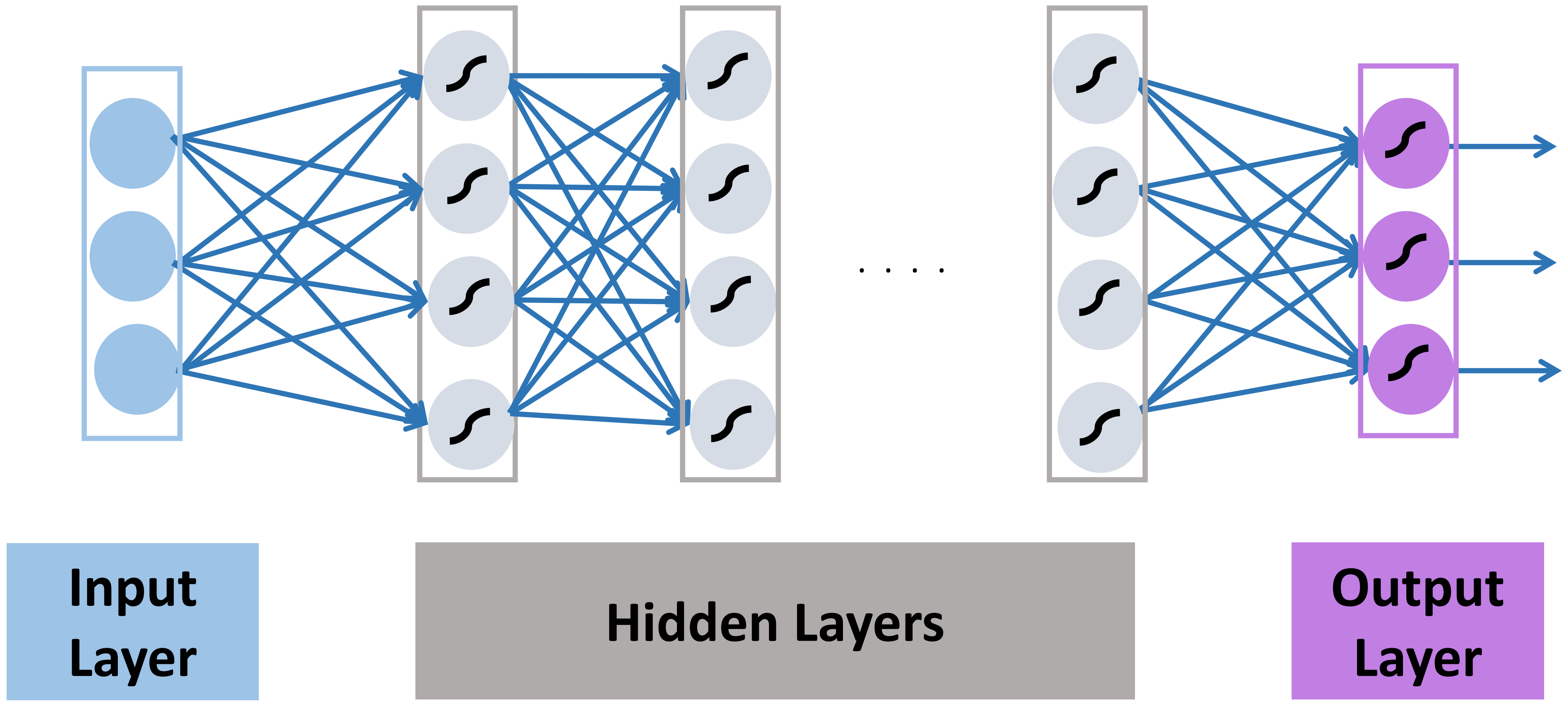
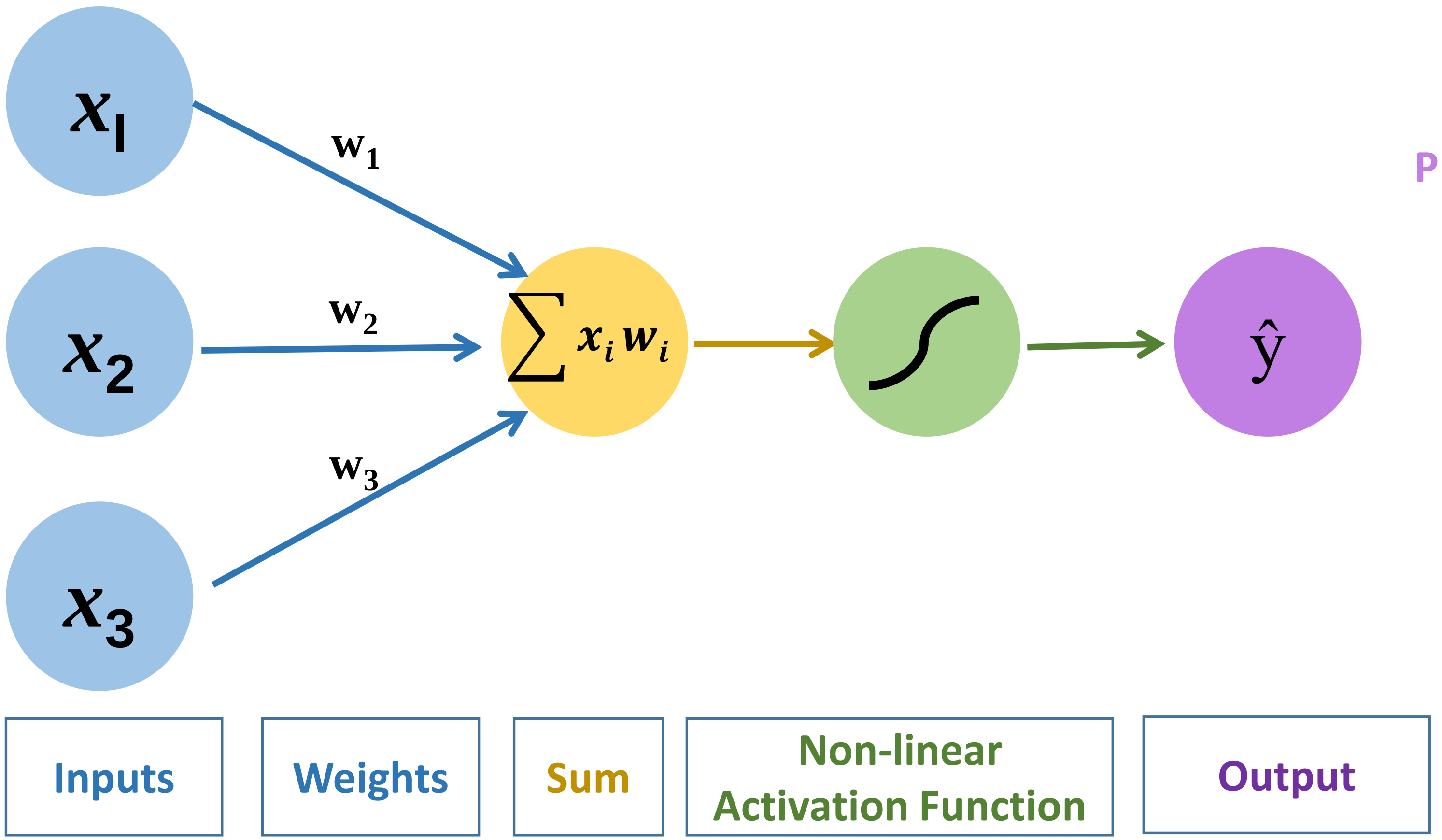
Artificial Neural Networks

Shubhandra Tripathi

Neural Network



Neuron/Perceptron



Predicted value

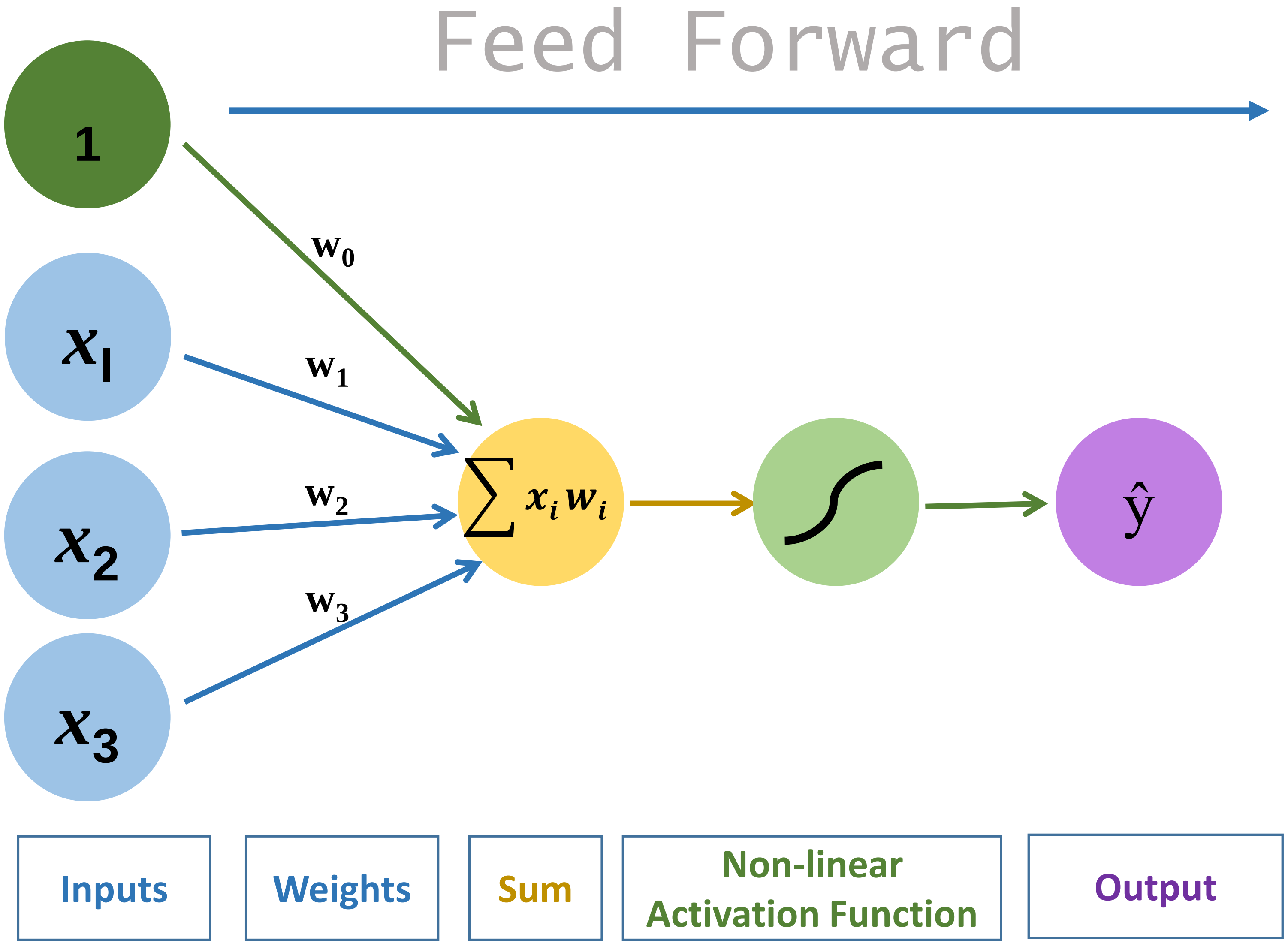
$$\hat{y} = g\left(\sum_{i=1}^{m} x_i w_i\right)$$

Activation Function

Inputs

Weights

Neuron/Perceptron



Predicted value

Activation Function

$$\hat{y} = g\left(\mathbf{w}_0 + \sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{w}_i\right)$$

$$\hat{y} = g\left(\mathbf{w}_0 + \mathbf{X}^T W\right)$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ . \\ . \\ \mathbf{x}_m \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ . \\ . \\ \mathbf{w}_m \end{bmatrix}$$

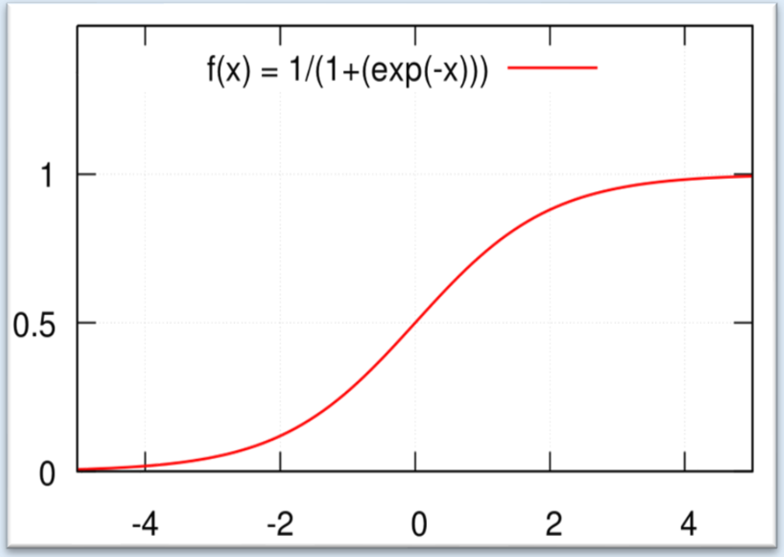
Inputs

Weights

Non-linear Activation Functions

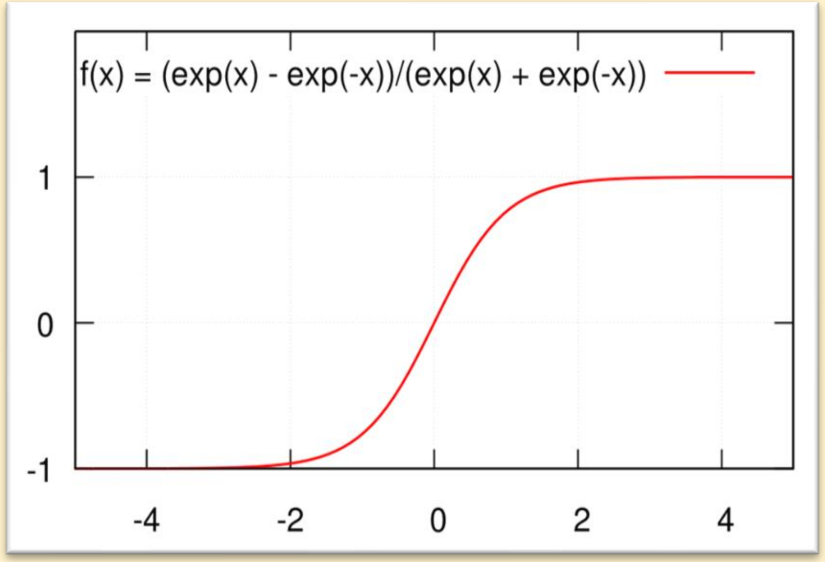
- Activation function introduces non-linearity in the neural network

Sigmoid Function



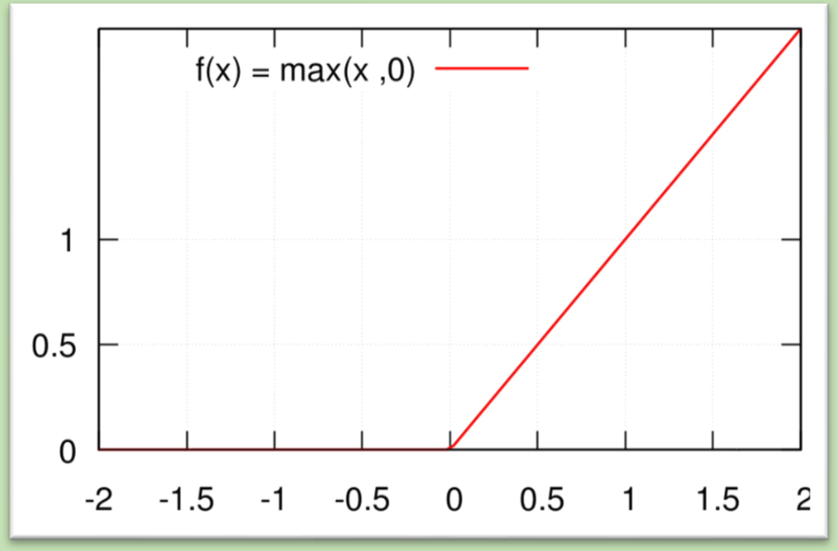
$$g(x) = \frac{1}{1+e^{-x}}$$

Hyperbolic Tangent



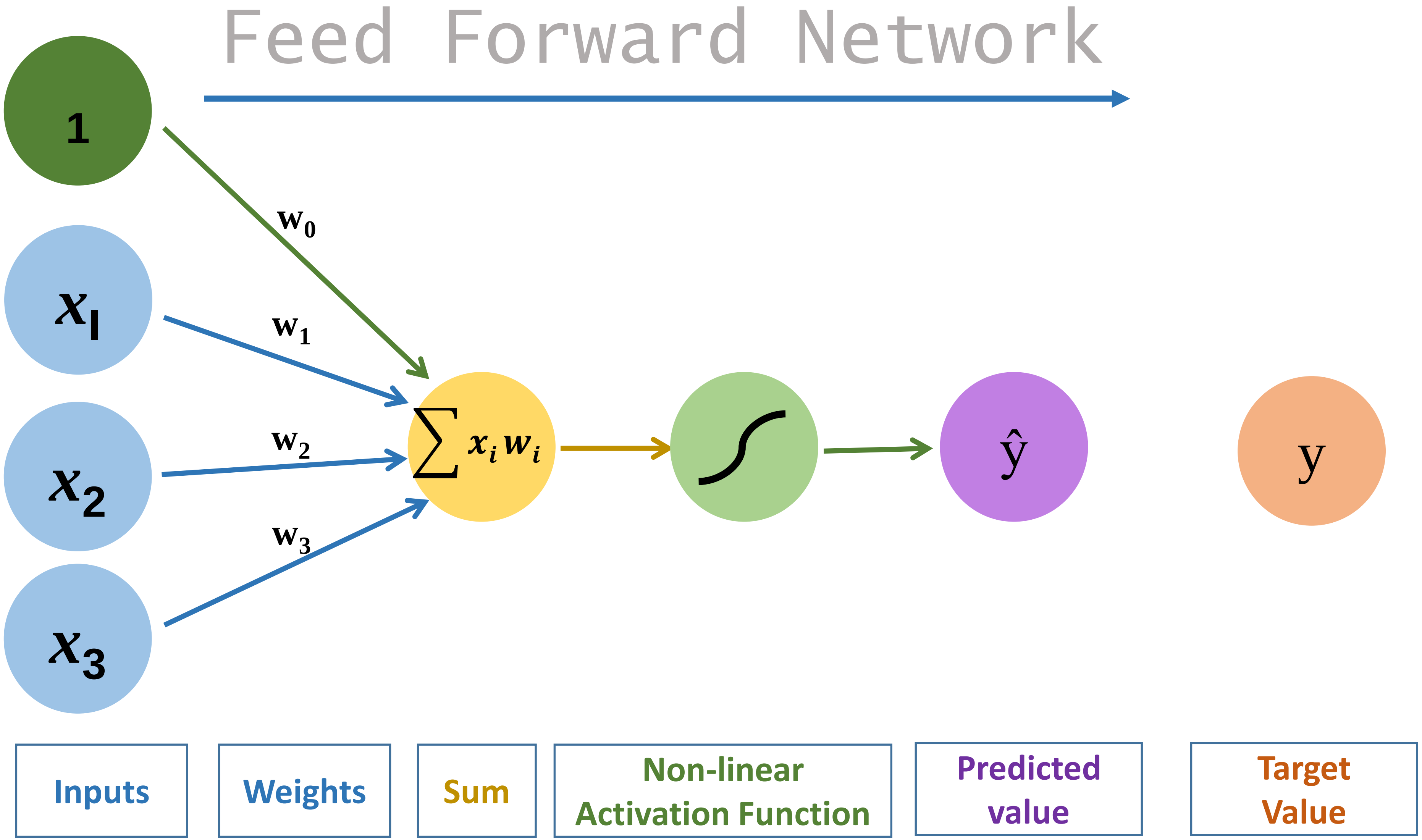
$$g(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

Rectified Linear Unit (ReLU)



$$g(x) = \max(0, x)$$

Neuron/Perceptron

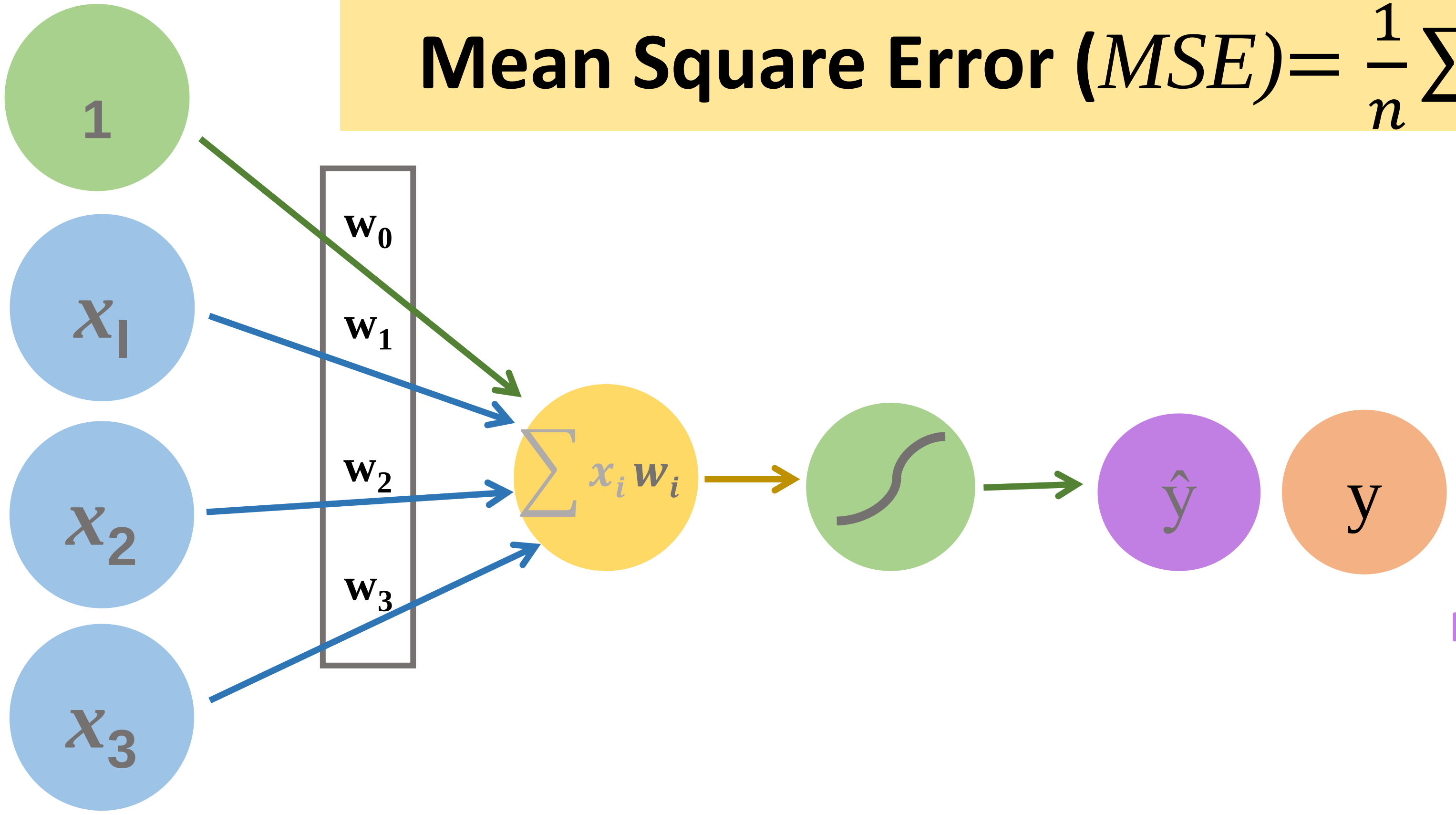


Loss Function

- The **loss function** measures the error incurred due to wrong prediction

$$\textbf{Mean Square Error } (MSE) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^2$$

Target value   Predicted value



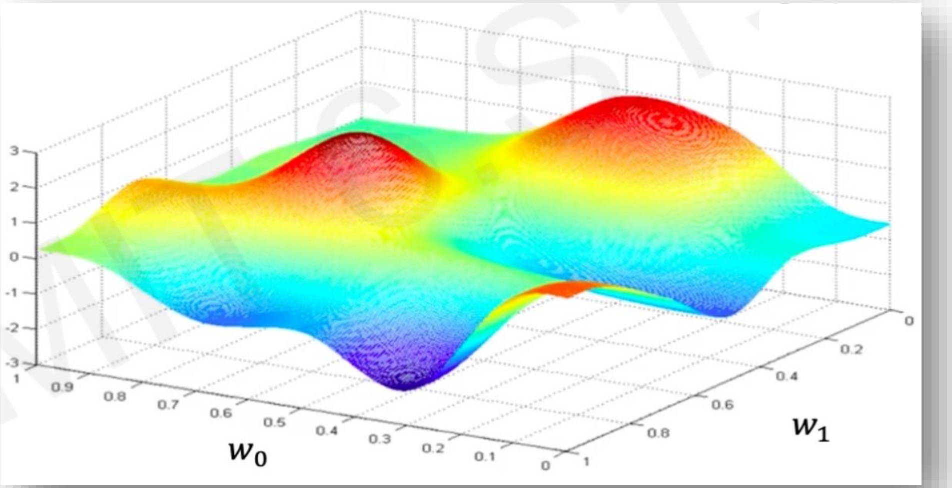
$$\hat{y} = g\,(\mathbf{w}_0 + \textstyle\sum_{i=1}^{m} \boldsymbol{x_i} \boldsymbol{w_i})$$

Predicted value   Activation Function   Inputs   Weights

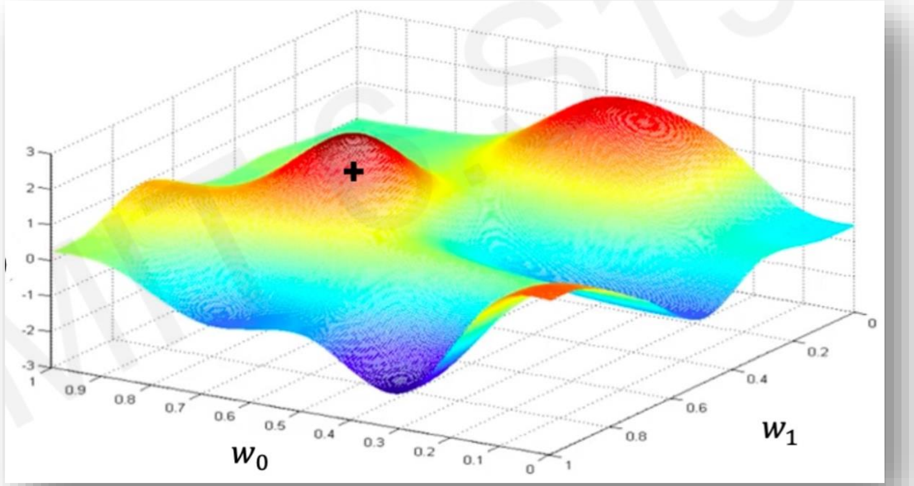
We want to find the **network weights** that can **achieve the minimum loss**

Loss Optimisation

$E(w_0, w_1)$



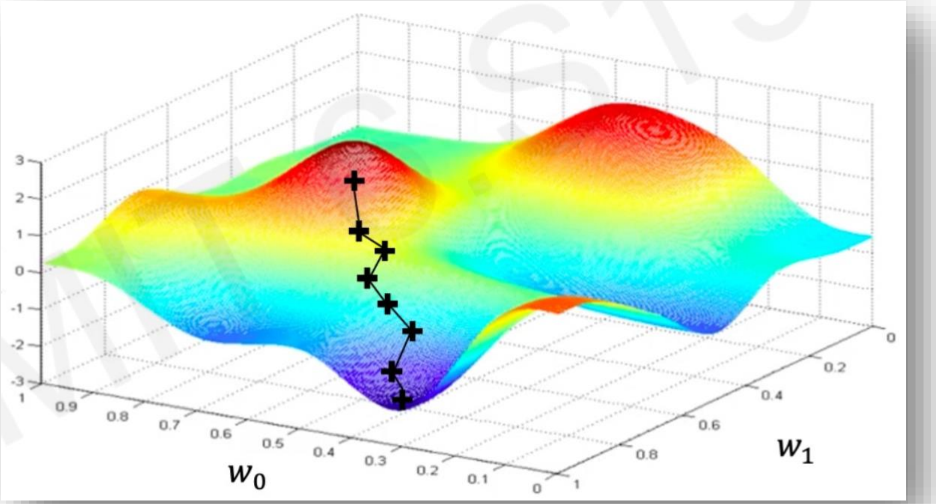
$E(w_0, w_1)$



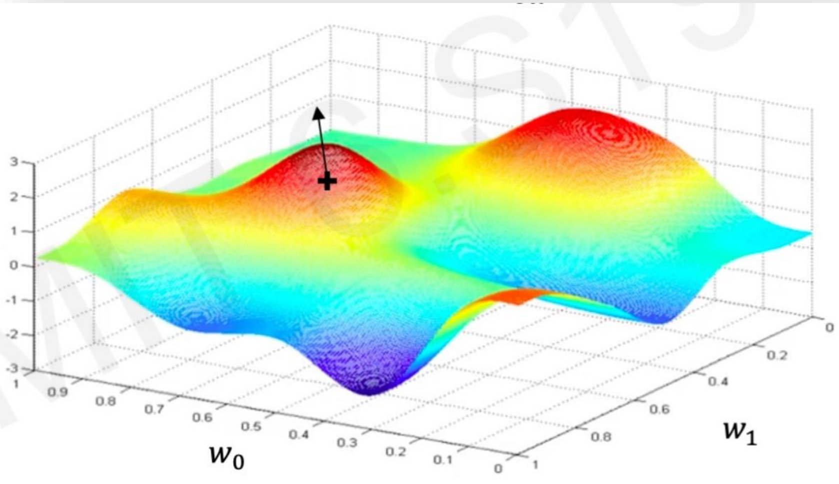
Take small steps in opposite direction of gradient and repeat until converged

Compute gradient, $\partial E(W)/\partial \mathbf{W}$

$E(w_0, w_1)$



$E(w_0, w_1)$



Calculating derivatives

$$\frac{\partial Error}{\partial w} = \frac{\partial Error}{\partial out\_o} * \frac{\partial out\_o}{\partial in\_o} * \frac{\partial in\_o}{\partial w}$$

Applying chain rule

$$\frac{\partial Error}{\partial out\_o} = \frac{\partial}{\partial out\_o}\left(\frac{1}{2} * (target - output)^2\right)$$

$$\frac{\partial Error}{\partial out\_o} = \left(\frac{1}{2} * 2 * (target - output)\right)\frac{\partial}{\partial out\_o}(target - output)$$

$$\frac{\partial Error}{\partial out\_o} = (target - output)\frac{\partial}{\partial out\_o}(-1)$$

$$\boldsymbol{\frac{\partial Error}{\partial out\_o} = (output - target)}$$

https://pub.towardsai.net/building-neural-networks-from-scratch-with-python-code-and-math-in-detail-i-536fae5d7bbf

Calculating derivatives

$$\frac{\partial out\_o}{\partial in\_o} = out\_o * (1 - out\_o)$$

$$out\_o = \left(\frac{1}{1+e^{-in\_o}}\right)$$ eq 1.

$$\frac{out\_o}{in\_o} = \frac{\partial}{\partial in\_o} * \left(\frac{1}{1+e^{-in\_o}}\right)$$

Value of out_o

$$\frac{out\_o}{in\_o} = \frac{\partial}{\partial in\_o} * \left(1+e^{-in\_o}\right)^{-1}$$

Applying chain rule along with power rule

$$\frac{out\_o}{in\_o} = -1\left(1+e^{-in\_o}\right)^{-2} * \frac{\partial}{\partial in\_o} * \left(1+e^{-in\_o}\right)$$

$$\frac{out\_o}{in\_o} = -1\left(1+e^{-in\_o}\right)^{-2} * \left(\frac{\partial}{\partial in\_o} + \frac{\partial}{\partial in\_o} * \left(e^{-in\_o}\right)\right)$$

$$\frac{out\_o}{in\_o} = -1\left(1+e^{-in\_o}\right)^{-2} * \left(0 + \frac{\partial}{\partial in\_o} * \left(e^{-in\_o}\right)\right)$$

$$\frac{out\_o}{in\_o} = -1\left(1+e^{-in\_o}\right)^{-2} * \left(e^{-in\_o} * \frac{\partial}{\partial in\_o}[-in\_o]\right)$$

$$\frac{out\_o}{in\_o} = -1\left(1+e^{-in\_o}\right)^{-2} * \left(e^{-in\_o} * -1\right)$$

$$\frac{out\_o}{in\_o} = \left(1+e^{-in\_o}\right)^{-2} * \left(e^{-in\_o}\right)$$

$$\frac{out\_o}{in\_o} = \frac{\left(e^{-in\_o}\right)}{\left(1+e^{-in\_o}\right)^2}$$

$$\frac{out\_o}{in\_o} = \frac{1 * \left(e^{-in\_o}\right)}{\left(1+e^{-in\_o}\right)\left(1+e^{-in\_o}\right)}$$

$$\frac{out\_o}{in\_o} = \frac{1 *}{\left(1+e^{-in\_o}\right)} * \frac{\left(e^{-in\_o}\right) + 1 - 1}{\left(1+e^{-in\_o}\right)}$$

$$\frac{out\_o}{in\_o} = \frac{1 *}{\left(1+e^{-in\_o}\right)} * \left(\frac{\left(1+e^{-in_o}\right)}{\left(1+e^{-in_o}\right)} - \frac{1}{\left(1+e^{-in_o}\right)}\right)$$

$$\frac{out\_o}{in\_o} = \frac{1 *}{\left(1+e^{-in\_o}\right)} * \left(1 - \frac{1}{\left(1+e^{-in_o}\right)}\right)$$

From equation 1

$$\frac{\partial out\_o}{\partial in\_o} = out\_o * (1 - out\_o)$$

Calculating derivatives

$$\frac{\partial Error}{\partial w} = \frac{\partial Error}{\partial out\_o} * \frac{\partial out\_o}{\partial in\_o} * \frac{\partial in\_o}{\partial w}$$

$$\frac{\partial in\_o}{\partial w} = \text{Input values}$$

Value of in_o

in_o = $\boldsymbol{w_1 x_1} + \boldsymbol{w_2 x_2} + \boldsymbol{w_3 x_3}$

All the values except w2 will be considered constant

$$\frac{\partial in\_o}{\partial w2} = 0 + \boldsymbol{x_2} + 0$$

$$\frac{\partial in\_o}{\partial w2} = \boldsymbol{x_2}$$

https://pub.towardsai.net/building-neural-networks-from-scratch-with-python-code-and-math-in-detail-i-536fae5d7bbf

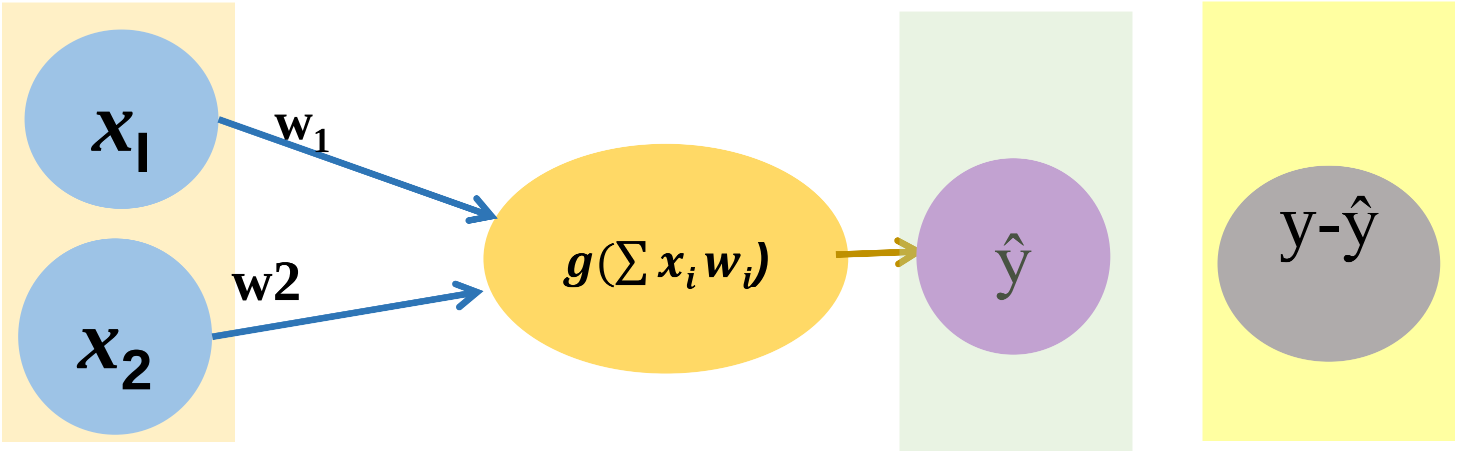
Calculating derivatives

$$\frac{\partial Error}{\partial w} = \frac{\partial Error}{\partial out\_o} * \frac{\partial out\_o}{\partial in\_o} * \frac{\partial in\_o}{\partial w}$$

$\frac{\partial in\_o}{\partial w}$= Input values

$$\frac{\partial out\_o}{\partial in\_o} = out\_o * (1 - out\_o)$$

$$\frac{\partial Error}{\partial out\_o} = (output - target)$$

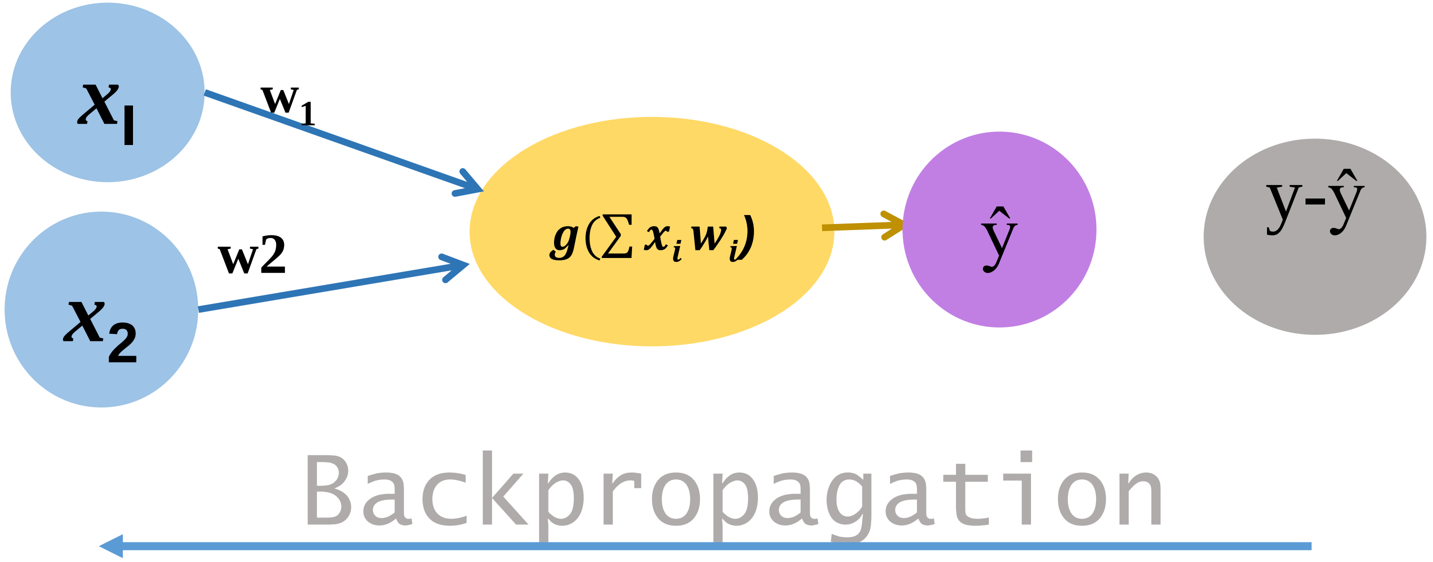


https://pub.towardsai.net/building-neural-networks-from-scratch-with-python-code-and-math-in-detail-i-536fae5d7bbf

Update Weights

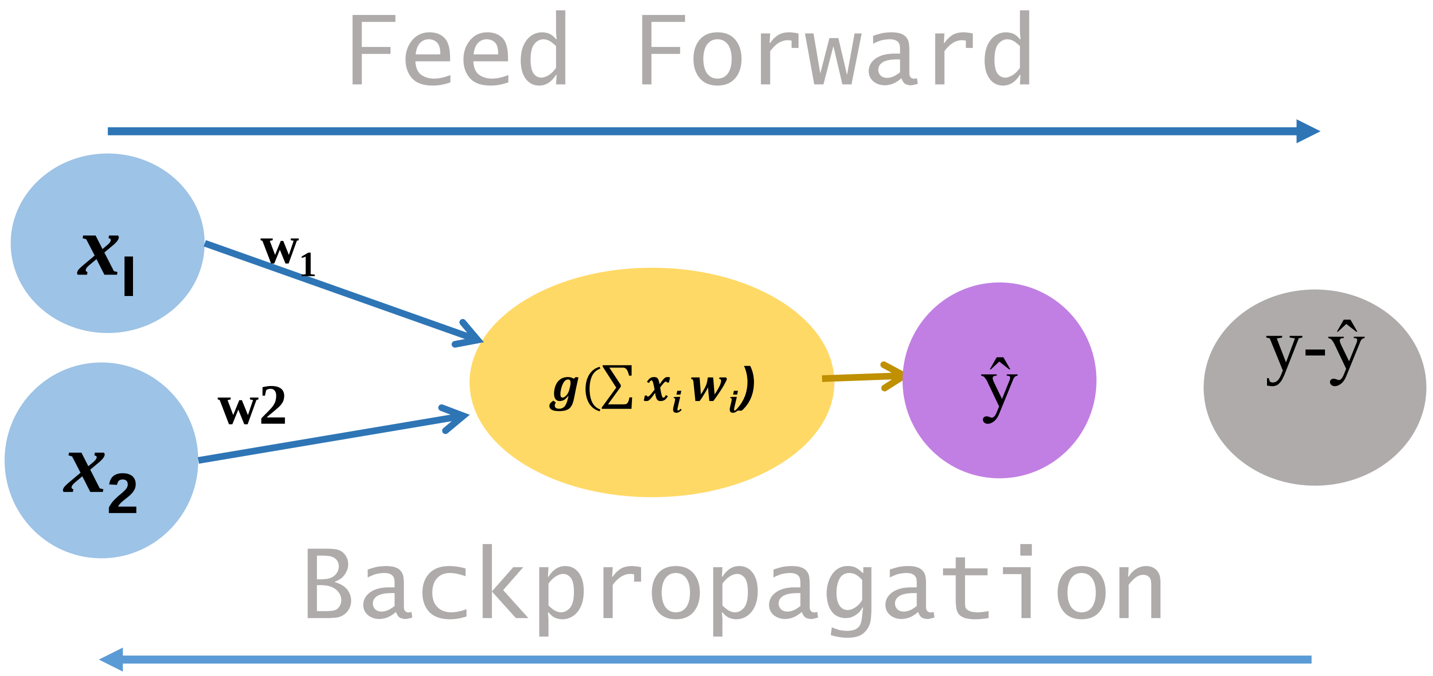
$$\mathbf{W}_{update} = W - \delta\left(\frac{\partial Error}{\partial W}\right)$$

Learning rate



https://pub.towardsai.net/building-neural-networks-from-scratch-with-python-code-and-math-in-detail-i-536fae5d7bbf

$$\hat{y} = g\left(\mathbf{w}_0 + \sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{w}_i\right)$$



$$\mathbf{W}_{update} = \boldsymbol{W} - \delta\left(\frac{\partial Error}{\partial W}\right)$$

https://pub.towardsai.net/building-neural-networks-from-scratch-with-python-code-and-math-in-detail-i-536fae5d7bbf

Algorithm

1. Initialize weights randomly

2. Loop until convergence:

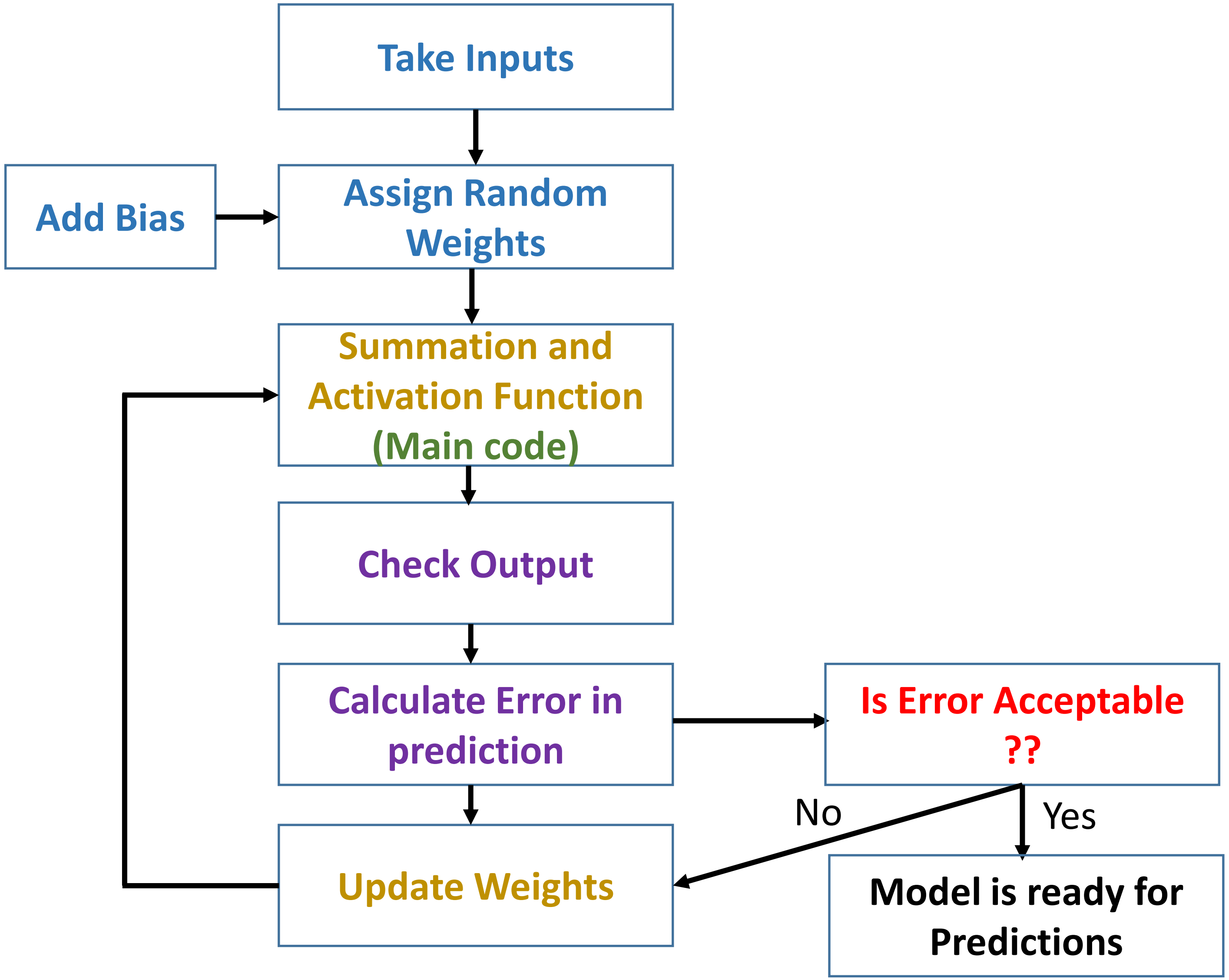
3. Compute gradient, $\frac{\partial Error}{\partial w}$

4. Update weights, $\mathbf{W}_{update} = \boldsymbol{W} - \boldsymbol{\delta}(\frac{\partial Error}{\partial W})$

5. Return optimized weights

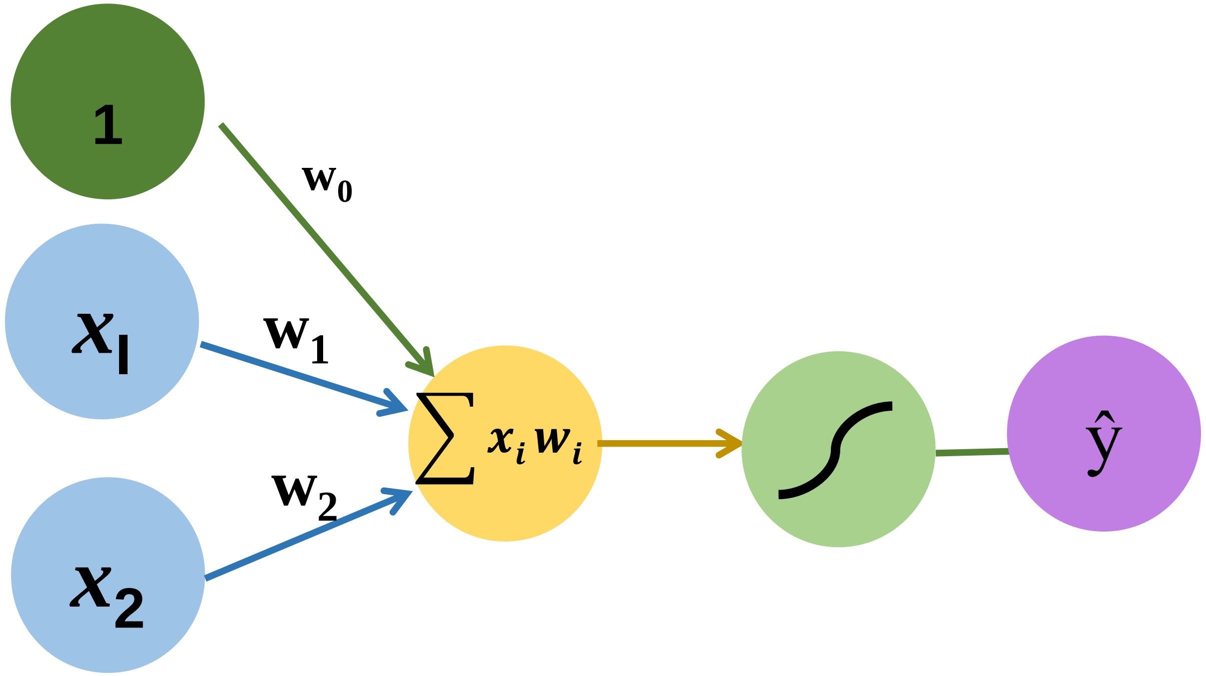
https://pub.towardsai.net/building-neural-networks-from-scratch-with-python-code-and-math-in-detail-i-536fae5d7bbf

Flow Chart


Take Inputs
Add Bias
Assign Random Weights
Summation and Activation Function
(Main code)
Check Output
Calculate Error in prediction
Is Error Acceptable ??
No
Yes
Update Weights
Model is ready for Predictions

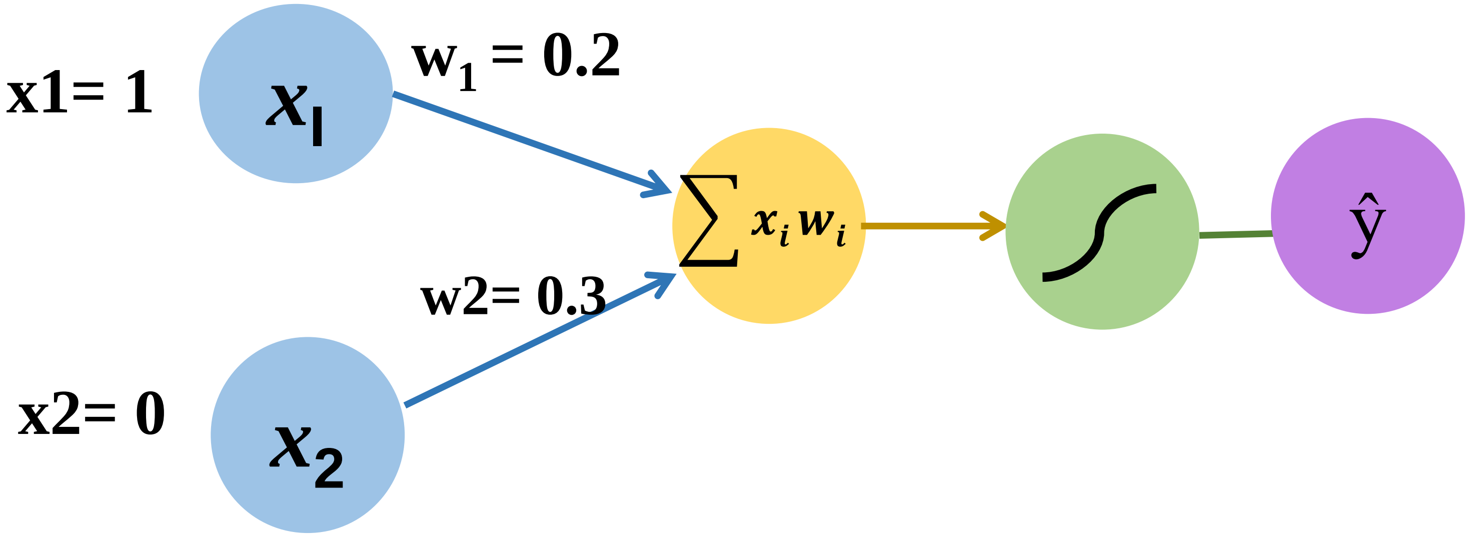
Example1: OR GATE

Input 1	Input2	Output
1	1	1
1	0	1
0	1	1
0	0	0



4X2 2X1 4X1

Example1: OR GATE



Calculating Input Manually

Input = w1*x1 + w2*x2
= 0.2*1 +0.3 *0
= 0.2+0
= 0.2

Calculating Output Manually

$$g(x) = \frac{1}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-0.2}}$$

$$= \frac{1}{1+0.81873}$$

$$= \frac{1}{1.81873}$$

= 0.54983

Calculating Error

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

= $(1- 0.54983)^2$

= $(0.45016)^2$

= 0.20264

Example1: OR GATE

$$\frac{\partial Error}{\partial w} = \frac{\partial Error}{\partial out\_o} * \frac{\partial out\_o}{\partial in\_o} * \frac{\partial in\_o}{\partial w}$$

$$\boldsymbol{\frac{\partial Error}{\partial out\_o} = (output \ - target)}$$

$$\boldsymbol{\frac{\partial out\_o}{\partial in\_o} = out\_o \ * (\ 1 - \ out\_o)}$$

$$\boldsymbol{\frac{\partial in\_o}{\partial w}} = \text{Input values}$$

Calculating Derivative

$$\frac{\partial Error}{\partial w1} = \frac{\partial Error}{\partial out\_o} * \frac{\partial out\_o}{\partial in\_o} * \frac{\partial in\_o}{\partial w}$$

= (0.54983 -1)*

(0.54983 *(1- 0.54983))*

1

=-0.11142

Calculating Derivative

$$\frac{\partial Error}{\partial w2} = \frac{\partial Error}{\partial out\_o} * \frac{\partial out\_o}{\partial in\_o} * \frac{\partial in\_o}{\partial w}$$

= (0.54983 -1)*

(0.54983 *(1- 0.54983))*

0

=0

Example1: OR GATE

$$\mathbf{W}_{update} = W - \delta(\frac{\partial Error}{\partial W})$$

Modified Weights

$\mathbf{W1}_{update}$ = W1– (0.05*-0.11142)

= 0.2+0.00557

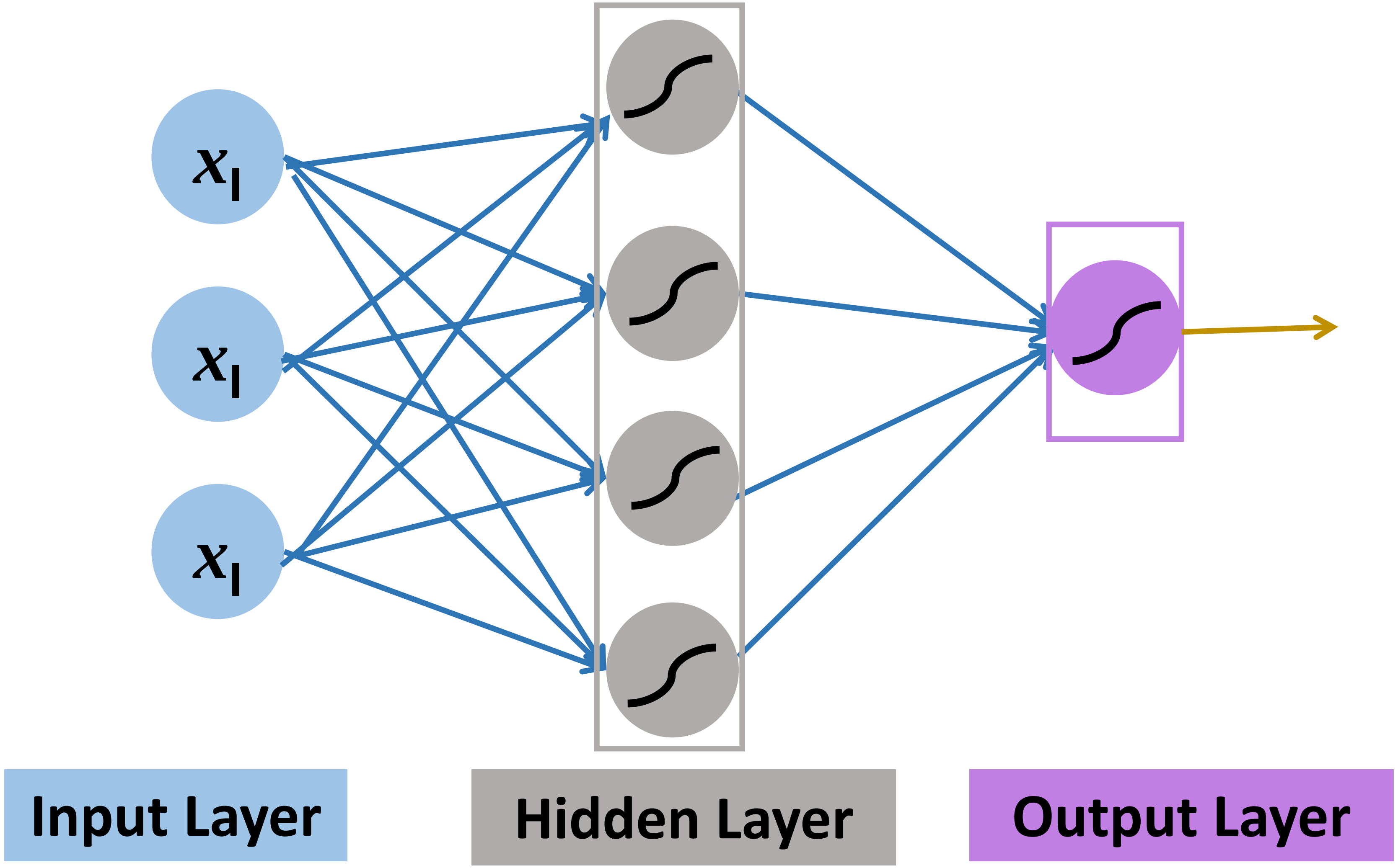
= 0.20557

Learning rate = 0.05

Example2:

Input 1	Input2	Input 3	Output
3	6	8	3
7	4	1	7
4	2	2	4
3	5	8	3
7	4	3	7
.	.	.	.
.	.	.	.
6	8	9	6

Example2: Neural Network



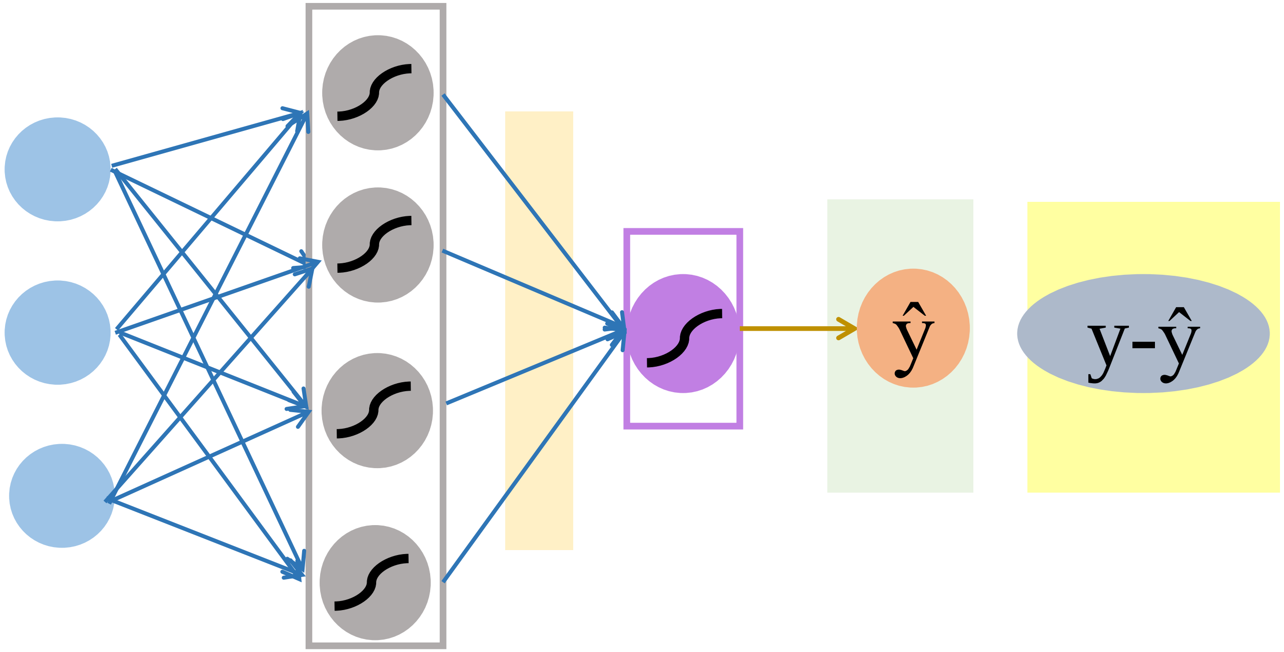
Example2:Gradient for Output Layer

$$\frac{\partial Error}{\partial wo} = \frac{\partial Error}{\partial out\_o} * \frac{\partial out\_o}{\partial in\_o} * \frac{\partial in\_o}{\partial wo}$$

$\frac{\partial in\_o}{\partial w}$ = Input values

$$\frac{\partial out\_o}{\partial in\_o} = out\_o * (1 - out\_o)$$

$$\frac{\partial Error}{\partial out\_o} = (output - target)$$



Example2:Gradient for Hidden Layer

$$\frac{\partial Error}{\partial out\_h} = \frac{\partial Error}{\partial ino\_o} * \frac{\partial in\_o}{\partial out\_h}$$

$$\frac{\partial Error}{\partial wo} = \frac{\partial Error}{\partial out\_o} * \frac{\partial out\_o}{\partial in\_o} * \frac{\partial in\_o}{\partial wo}$$

$$\frac{\partial Error}{\partial in\_o} = \frac{\partial Error}{\partial out\_o} * \frac{\partial out\_o}{\partial in\_o}$$

$$\frac{\partial Error}{\partial wh} = \frac{\partial Error}{\partial out\_h} * \frac{\partial out\_h}{\partial in\_h} * \frac{\partial in\_h}{\partial wh}$$

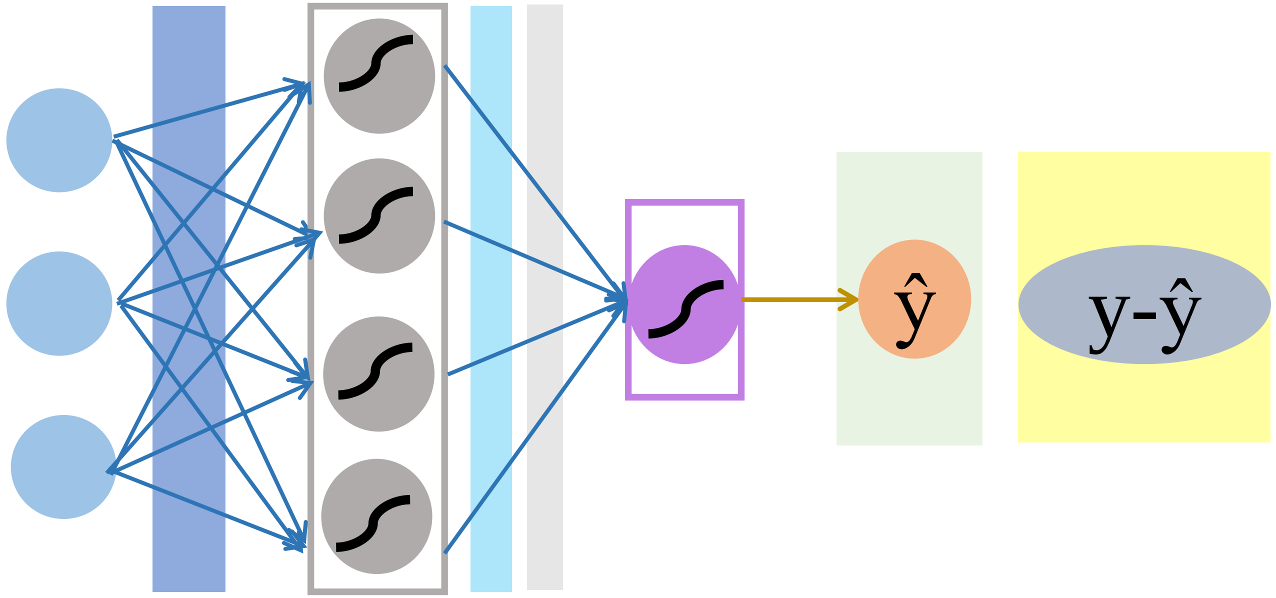
$$\boldsymbol{\frac{\partial Error}{\partial out\_o} = (output - target)}$$

$$\frac{\partial out\_h}{\partial in\_h} = out\_h*(1 - out\_h)$$

$$\frac{\partial in\_h}{\partial wh} = input\ value$$

$$\boldsymbol{\frac{\partial out\_o}{\partial in\_o} = out\_o * (1 - out\_o)}$$

$$\frac{\partial in\_o}{\partial out\_h} = output\ weighs = wo$$



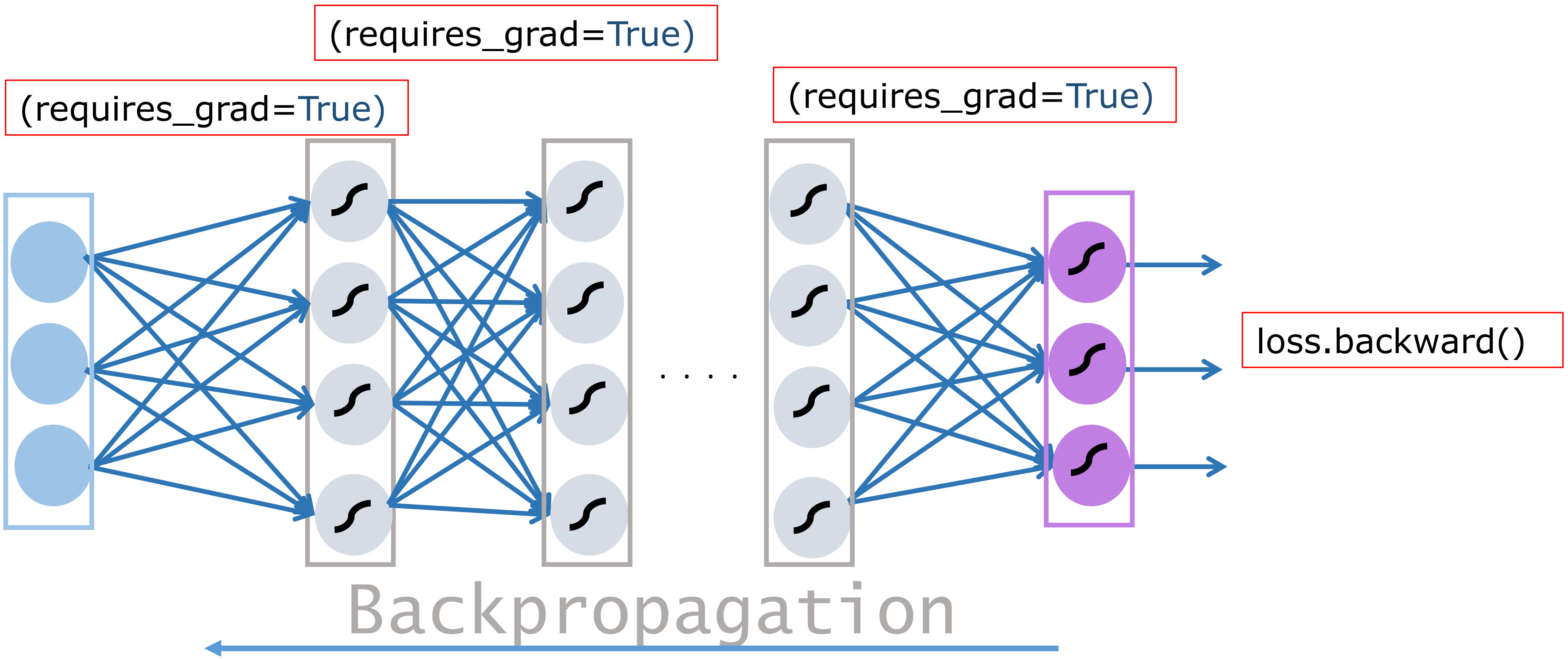
Gradient calculation

- Calculating gradient for Neural Network with multiple hidden layers is a tedious task

- Calculates gradient automatically

Gradient calculation

PyTorch



Thank you