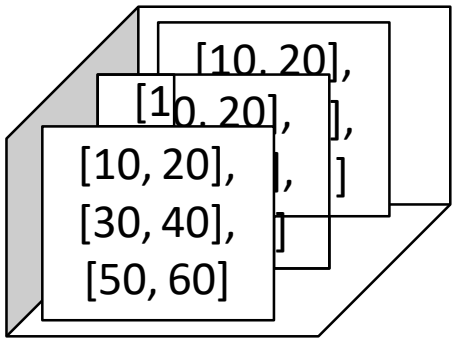
Rank	Mathematical Entity
1	Vector

[10, 20]

Rank	Mathematical Entity
2	Matrix

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

Rank	Mathematical Entity
3	Tensor



Tensor – Dimensions

Rank	Mathematical Entity
0	Scalar

10

Rank	Mathematical Entity
1	Vector

[10, 20]

Rank	Mathematical Entity
2	Matrix

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

Rank	Mathematical Entity
3	Tensor

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

Tensor – Shapes

Rank	Mathematical Entity
0	Scalar

10

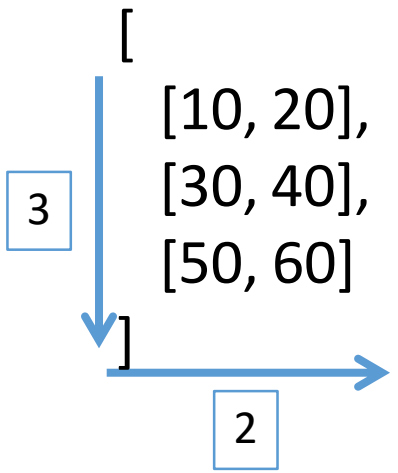
( )

Rank	Mathematical Entity
1	Vector

[10, 20]

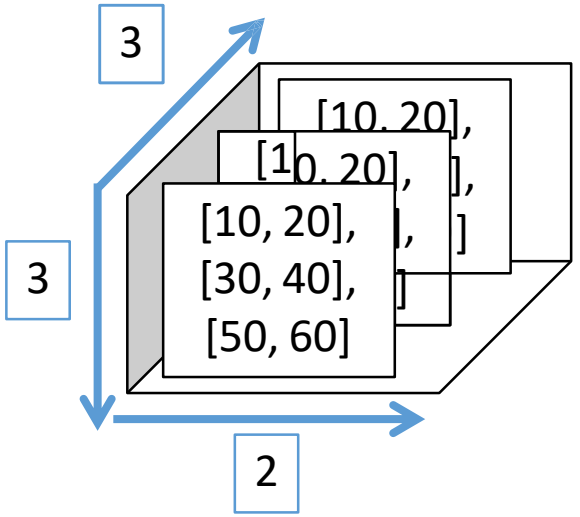
(2,)

Rank	Mathematical Entity
2	Matrix



(3, 2)

Rank	Mathematical Entity
3	Tensor



(3, 3, 2)

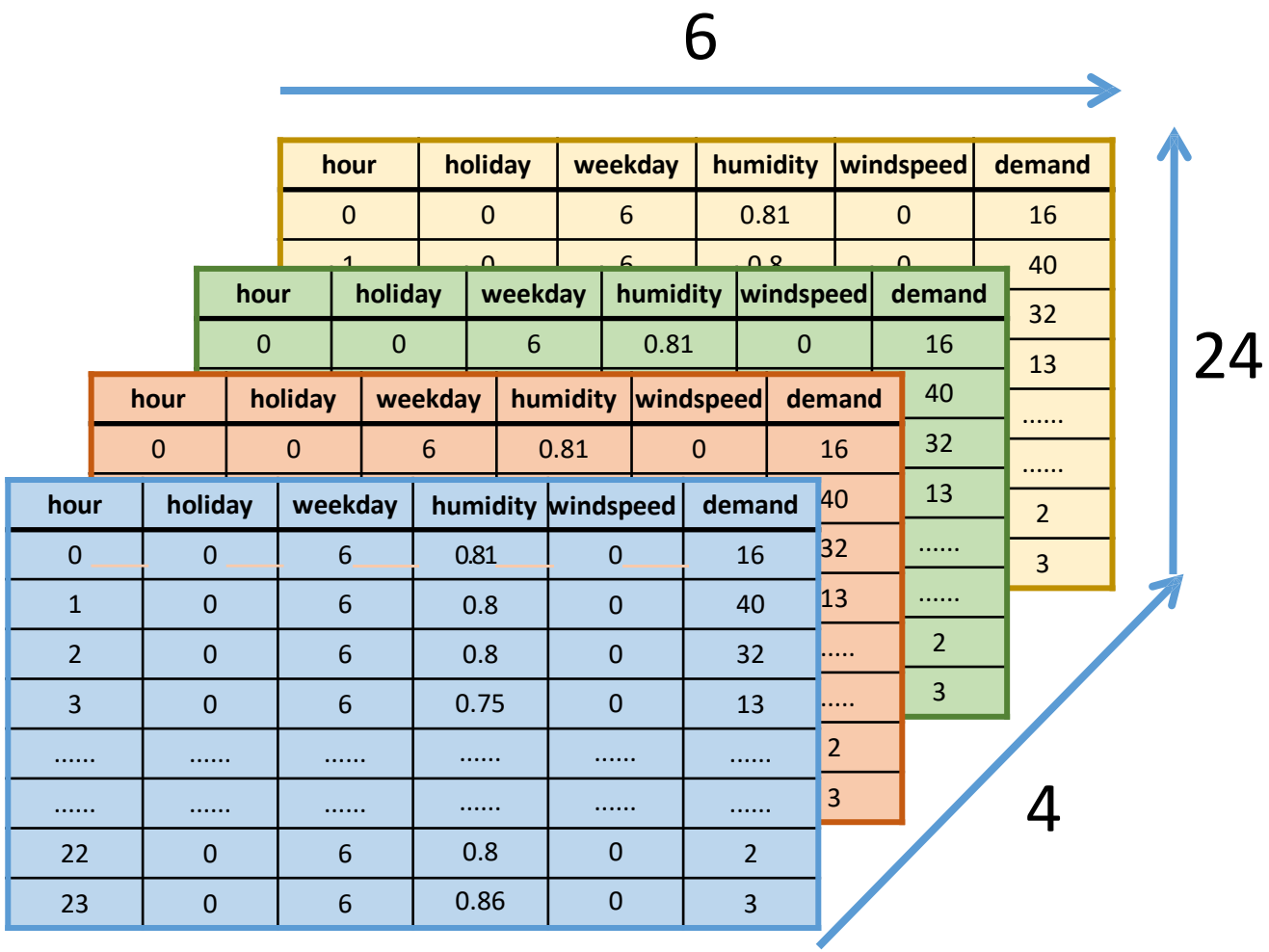
Tensor – Examples

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

Rank of this Tensor → 2

Shape of this Tensor → (9,5)

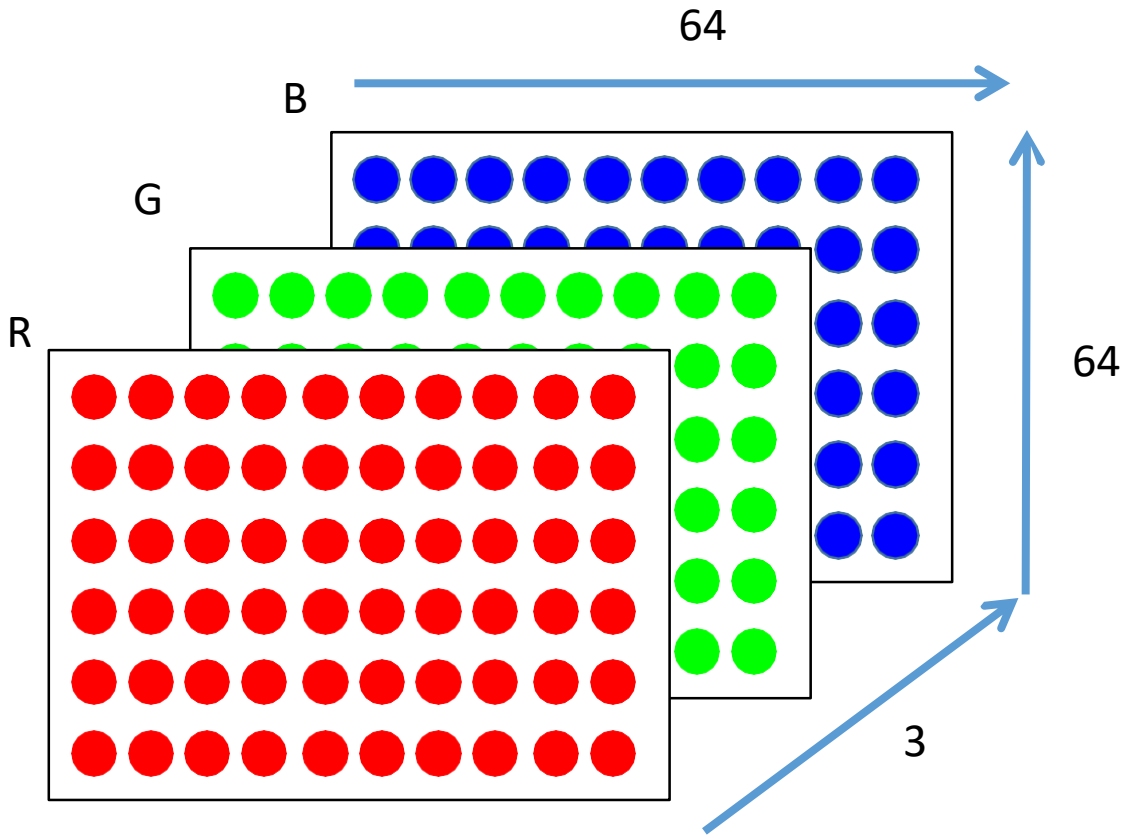
Tensor – Examples



Rank of this Tensor → 3

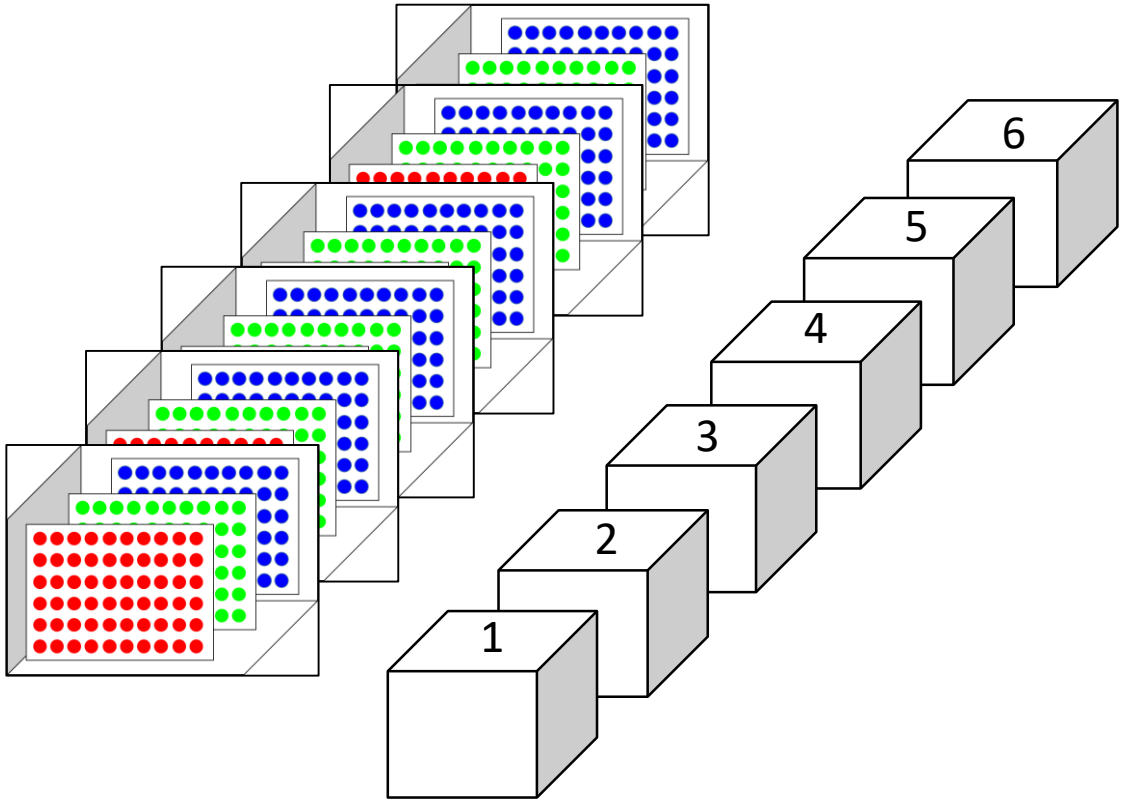
Shape of this Tensor → (4, 24, 6)

Tensor – Examples



Shape of this Tensor → (3, 64, 64)

Tensor – Examples



Shape of this Tensor → (6, 3, 64, 64)

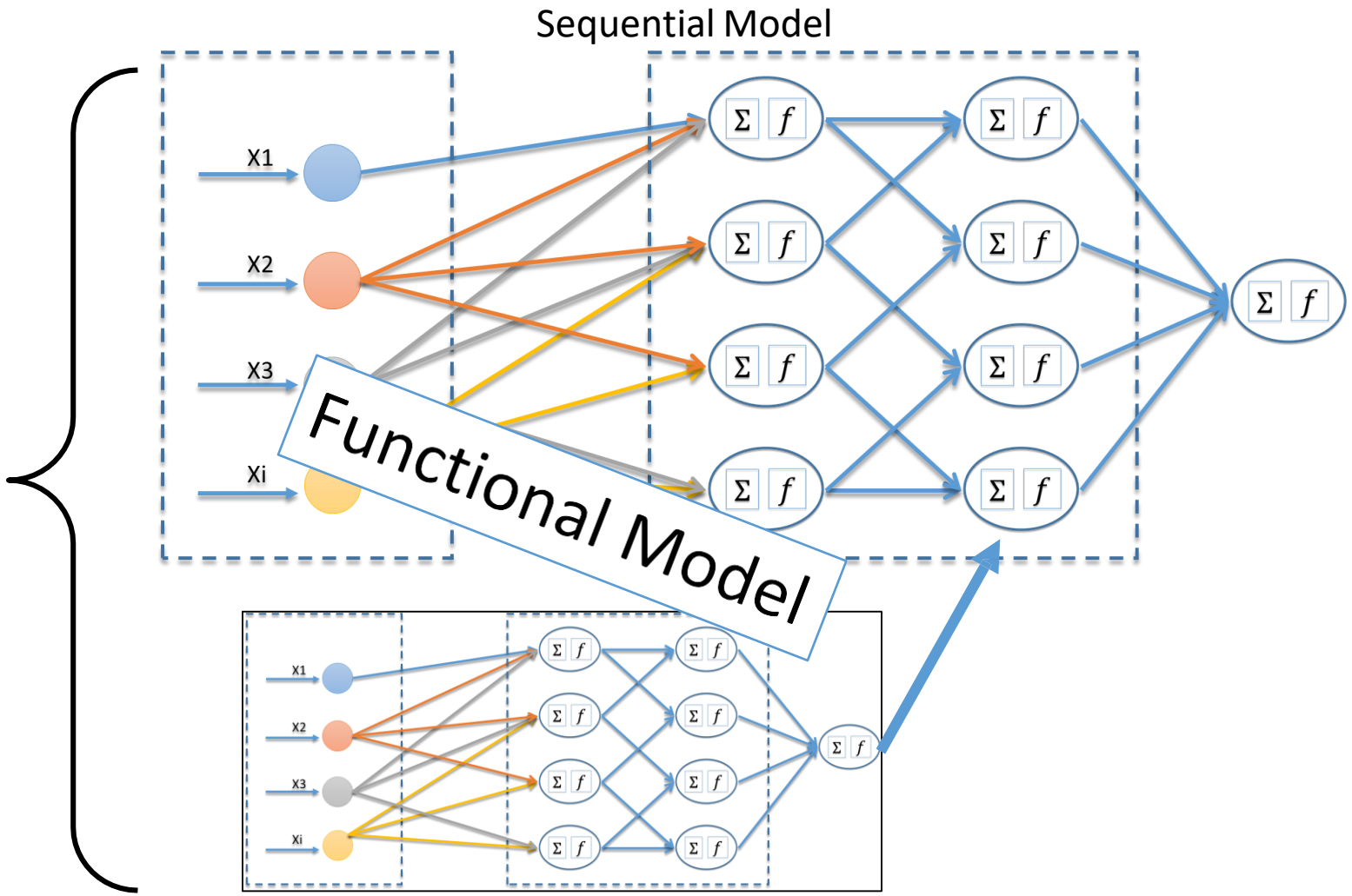
Model Types

Keras APIs

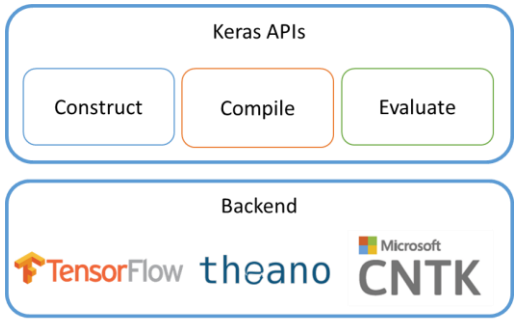
Construct | Compile | Evaluate

Backend

TensorFlow theano Microsoft CNTK

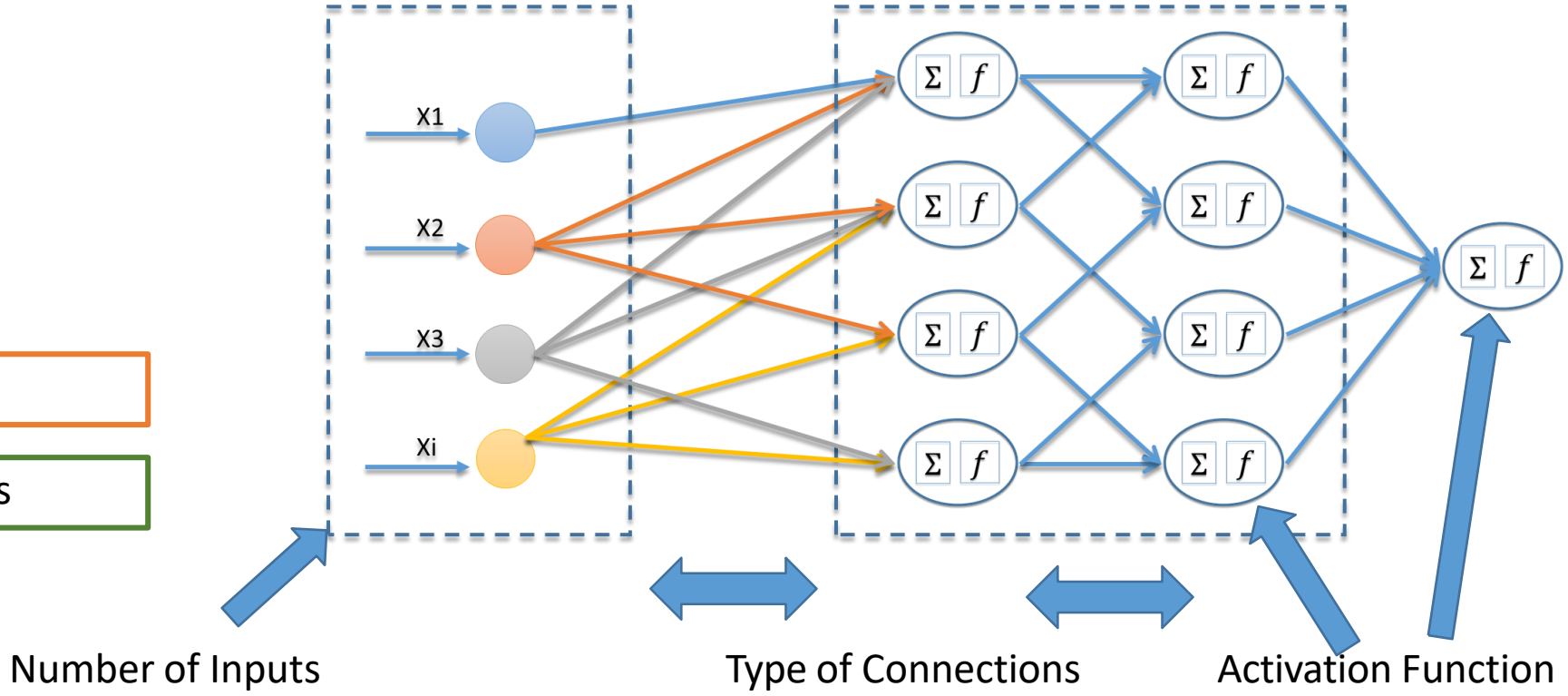


Model Building Steps



Step 1 – Specify/Add Layers

Step 2 – Define Layer Characteristics



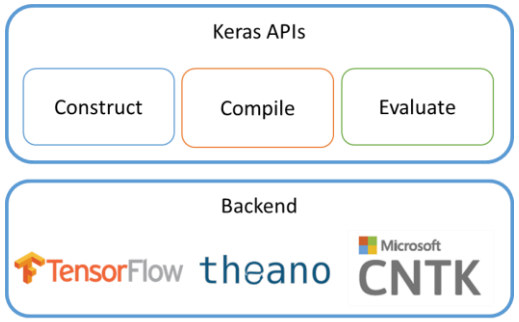
Constraints

$$\begin{pmatrix} W1 \\ W2 \\ W3 \\ Wn \end{pmatrix}$$

Initialise

Regularize

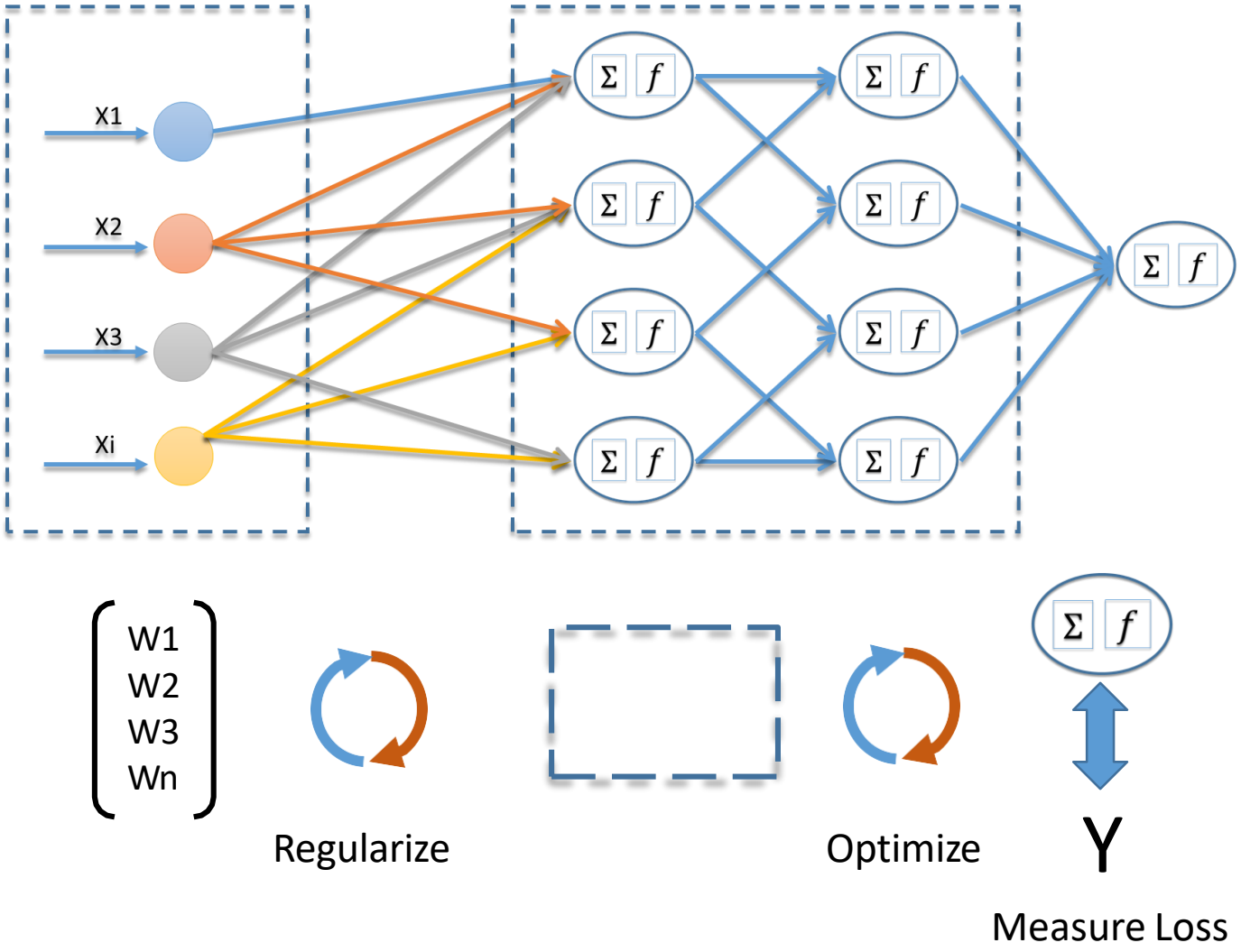
Model Building Steps



Step 1 – Specify/Add Layers

Step 2 – Define Layer Characteristics

Step 3 – Compile the Model

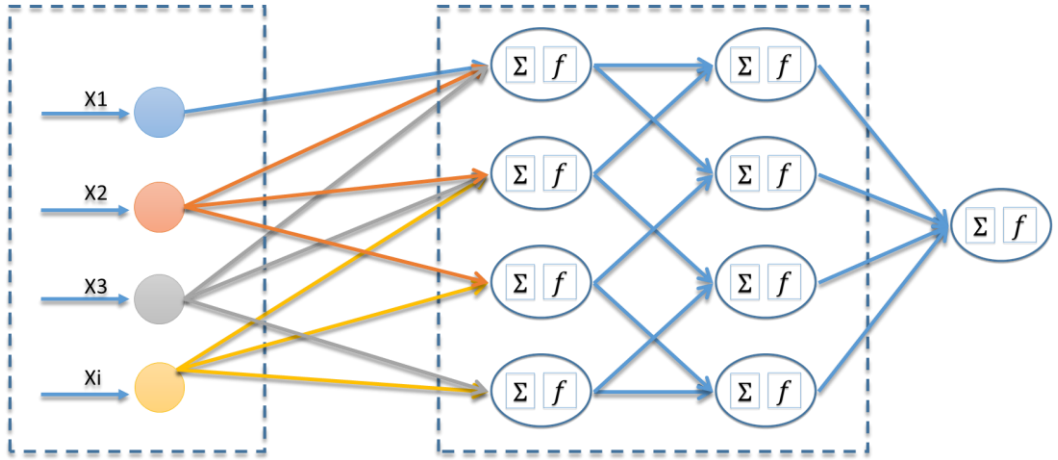


Model Building Steps

Step 1 – Specify/Add Layers	Number of Layers	Types of Layers
Step 2 – Define Layer Characteristics	Number of Inputs Type of Connections Activation Function	Constraints Initialise Regularize
Step 3 – Compile the Model	Optimize	Measure Loss
Step 4 – Evaluate the Model	Metrics to measure performance	

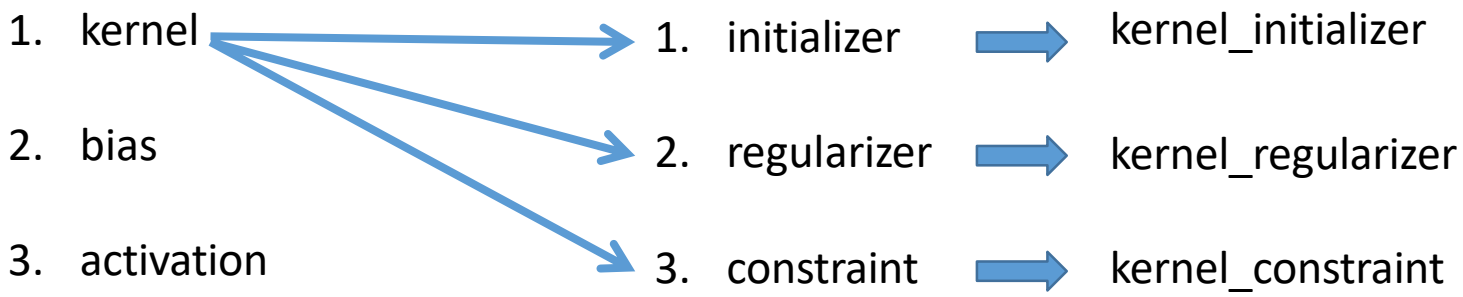
Types of Layers

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



Dense layer parameters

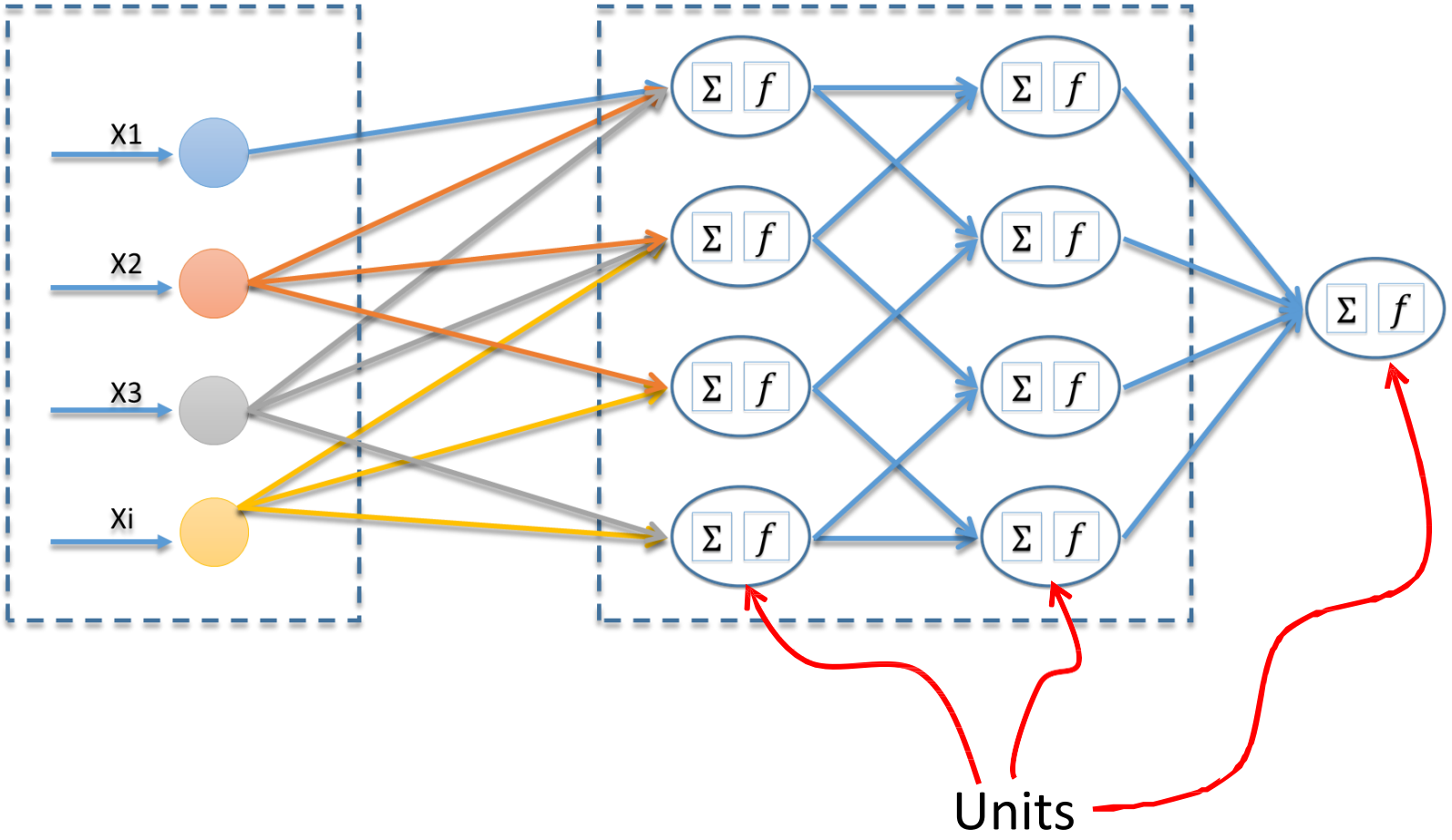
- units
- kernel_initializer
- kernel_regularizer
- kernel_constraint
- use_bias
- bias_initializer
- bias_regularizer
- bias_constraint
- activation
- activity_regularizer



Important Terms

- Units
- Kernel
- Bias
- Activation Function

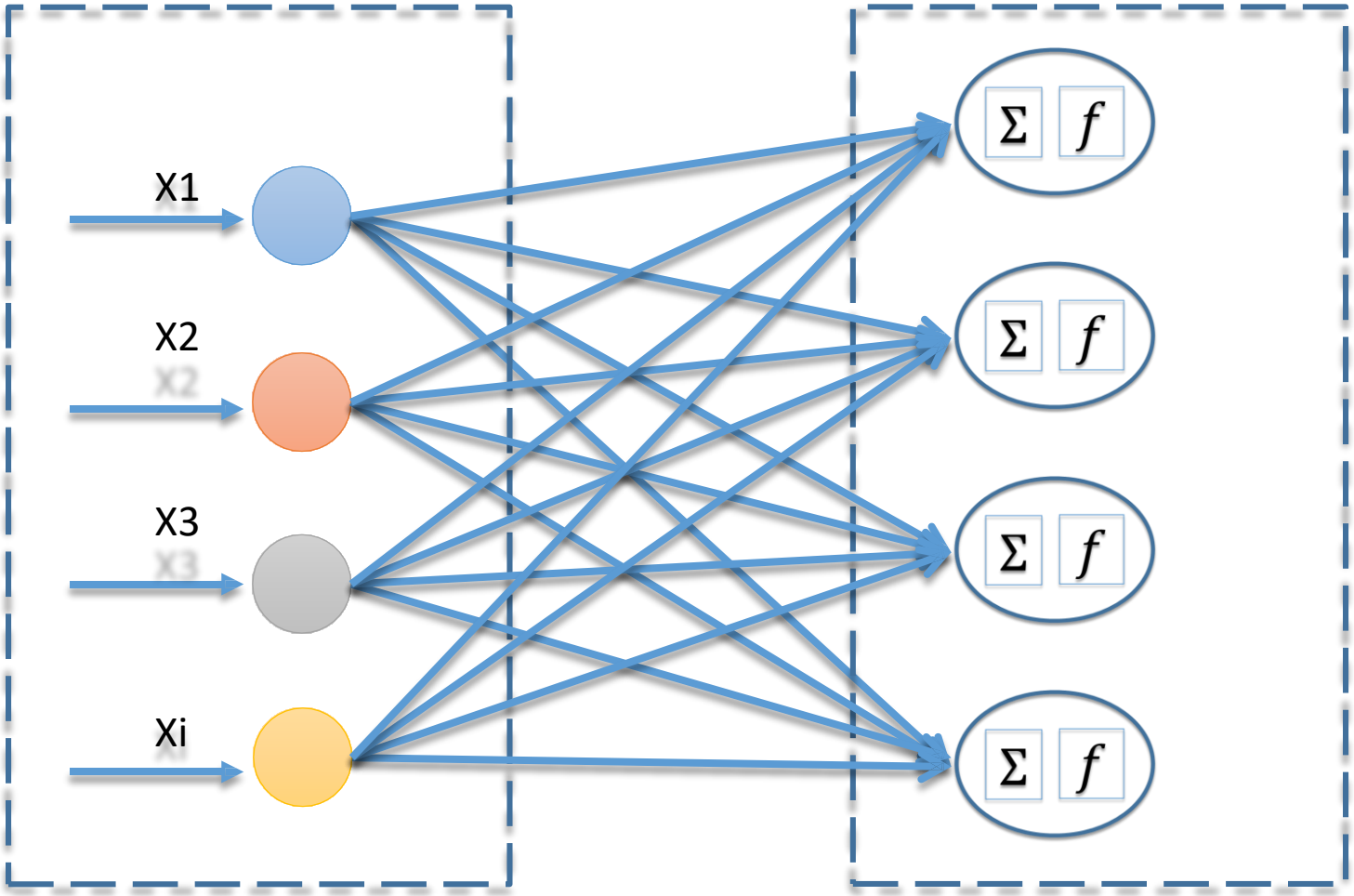
Units



Important Terms

- Units
- **Kernel**
- Bias
- Activation Function

Kernel

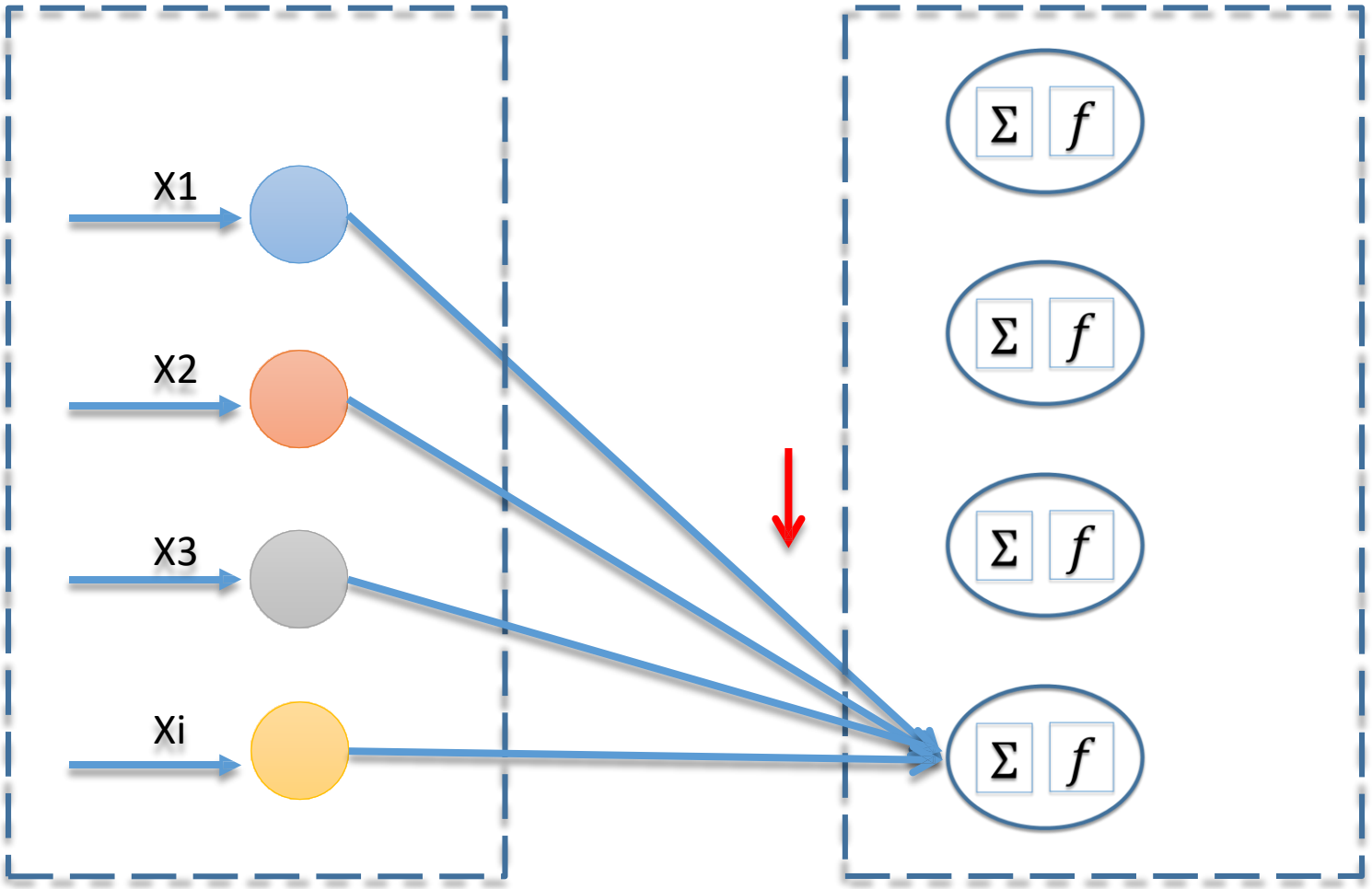


$$\begin{pmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{pmatrix}$$

Important Terms

- Units
- Kernel
- **Bias**
- Activation Function

Bias

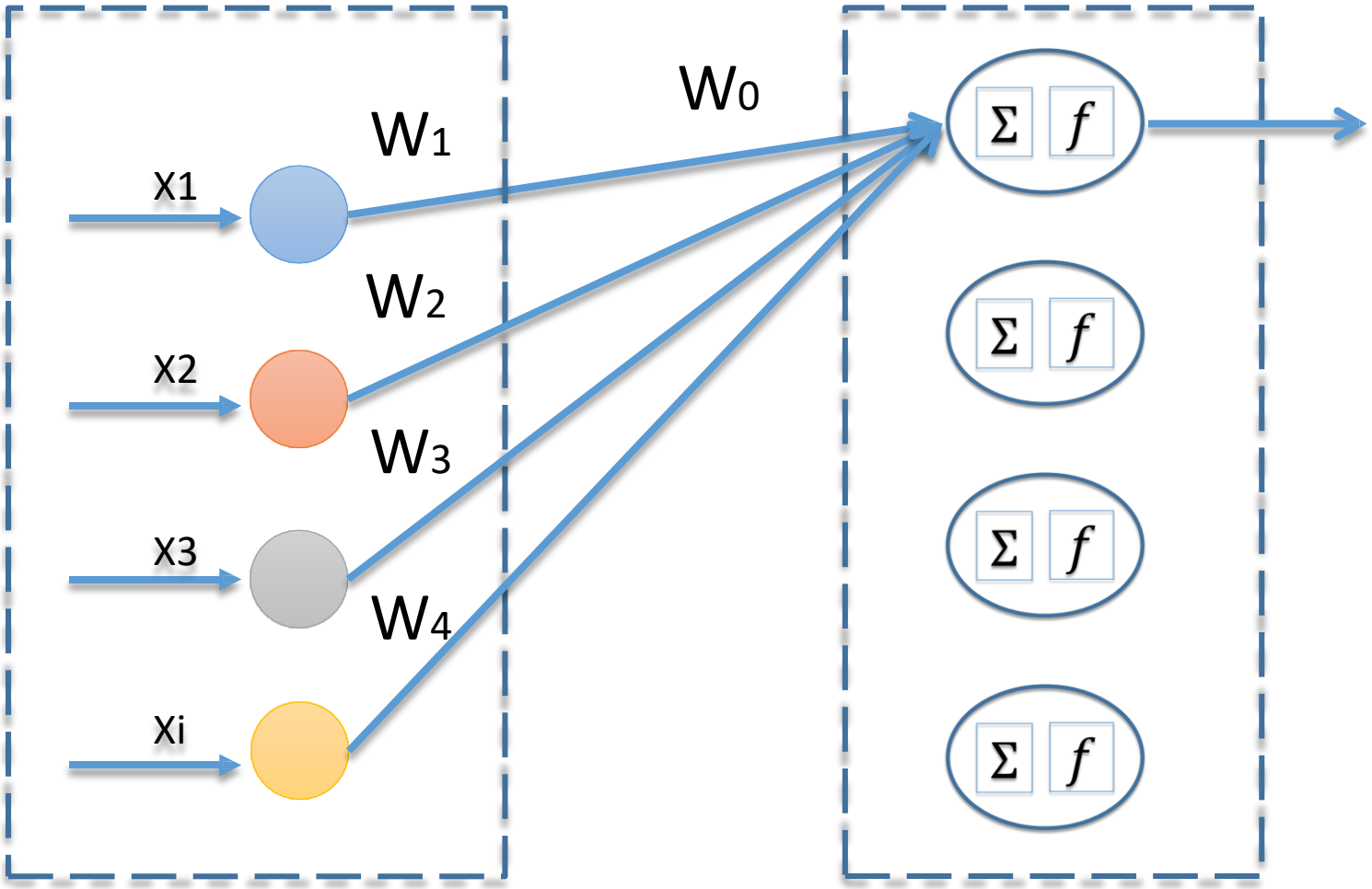


$$\begin{pmatrix} W_{01} \\ W_{02} \\ W_{03} \\ W_{04} \end{pmatrix}$$

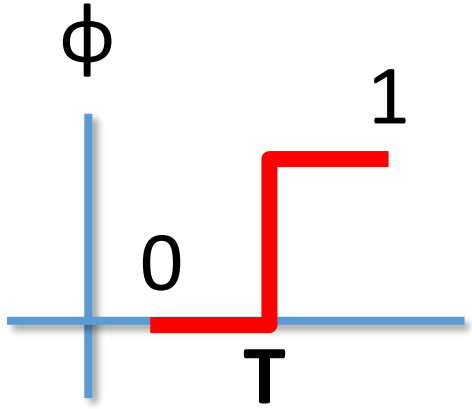
Important Terms

- Units
- Kernel
- Bias
- **Activation Function**

Activation Function



$$\Phi = f\left(\sum_{i=1}^{n} x_i w_i\right)$$

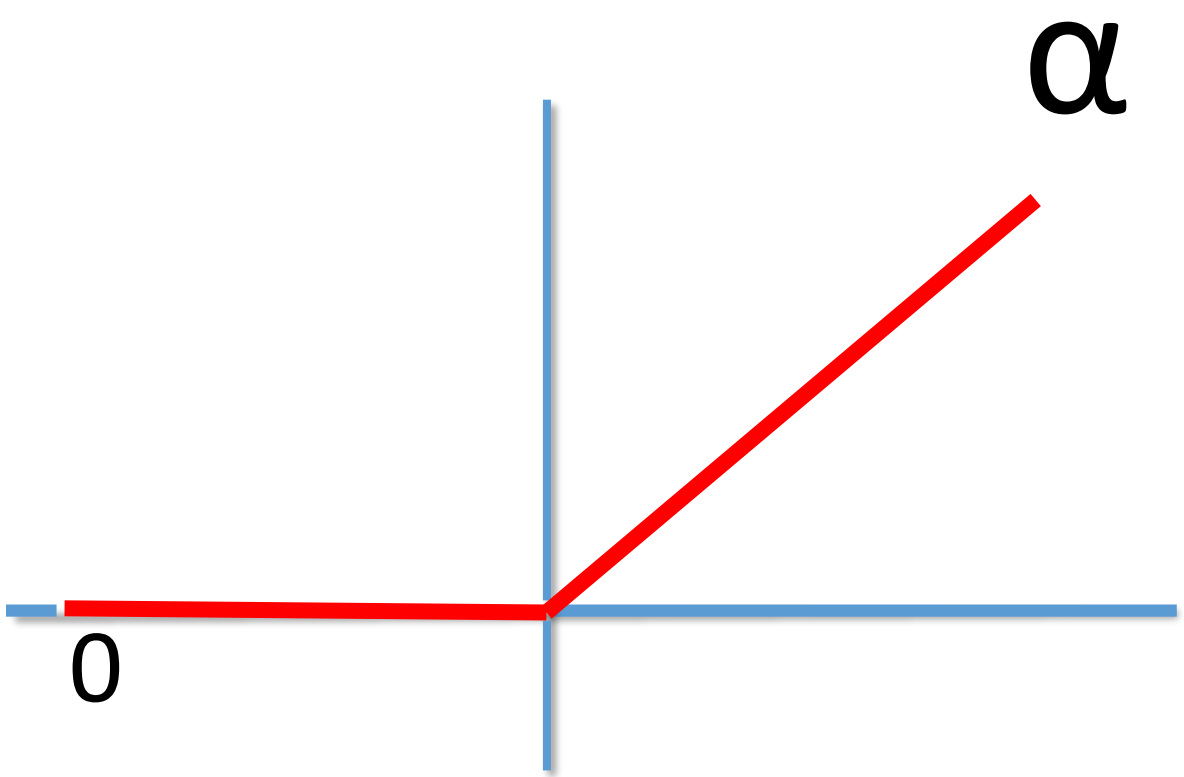


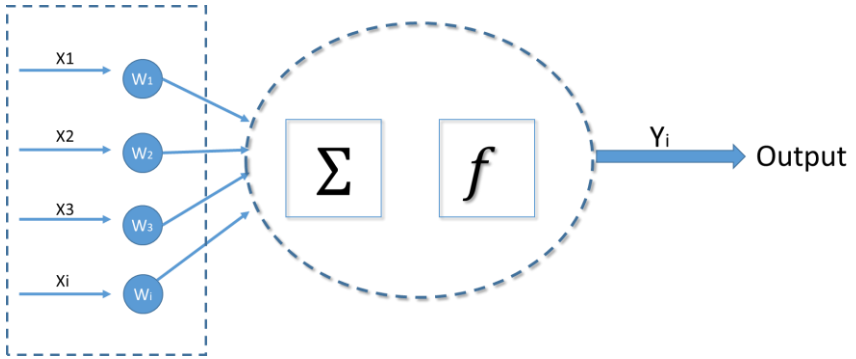
Types of Activation Functions

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

Rectified Linear Unit (ReLU)

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

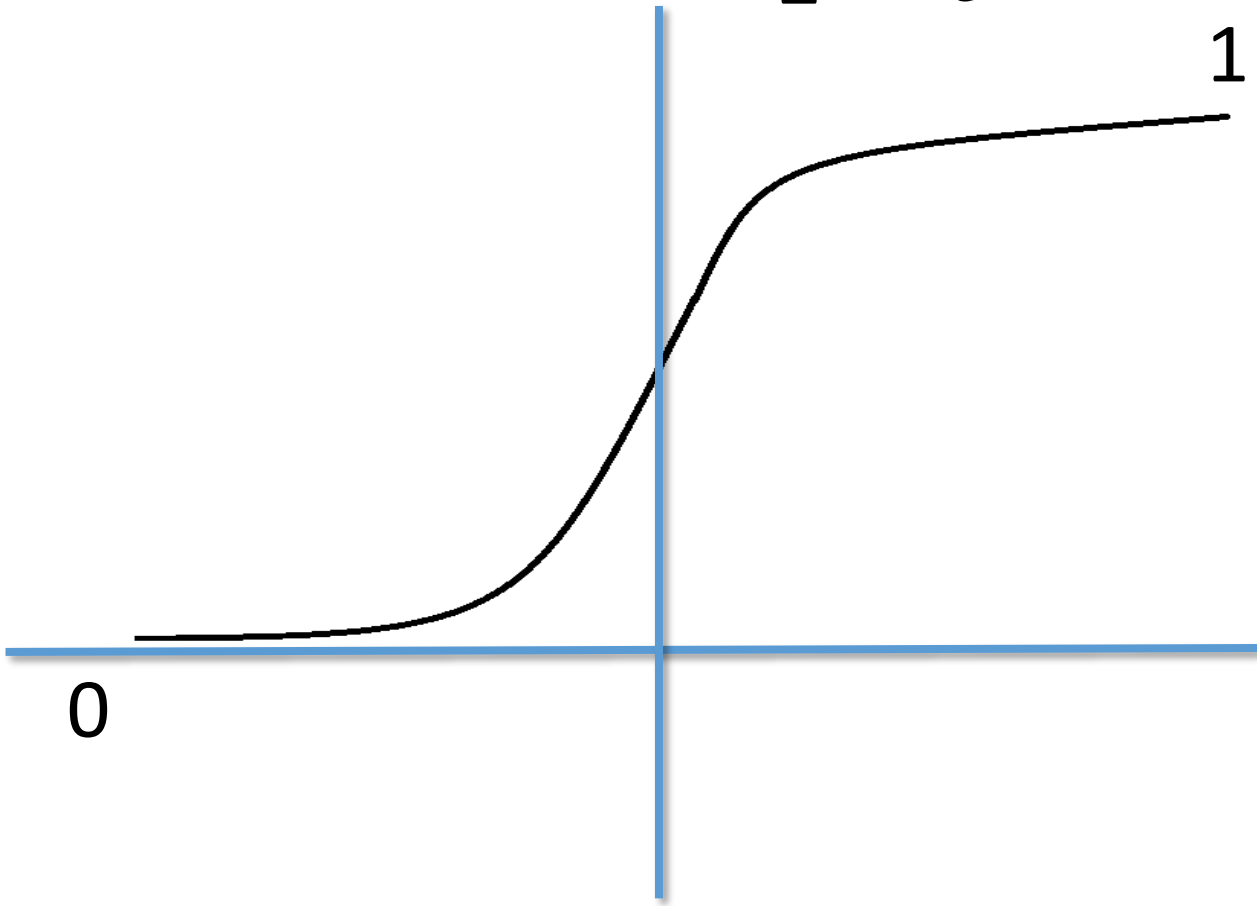


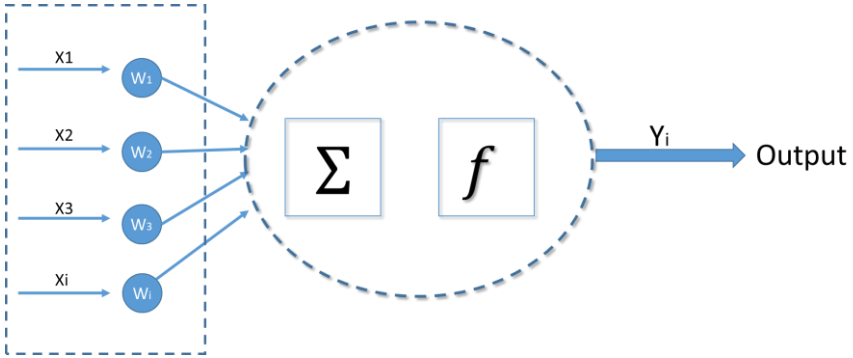


$$\alpha = \sum_{i=1} x_i w_i$$

Sigmoid Function

$$\Phi = \frac{1}{1 + e^{-\alpha}}$$

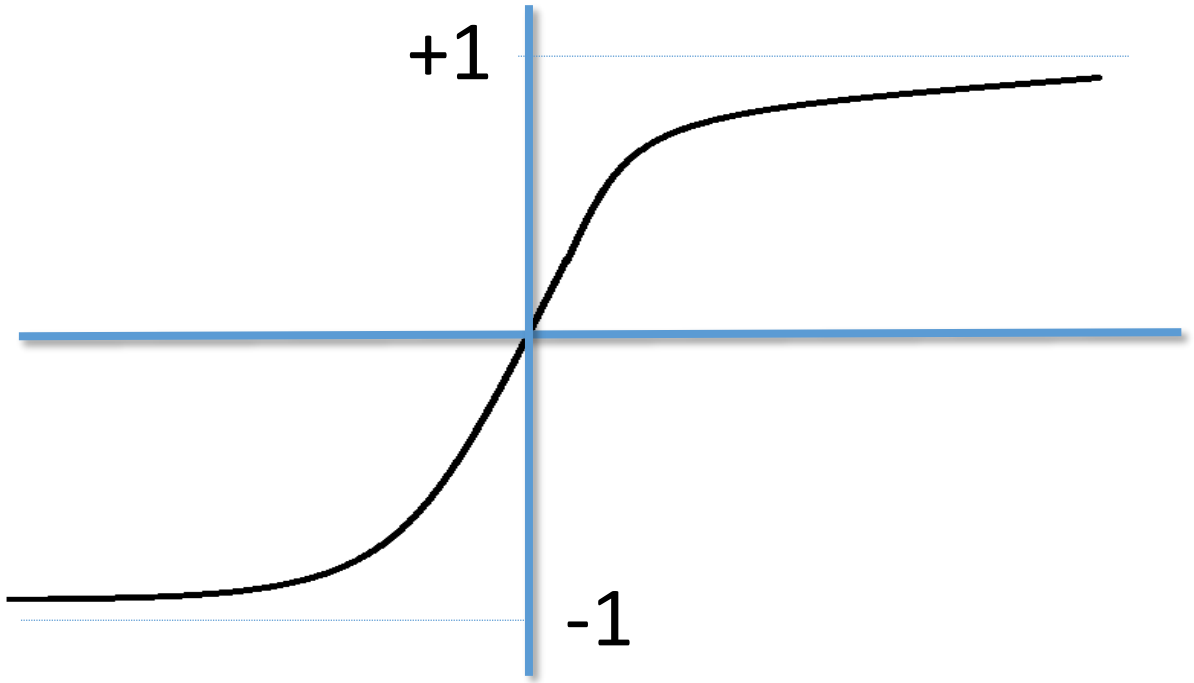


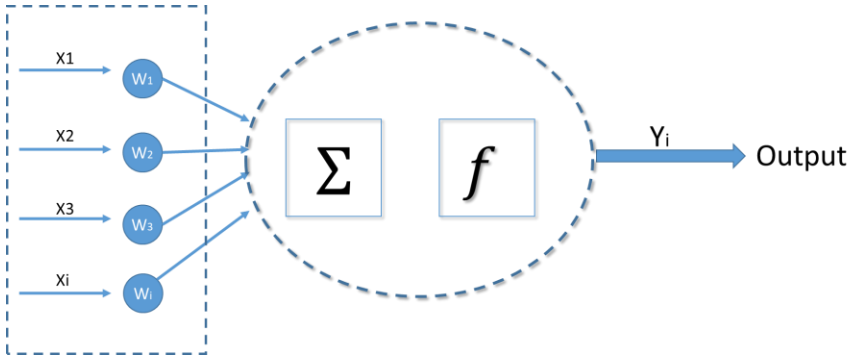


$$\alpha = \sum_{i=1} x_i w_i$$

TanH Function

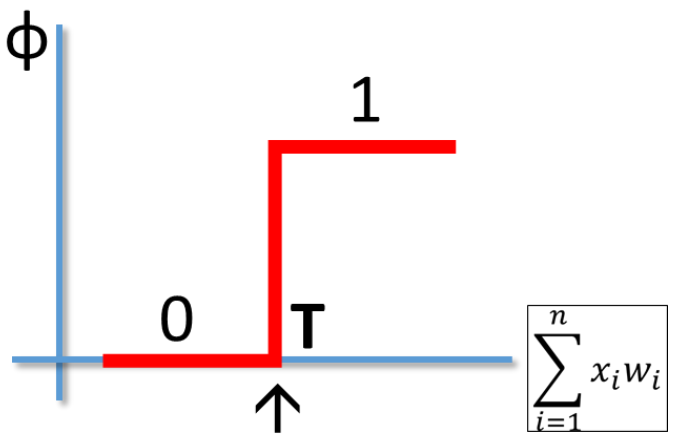
$$\Phi = \frac{e^{\alpha} - e^{-\alpha}}{e^{\alpha} + e^{-\alpha}}$$



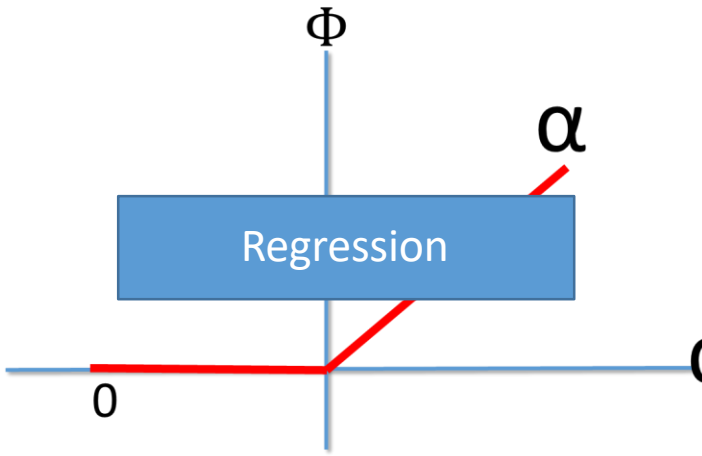


$$\alpha = \sum_{i=1} x_i w_i$$

Activation Functions

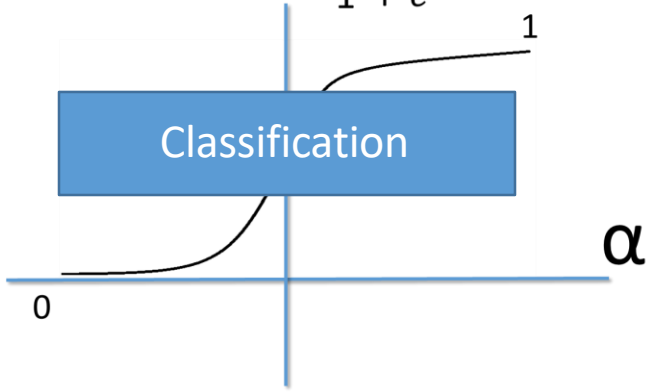


$$\sum_{i=1}^{n} x_i w_i$$



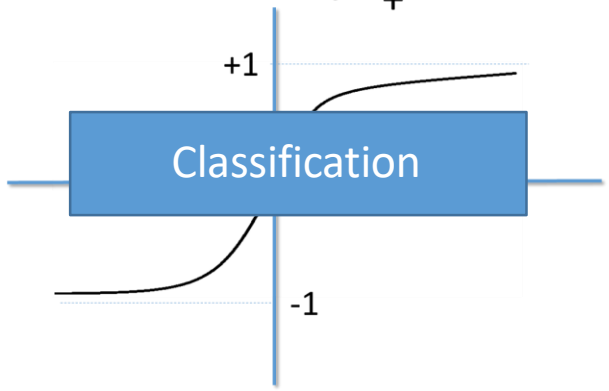
$$\alpha = \sum_{i=1}^{n} x_i w_i$$

$$\Phi = \frac{1}{1 + e^{-\alpha}}$$



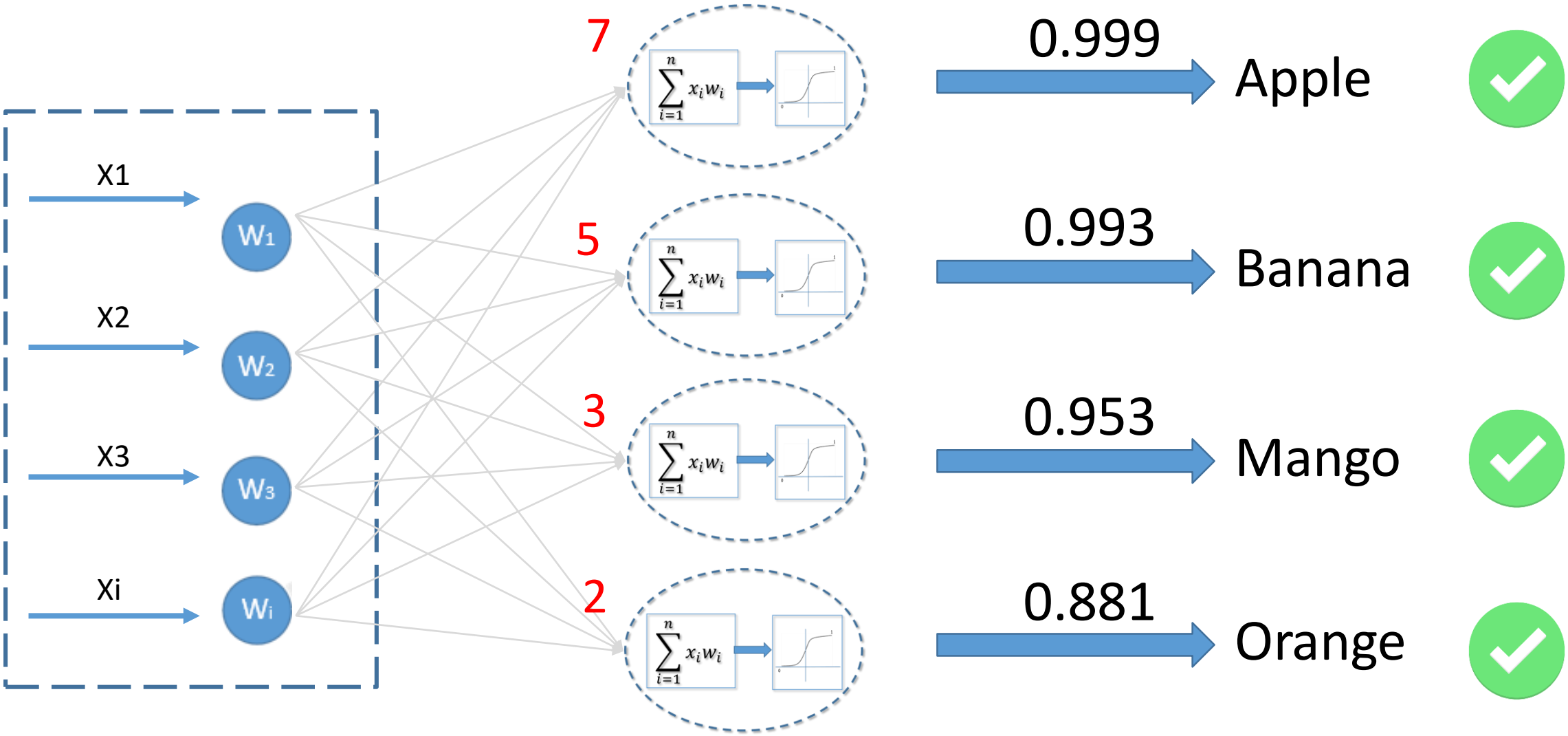
$$\alpha = \sum_{i=1}^{n} x_i w_i$$

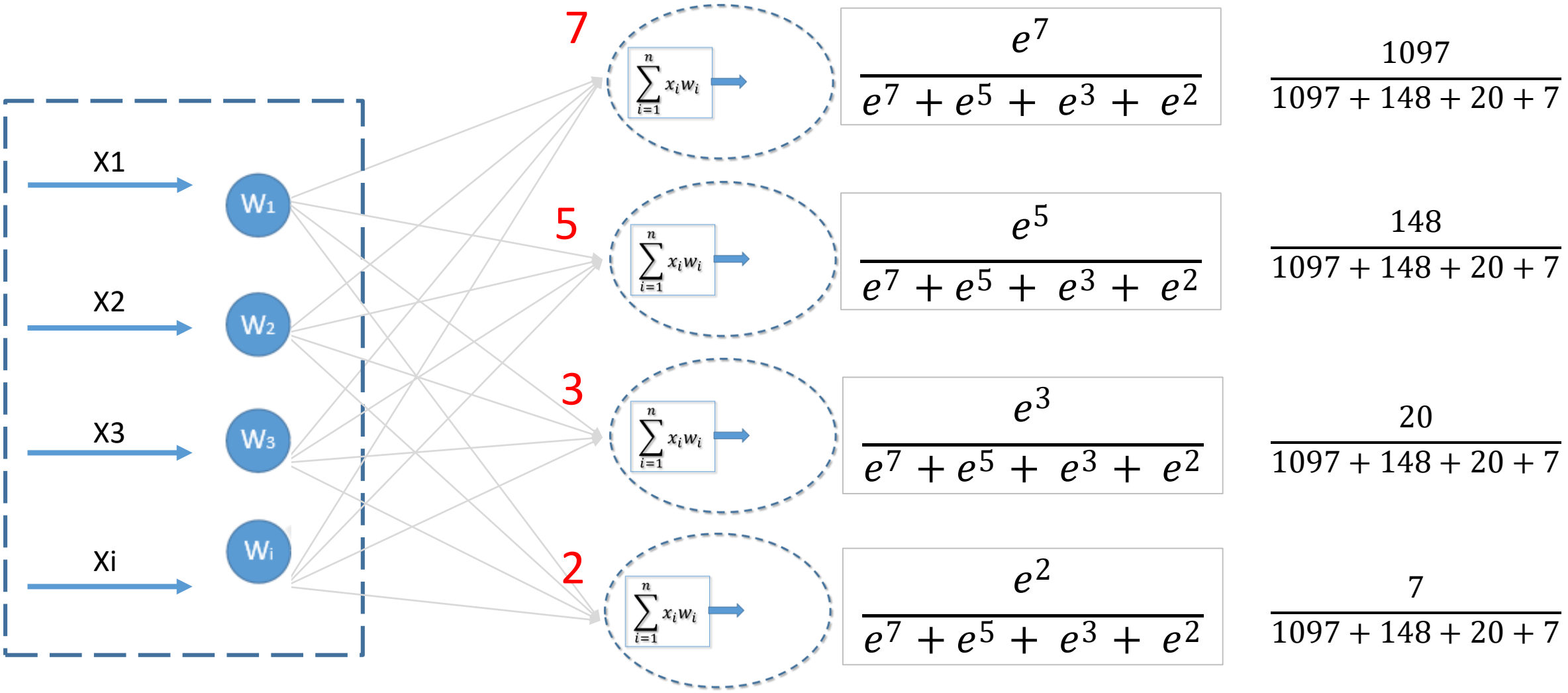
$$\Phi = \frac{e^{\alpha} - e^{-\alpha}}{e^{\alpha} + e^{-\alpha}}$$

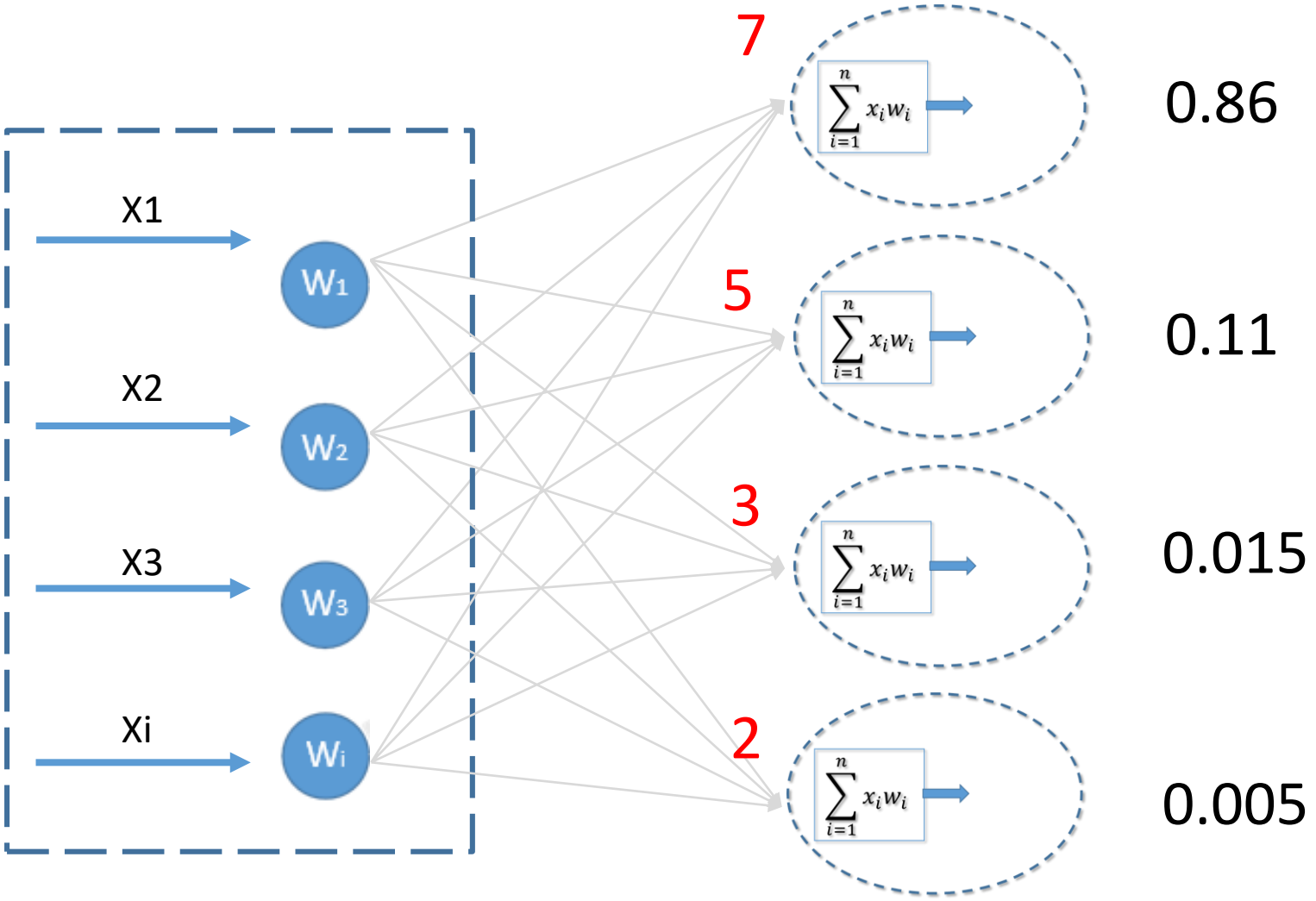


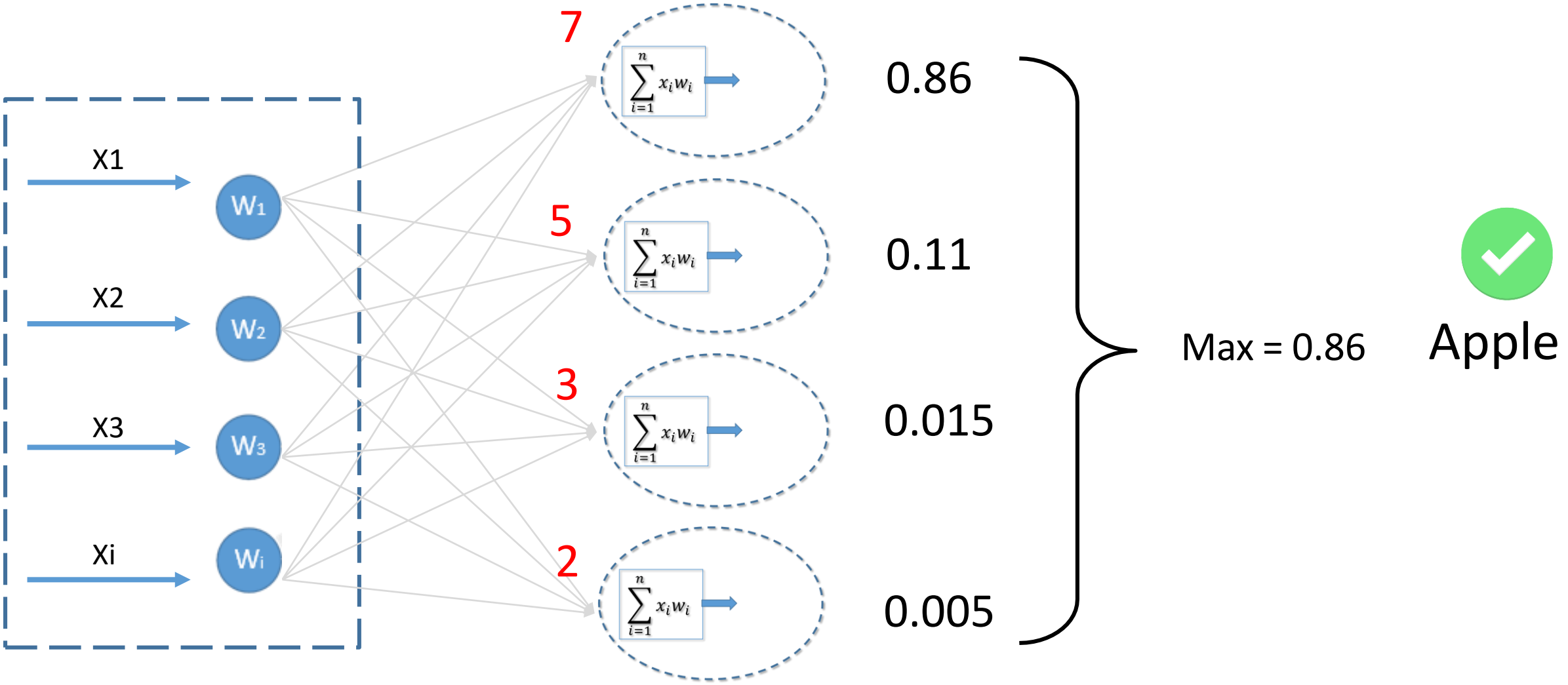
$$\alpha = \sum_{i=1}^{n} x_i w_i$$

Softmax Activation


X1
X2
X3
Xi
W1
W2
W3
Wi
7
5
3
2
$\sum_{i=1}^{n} x_i w_i$
0.999
Apple
0.993
Banana
0.953
Mango
0.881
Orange


X1
X2
X3
Xi
W1
W2
W3
Wi
7
5
3
2
$\sum_{i=1}^{n} x_i w_i$
$\frac{e^7}{e^7 + e^5 + e^3 + e^2}$
$\frac{e^5}{e^7 + e^5 + e^3 + e^2}$
$\frac{e^3}{e^7 + e^5 + e^3 + e^2}$
$\frac{e^2}{e^7 + e^5 + e^3 + e^2}$
$\frac{1097}{1097 + 148 + 20 + 7}$
$\frac{148}{1097 + 148 + 20 + 7}$
$\frac{20}{1097 + 148 + 20 + 7}$
$\frac{7}{1097 + 148 + 20 + 7}$


X1
X2
X3
Xi
W1
W2
W3
Wi
7
5
3
2
$\sum_{i=1}^{n} x_i w_i$
0.86
0.11
0.015
0.005


X1
X2
X3
Xi
W1
W2
W3
Wi
7
5
3
2
$\sum_{i=1}^{n} x_i w_i$
0.86
0.11
0.015
0.005
Max = 0.86
Apple

Softmax Function

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

Loss Functions

Loss Functions in Keras

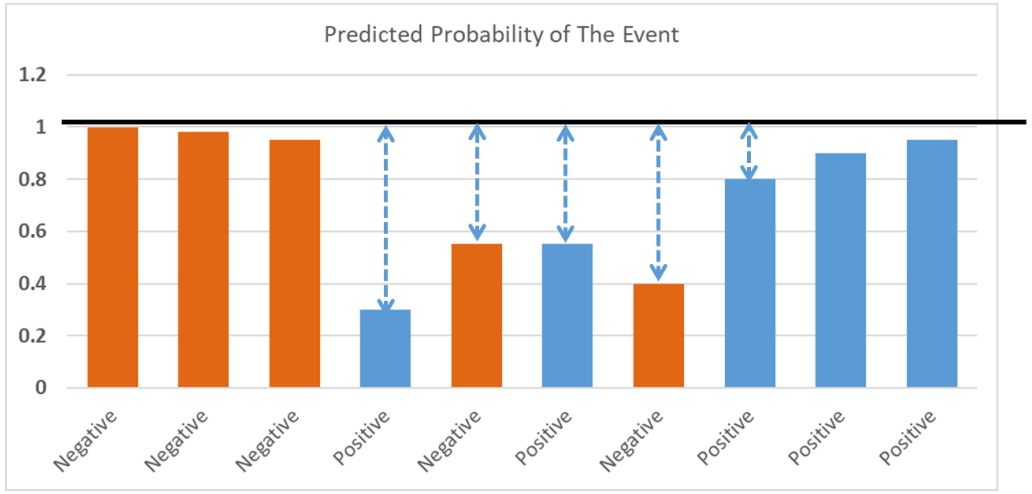
- **Regression**
  - Mean Squared Error
  - Mean Absolute Error
  - Mean Absolute Percentage Error
  - Mean Squared Logarithmic Error
  - L1-Norm
  - L2-Norm
- **Classification**
  - Categorical_crossentropy
  - Binary_crossentropy
  - Sparse_categorical_crossentropy
  - Hinge
  - Squared_hinge
  - Categorical_hinge

Regression Loss Functions

Name	Formula	Usage/Characteristics
Mean Squared Error or MSE	$\frac{1}{n}\sum_{i=1}^{n}(yi - \hat{y}i)^2$	1. One of the most popular 2. Used when predicted values are small
Mean Squared Logarithmic Error	$\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2$	1. Used when predicted values are large and we do not want to penalise the large error
L2-Norm	$\sum_{i=1}^{n}(yi - \hat{y}i)^2$	Same as MSE without the mean.
Mean Absolute Error	$\frac{1}{n}\sum_{i=1}^{n}\lvert yi - \hat{y}i\rvert$	1. Robust but unstable solution 2. Gradient is difficult to calculate
Mean Absolute Percentage Error	$\frac{100\%}{n}\sum_{i=1}^{n}\left\lvert\frac{y_i - \hat{y}_i}{y_i}\right\rvert$	1. For zero values could pose a challenge 2. Does not always remain below 100%
L1-Norm	$\sum_{i=0}^{n}\lvert y_i - \hat{y}_i\rvert$	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

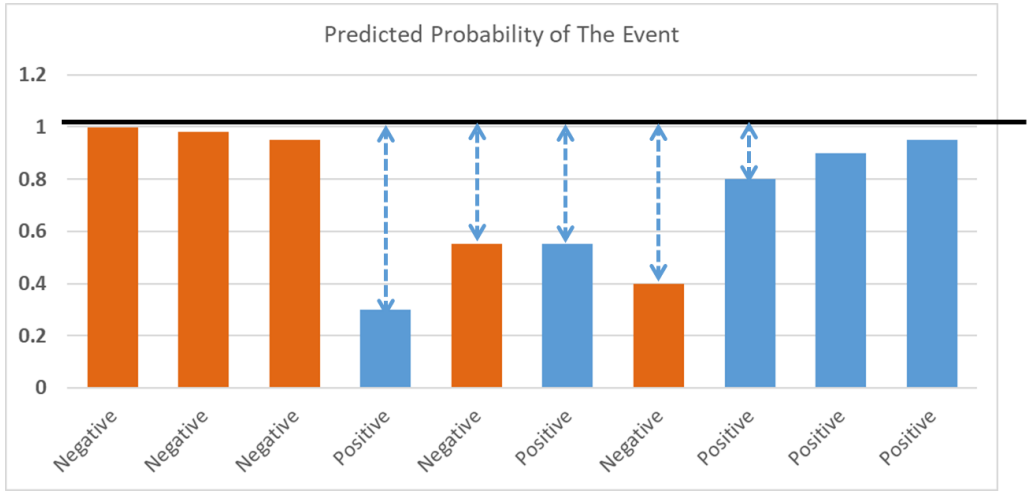
Categorical_crossentropy — Multi-Class Classification

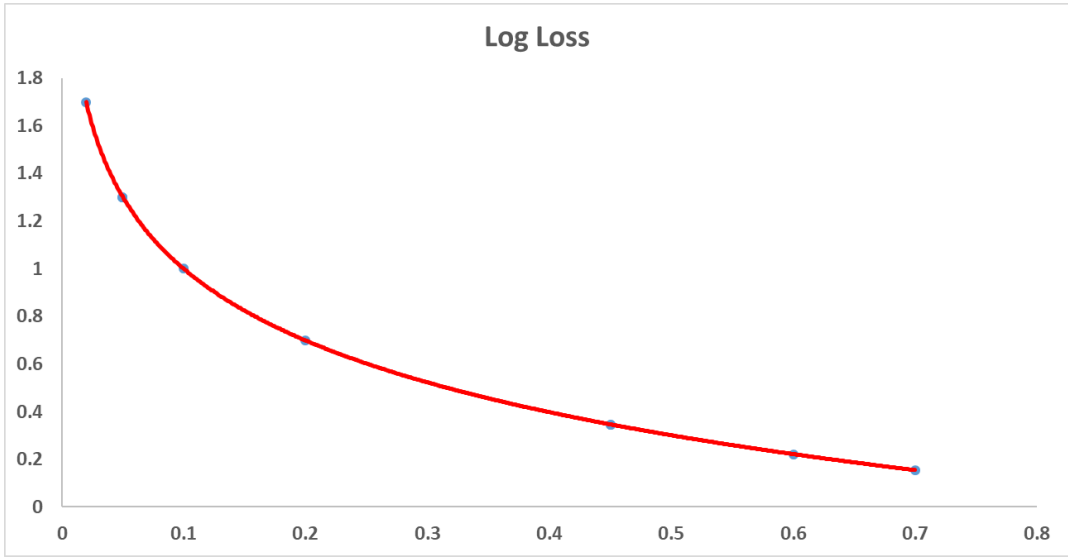
0
1
2
3

Sparse_categorical_ crossentropy — Hot-encoded Multi Classification

0	1	0
1	0	0
0	0	1

Cross Entropy – Measure of Impurity/Uncertainty

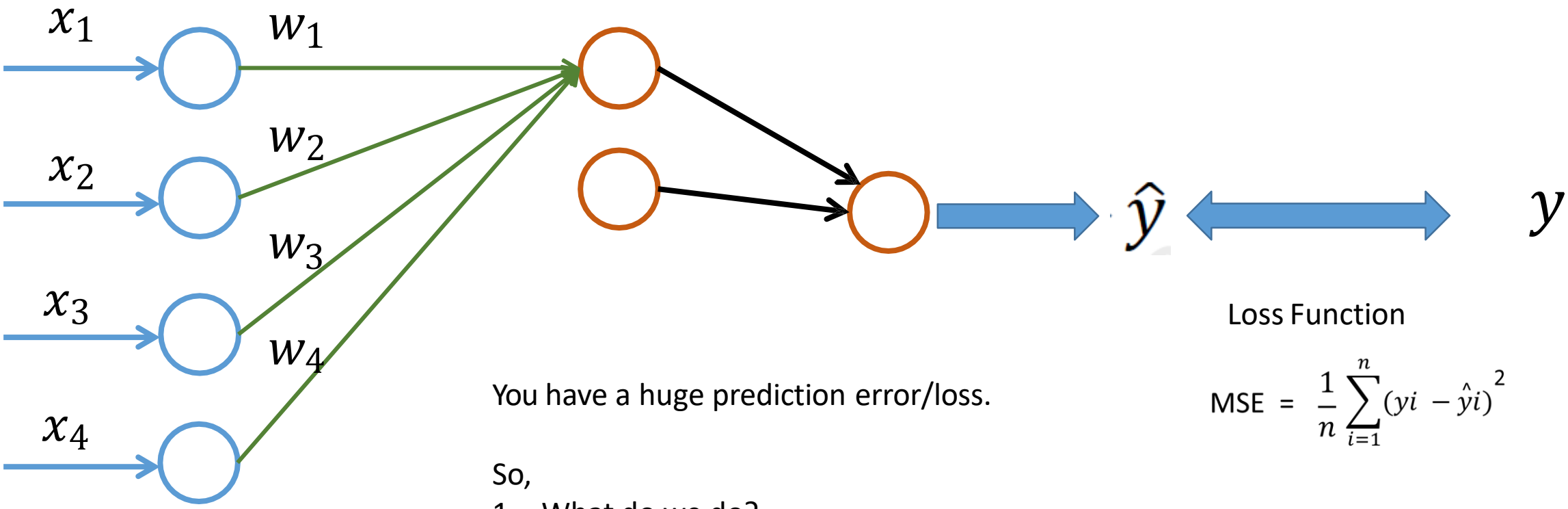




$$H(p,q) = -\sum_{i=1}^{x} p(x) \log q(x)$$

What is Optimization?

What is Optimization?



Loss Function

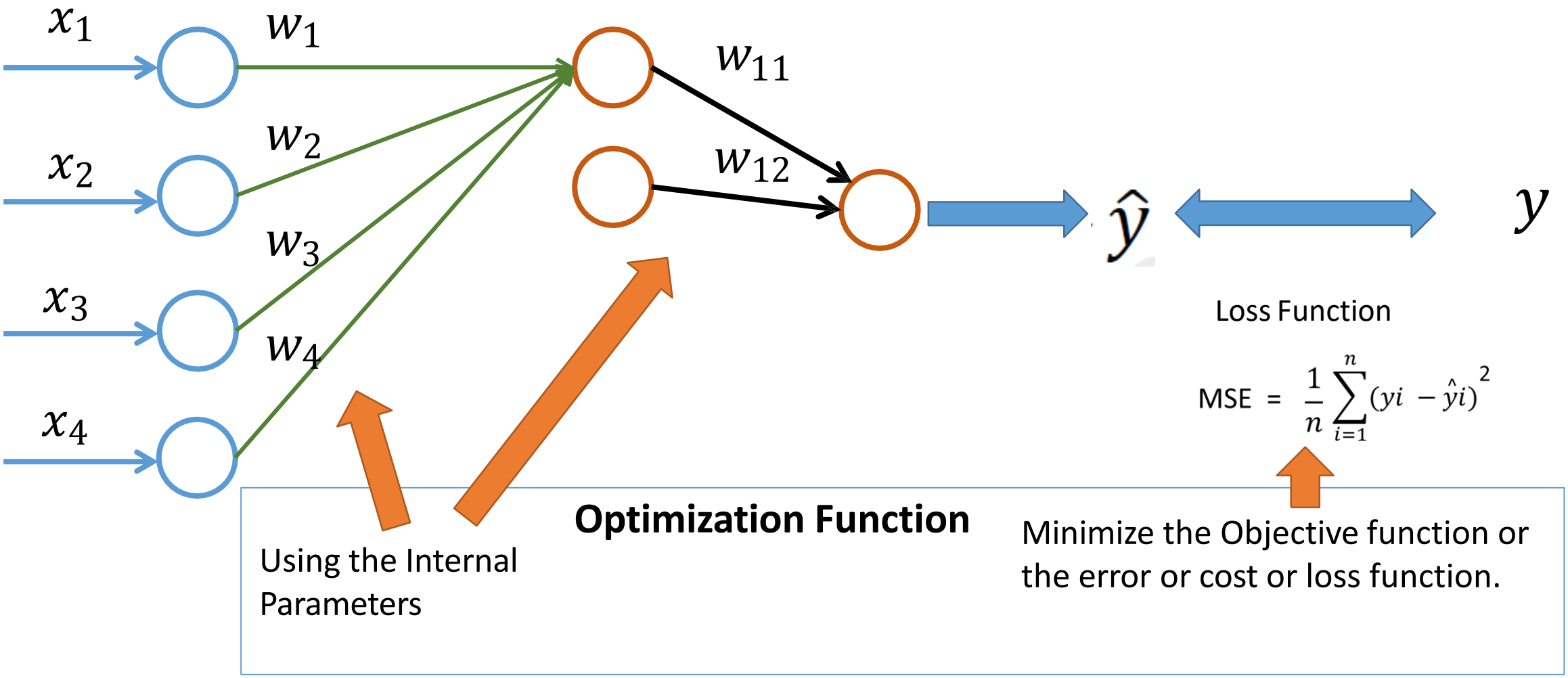
MSE = $\frac{1}{n}\sum_{i=1}^{n}(yi - \hat{y}i)^2$

You have a huge prediction error/loss.

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?

What is Optimization?

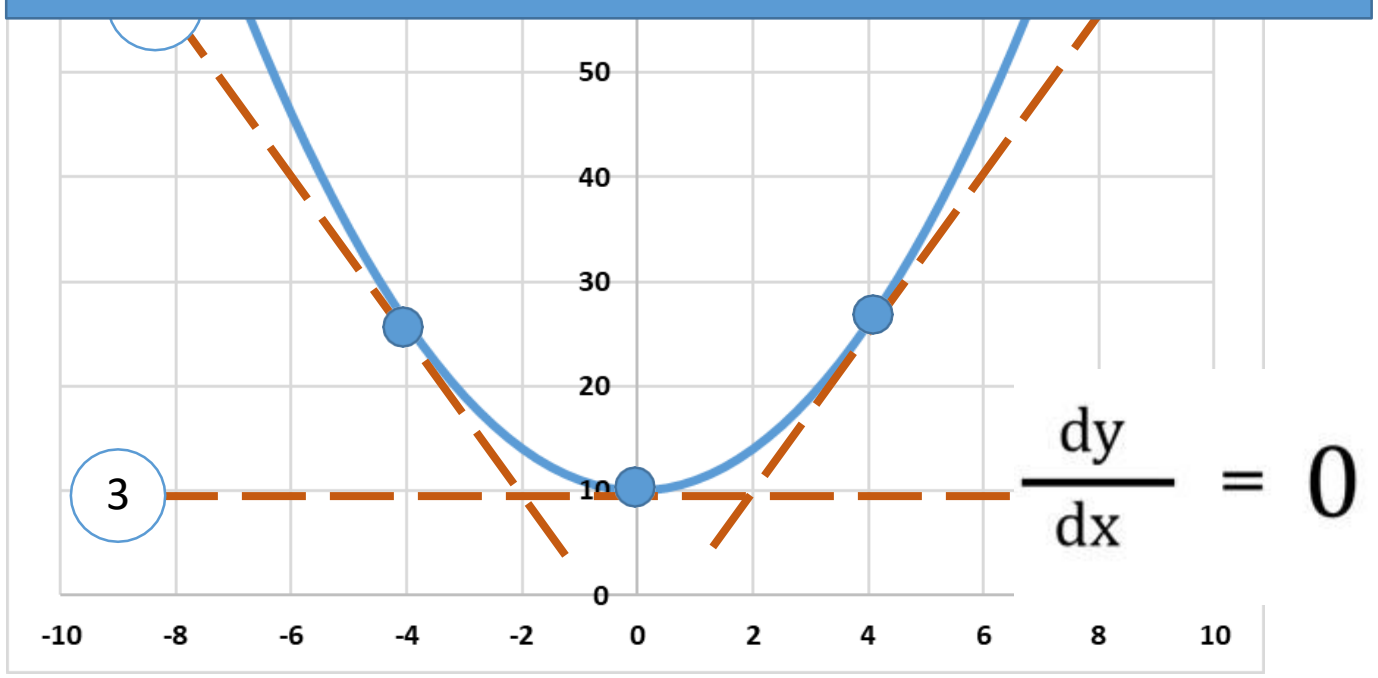


Optimization using First Order Derivatives

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

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

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



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

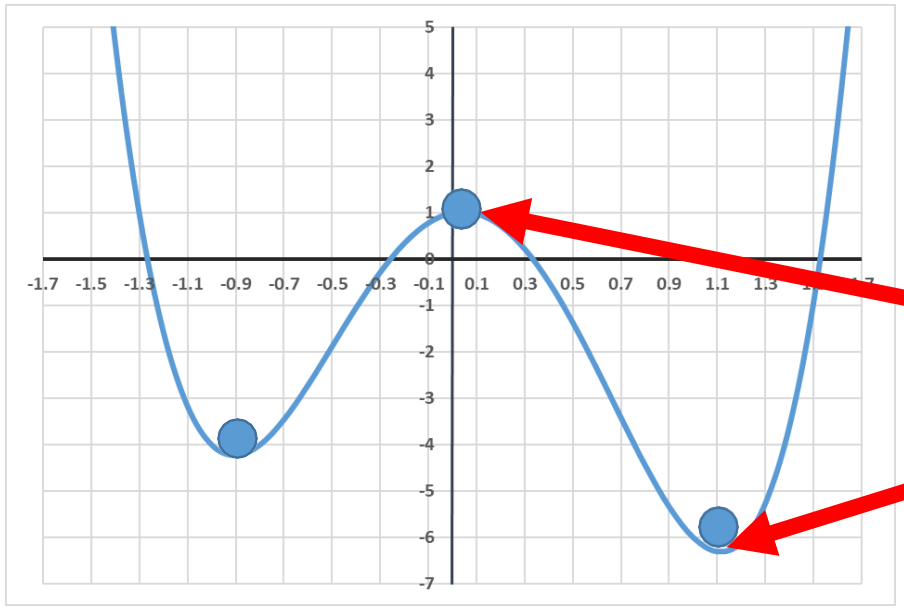
2 $\frac{dy}{dx} = +8$

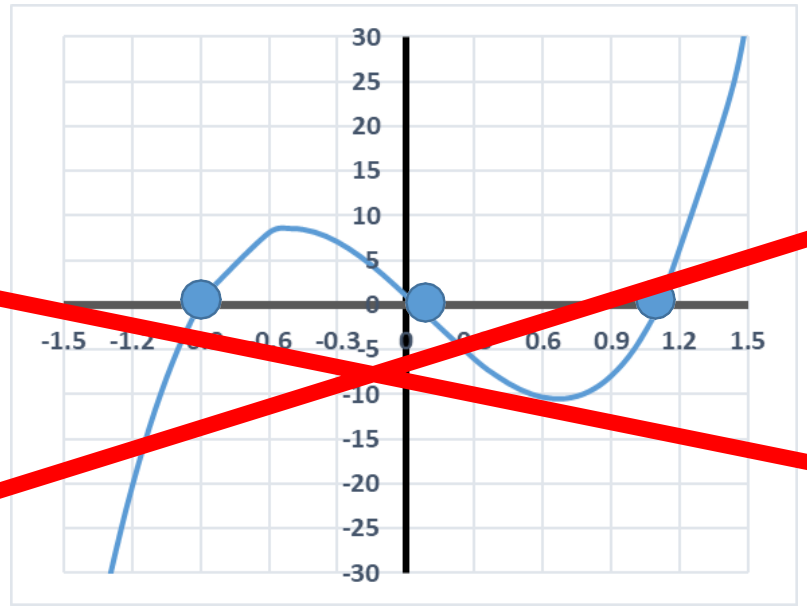
Optimization using Second Order Derivative

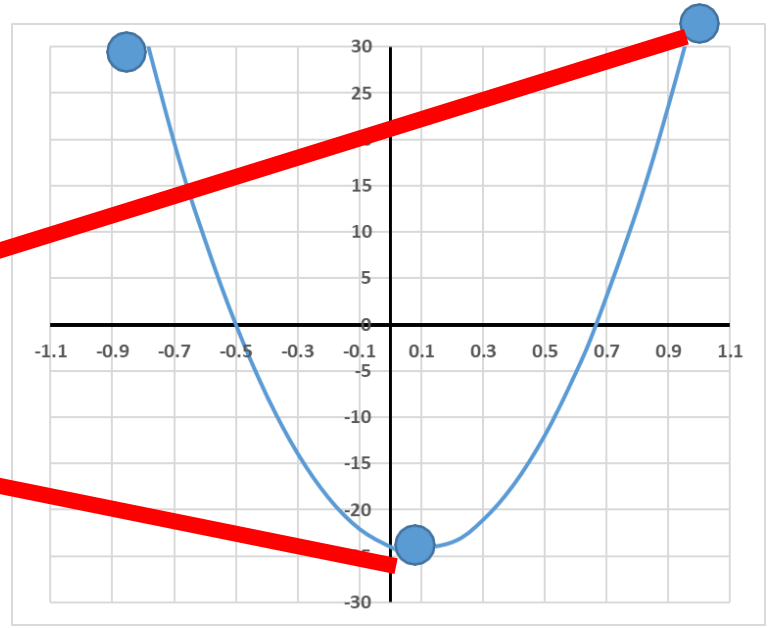
$$y = 6x^4 - 2x^3 - 12x^2 + x + 1$$

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

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







-0.9054 0.04131 1.1141

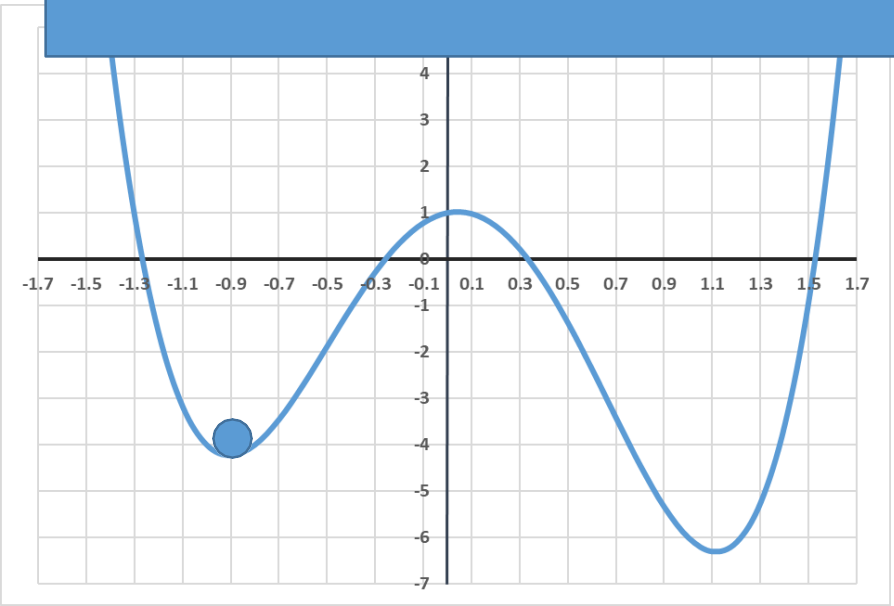
-0.9054 0.04131 1.1141

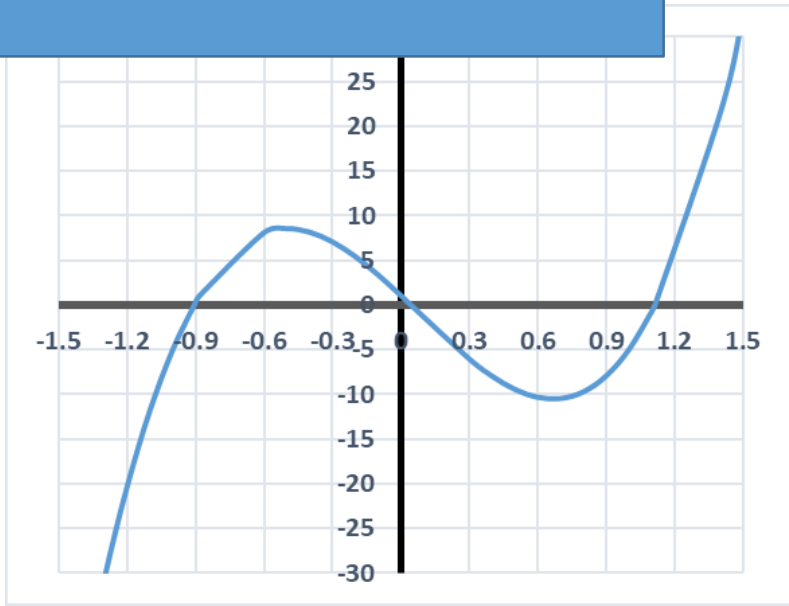
-0.9054 0.04131 1.1141

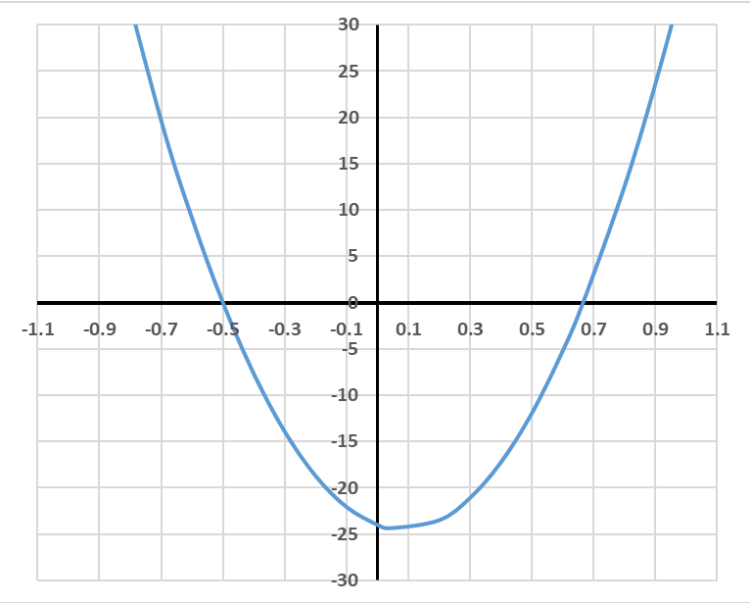
Optimization using Second Order Derivative

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

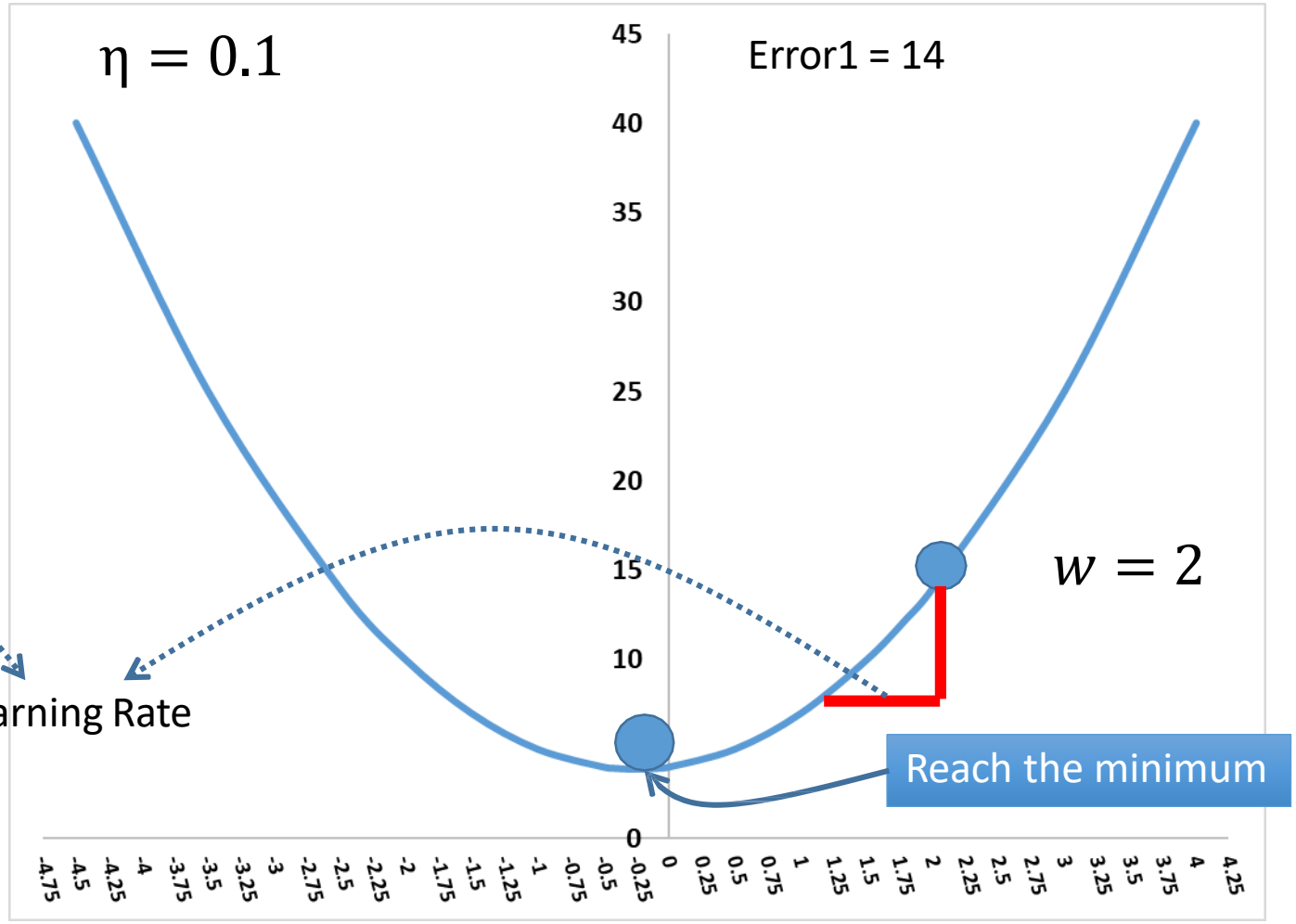
Let's Solve the gradient descent

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

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

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

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



Let's Solve the gradient descent

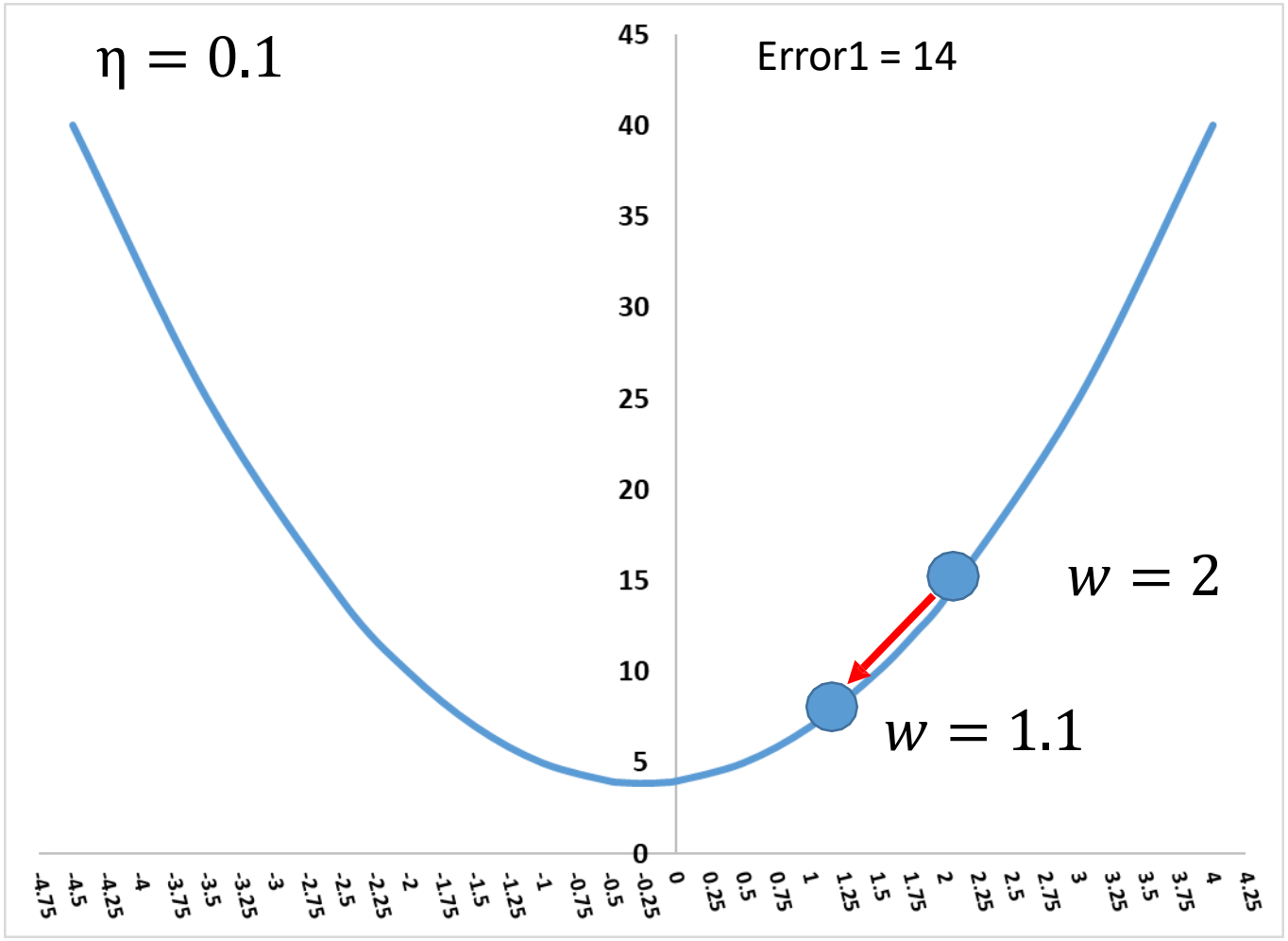
$$f(w) = 2w^2 + w + 4$$

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

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

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

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



Let's Solve the gradient descent

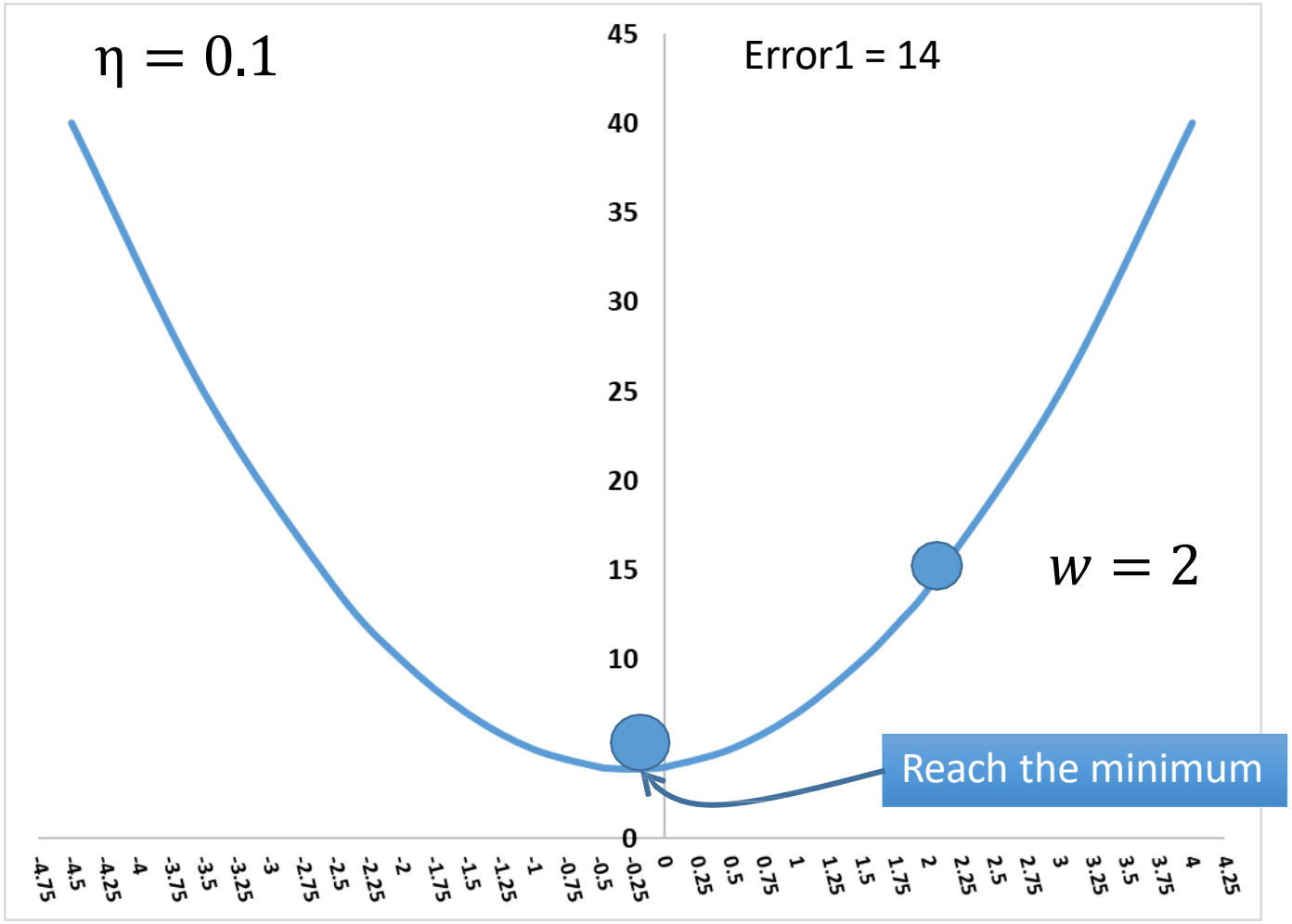
$$f(w) = 2w^2 + w + 4$$

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

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

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

$$w_{i+1} = 2 - 0.1(4 * 2 + 1)$$
$$= 2 - 0.9$$



Let's Solve the gradient descent

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

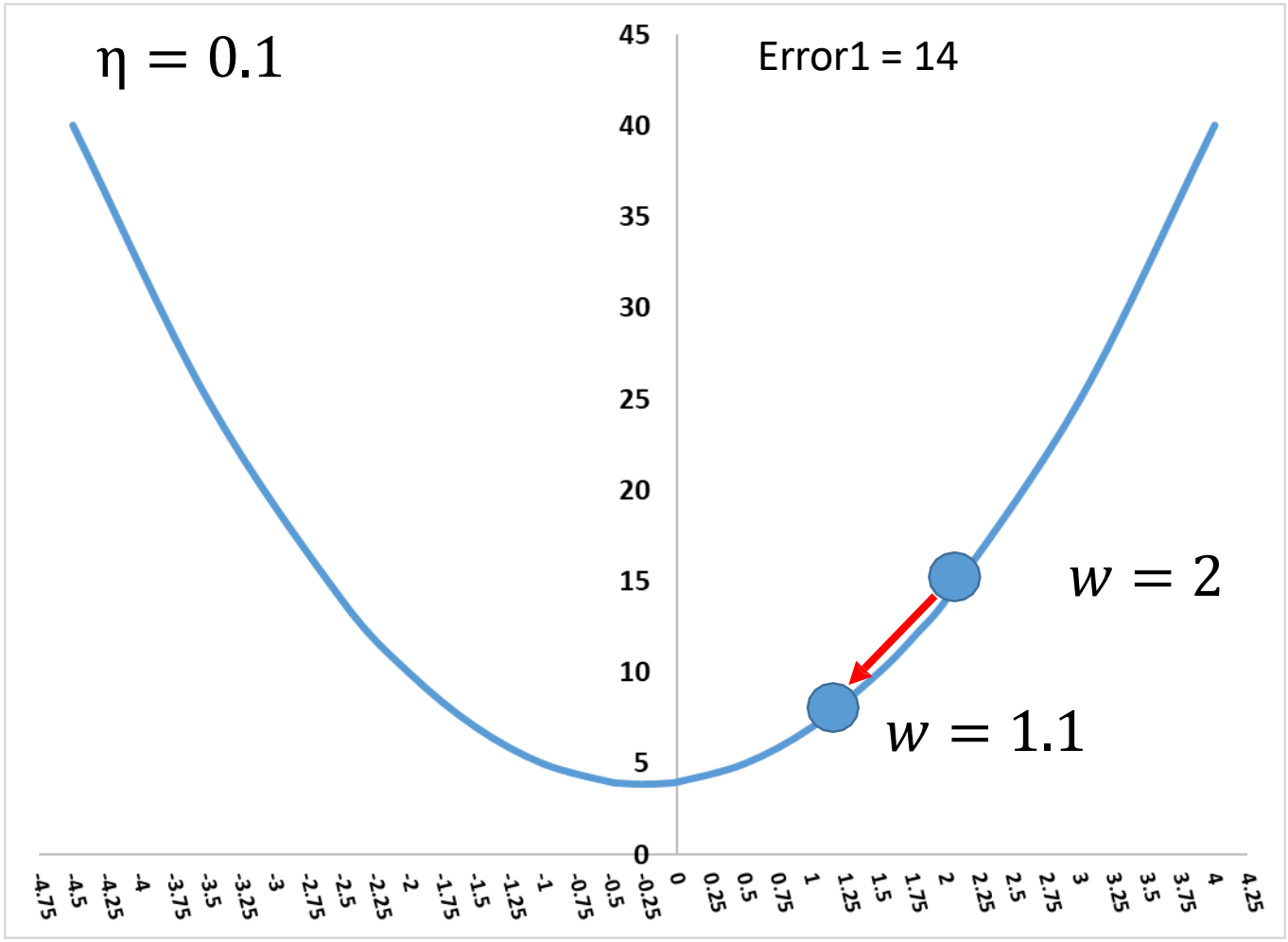
$$f'(w) = 4w + 1$$

<u>Iteration 1</u>

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

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

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



Let's Solve the gradient descent

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

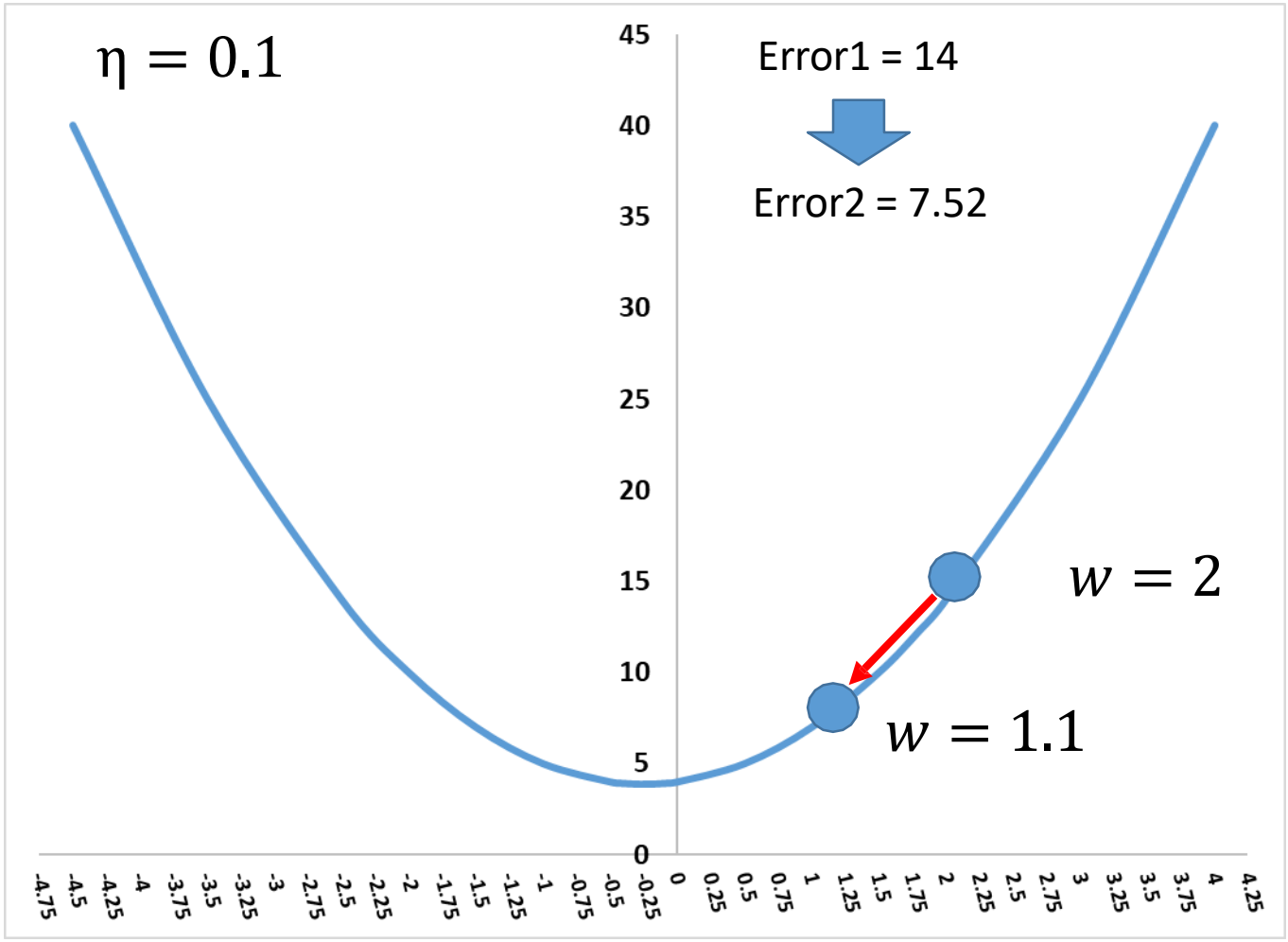
$$f'(w) = 4w + 1$$

<u>Iteration 2</u>

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

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

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



Let's Solve the gradient descent

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

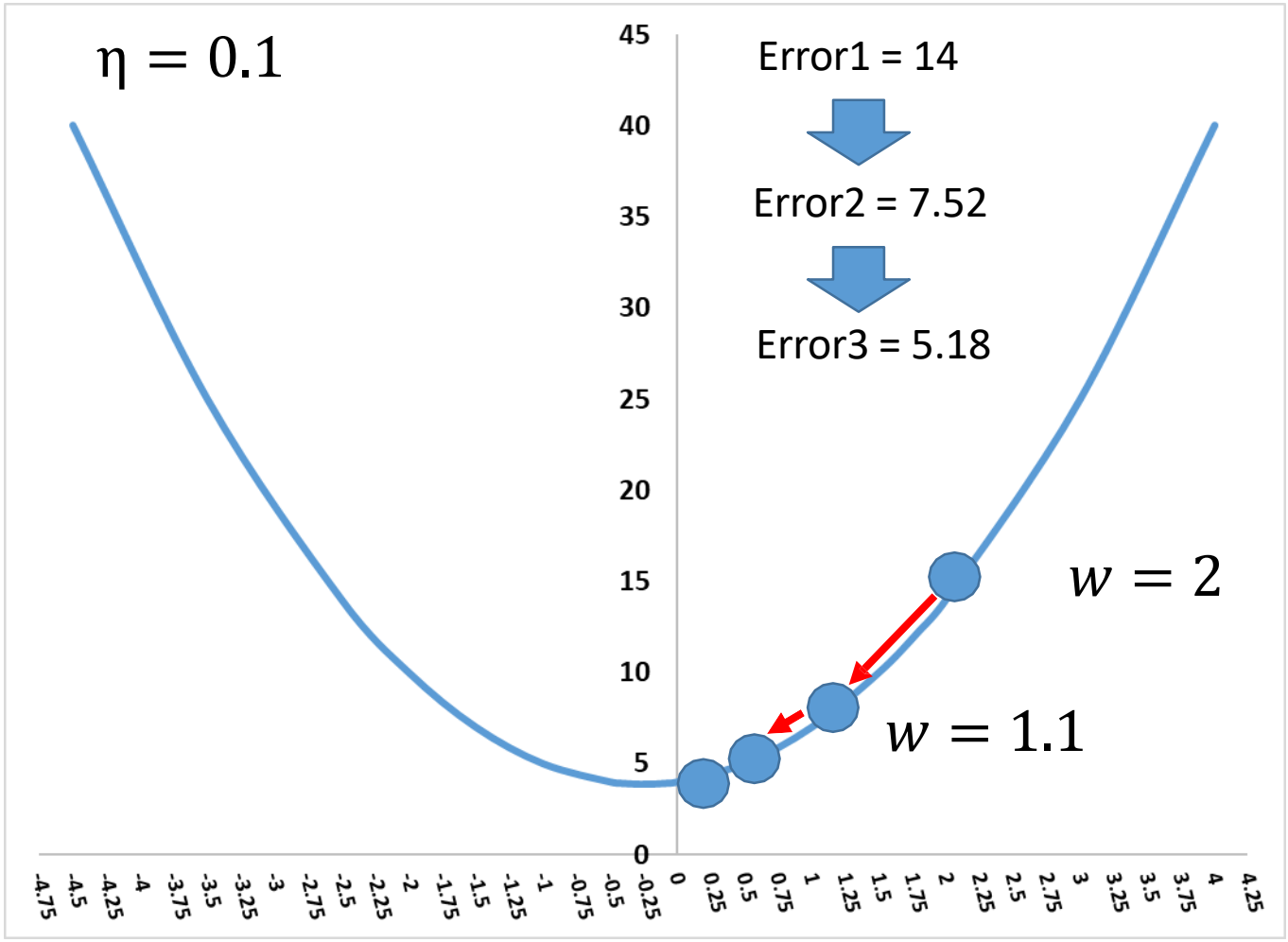
$$f'(w) = 4w + 1$$

<u>Iteration 3</u>

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

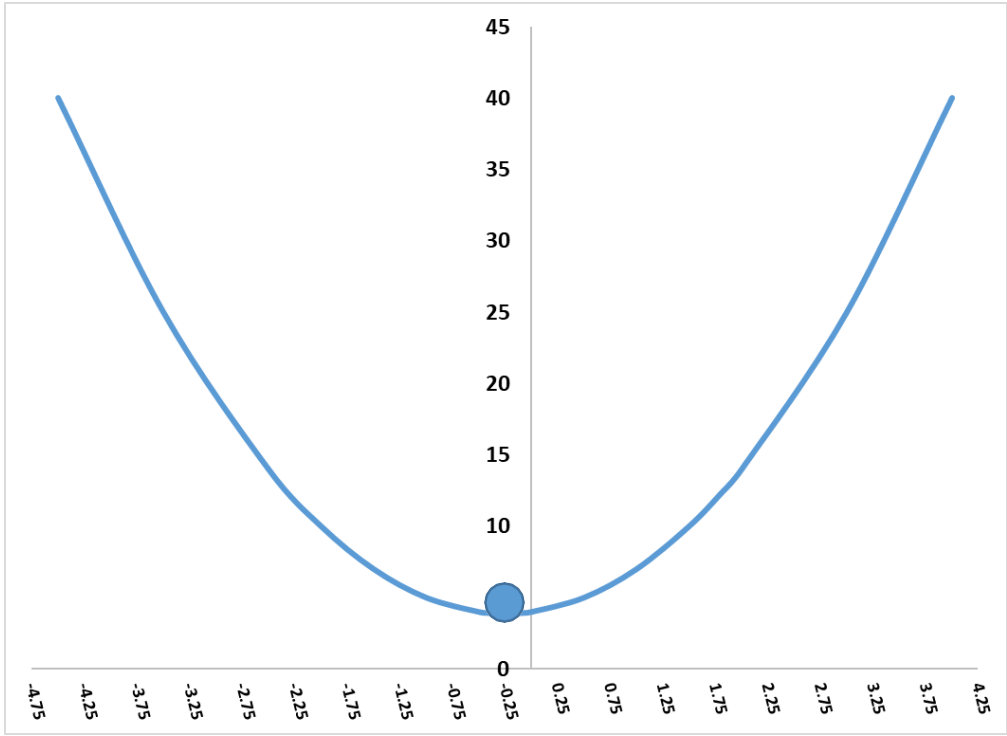
$$w_{i+1} = w_i - \eta * f'(w)$$

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



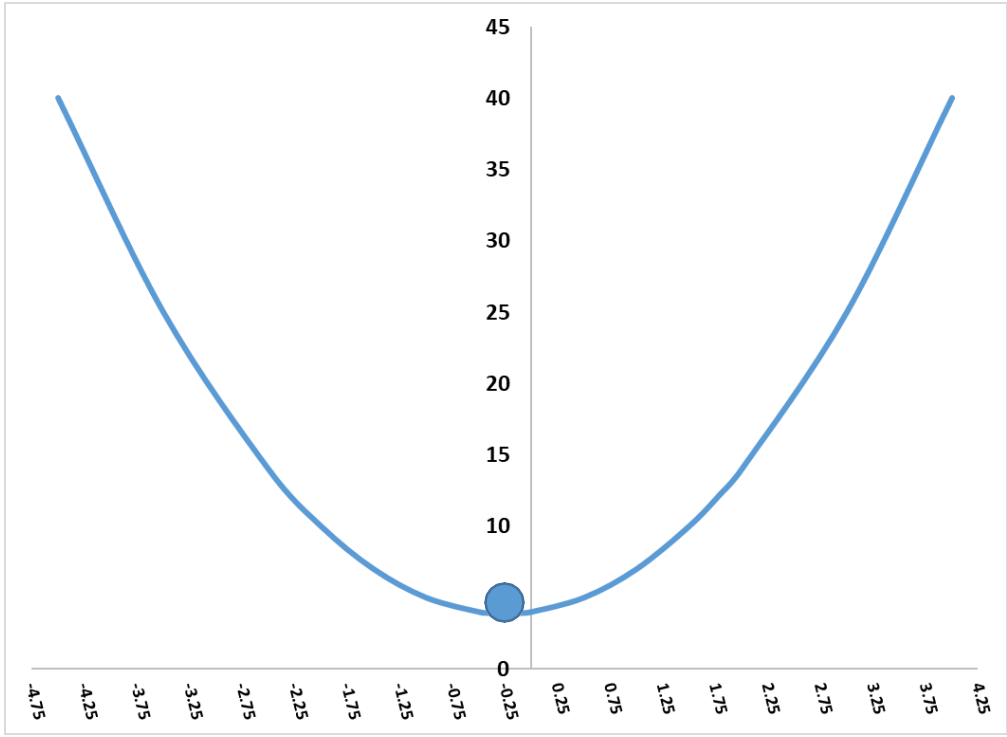
Iteration	W	f (W)	f'(W)	$w_{i+1} = w_i - \eta * f'(w)$
1	2.00000	14.00000	9.00000	1.10000
2	1.10000	7.52000	5.40000	0.56000
3	0.56000	5.18720	3.24000	0.23600
4	0.23600	4.34739	1.94400	0.04160
5	0.04160	4.04506	1.16640	-0.07504
6	-0.07504	3.93622	0.69984	-0.14502
7	-0.14502	3.89704	0.41990	-0.18701
8	-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

$\eta = 0.1$



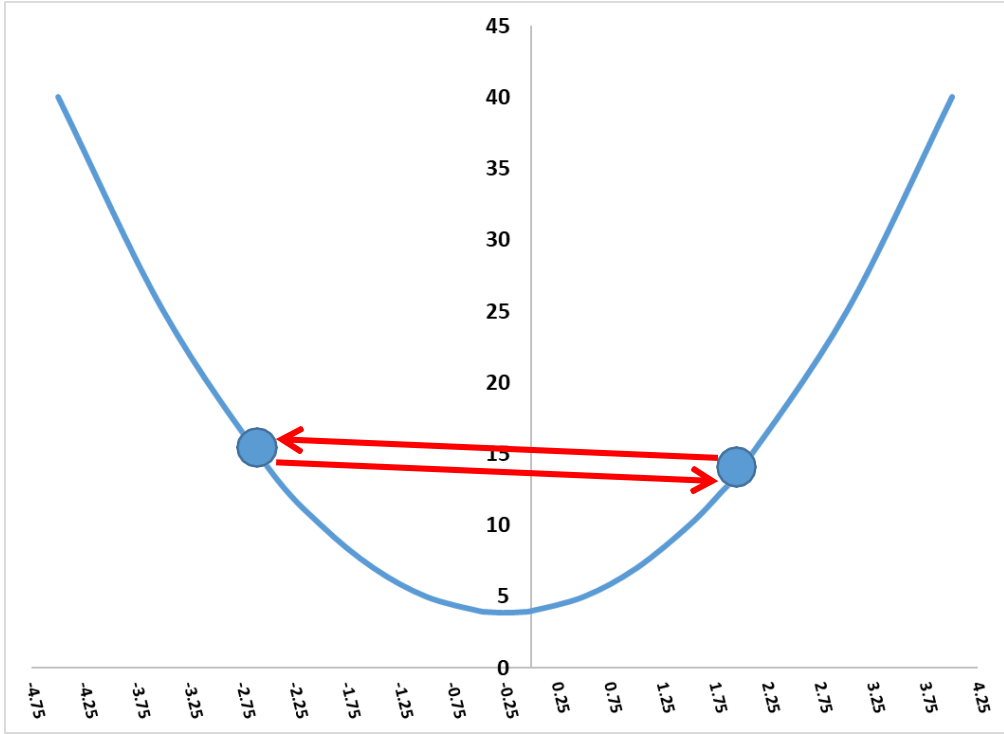
Iteration	W	f (W)	f'(W)	$w_{i+1} = w_i - \eta * f'(w)$
1	2.00000	14.00000	9.00000	0.20000
2	0.20000	4.28000	1.80000	-0.16000
3	-0.16000	3.89120	0.36000	-0.23200
4	-0.23200	3.87565	0.07200	-0.24640
5	-0.24640	3.87503	0.01440	-0.24928
6	-0.24928	3.87500	0.00288	-0.24986
7	-0.24986	3.87500	0.00058	-0.24997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

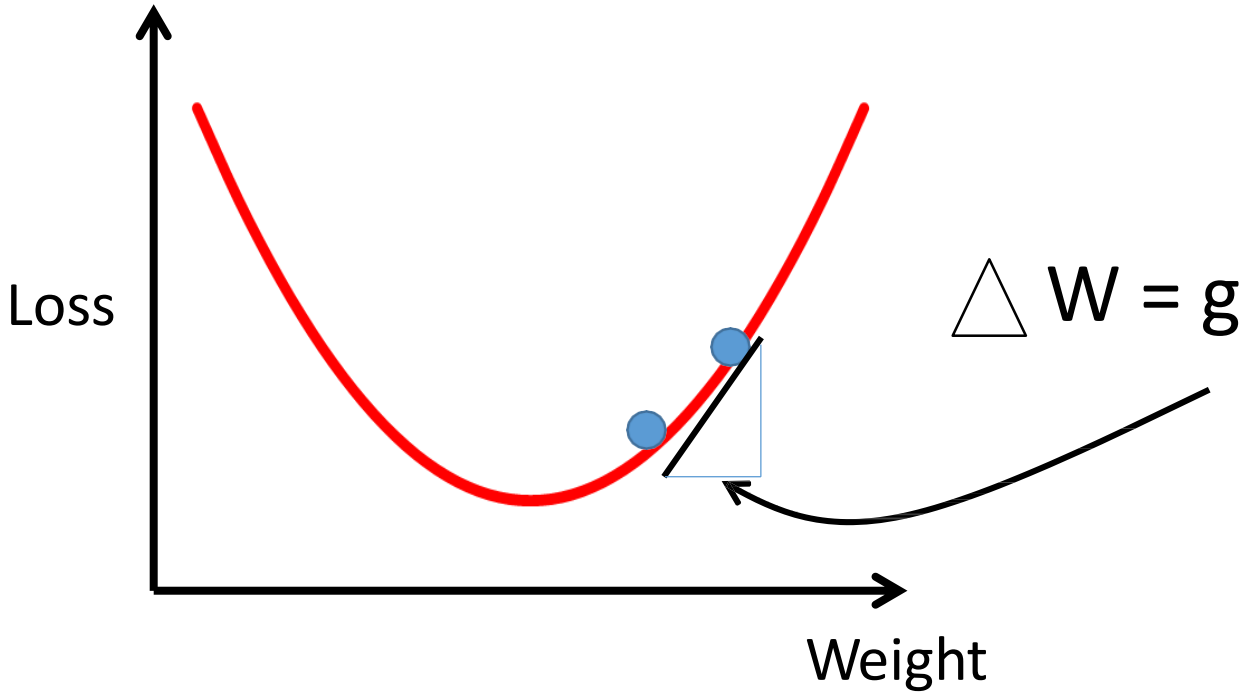
$$error\ or\ loss = \frac{1}{2}\ (y\ -\ \hat{y})^2$$

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

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

Gradient Descent

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



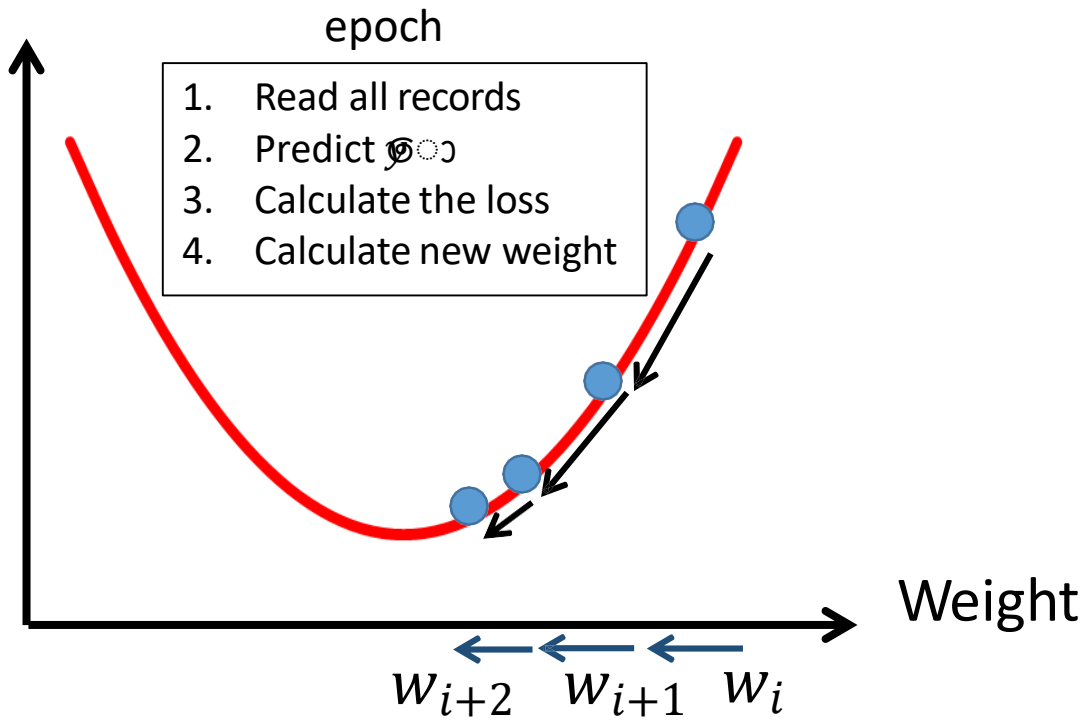
$$w_i = w_j - \eta \Delta w$$

$\eta = Learning\ Rate$

Gradient Descent

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

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

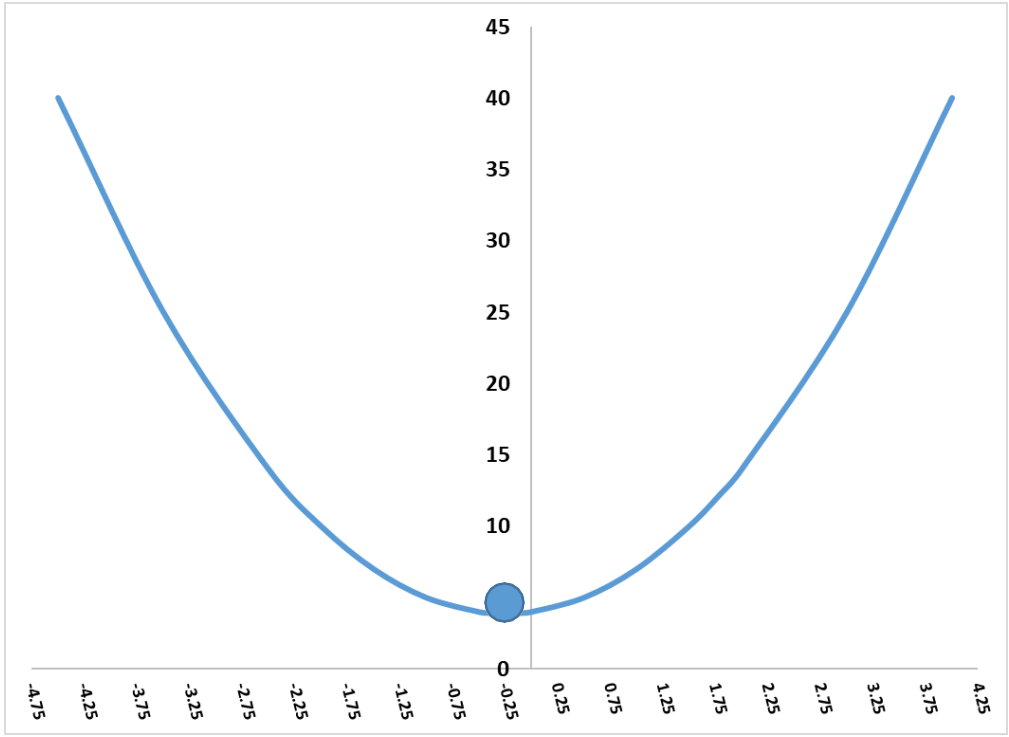


Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…

$n = 10{,}000$

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

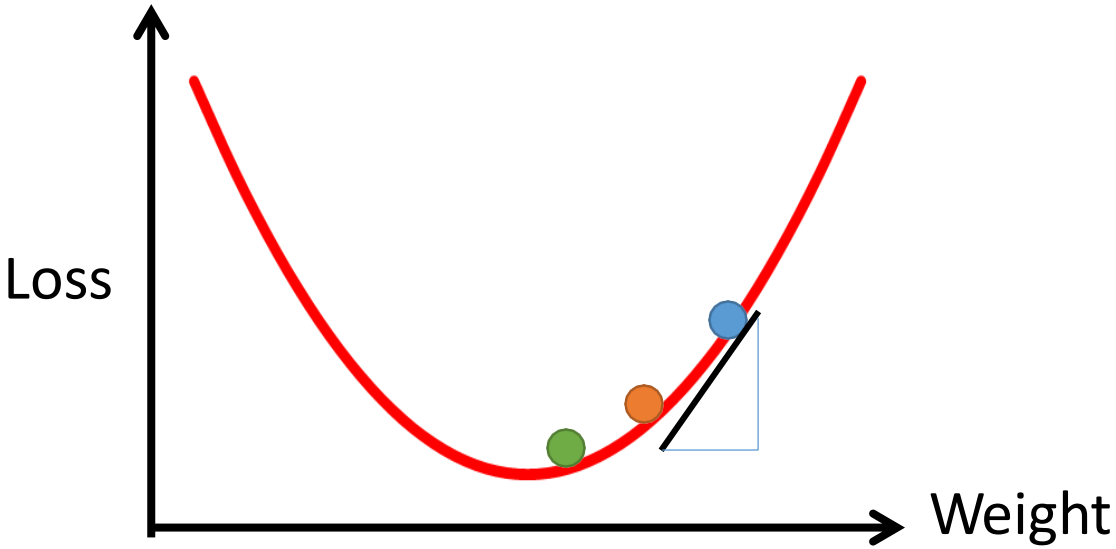
210,000 read operations

Batch Gradient Descent

X1	X2	….	Xn

Sum of All before taking one step (epoch)

Long time to reach the bottom

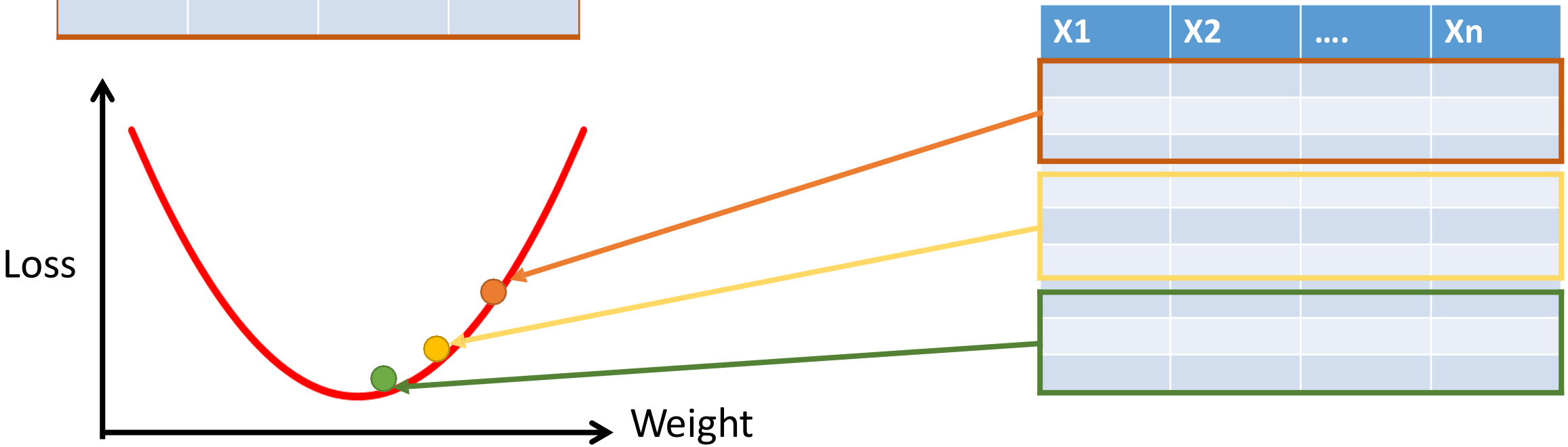


$$w_{i+1} = w_i - \eta * f'(w)$$

Mini Batch Gradient Descent

X1	X2	....	Xn

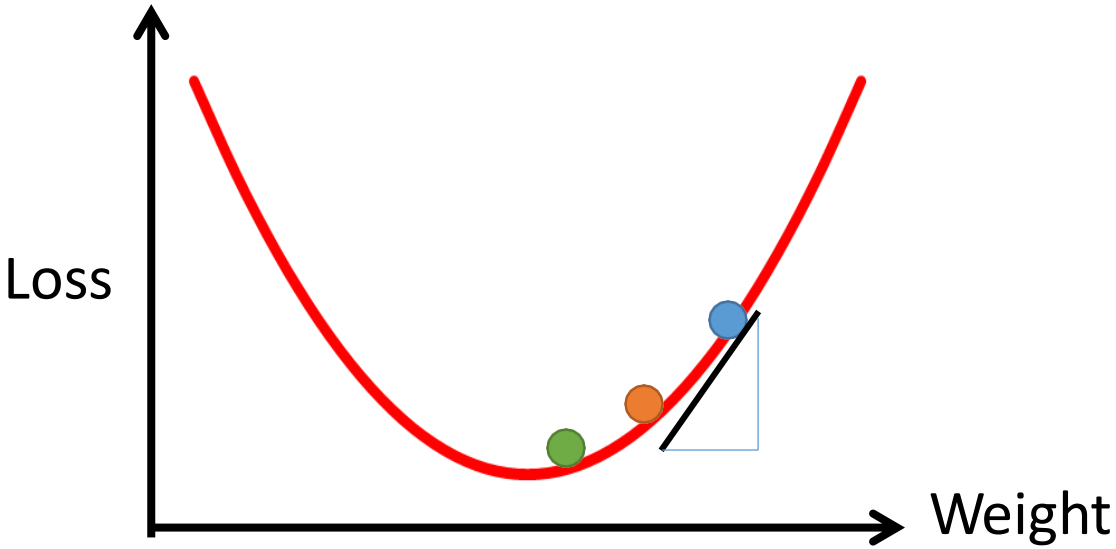
Mini-Batch Gradient Descent



Online or Stochastic Gradient Descent

X1	X2	….	Xn

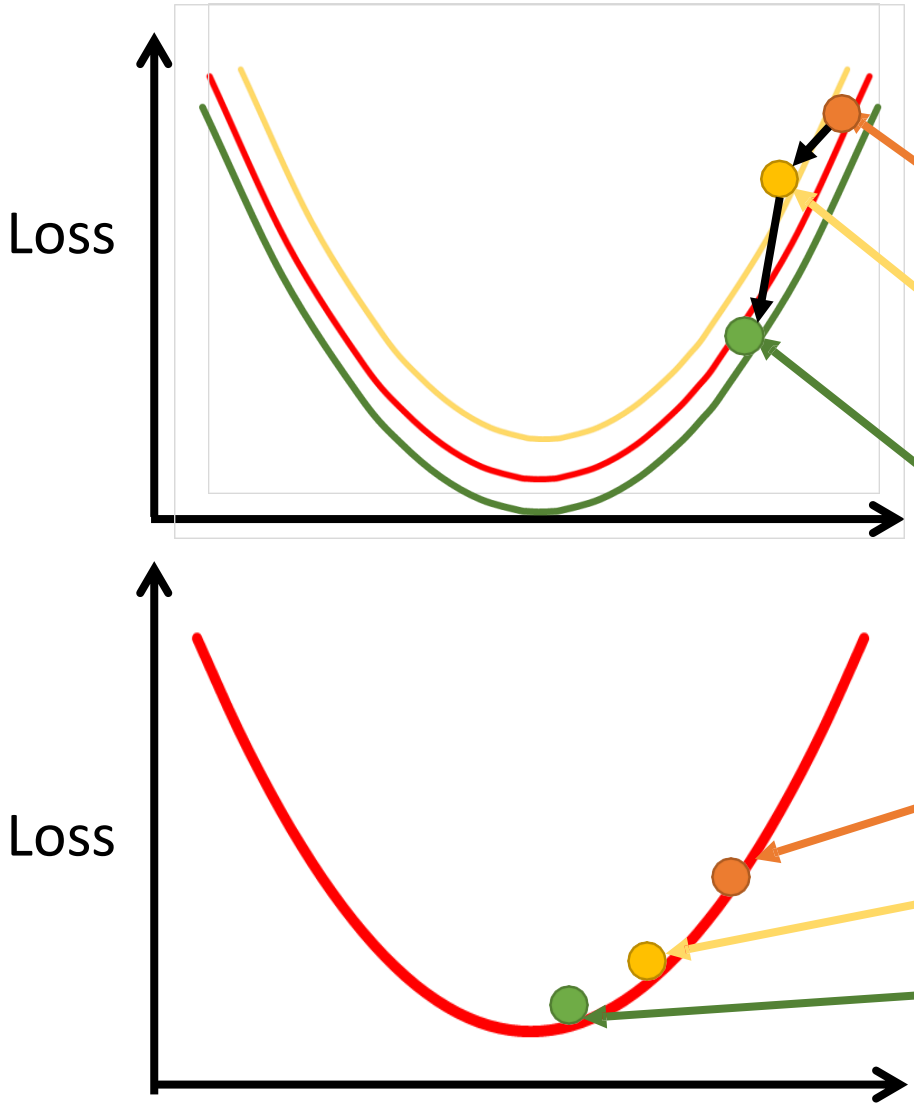
One observation at a time



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

SGD With Momentum

Mini Batch Gradient Descent



Loss Of Batch 1 = 14

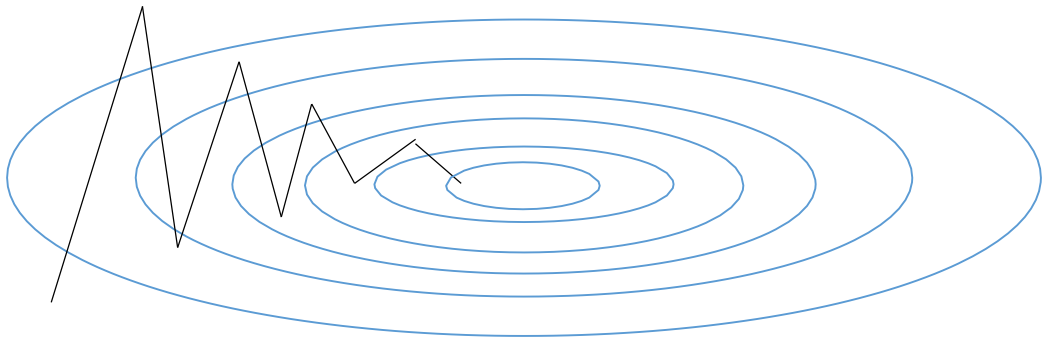
Loss of Batch 2 = 14.5

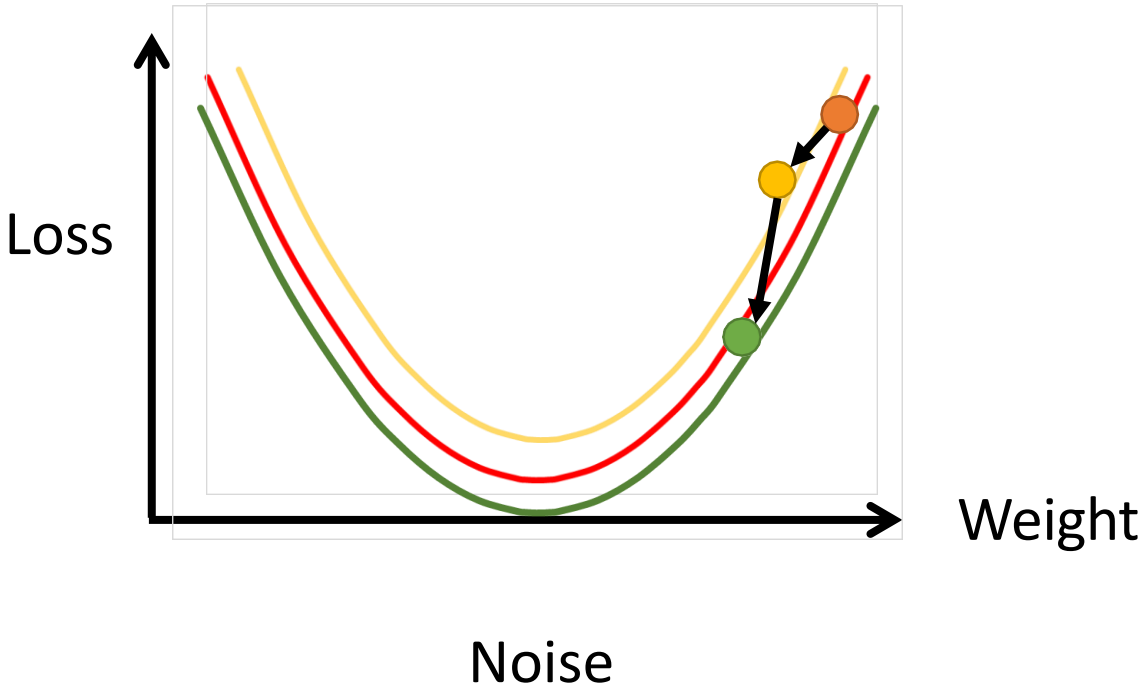
Loss Of Batch 3 = 13.5

Iteration 1

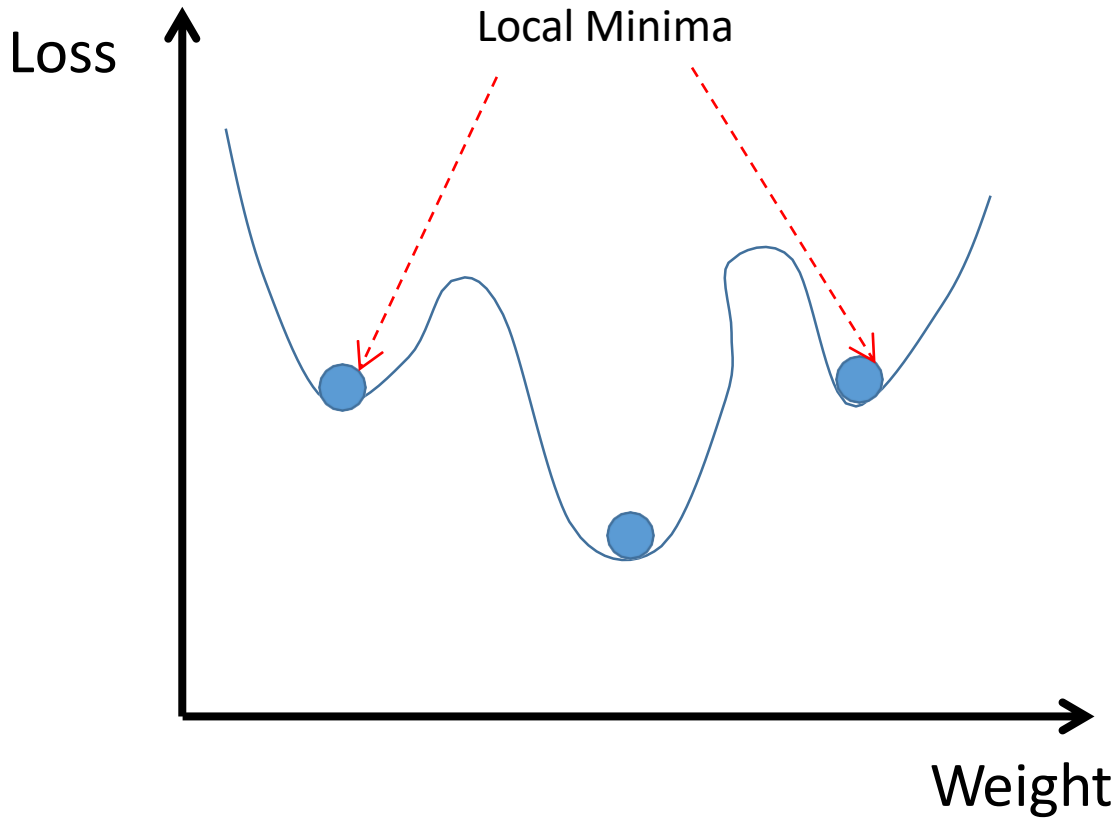
X1	X2	....	Xn

Mini Batch Gradient Descent

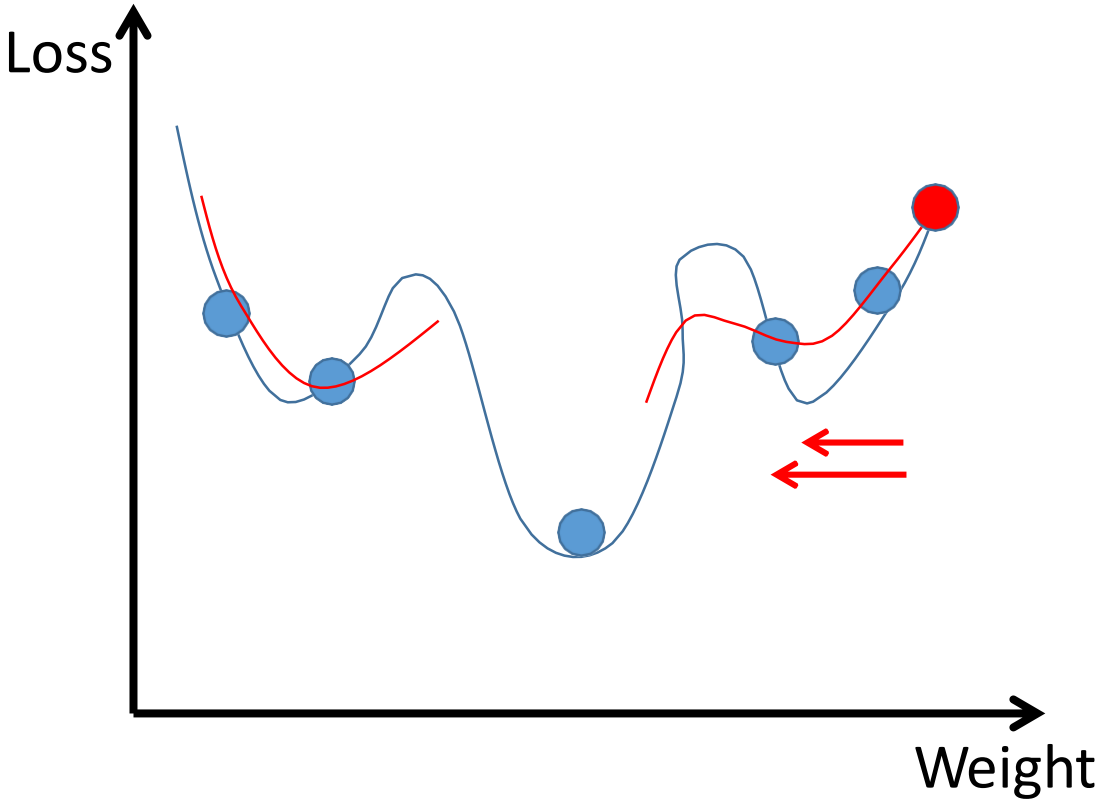




Problem of Local Minima



Add Momentum to the Weight



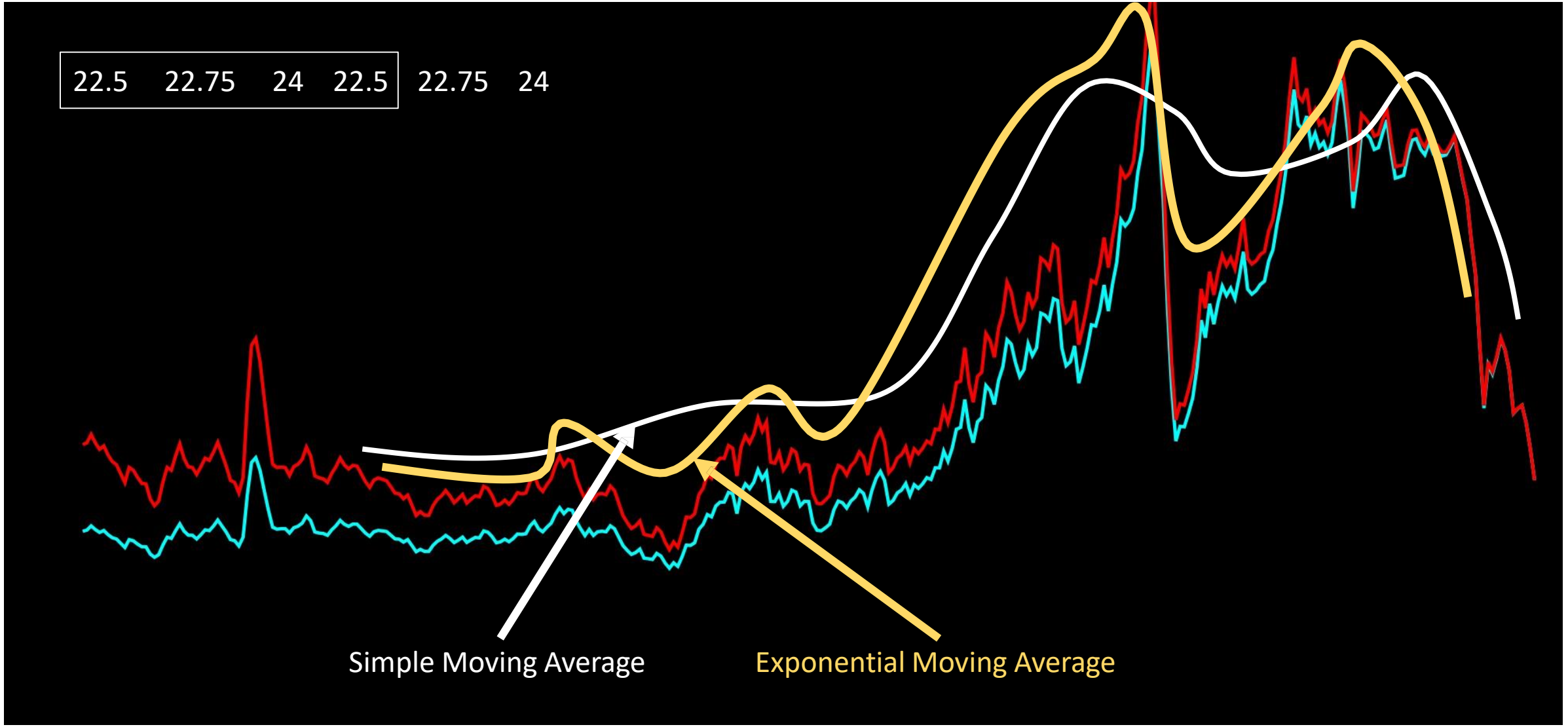
$$w_{i+1} = w_i - \eta * \frac{\partial L}{\partial w}$$

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

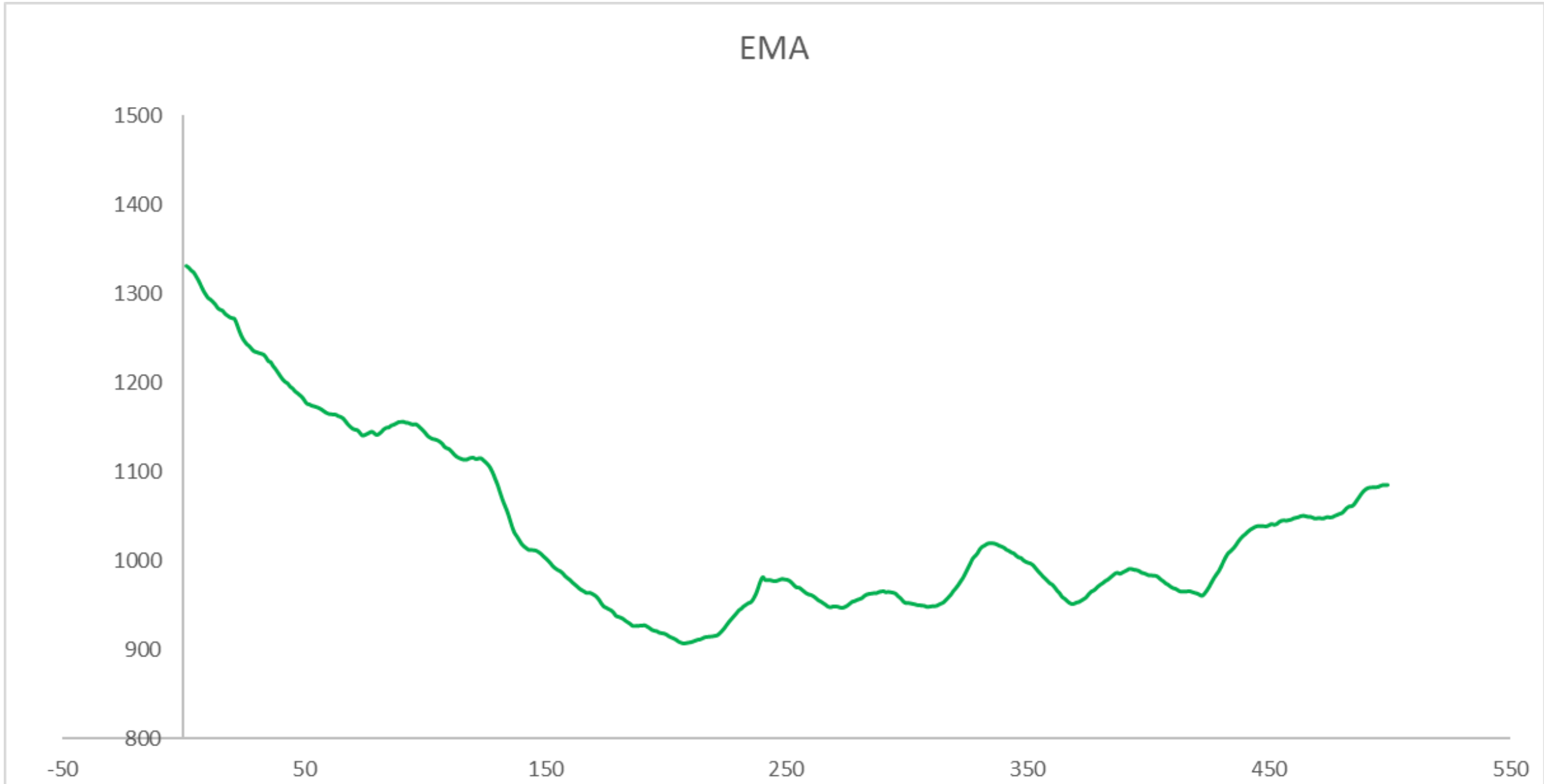
Momentum

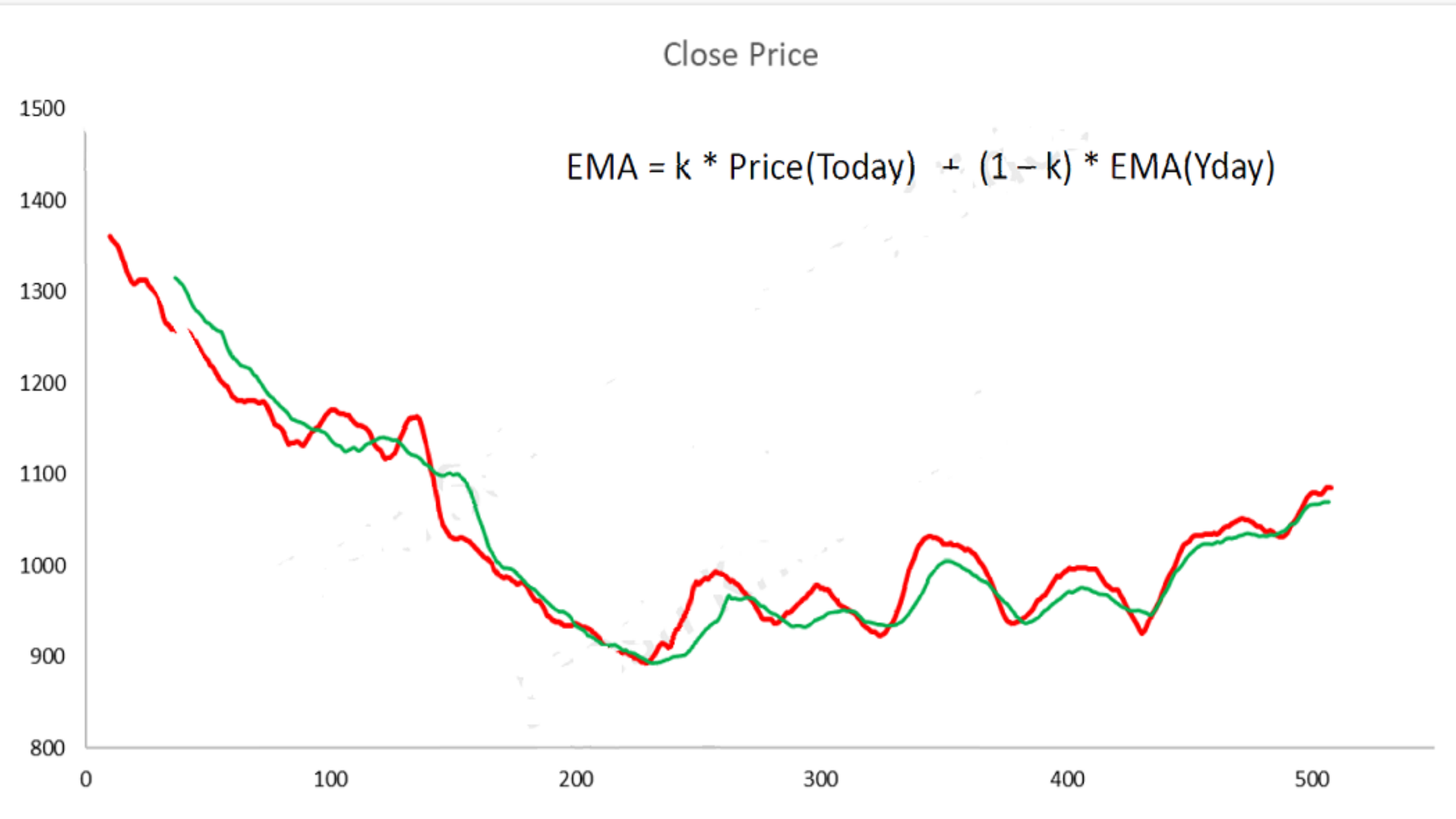
Understand the Exponential Moving Average

Moving Average




Close Price
1500
1400
1300
1200
1100
1000
900
800
0
100
200
300
400
500


EMA
1500
1400
1300
1200
1100
1000
900
800
-50
50
150
250
350
450
550


Close Price
EMA = k * Price(Today) + (1 – k) * EMA(Yday)
1500
1400
1300
1200
1100
1000
900
800
0
100
200
300
400
500

SGD with Momentum

EMA = k * Price(Today) + (1 – k) * EMA(Yday)

Velocity (V) Prev Velocity Momentum β Gradient

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

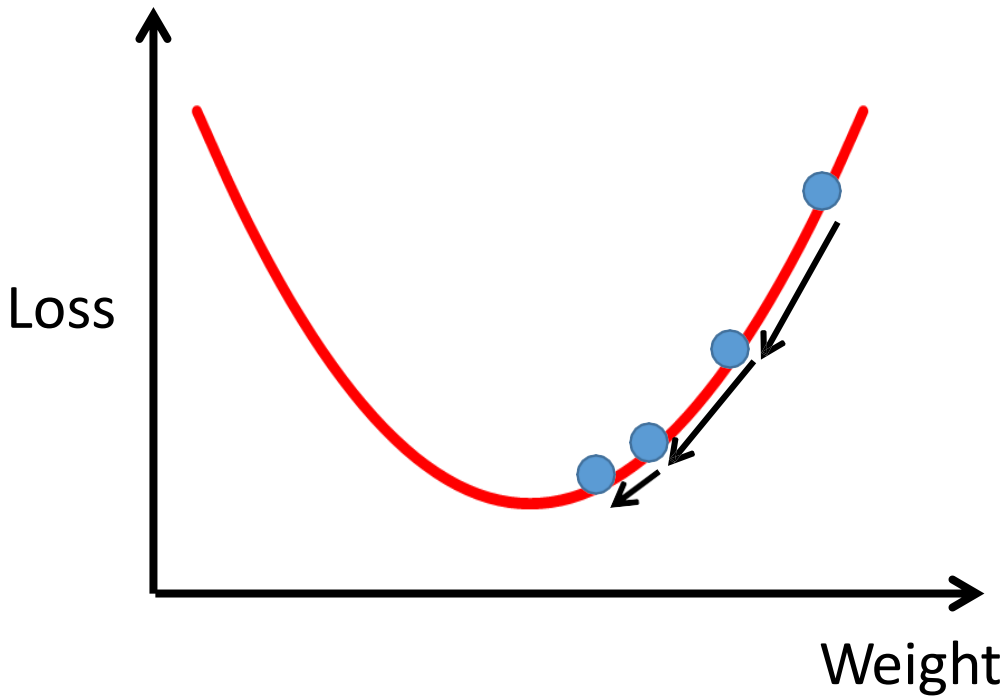
SGD with Momentum

EMA = k * Price(Today) + (1 – k) * EMA(Yday)

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

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

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

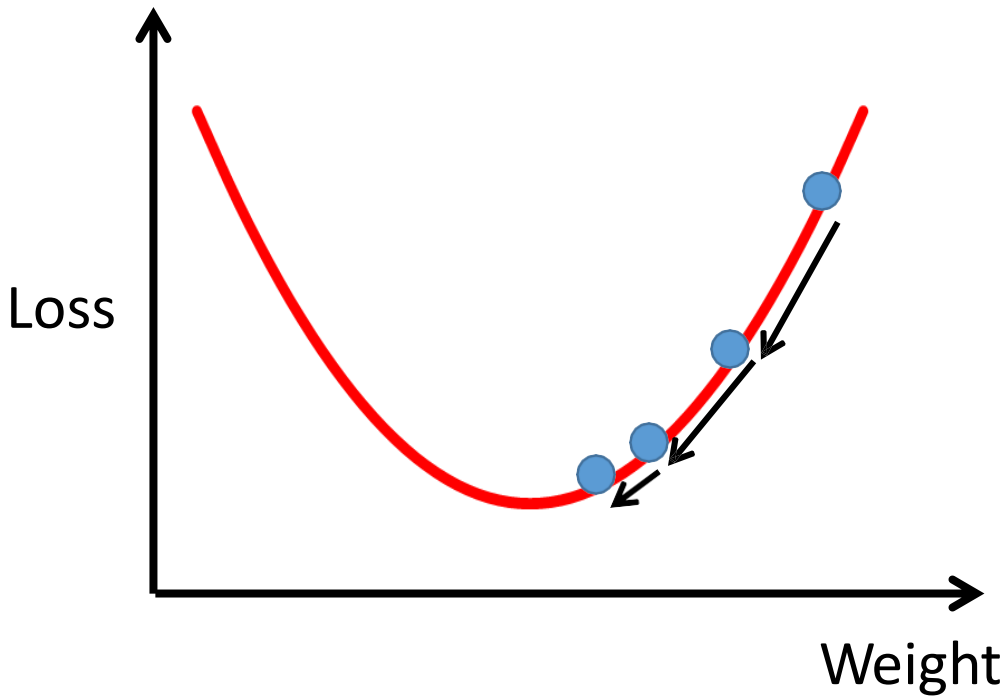


SGD with Momentum

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

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

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



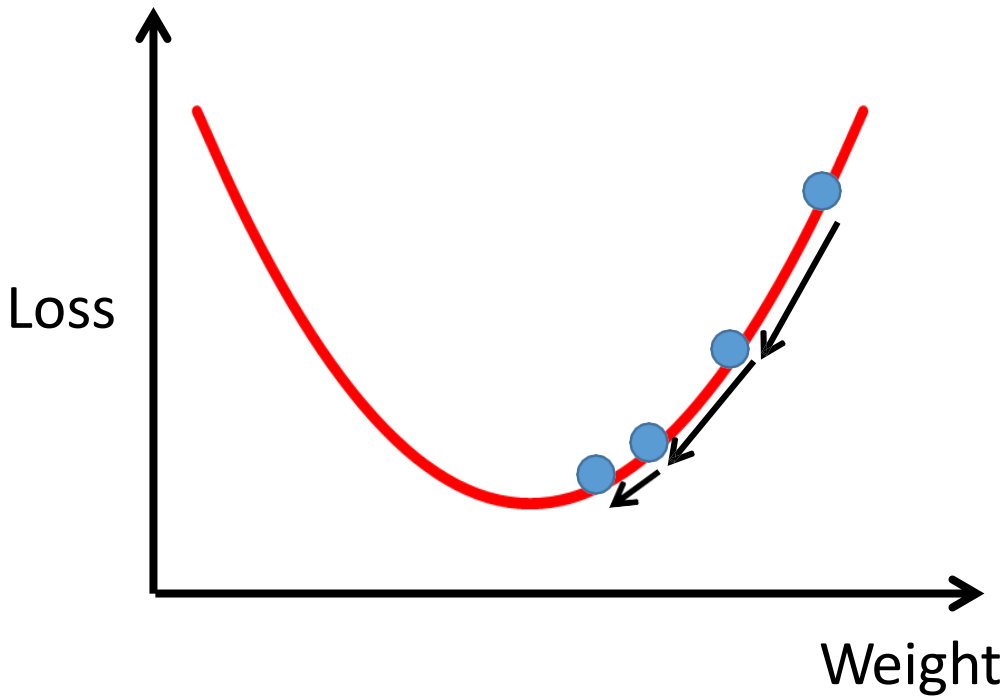
$$V_i = \beta * V_{i-1} + (1 - \beta) * \frac{\partial L}{\partial w}$$

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

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

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

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

SGD with Momentum

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

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

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

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

SGD with Momentum

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

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

SGD with Momentum

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

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

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

SGD with Momentum

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

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

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

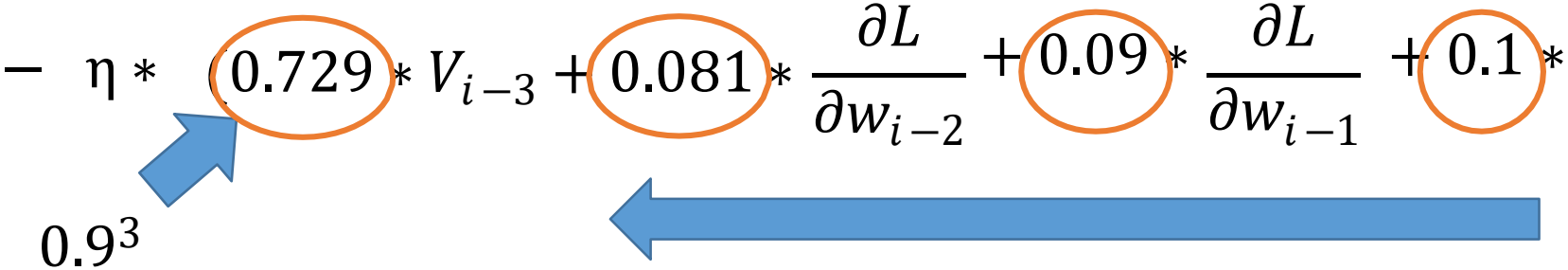
SGD with Momentum

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

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

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

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



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

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

RMSProp

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

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

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

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

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

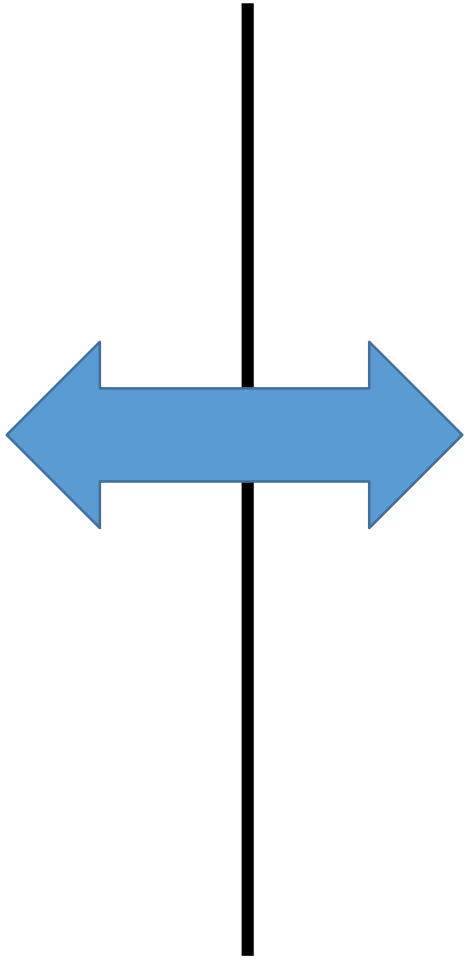
SGD with Momentum

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

RMSProp

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

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



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

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

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

Adam Optimization

SGD with Momentum

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

RMSProp

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

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

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

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

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

SGD with Momentum

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

RMSProp

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

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

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

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

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

Bias Correction

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

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

SGD with Momentum

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

RMSProp

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

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

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

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

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

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

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

Adam

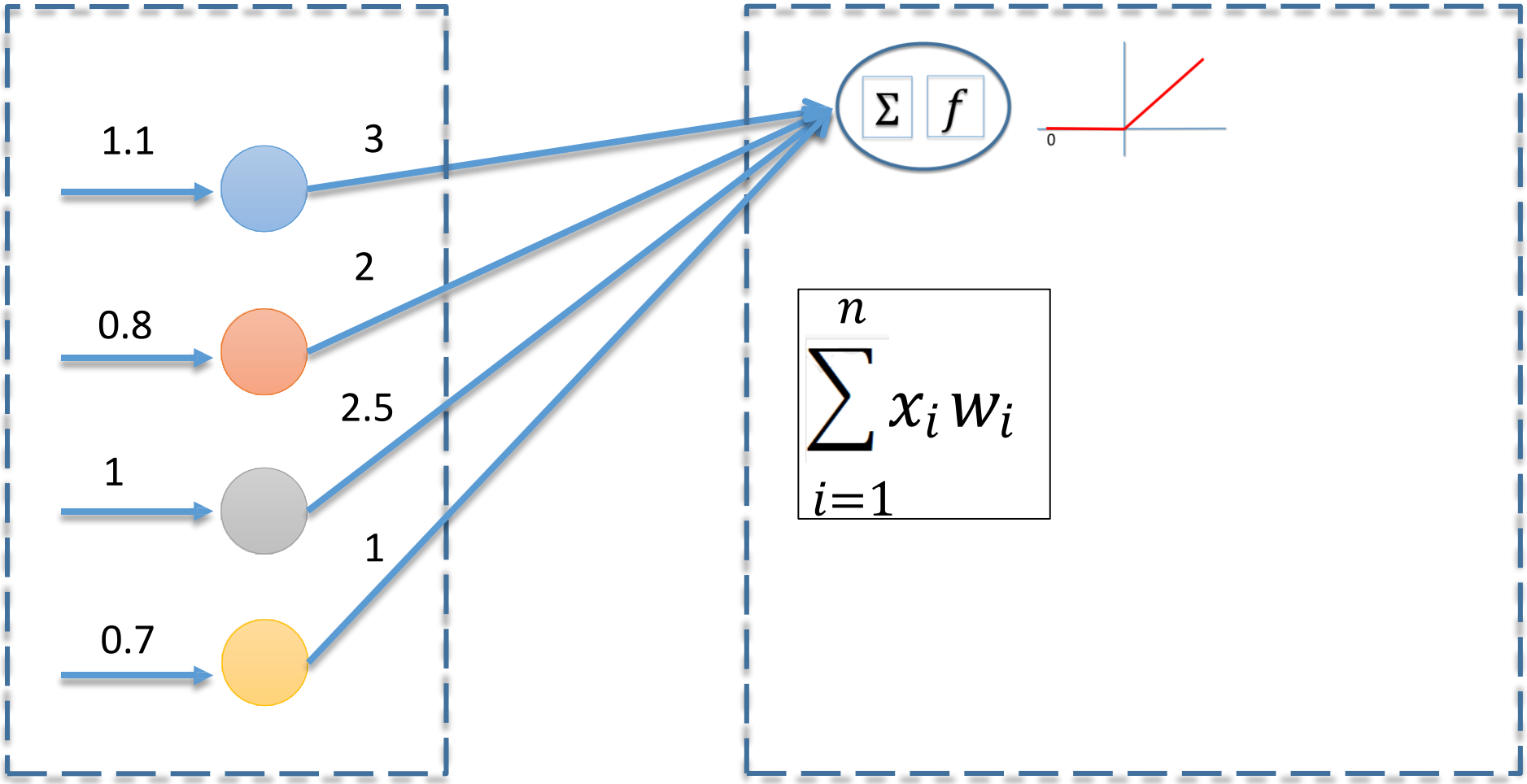
$$w_{i+1} = w_i - \frac{\eta * \hat{v}_i}{\epsilon + \sqrt{\hat{w}_{ave}}}$$

$$\hat{v}_i = \frac{v_i}{1 - \beta_1} \quad \Rightarrow \quad V = \beta_1 * V_{i-1} + (1 - \beta_1) * \frac{\partial L}{\partial w_i}$$

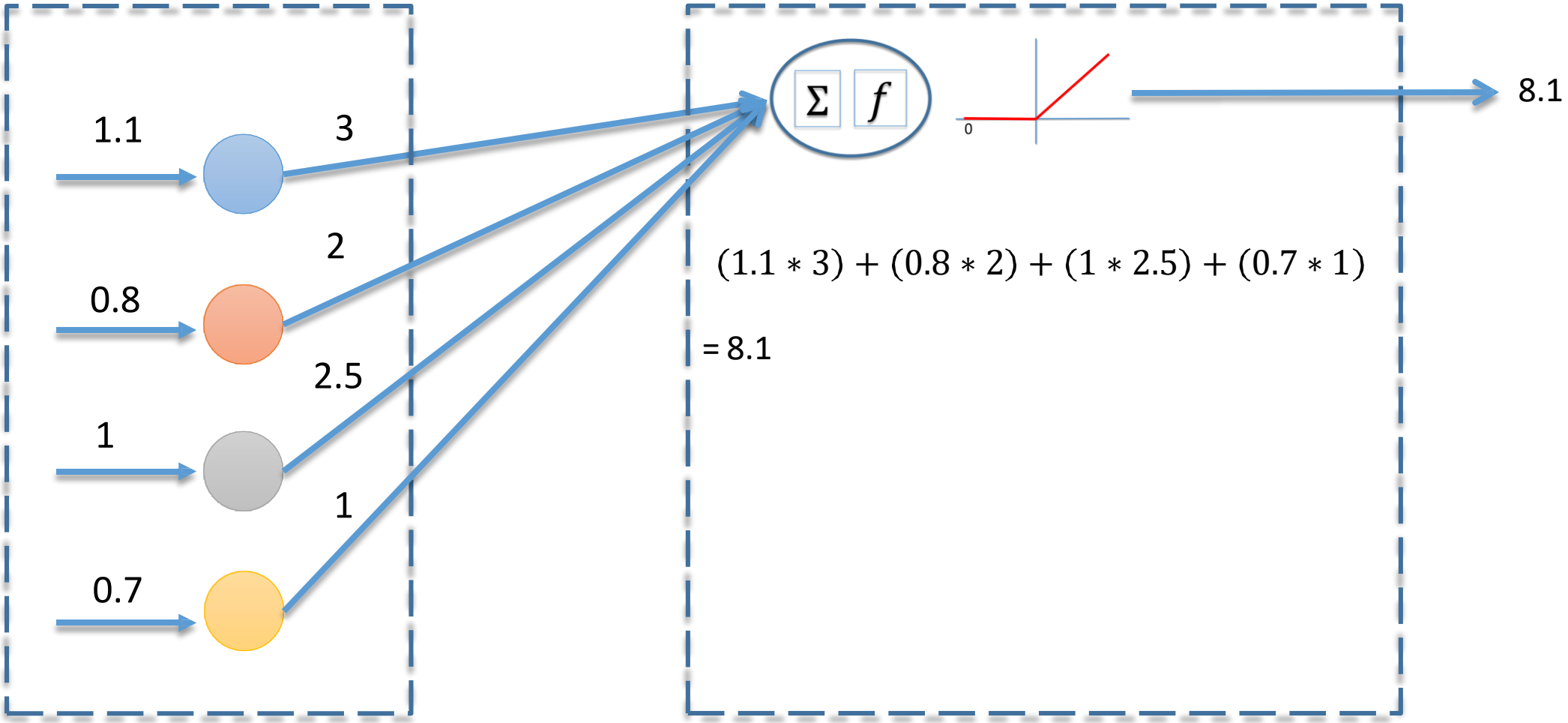
$$\hat{w}_{ave} = \frac{w_{ave}}{1 - \beta_2} \quad \Rightarrow \quad w_{ave(i)} = \beta_2 * w_{a(i-1)} + (1 - \beta_2) * \left(\frac{\partial L}{\partial w}\right)^2$$

Vanishing/Exploding Gradients

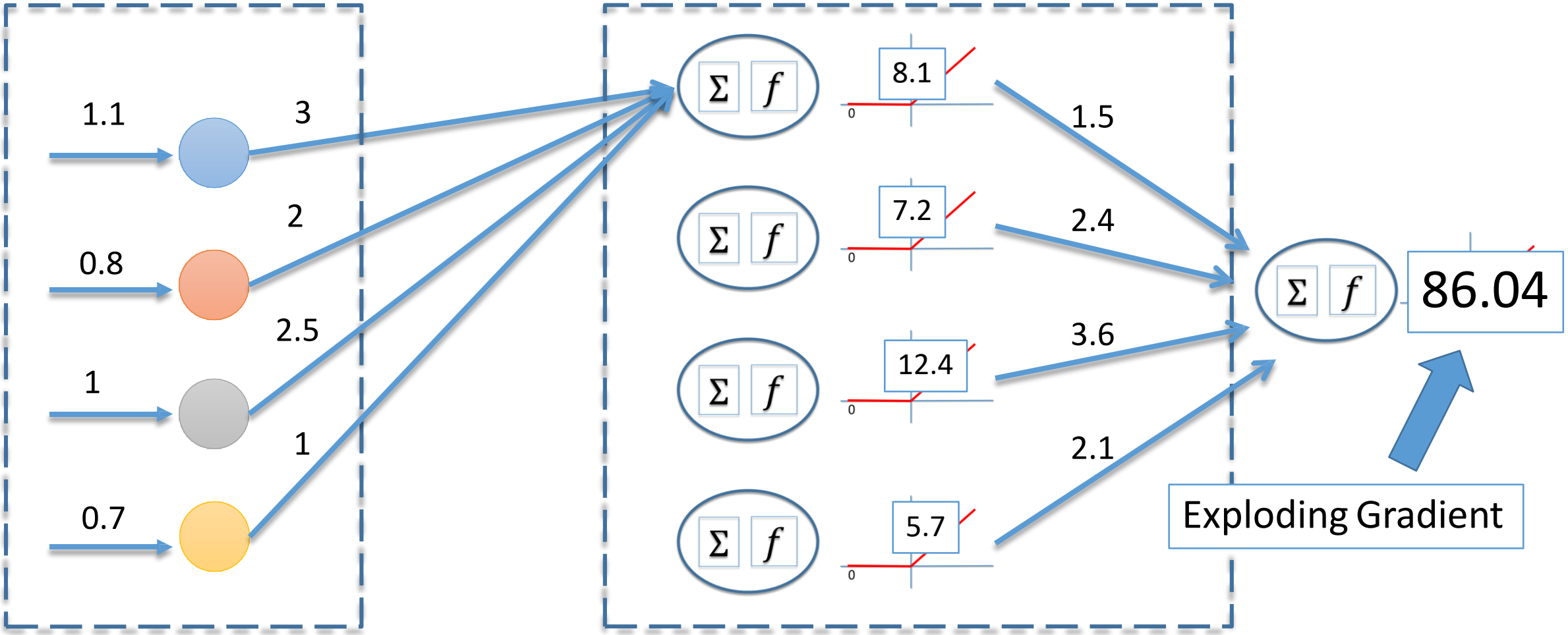
High Weights for all



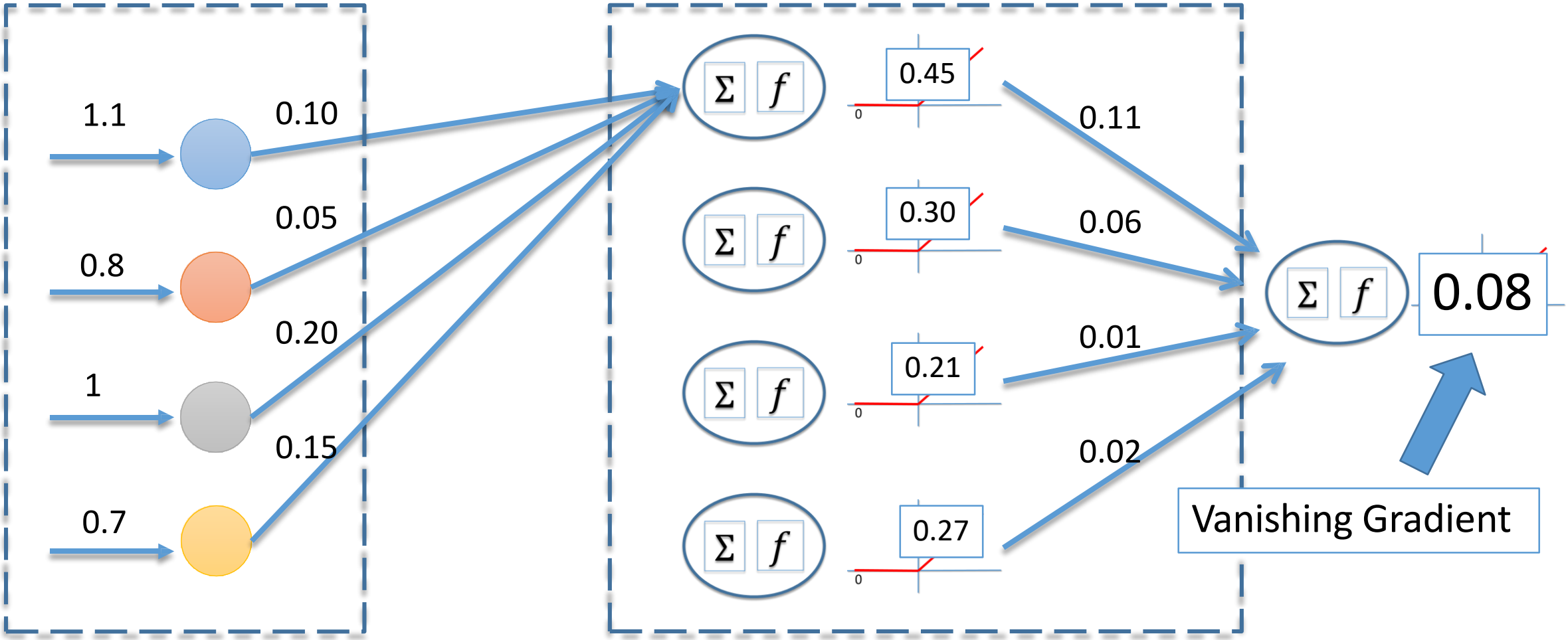
High Weights for all



High Weights for all



Too Low Weights for all



Initializers

Types of Initializers

Constants

- Zeros
- Ones
- Constant

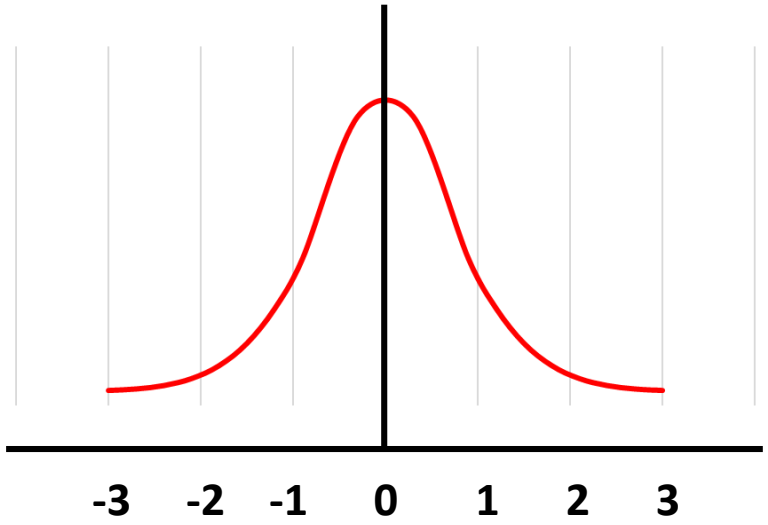
Normal

- RandomNormal
- glorot_normal
- he_normal
- lecun_normal
- TruncatedNormal

Uniform

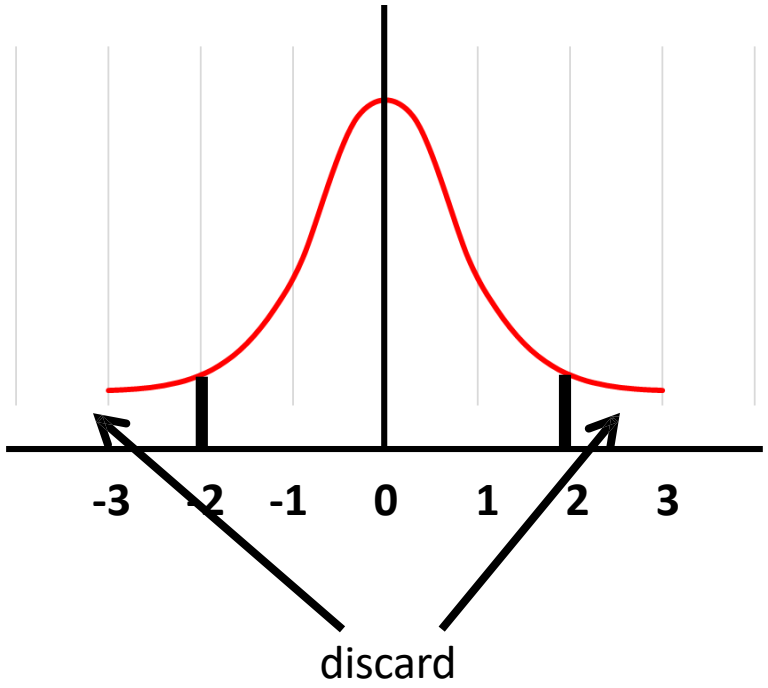
- RandomUniform
- glorot_uniform
- he_uniform
- lecun_uniform

RandomNormal



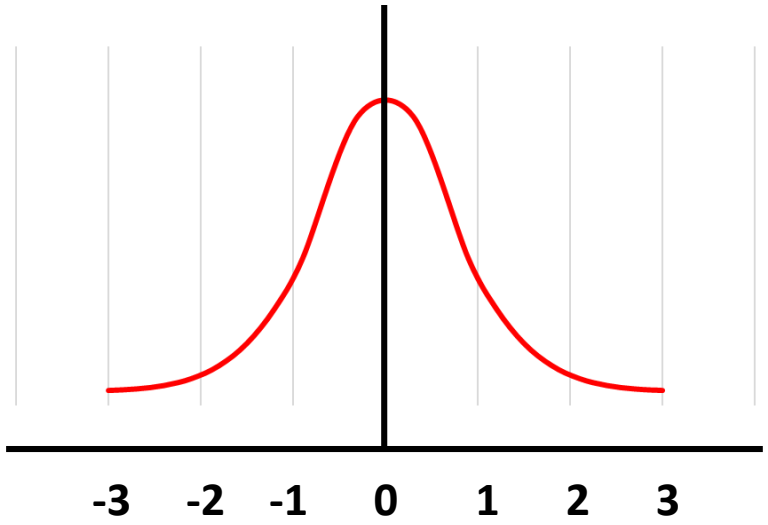
init = *keras.initializers.***RandomNormal**(mean=0.0, stddev=0.05, seed=1234)

TruncatedNormal



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

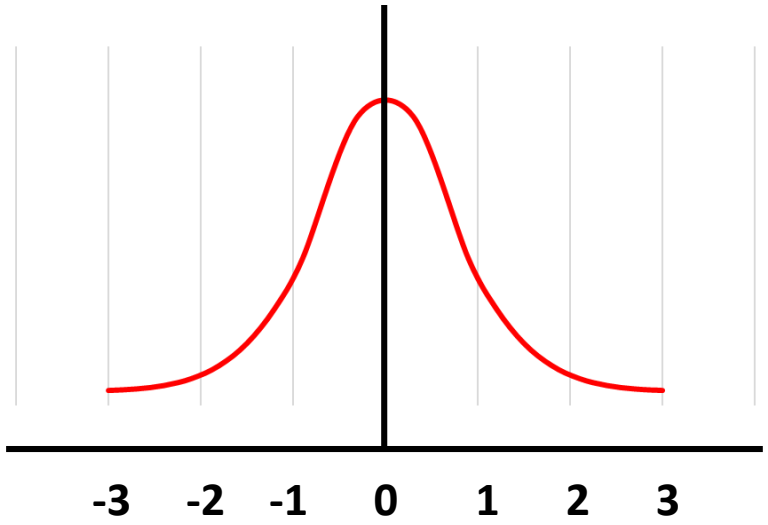
glorot_normal or Xavier Normal Initializer



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

$$\sigma = \sqrt{\frac{2}{fan\_in + fan\_out}}$$

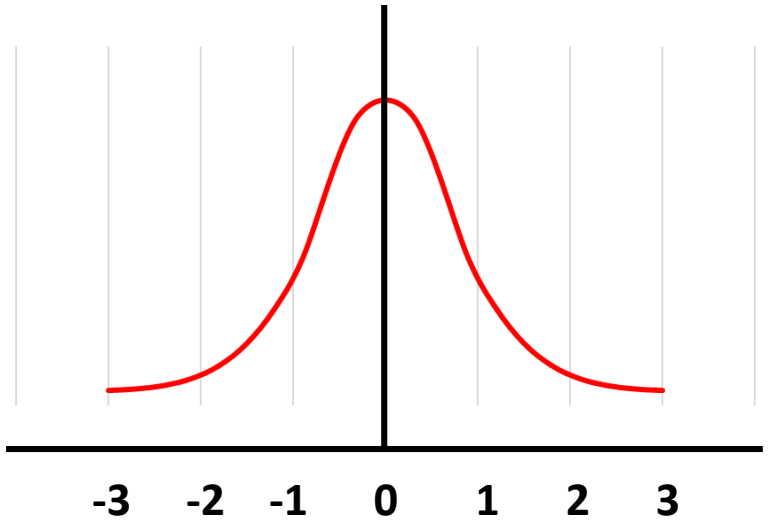
he_normal



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

$$\sigma = \sqrt{\frac{2}{fan\_in}}$$

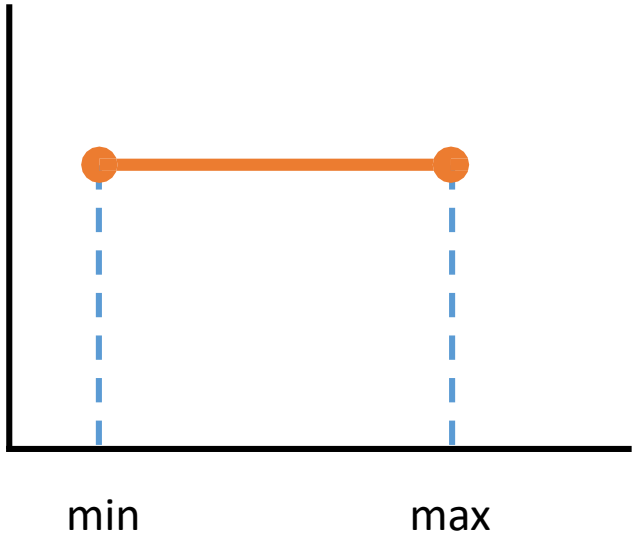
lecun_normal



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

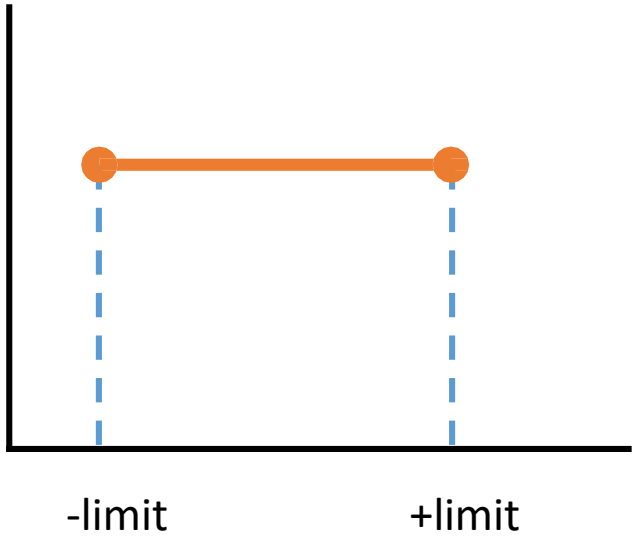
$$\sigma = \sqrt{\frac{1}{fan\_in}}$$

RandomUniform



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

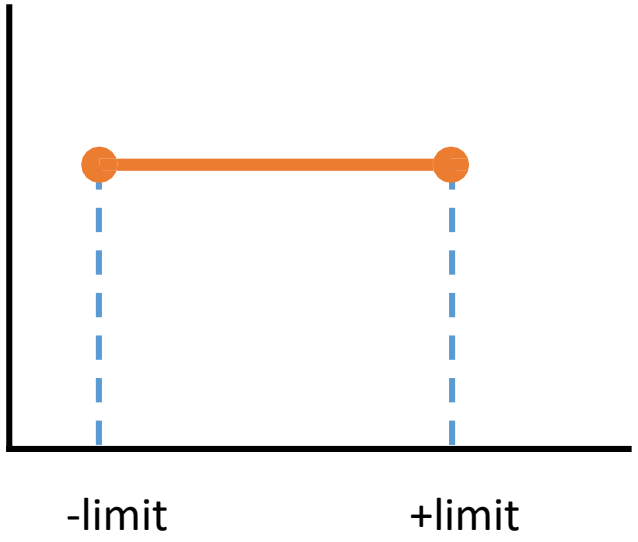
glorot_uniform



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

$$limit = \sqrt{\frac{6}{fan\_in + fan\_out}}$$

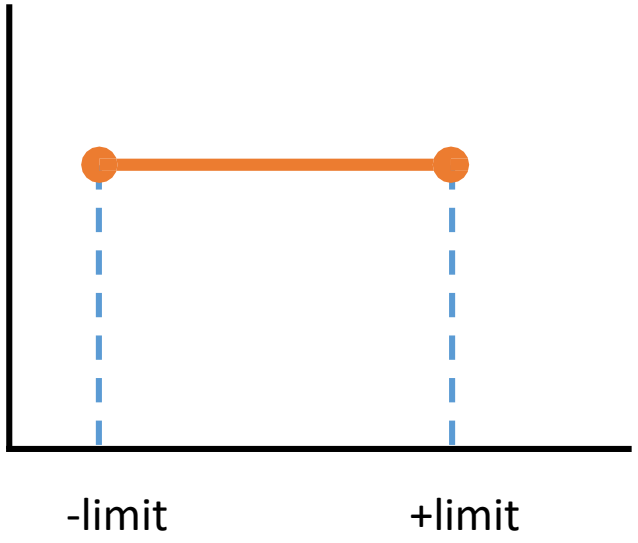
he_uniform



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

$$limit = \sqrt{\frac{6}{fan\_in}}$$

lecun_uniform



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

$$limit = \sqrt{\frac{3}{fan\_in}}$$