Deep Learning
Introduction to Neural Network.

Siman Giri

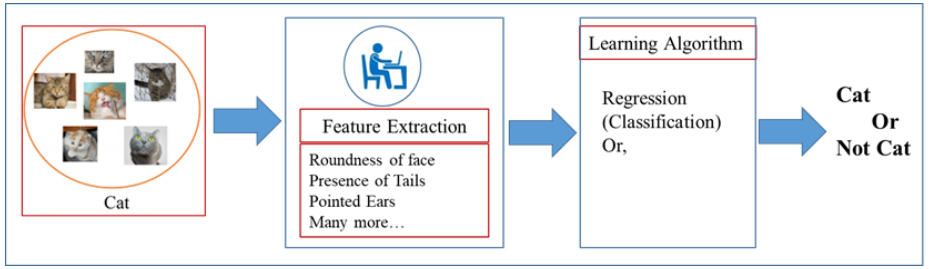
Deep Learning

Some Motivations.

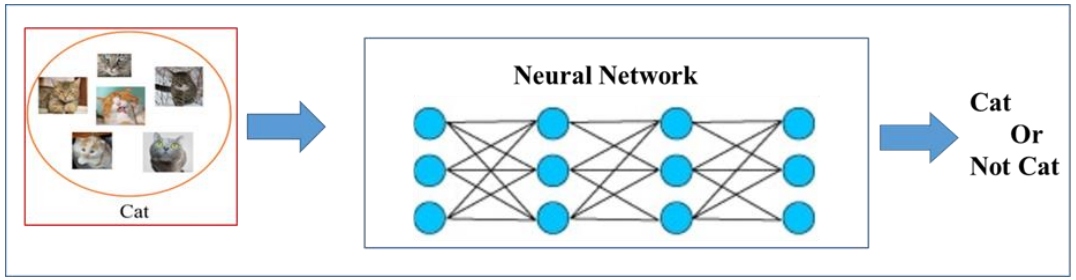
Machine Learning → Deep Learning.

Machine Learning

- One crucial aspect of machine learning approaches to solving problems is that human and often undervalued engineering plays an important role.
  - A human still has to frame the problem,
    - acquire and organize data, design a space of possible solutions,
    - select a learning algorithm and its parameters,
    - apply the algorithm to the data,
    - validate the resulting solution to decide whether it's good enough to use, etc.
- These steps are of great importance.

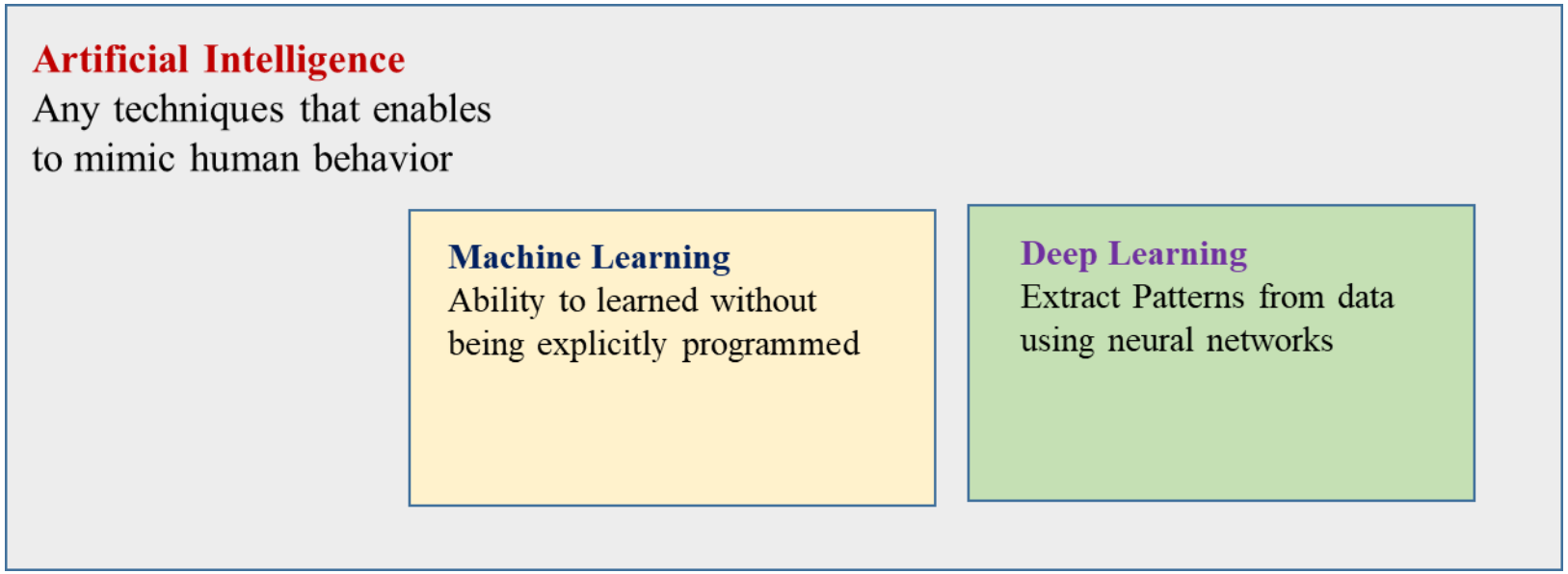


Deep Learning



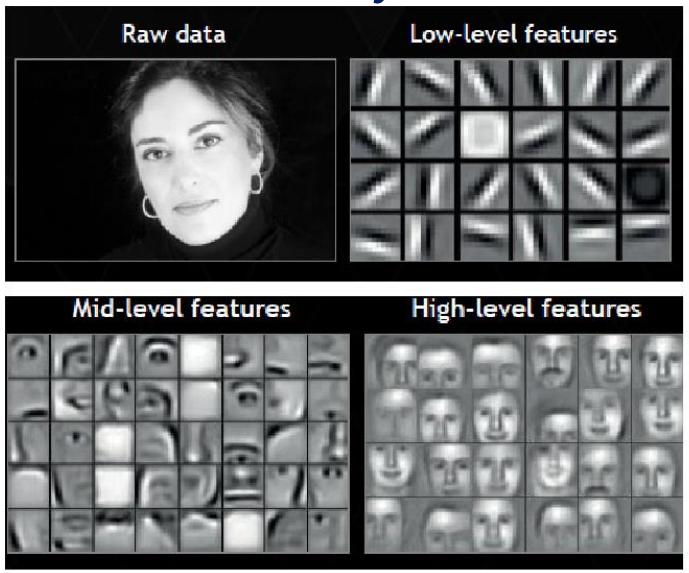
Machine Learning → Deep Learning.

- **What is Deep Learning?**
  - Teaching computers **how to learn a task** directly **from raw data**.



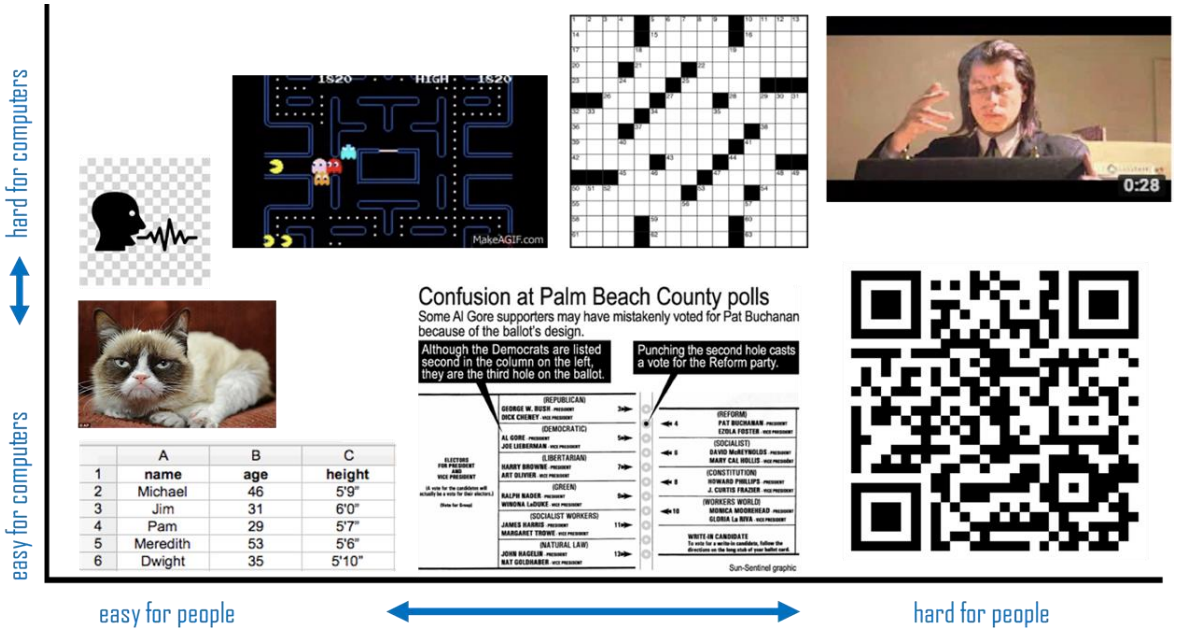
Machine Learning → Deep Learning.

- **Why Deep Learning?**
  - Extracting a features manually does not seems a good idea?
  - Hand engineering a feature is time consuming, confusing and many time inapplicable.
- **Can we learn the underlying features directly from data?**



Machine Learning → Deep Learning.

- Dataset: Formats

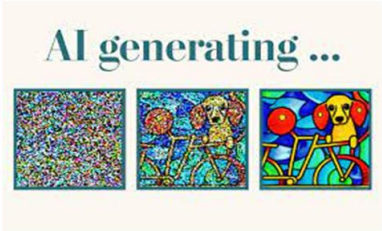


Deep Learnings are Taking Over!!!

- Deep learning have become one of the main approaches to AI
- They have been successfully applied to various pattern recognition, prediction, and analysis problems
- In many problems they have established the state of the art
  - Often exceeding previous benchmarks by large margins
  - Sometimes solving problems you couldn't solve using earlier ML methods
-

Some Modern Breakthroughs Of Deep Learning.

Text to image generator:



Large Language Models:

Alpha Folds:

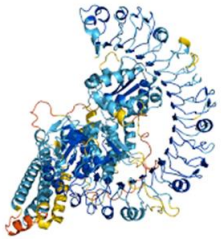
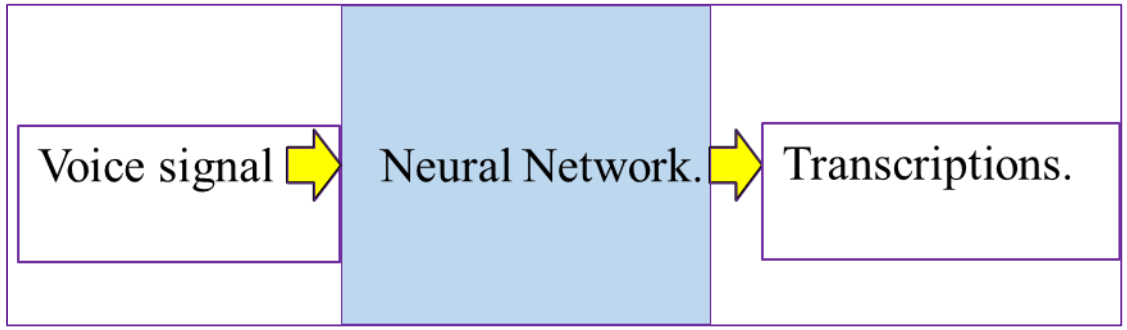


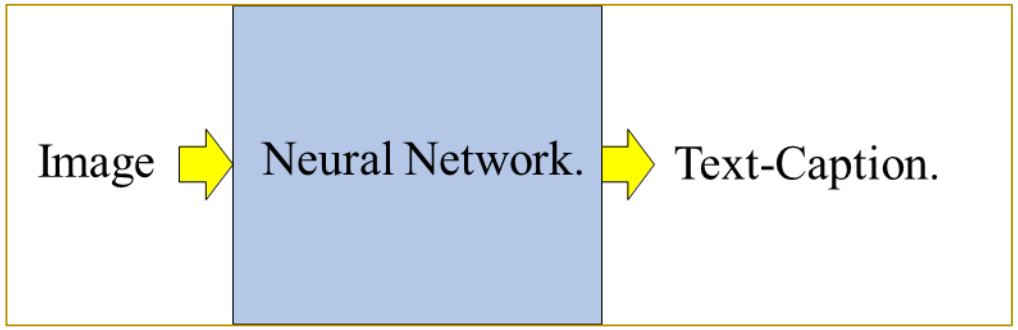
Image and Computer Vision Application

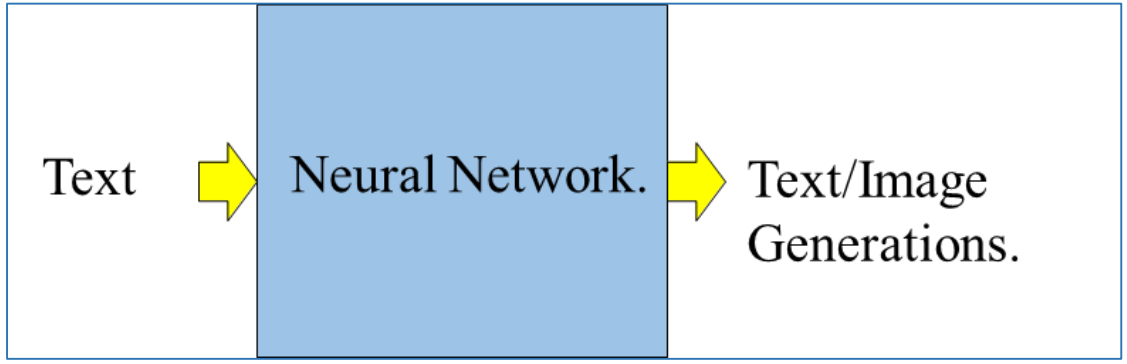
Text and Audio Signal Analysis.

Translations.

Common Things in above Breakthroughs.







What are Neural Networks?

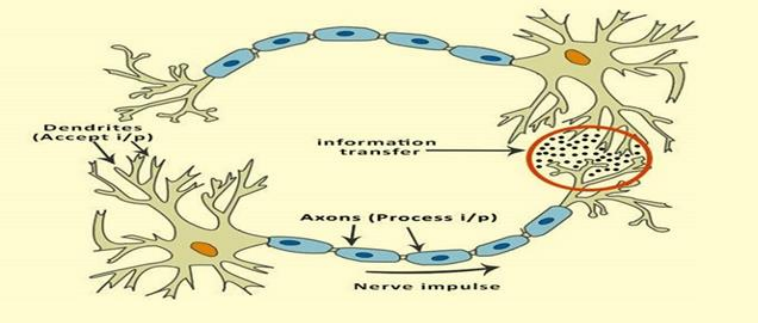
Inspired by Human ability of thinking?
Cognitive and Learning Abilities of Human Brain.

1. Neurons → ML →Neural Networks.

From Biological Inspiration to Computational Models.

1.1 Biological Neurons.

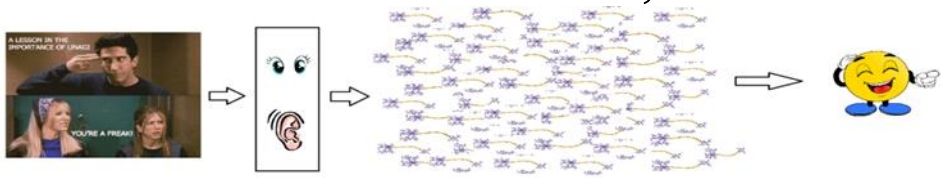
- The structure of Biological Neurons:
  - **Dendrite**: receives signals from other neurons
  - **Synapse:** point of connection to other neurons
  - **Soma:** process the information
  - **Axon:** transmits the output of this neuron



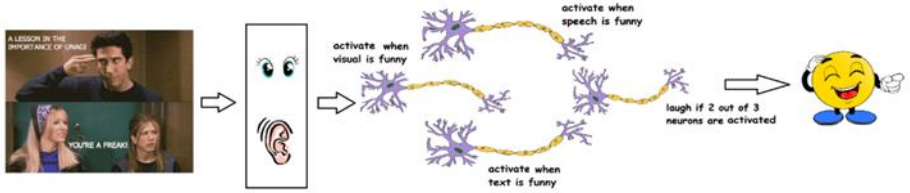
- How neurons transmits the information?
  - **There are almost billions neurons.**

- Some Cartoonish illustrations:

- There are 100 Billion Neurons;



- Do all act at once? Or division of Work?



1.2 Biological Neurons→ Computational Model.

- **Observations from working of Neurons:**
  - Interconnected network of hundreds to thousands of neurons works in parallel
  - For neurons to **activate** or fire or spike **certain threshold** must be passed
  - **Inputs** are receives and can be of:
    - Excitatory inputs/Synapses:
      - Transmits weighted input to the neuron.
    - Inhibitory inputs/Synapses:
      - Any signals from an inhibitory inputs prevents neuron from firing, regardless of other inputs.

- **Neurons as Biological Computational devices:**
  - How can we **model the single neuron** based on the above observation?
    - A Neuron can either fire or not-fire, thus output of neuron can be **binary i.e. 0 or 1**
    - As output can only be 0 or 1, if we can make **collection of input also be binary**, we can create a **threshold function using a logic operator.**

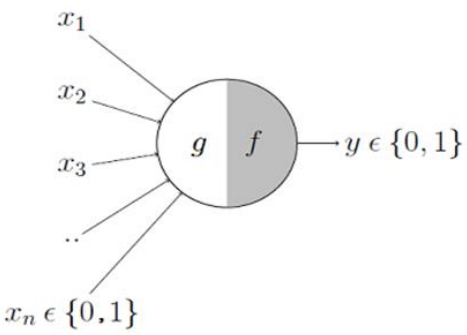
1.3 The McCulloch-Pitts Artificial Neuron

- The first computational model of a neuron was proposed by Warren **MuCulloch** (neuroscientist) and **Walter Pitts** (logician) in 1943.
- Based on there thought on what was the **fundamentals elements** to represent **computation** in **biological neurons** they proposed a model also known as **linear threshold gate or threshold logic units** or **MCP neurons.**
- It is **highly simplified representation**, thus **we can not make conclusion about neuron based** on properties of **MCP neurons.**

- Mathematical Formalization of MCP.
- MCP describes the activity of single neuron with **two states:** firing(1) or not firing(0).
  - $y = 0$ **if any** x_i **is inhibitory**, else:

$$g(x_1, x_2, x_3 \ldots, x_n) = g(X) = \sum_{i=1}^{n} x_i$$

  - if: $g(X) \geq T$
    - $y = f(g(X)) = 1$
  - $else$: $g(X) < T$
    - $y = f(g(X)) = 0$



1.4 Limitation of MCP Neurons

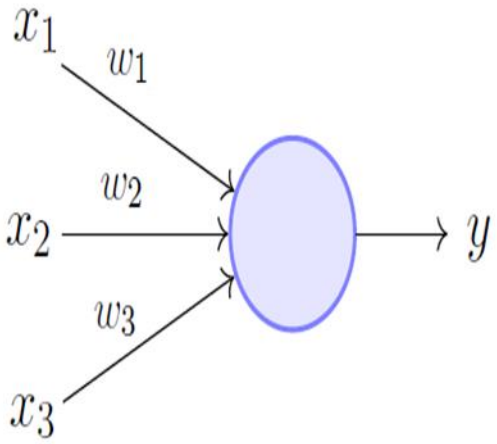
- A single **McCulloch Pitts Neuron** can be used to represent boolean functions which are linearly separable.
  - **Linear separability** (for boolean functions) :
    - There exists a line (plane) such that
      - all inputs which produce a 1 lie on one side of the line (plane)
      - and all inputs which produce a 0 lie on other side of the line (plane)
- The **MCP Neuron Architecture** lacks **several characteristics** of **biological networks**:
  - **Complex connectivity patterns i.e. represents single neuron only**
  - **Processing of continuous values**
  - **A measure of importance**
  - **A learning procedure**
- Regardless of limitation, MCP is considered **a significant** first step towards development of ANN models
  - **Towards Better Model…**

2. The Perceptron

- **Frank Rosenblatt**
  - **Psychologist, Logician**
  - **Inventor of the solution to everything, aka the Perceptron (1958).**
- **"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence," New York Times (8 July) 1958.**
- **"Frankenstein Monster Designed by Navy That Thinks," Tulsa, Oklahoma Times 1958.**

2.1 The Perceptron?

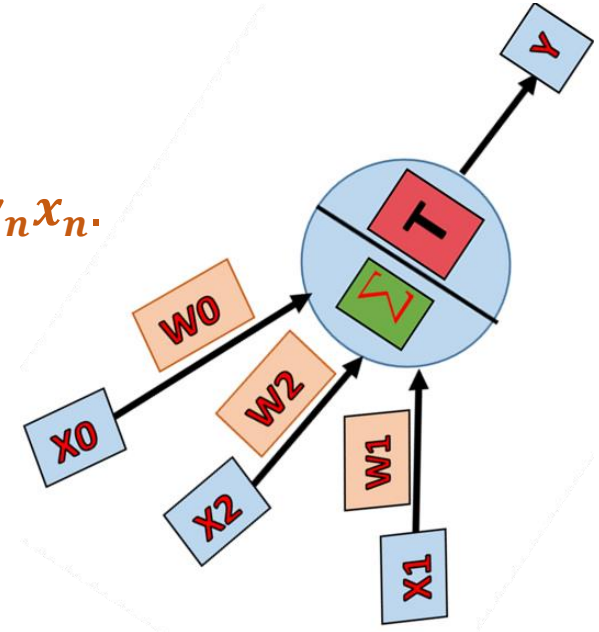
- The ***perceptron*** model, proposed by Rosenblatt and generalized by Minsky-Papert, is a more general computational model than McCulloch-Pitts neuron.
- Perceptron in general can classify the data into two parts.
  - Therefore, it is also known as **Linear Binary Classifier.**
- A measure of importance
  - **the concept of numerical weights**.
    - **the weights are learned during learning**.
  - **It can take any continuous input**.
- What about **threshold function**?
  - Formalize Mathematically.
- **Learning**?



2.2 Perceptron – Simplified Model.

- In it simplest form, the mathematical formulation is as follows:
- Collection of inputs:
- $g(x_1, x_2, x_3 \ldots, x_n) = g(X) = \sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + \ldots\ldots + w_n x_n$.
- Output will be:

$$y = \begin{cases} 1 \; if \; \sum_{i=0}^{n} w_i x_i \geq T \\ -1 \; if \; \sum_{i=0}^{n} w_i x_i < T \end{cases}$$



- I do not want to prefixed my threshold, I can rewrite my equation as

$$y = \begin{cases} 1 \; if \; \sum_{i=0}^{n} w_i x_i - T \geq 0 \\ -1 \; if \; \sum_{i=0}^{n} w_i x_i - T < 0 \end{cases}$$

- Let's replace T with w_0:

$$y = \begin{cases} 1 \; if \; w_0 + \sum_{i=0}^{n} w_i x_i \geq 0 \\ -1 \; if \; w_0 + \sum_{i=0}^{n} w_i < 0 \end{cases}$$

2.3 Perceptron Learning Algorithm.

- Perceptron Learning Algorithm:

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize **w** randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

- For any finite set of linearly separable labeled samples, the Perceptron Learning Algorithm will halt after a finite number of iterations.
- Ideal update would be: if weights w changed in such a way that allows network's prediction to be as closer as possible to actual output, i.e. reduce $(y - \hat{y})$.

- As proposed by Rosenblatt learning for perceptron
  - The network is shown a set of **training samples**.
  - Each training sample has it's own set of **features** and **label** .
  - Sample features are fed through the network so it's **predicted output** can be calculated.
  - **Weights** are adjusted based on the error function i.e. difference between predicted output and real output i.e. $(\boldsymbol{y} - \widehat{\boldsymbol{y}})$.
  - This is an **iterative process**, as multiple samples are shown to the network and weights are updated.

2.4 Minsky and Papert Correction

- "In the book published by Minsky and Papert in 1969, the authors implied that, since a single artificial neuron is incapable of implementing some functions such as the XOR logical function, larger networks also have similar limitations, and therefore should be dropped. Later research on three-layered perceptrons showed how to implement such functions, therefore saving the technique from obliteration."

OR Function.

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$

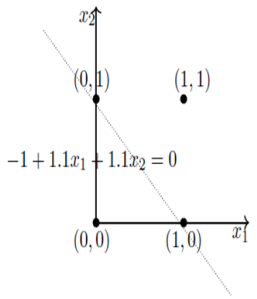
$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$

$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$

$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 > -w_0$

One possible solution is

$w_0 = -1, w_1 = 1.1, w_2 = 1.1$



XOR Function.

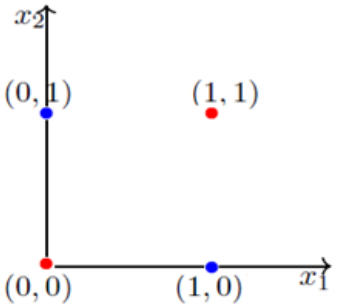
x_1	x_2	XOR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$

$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$

$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$

$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$

$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 < -w_0$



2.5 Strength and Limitation of Perceptron.

Strength

- The perceptron is a powerful problem solver given its simplicity.
- If a problem has a **linearly separable** solution, then it is proved that the perceptron can always **converge towards an optimal solution.**

Limitations: Lesson Learned.

- Many of the real-world problems **are not linearly separable**.
- It is very likely that using a layer of perceptrons will not be the best solution for your problem because it will be unable to find the **optimal boundary that separates your classes**.
- We used linear functions to **aggregate** inputs, which limits scope of network to learn non-linear decision boundary. Thus **XOR problem**.

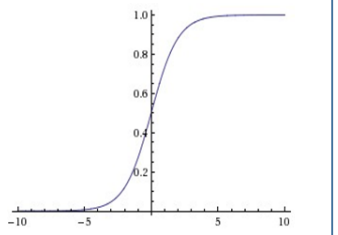
3. Building Neural Network with perceptrons.

3.1 Common Activation Functions

- Introducing Non-Linearity in our Perceptron:
- The activation function does the **non-linear transformation to the input**, making it **capable to learn and perform more complex tasks**.
- The activation function should be **differentiable** or the concept of updating weights (**Backpropagation**)fails, which is the core idea of deep learning
- Some Common Activation Function:

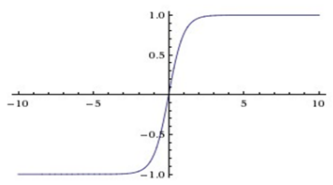
Sigmoid Function

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



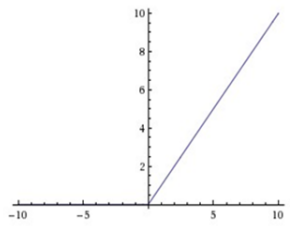
Tanh

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



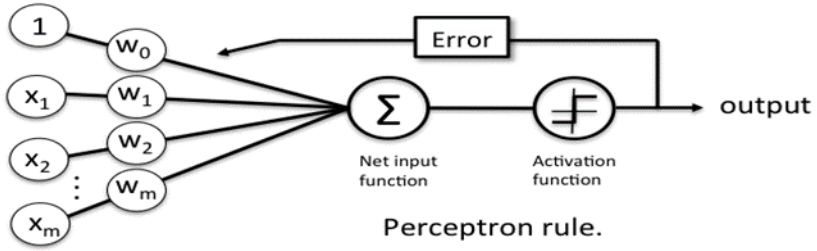
ReLU

- f(x)=max(0,x).

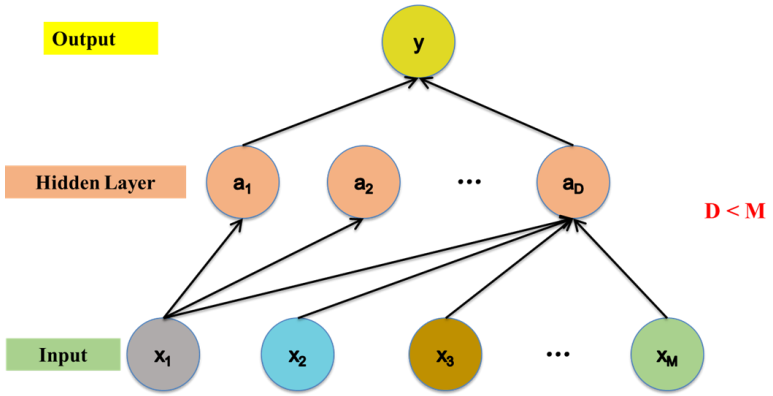


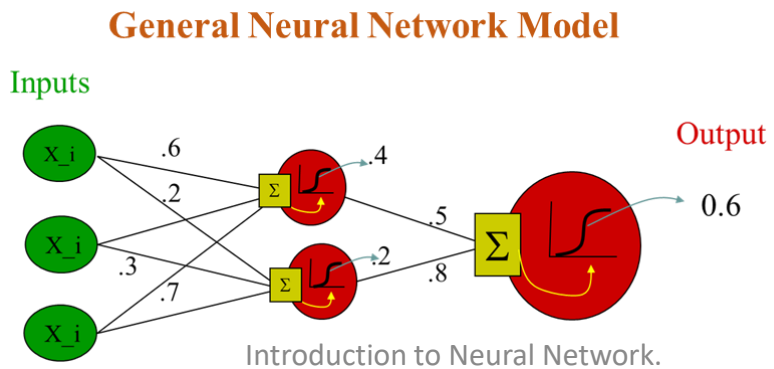
3.2 Network of Neurons.

- Single Neuron with activation functions:



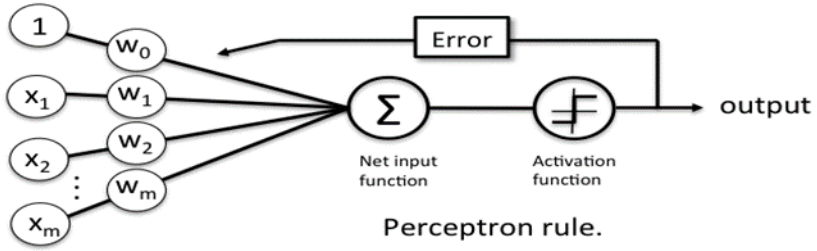
- Stacking a layer of neurons to build a neural network:



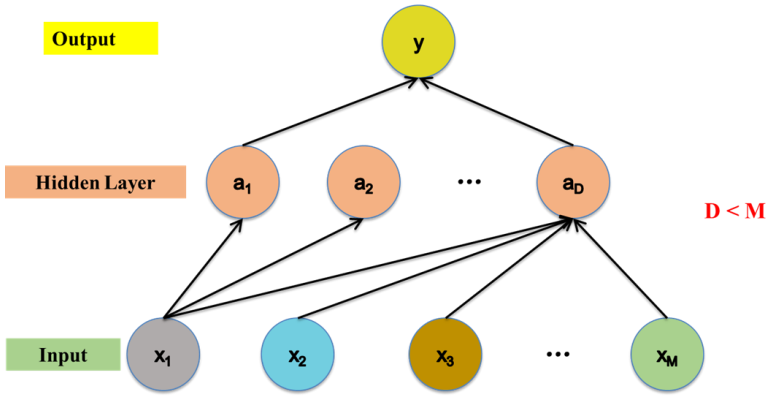


3.2 Network of Neurons.

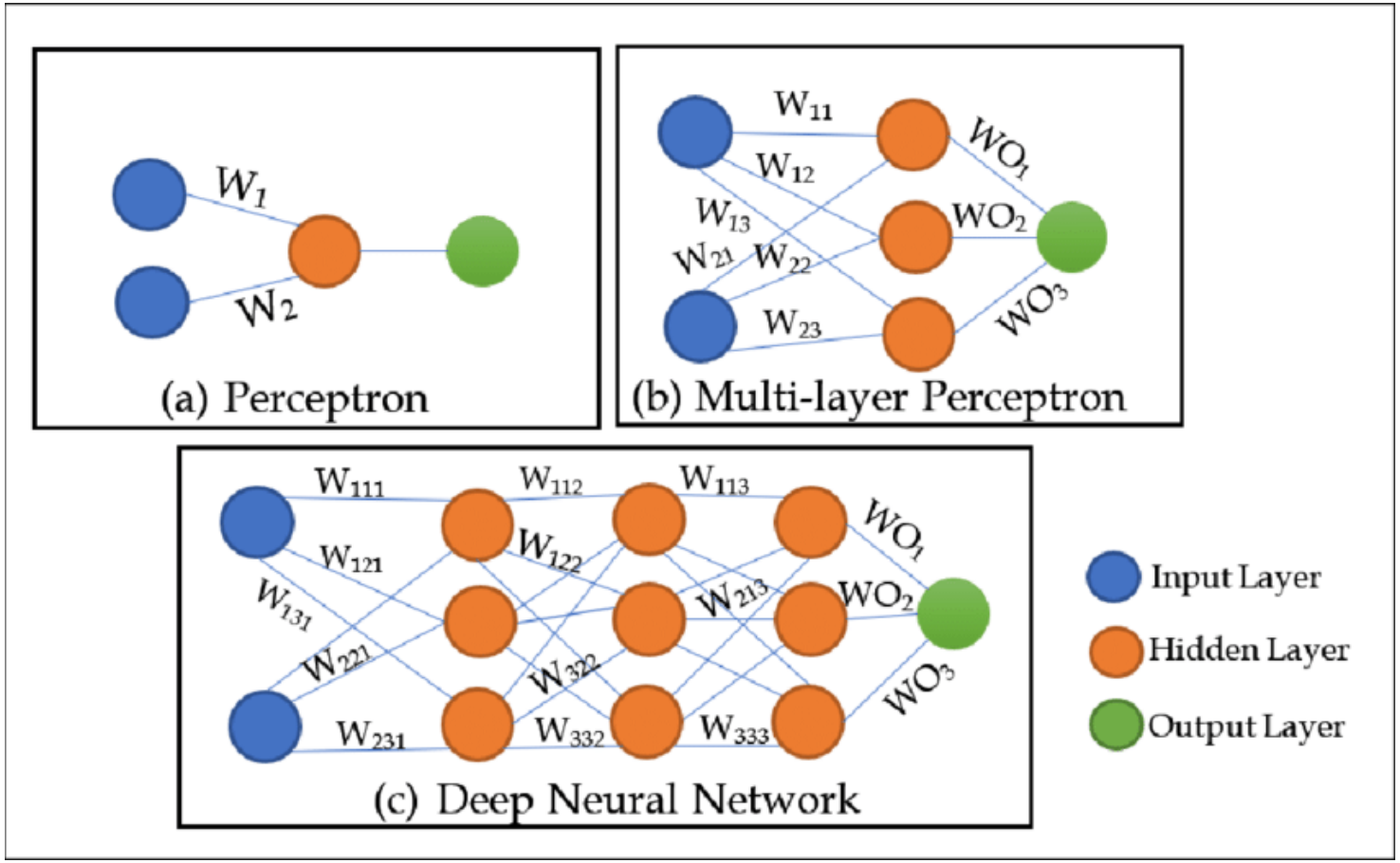
- Single Neuron with activation functions:



- Stacking a layer of neurons to build a neural network:



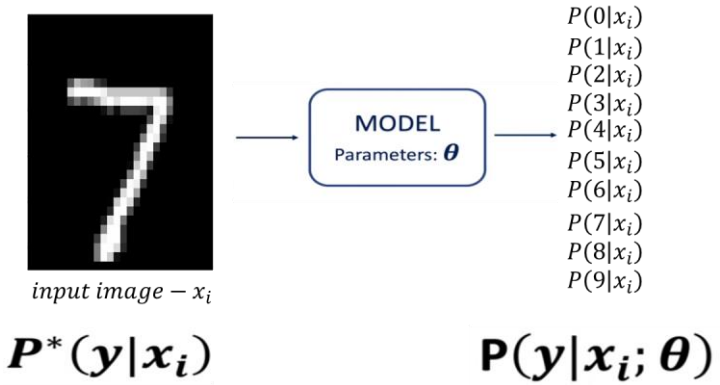
3.3 Shallow to Deeper Neural Networks.



3.4 Problem Setup in Neural Network.

- Idea is to learn the best value of weights.
- Mathematical Formalizations:
  - We want to find the network weights that achieve the lowest loss:

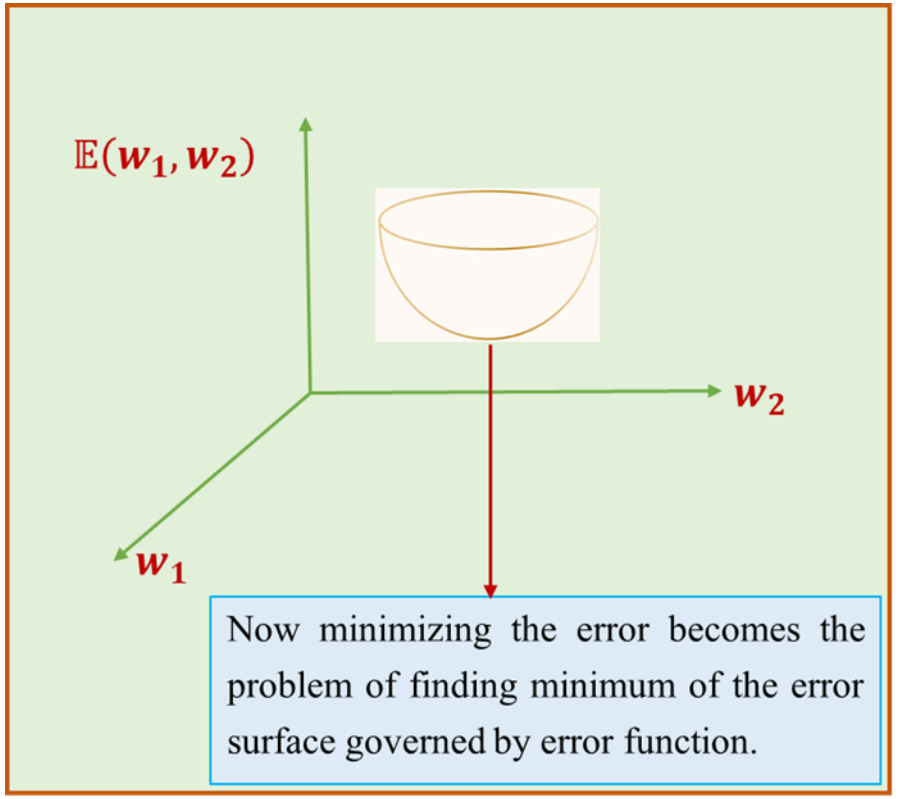
$$w^* = argmin_W \left(\frac{1}{n}\right)\left(\sum_{i=1}^{n} L(f(x^i; W), y^i)\right)$$



4. Training of Neural Network.

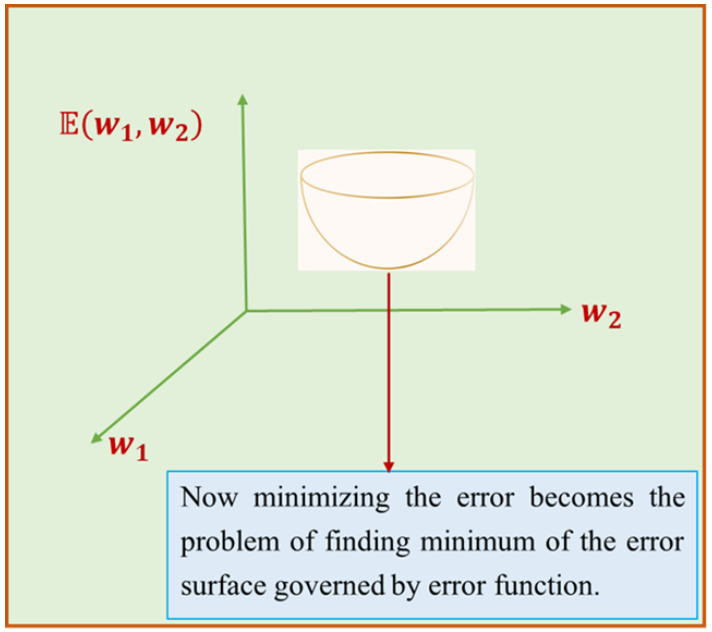
4.1 Error/Loss Surface.

- Question:
  - What kind of plot we get if plotted error vs. weights?
- **Example:**
  - Consider a linear regression with two inputs $\boldsymbol{X \in \{x_1, x_2\}}$ and assume there is no intercept i.e.
    - $\boldsymbol{\hat{y} = f_{w_1,w_2} = w_1 x_1 + w_2 x_2.}$
  - In this case loss function (square error) will generate a convex error surface with the shape of bowl: →

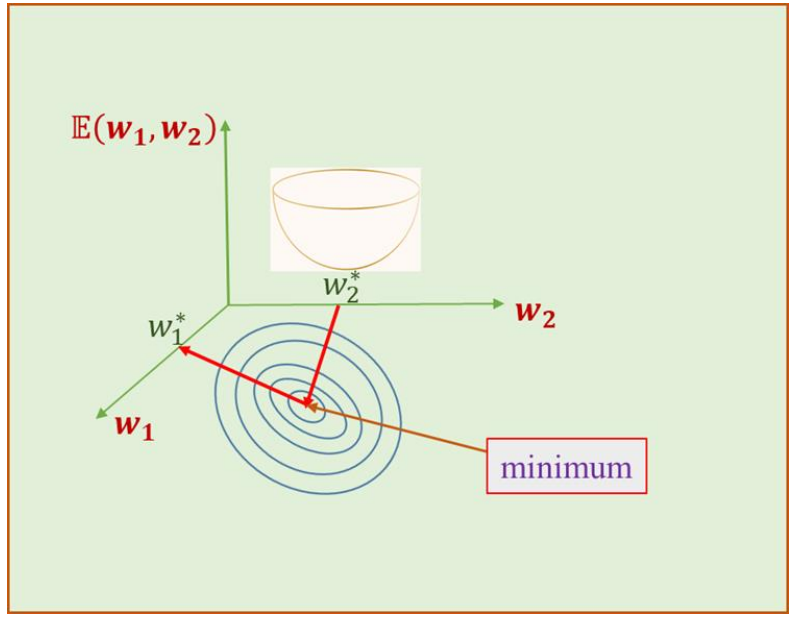


How do we minimize a function?

4.1 Error/Loss Surface.



How do we minimize a function?
Taking a derivative.



Minimizing a **convex function** in Optimization is a Convex Optimization problem.
Such problem can be solved using various iterative algorithms, one such form is:
Gradient Descent!!!!

4.2 Gradient Descent: Idea

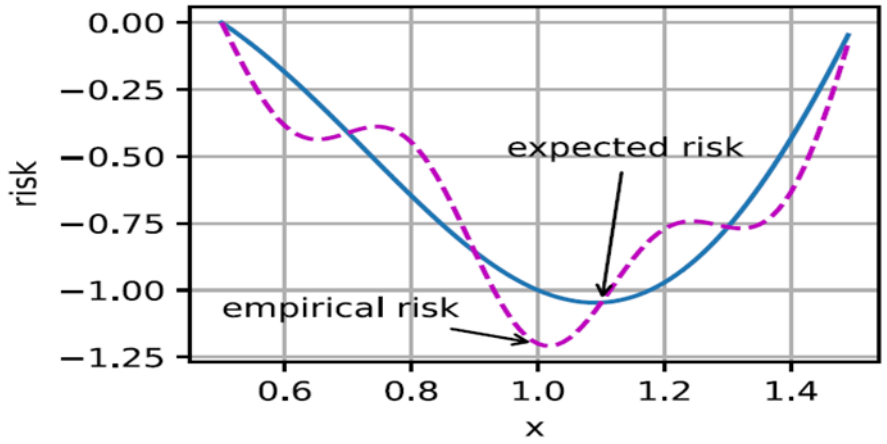
- It is an iterative methods used to compute minimum.
- The gradient $\boldsymbol{\nabla L}$ at any point is the direction of the steepest increase. The negative gradient is the direction of steepest decrease.
- By following the –ve gradient, we can eventually find the lowest point.
- This method is called Gradient Descent.
- Algorithm:
  - For some cost/loss functions: $\mathbb{E}(\boldsymbol{w_0}, \dots, \boldsymbol{w_d})$.
  - Start off with some guesses for $\boldsymbol{w_0}, \dots, \boldsymbol{w_d}$
    - It does not really matter what values you start off with, but a common choice is to set them all initially to zero
  - Repeat until Convergence:{

$$w_{new} := \mathrm{w_{old}} - \alpha \frac{\partial\, \mathbb{E}(\boldsymbol{w_0}, \dots, \boldsymbol{w_d})}{\partial w}$$

}

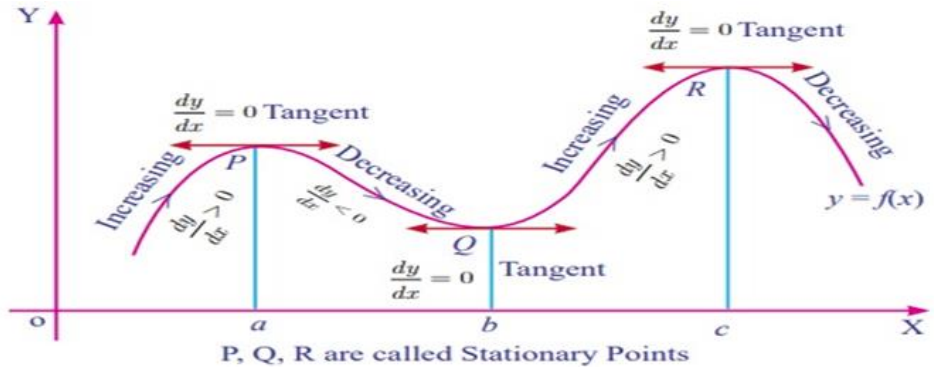
4.3 Error Surface in Practice.

- For a given empirical function g (dashed purple curve), optimization algorithms attempt to find the point of minimum empirical risk
- The expected function f (blue curve) is obtained given a limited amount of training data examples
- ML algorithms attempt to find the point of minimum expected risk, based on minimizing the error on a set of testing examples
    - Which may be at a different location than the minimum of the training examples
    - And which may not be minimal in a formal sense



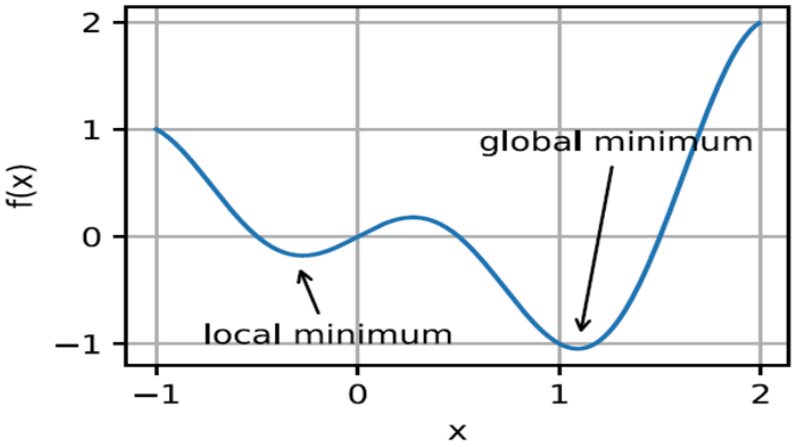
4.4 Stationary Points

- **Stationary points** (or critical points) of a differentiable function f(x) of one variable are the points where the derivative of the function is zero, i.e., $f'(x) = 0$
- The stationary points can be:
  - **Minimum**, a point where the derivative changes from negative to positive
  - **Maximum**, a point where the derivative changes from positive to negative
  - **Saddle point**, derivative is either positive or negative on both sides of the point
- The minimum and maximum points are collectively known as extremum points



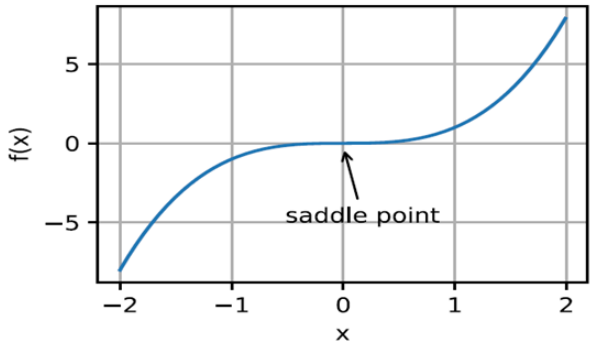
4.5 Local Minima

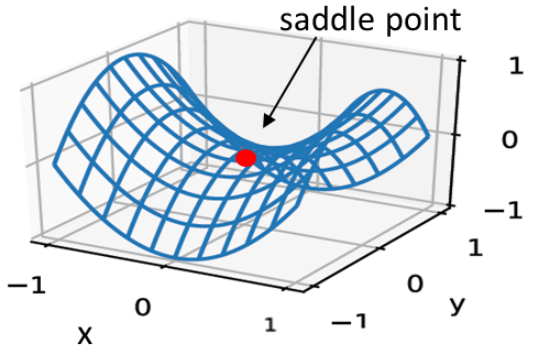
- Among the challenges in optimization of model's parameters in DL involve local minima, saddle points, vanishing gradients
- For an objective function $f(x)$, if the value at a point x is the minimum of the objective function over the entire domain of x, then it is the ***global minimum***
- If the value of $f(x)$ at x is smaller than the values of the objective function at any other points in the vicinity of x, then it is the ***local minimum***
  - The objective functions in DL usually have many local minima
    - When the solution of the optimization algorithm is near the local minimum, the gradient of the loss function approaches or becomes zero (vanishing gradients)
    - Therefore, the obtained solution in the final iteration can be a local minimum, rather than the global minimum



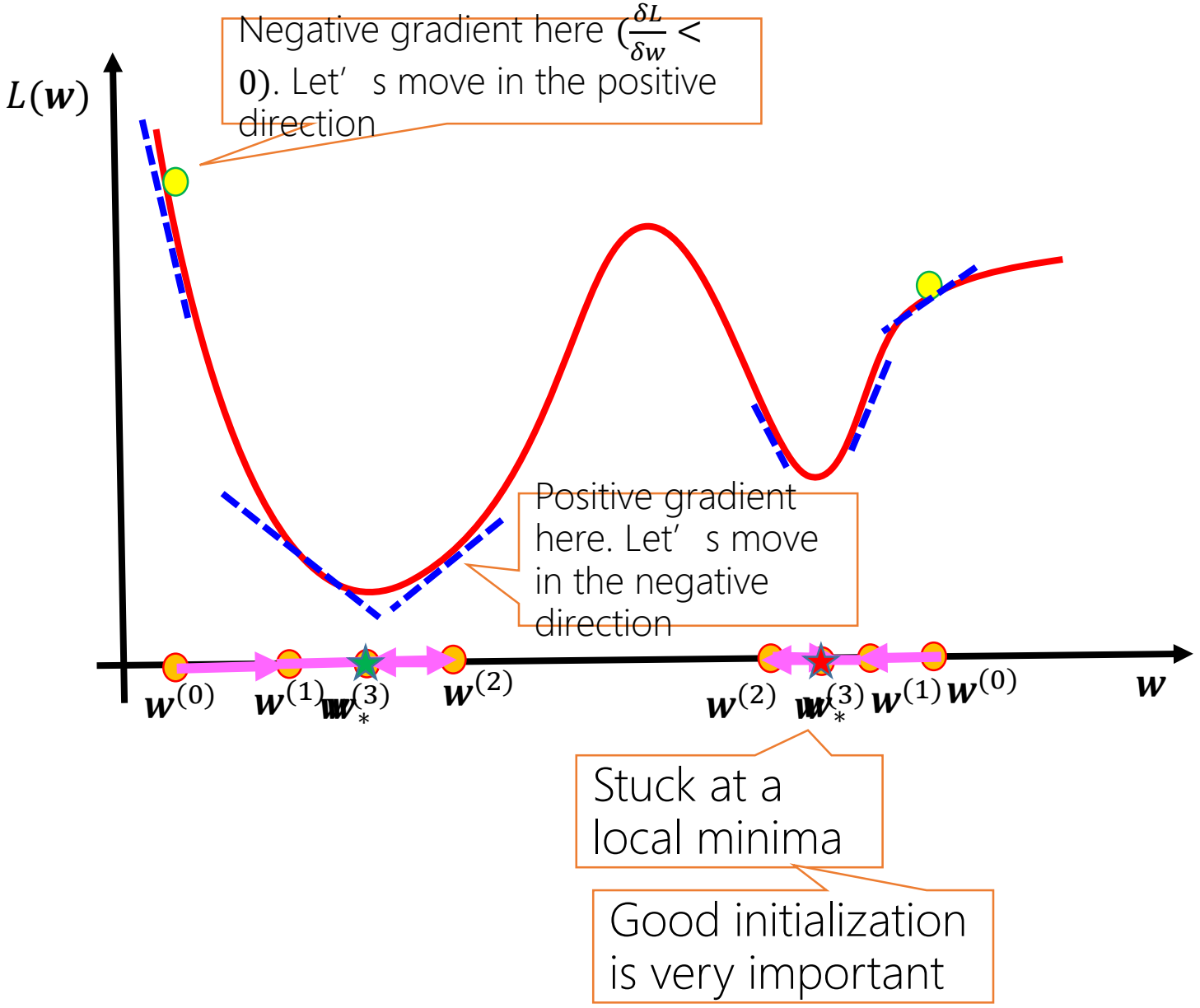
4.6 Saddle Points

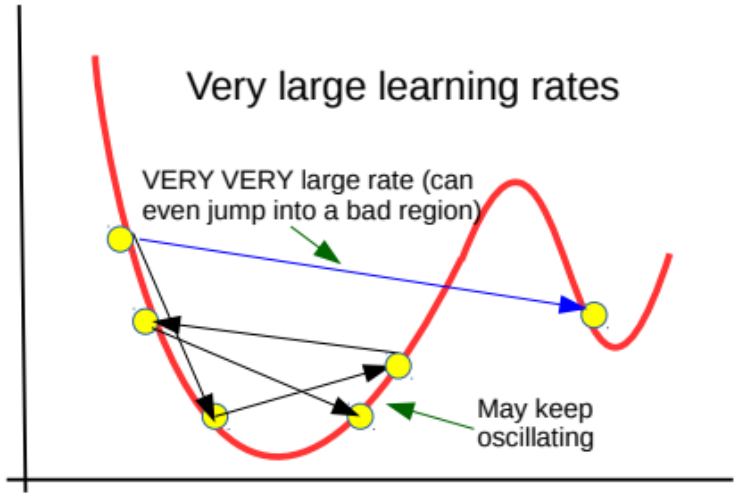
- The gradient of a function $f(x)$ at a saddle point is 0, but the point is not a minimum or maximum point
  - The optimization algorithms may stall at saddle points, without reaching a minima
- Note also that the point of a function at which the sign of the curvature changes is called an inflection point
  - An inflection point ($f''(x) = 0$) can also be a saddle point, but it does not have to be
- For the 2D function (right figure), the saddle point is at (0,0)
  - The point looks like a saddle, and gives the minimum with respect to x, and the maximum with respect to y

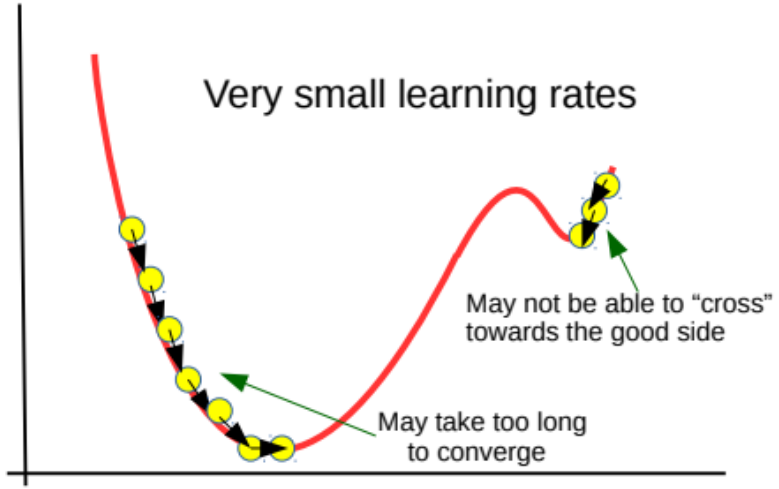




4.7 Gradient Descent: An Illustration

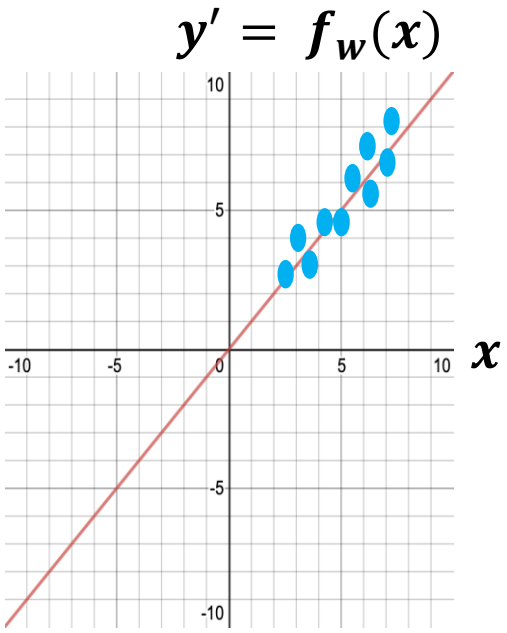


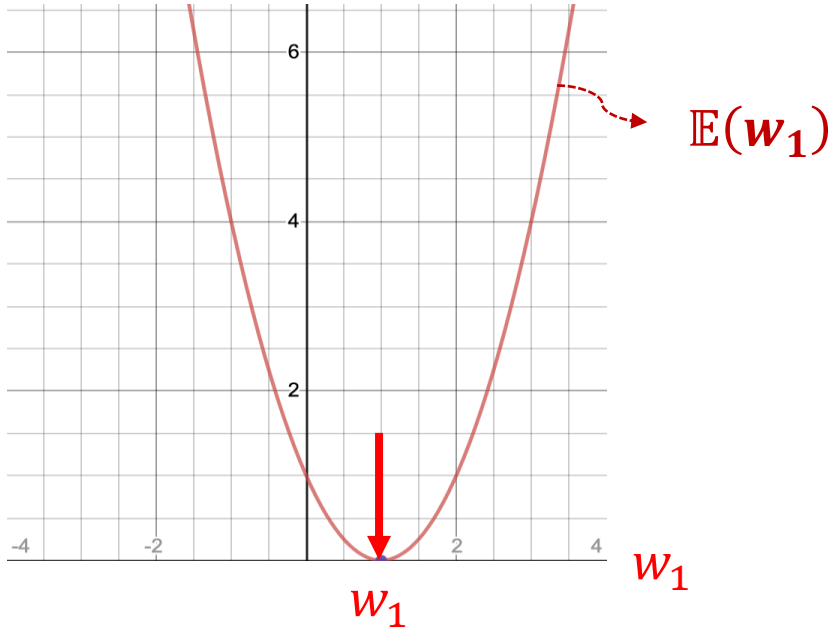




4.8 Gradient Descent - Impact of Partial Derivative.

- Our Objective is to minimize $J(\theta_1)$ for the model represented as $y' = f_w(x) = w_1 x$.



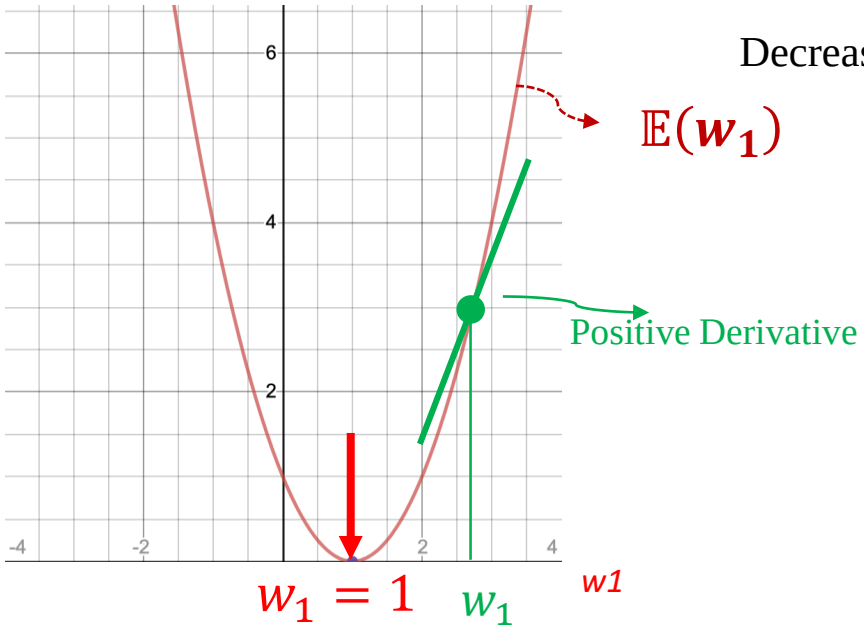


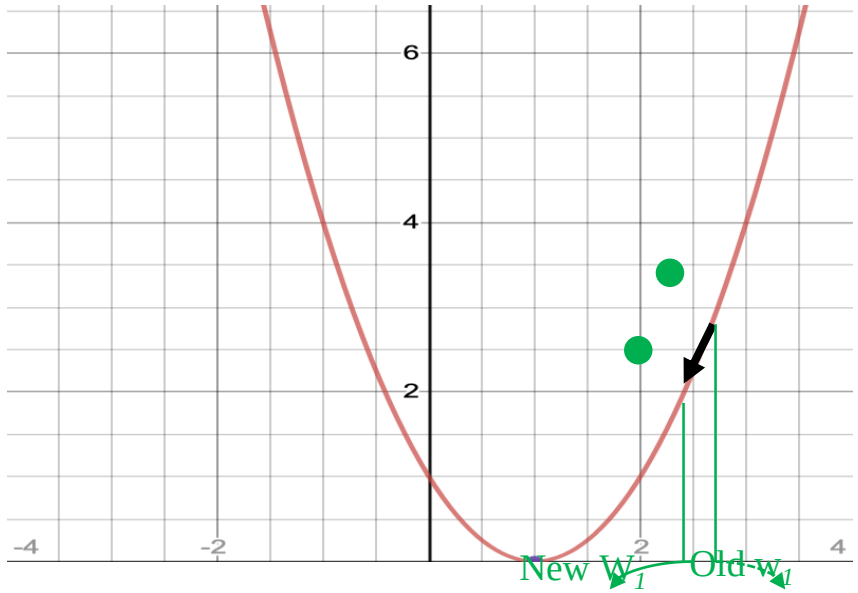
4.8 Gradient Descent-Impact of Partial Derivative.

- Our Objective is to minimize $\mathbb{E}(\boldsymbol{w_1})$ for the model represented as
  - $\hat{y} = \boldsymbol{f_w}(\boldsymbol{x}) = \boldsymbol{W_1 x}.$

$$w_1 = w_1 - \alpha \frac{\boldsymbol{d}\, \boldsymbol{E}(\boldsymbol{w_1})}{\boldsymbol{d}\, w\_1_j}$$
$$= w_1 - \alpha\, (Positive\ Number)$$

Decrease w_1 by a certain value



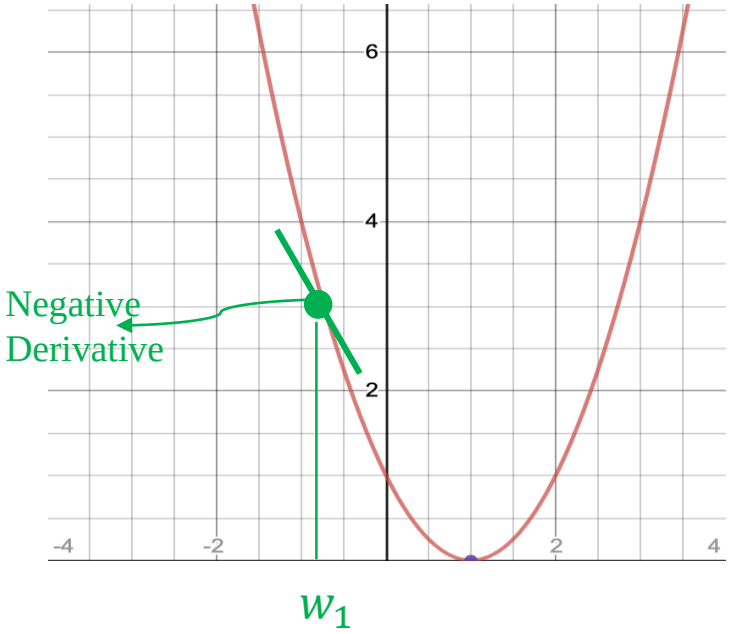


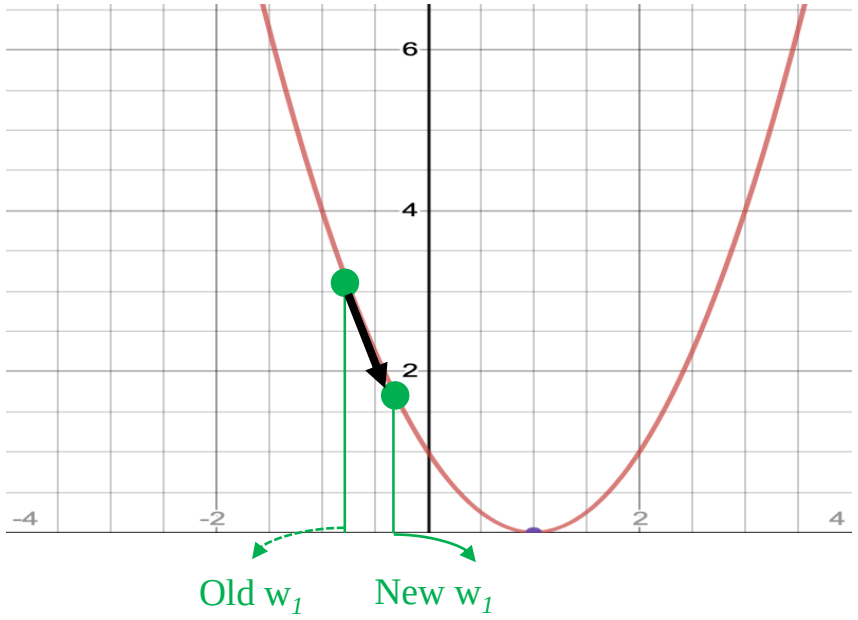
4.8 Gradient Descent-Impact of Partial Derivative.

- Our Objective is to minimize $\mathbb{E}(\boldsymbol{w_1})$ for the model represented as
  - $\hat{y} = \boldsymbol{f_w}(\boldsymbol{x}) = \boldsymbol{W_1 x}$.

$$w_1 = \mathrm{w}_1 - \alpha \frac{\boldsymbol{d}\, \boldsymbol{J}(\boldsymbol{\theta_1})}{\boldsymbol{d}\, \theta_j}$$
$$= \mathrm{w}_1 - \alpha\, (Negative\ Number)$$

Increase w_1 by a certain value



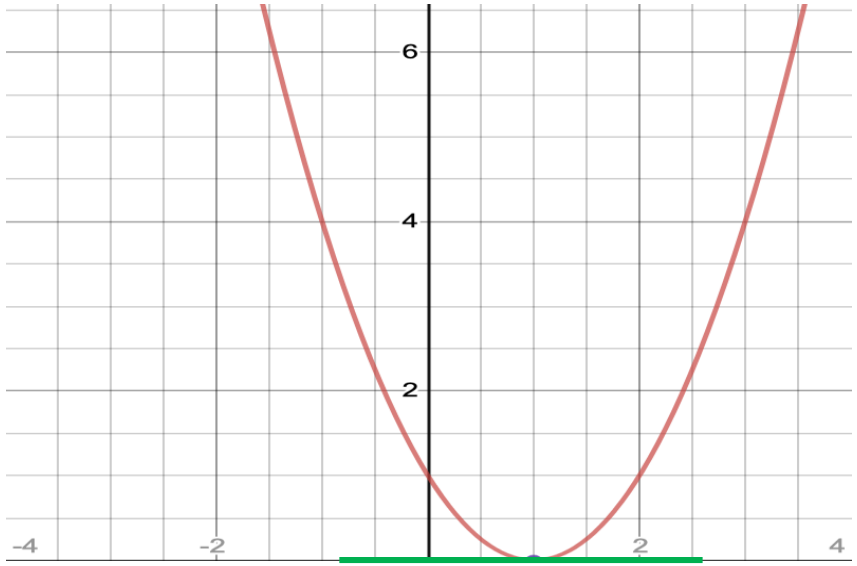


4.8 Gradient Descent-Impact of Partial Derivative.

- Our Objective is to minimize $\mathbb{E}(\boldsymbol{w_1})$ for the model represented as
  - $\hat{y} = \boldsymbol{f_w}(\boldsymbol{x}) = \boldsymbol{W_1 x}$.

$$w_1 = w_1 - \alpha \frac{\boldsymbol{d}\, \boldsymbol{E}(\boldsymbol{w_1})}{\boldsymbol{d}\, w_j}$$
$$= w_1 - \alpha\, (Zero)$$

w_1 remains same.

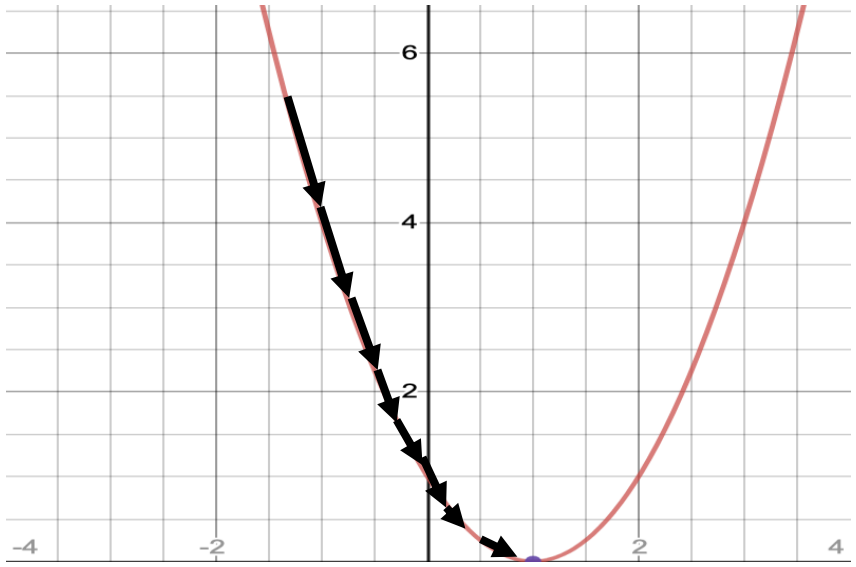


Derivative = 0

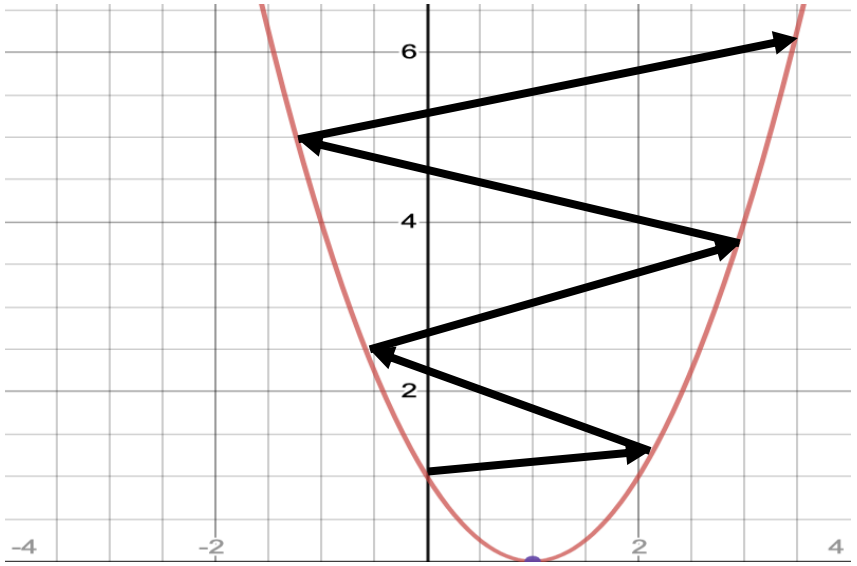
4.9 Gradient Descent-Impact of Learning Rate.

- Our Objective is to minimize $\mathbb{E}(\boldsymbol{w_1})$ for the model represented as
  - $\hat{y} = \boldsymbol{f_w}(\boldsymbol{x}) = \boldsymbol{W_1 x}.$

$$w_1 = w_2 - \alpha \frac{\boldsymbol{d}\, \boldsymbol{E}(\boldsymbol{w_1})}{\boldsymbol{d}w_1}$$
$$= \alpha: Too\ small\ number.$$

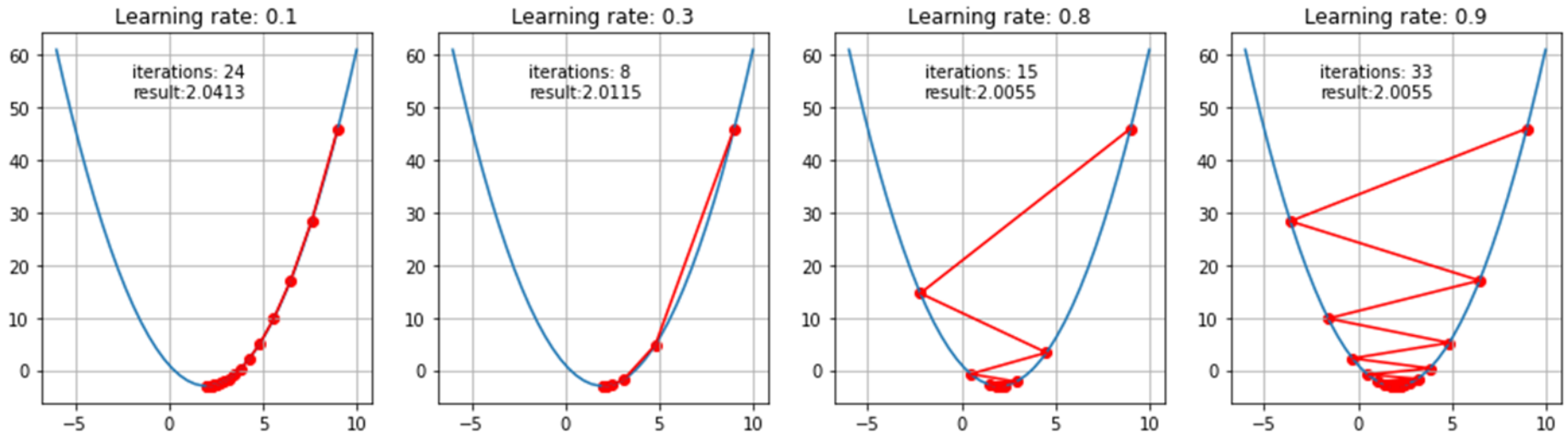


$$w_1 = w_1 - \alpha \frac{\boldsymbol{d}\, \boldsymbol{E}(\boldsymbol{\theta_1})}{\boldsymbol{d}\, w_j}$$
$$= \alpha: Too\ Large\ number.$$



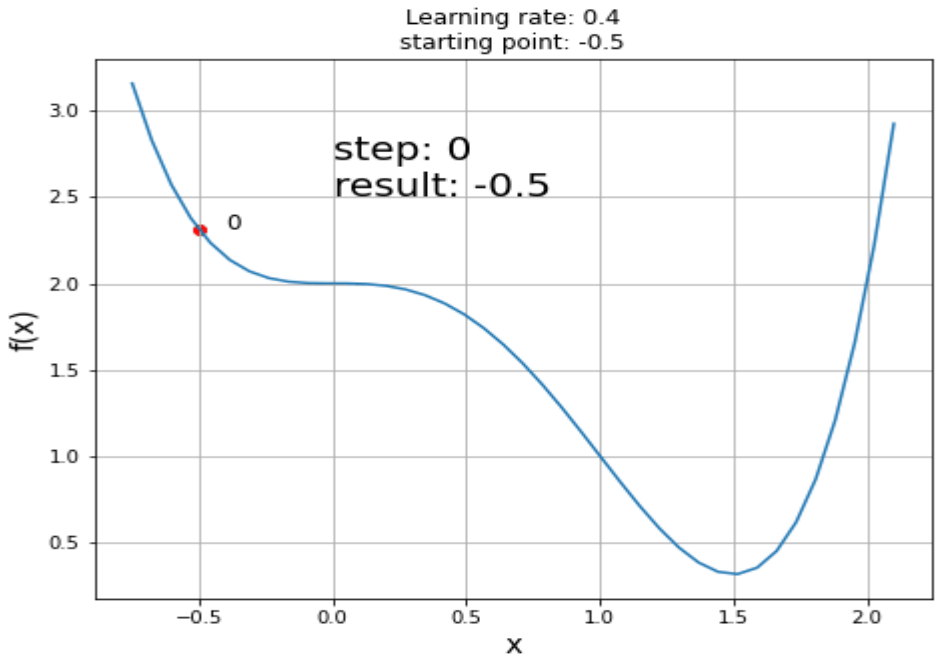
4.9 Gradient Descent-Learning Rate and Convergence.

- High Learning Rate does not Guarantees speedy Convergence.



4.10 Gradient Descent – Starting Point and Convergence.

- Convergence speed also depends on our starting point.

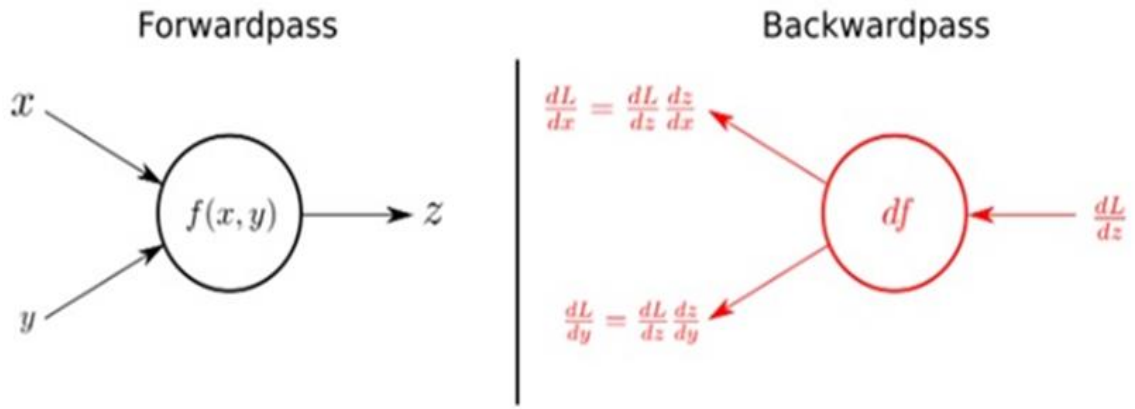


5. Computing Gradients

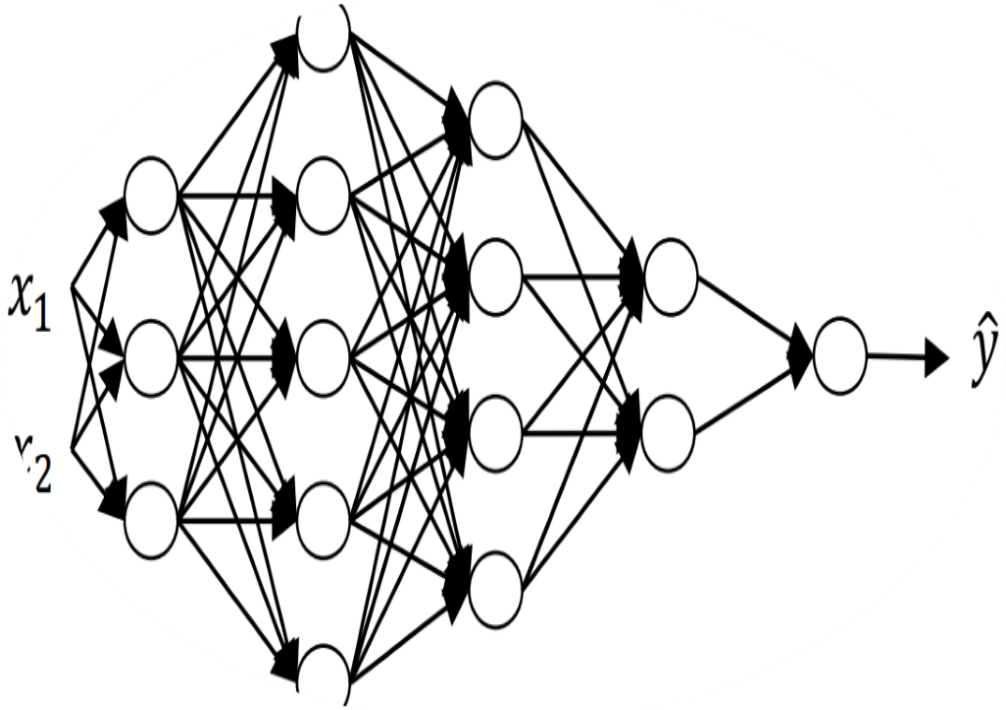
Forward and Backward Propagation.

5.1 Forward and Backward Propagations.

- The weights in Multi layer networks are learned with the combinations of forward and backward propagations.
- a network forward propagates activation to produce an output and it backward propagates error to determine weight changes



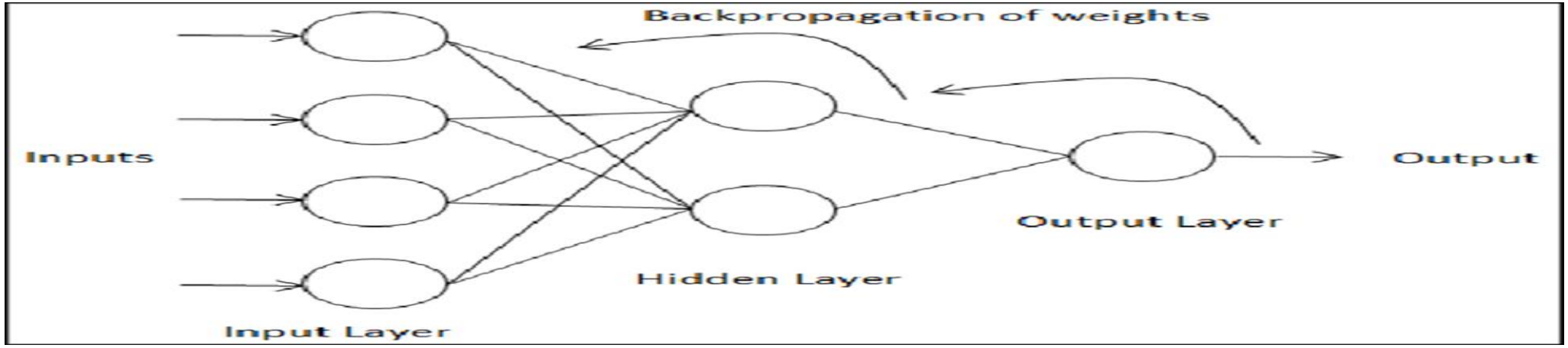
5.2 Feed Forward Calculations.



$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}\left(Z^{[L]}\right) = \hat{Y}
\end{aligned}
$$

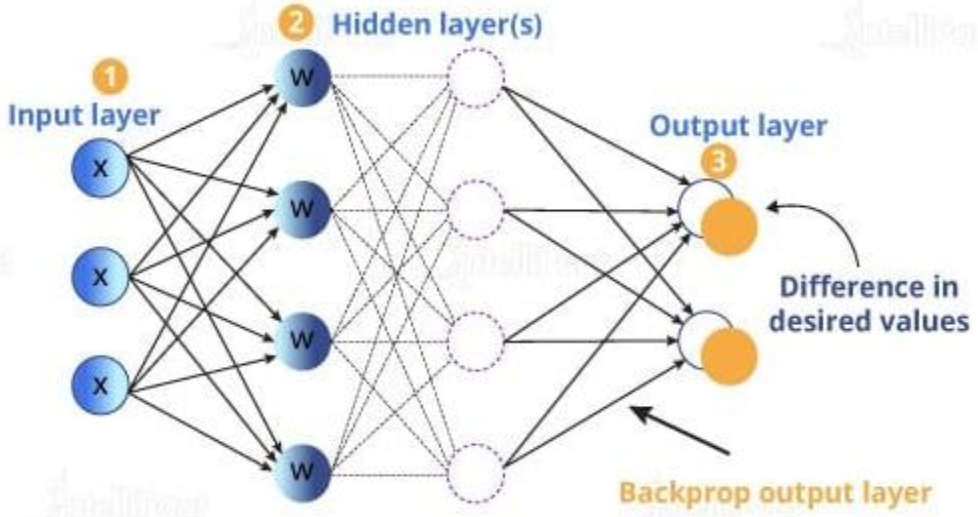
5.3 Back-Propagation.

- Backpropagation is a technique used by deep layer networks to find the error of the network.
- The error is calculated by comparing an expected output with a predicted output, this algorithm then propagated these errors backward to update weights and biases.



5.3 Backpropagation Requirements

- Backprop Requires following three things:
  1. Dataset: in the pairs of input-output i.e. (x_i, y_i).
  2. A feed-forward neural network;
  3. An Error Function, E which defines the error between the desired output and calculated output.



5.4 Backpropagation Algorithm.

Algorithm 2 Backpropagation algorithm for feedforward networks.

Require: a set of training examples D, learning rate α.

1. Create a feed-forward network with n_{in} inputs, n_h hidden units, and n_o output units.
2. Initialize all network weights to *small* random numbers.
3. **Repeat** until the termination condition is met:

   For each training example $(\mathbf{x}, \mathbf{t}) \in D$ **do:**

   Propagate the input $\mathbf{x}$ forward through the network, i.e.:

   1. Input $\mathbf{x}$ to the network and compute the output a_k of units in the output layer.

   Backpropagate the errors through the network:

   1. **For** each network output unit k, calculate its error term δ_k:

$$\delta_k \leftarrow -a_k(1 - a_k)(t_k - a_k) \tag{3}$$

   2. **For** each hidden unit h, calculate its error term δ_h:

$$\delta_h \leftarrow a_h(1 - a_h) \sum_k \delta_k w_{kh} \tag{4}$$

   3. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

   where

$$\Delta w_{ji} = -\alpha \delta_j x_{ji}$$

Notations:

The subscript k denotes the output layer.

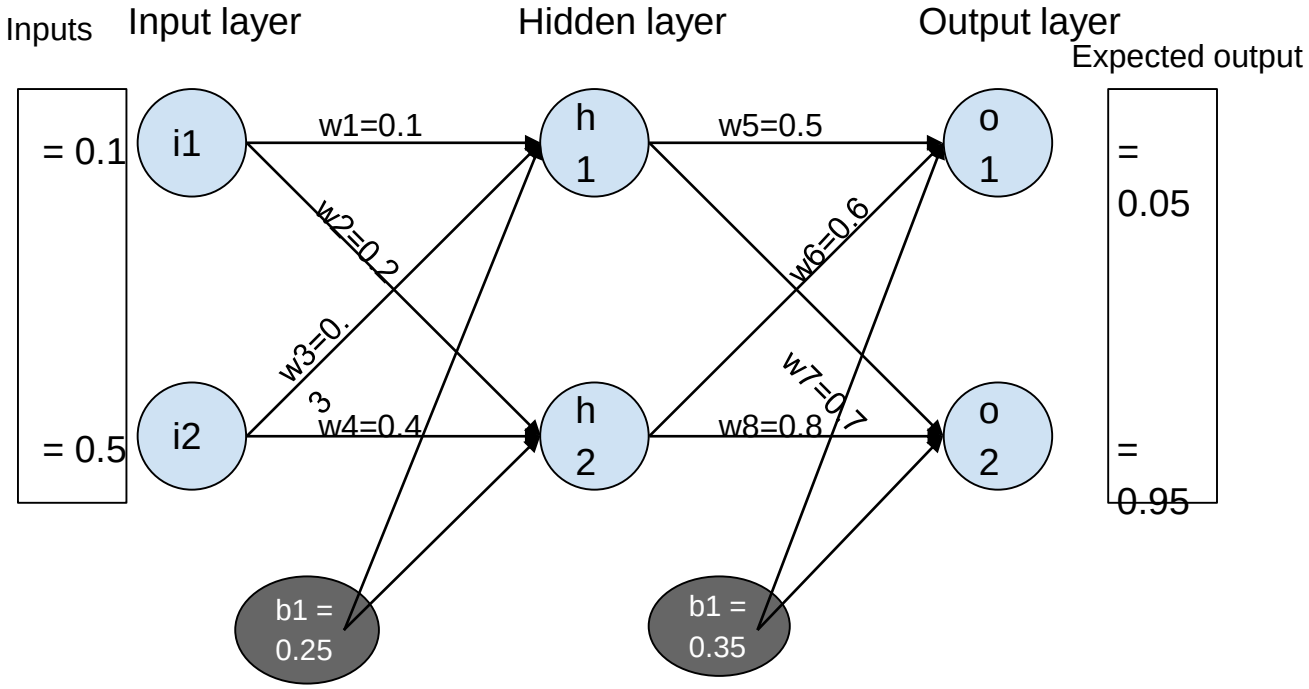
The subscript j denotes the hidden layer.

The subscript i denotes the input layer

6. Forward and Backward Pass

Example

Solve Neural Network below:



Notations:

The Neural Network has three layers:

- Input Layer with two units.
- Hidden Layer with two Units.
- Output Layer with two Units.

Weights for all the connections are denoted as:

- w1 to w8

Biases for Hidden and Output layers are:

- b1 and b2

Task:

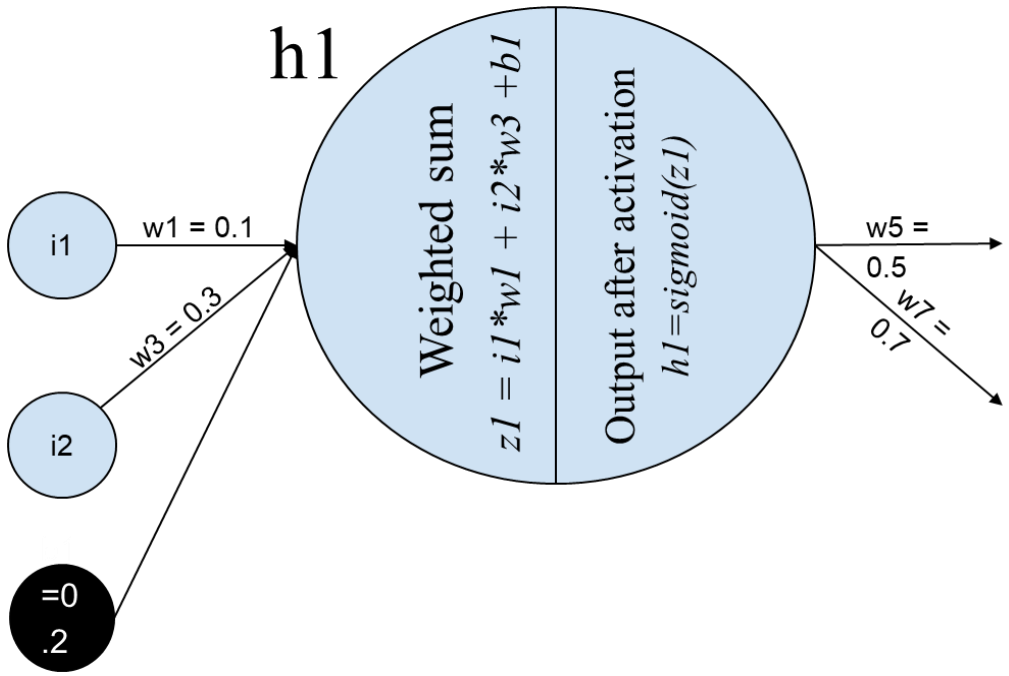
Using two inputs i1 and i2:

Perform a **forward pass** through the network to **compute total error**.

Perform a backward pass to **propagate the error** within the network and update the weights accordingly.

Forward Propagation.

Computation on Single Sigmoid Neuron is as:



Let's Get Started with forward pass:
For h1:

$$sum_{h1} = i_1 * w_1 + i_2 * w_3 + b_1$$

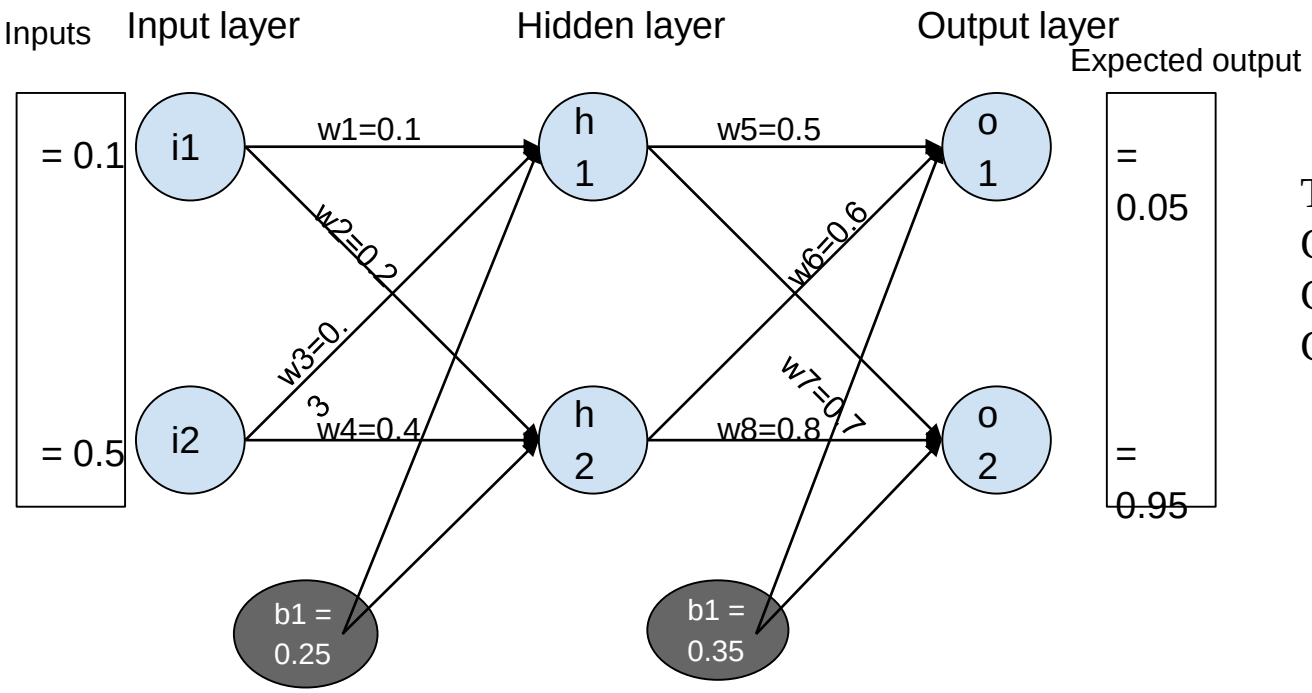
$$sum_{h1} = 0.1 * 0.1 + 0.5 * 0.3 + 0.25 = 0.41$$

Now, pass the weighted sum through the logistic function:

$$output_{h1} = \frac{1}{1 + e^{-sum_{h1}}}$$

$$output_{h1} = \frac{1}{1 + e^{-0.41}} = 0.60108$$

Forward Propagation.



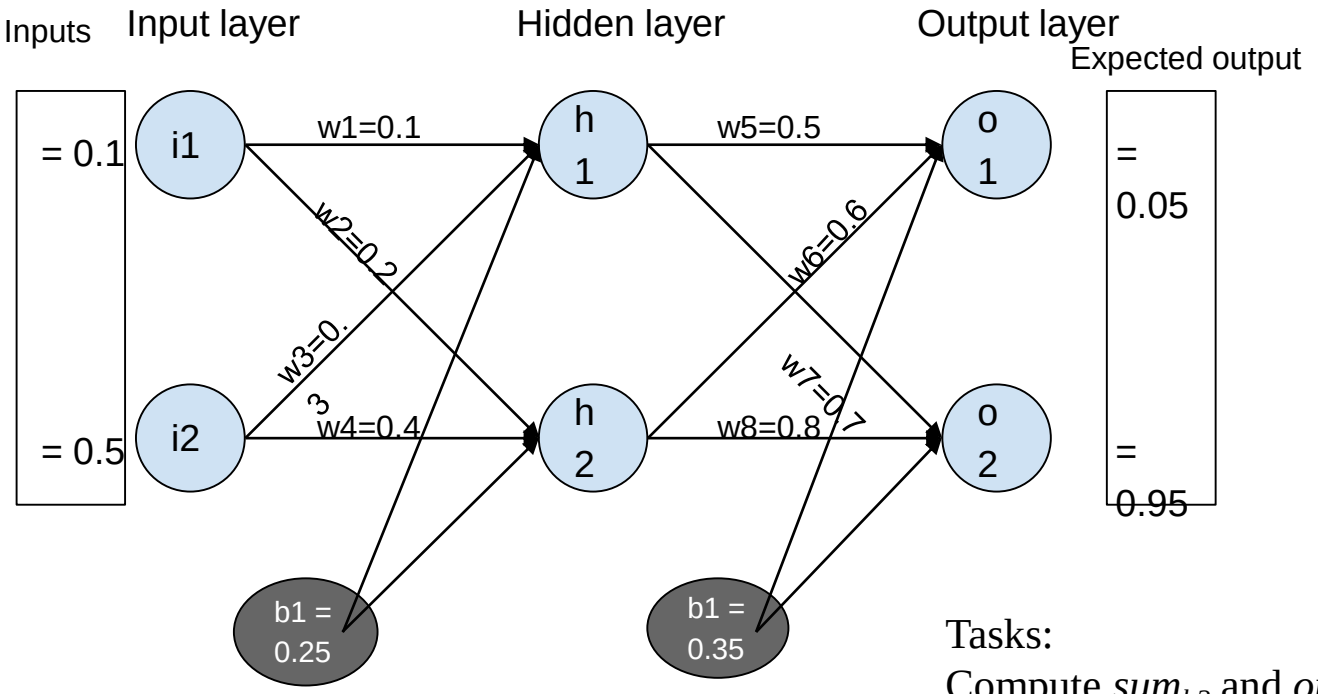
Tasks:

Compute sum_{h2} and $output_{h2}$ for our second node *h2*.

Compute sum_{o1} and $output_{o1}$ for the node *o1*.

Compute sum_{o2} and $output_{o2}$ for the node *o2*.

Forward Propagation.



Tasks:

Compute sum_{h2} and $output_{h2}$ for our second node *h2*. [**Answer**: 0.47, 0.61538]

Compute sum_{o1} and $output_{o1}$ for the node *o1*. [**Answer**: 1.01977, 0.73492]

Compute sum_{o2} and $output_{o2}$ for the node *o2*. [**Answer**: 1.26306, 0.77955]

Forward Propagation.

Compute Total Error:

The Expected Outputs:

$target_{o1} \rightarrow 0.05$

$target_{o2} \rightarrow 0.95$

We will now Compute the errors, with formula:

$$E_1 = \frac{1}{2}(target_1 - output_{o1})^2$$

$$E_2 = \frac{1}{2}(target_2 - output_{o2})^2$$

$$E_{total} = E_1 + E_2$$

Tasks:

Compute *E1*, *E2*, and E_{total} .

Forward Propagation.

Compute Total Error:

The Expected Outputs:

$target_{o1} \rightarrow 0.05$

$target_{o2} \rightarrow 0.95$

We will now Compute the errors, with formula:

$$E_1 = \frac{1}{2}(target_1 - output_{o1})^2$$

$$E_2 = \frac{1}{2}(target_2 - output_{o2})^2$$

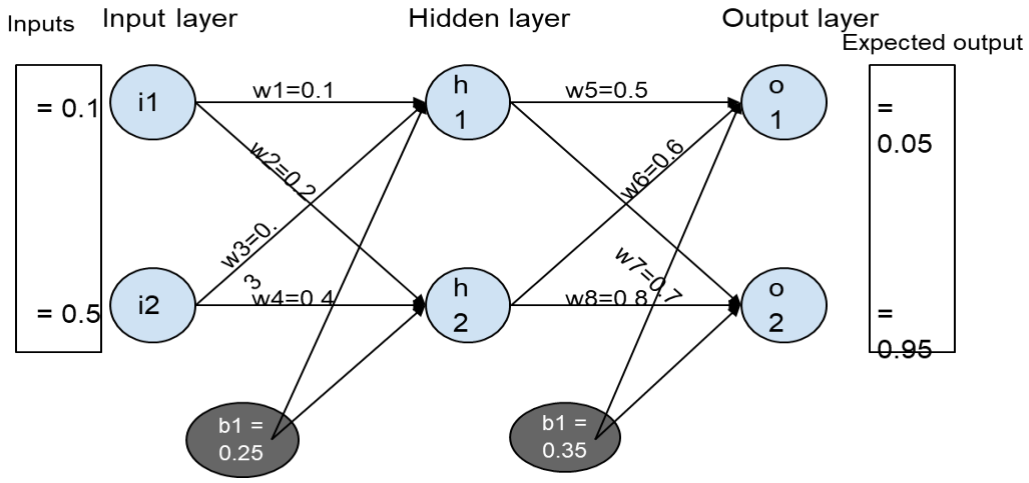
$$E_{total} = E_1 + E_2$$

Tasks:

Compute *E1*, *E2*, and E_{total}. [**Answer**: 0.23456, 0.01452, 0.24908]

Congratulations you completed Forward Pass!!!!

Backward Propagation.



Task: Update weight at *w5*

- The purpose of the backward pass, also known as Backpropagation, is to distribute the overall error across the network .
- Modify the weights to minimize the cost function (loss). The weights are updated in a manner that ensures that the subsequent forward pass employs the updated weights.
- Decrease the overall error by a specific margin until the minima is achieved.
- Computes how much contribution each weight has on corresponding error.
- If we closely look at the example neural network, we can see that *E1* is affected by $output_{o1}$, $output_{o1}$ is affected by sum_{o1} , and sum_{o1} is affected by w5.
- From the chain Rule:

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$$

Task: Update weight at *w5*.

- We have; $\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$

Let's Breakdown each component of the above chain separately:

Component1: partial derivative of Error w.r.t Output.

Here, E_{total} is given by:

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

$$E_{total} = \frac{1}{2}(target_1 - output_{o1})^2 + \frac{1}{2}(target_2 - output_{o2})^2$$

$$\frac{\partial E_{total}}{\partial output_{o1}} = 2 * \frac{1}{2} * (target_1 - output_{o1}) * -1$$

$$= output_{o1} - target_1$$

Task: Update weight at *w5*.

- We have; $\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$

Let's Breakdown each component of the above chain separately:

Component2: partial derivative of Output w.r.t Sum.

Output segment of a neural network unit employs non-linear activation functions.
The activation function chosen for this example is the sigmoid function.
The derivative of sigmoid is:

$\sigma(x) = \frac{1}{1+e^{-x}}$ $\frac{\mathrm{d}}{\mathrm{dx}}\sigma(x) = \sigma(x)(1-\sigma(x))$ Derivative of Logistic→ output(1-output)

Hence:

$$\frac{\partial output_{o1}}{\partial sum_{o1}} = output_{o1}(1 - output_{o1})$$

Task: Update weight at *w5*.

- We have; $\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$

Let's Breakdown each component of the above chain separately:

Component3: partial derivative of Sum w.r.t *w5*.

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2$$

Hence:

$$\frac{\partial sum_{o1}}{\partial w5} = output_{h1}$$

Let's Put all together:

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$$

$$\frac{\partial E_{total}}{\partial w5} = [output_{o1} - target_1] * [output_{o1}(1 - output_{o1})] * [output_{h1}]$$

Compute the above derivative $dE_{total}/dw5$.

Task: Update weight at *w5*.

- We have; $\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$

Let's Breakdown each component of the above chain separately:

Component3: partial derivative of Sum w.r.t *w5*.

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2$$

Hence:

$$\frac{\partial sum_{o1}}{\partial w5} = output_{h1}$$

Let's Put all together:

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$$

$$\frac{\partial E_{total}}{\partial w5} = [output_{o1} - target_1] * [output_{o1}(1 - output_{o1})] * [output_{h1}]$$

Compute the above derivative $dE_{total}/dw5$. [**Answer**: 0.08020]

Task: Update weight at *w5*.

Compute the above derivative $dE_{total}/dw5$. [**Answer**: 0.08020]

The *new_w5* is

$$new\_w_5 = w5 - n * \frac{\partial E_{total}}{\partial w5}$$

Here: n is learning rate set at-0.6

Task: Compute *new_w5*.

Similarly find:
$dE_{total}/dw6$, *new_w6*.
$dE_{total}/dw7$, *new_w7*.
$dE_{total}/dw8$, *new_w8*.

Task: Update weight at *w5*.

Compute the above derivative $dE_{total}/dw5$. [**Answer**: 0.08020]

The *new_w5* is

$$new\_w_5 = w5 - n * \frac{\partial E_{total}}{\partial w5}$$

Here: n is learning rate set at-0.6

Task: Compute *new_w5*. [**Answer**: 0.45187]

Similarly find:
$dE_{total}/dw6$, *new_w6*. [**Answer**: 0.08211, 0.55073]
$dE_{total}/dw7$, *new_w7*. [**Answer**: -0.01760, 0.71056]
$dE_{total}/dw8$, *new_w8*. [**Answer**: −0.01802, 0.81081]

Task: Update Weight *w1, w2, w3, w4.*

First let's compute $dE_1/dw1$:

$$\frac{\partial E_1}{\partial w1} = \frac{\partial E_1}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

First Component:

$$\frac{\partial E_1}{\partial output_{o1}} = output_{o1} - target_1$$

We already computed the second component.

Third Component:

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2$$

$$\frac{\partial sum_{o1}}{\partial output_{h1}} = w5$$

Fourth Component:

$$\frac{\partial output_{h1}}{\partial sum_{h1}} = output_{h1} * (1 - output_{h1})$$

Fifth Component:

$$sum_{h1} = i_1 * w_1 + i_2 * w_3 + b_1$$

$$\frac{\partial sum_{h1}}{\partial w1} = i_1$$

Putting it all together Compute $dE_1/dw1$:

Task: Update Weight *w1, w2, w3, w4.*

First let's compute $dE_2/dw1$:

$$\frac{\partial E_2}{\partial w1} = \frac{\partial E_2}{\partial output_{o2}} * \frac{\partial output_{o2}}{\partial sum_{o2}} * \frac{\partial sum_{o2}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

First Component:

$$\frac{\partial E_2}{\partial output_{o2}} = output_{o2} - target_2$$

We already computed the second component.

Third Component:

$$sum_{o2} = output_{h1} * w_7 + output_{h2} * w_8 + b_2$$

$$\frac{\partial sum_{o2}}{\partial output_{h1}} = w7$$

Fourth Component:

$$\frac{\partial output_{h1}}{\partial sum_{h1}} = output_{h1} * (1 - output_{h1})$$

Fifth Component:

$$sum_{h1} = i_1 * w_1 + i_2 * w_3 + b_1$$

$$\frac{\partial sum_{h1}}{\partial w1} = i_1$$

Putting it all together Compute $dE_2/dw1$:

Task: Update Weight *w1, w2, w3, w4.*

Compute $dE_{total}/dw1$

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_1}{\partial w1} + \frac{\partial E_2}{\partial w1}.$$

new_w1 is:

$$new\_w_1 = w1 - n * \frac{\partial E_{total}}{\partial w1}$$

Similarly Find:
$dE_{total}/dw2$, *new_w2* [**Answer**: 0.19919]
$dE_{total}/dw3$, *new_w3* [**Answer**: 0.29667]
$dE_{total}/dw4$, *new_w4* [**Answer**: 0.39597]

Thank You!!!
For Collaboration:
simangiri@heraldcollege.edu.np