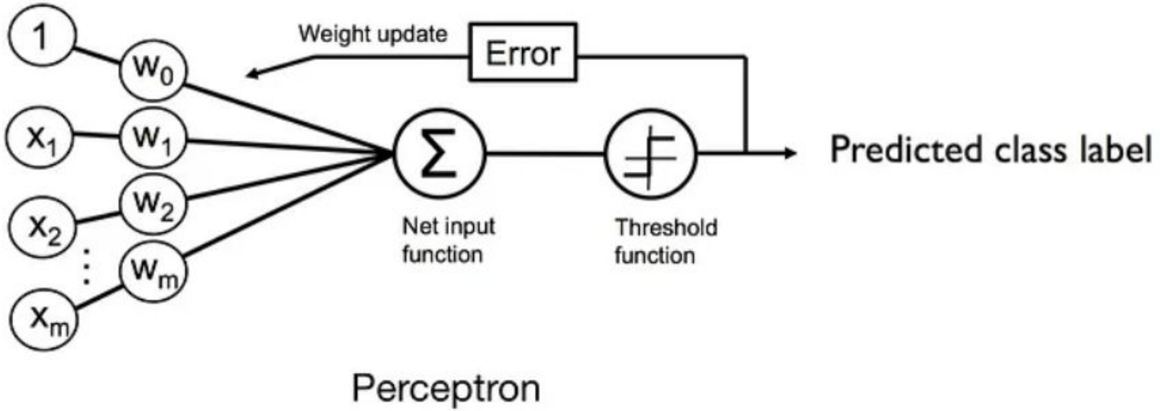
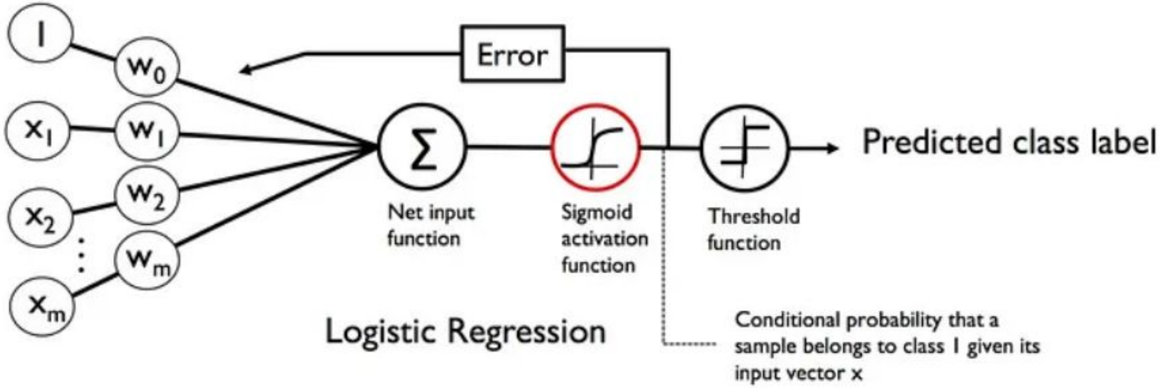
Artificial Neural Network (ANN) training

Dr. Sagarika B

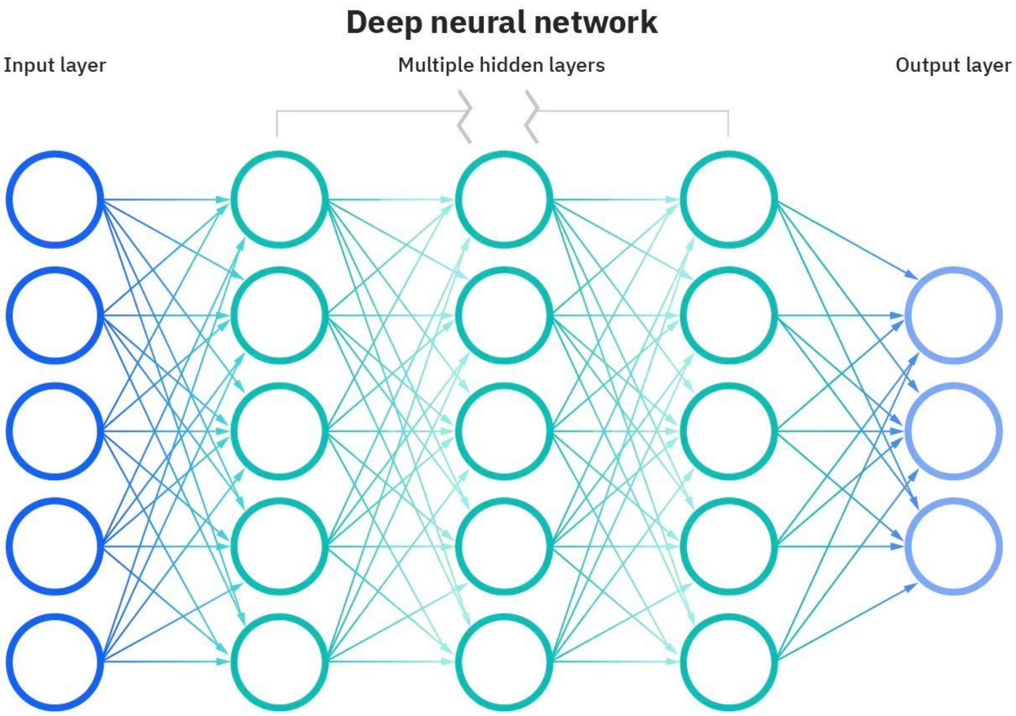
Single layer neuron / perceptron





Multi layer perceptron or Artificial Neural network

1. The **Input** layer (interface to accept data)
2. The **Hidden** layer(s) (Actual Neurons)
3. The **Output** layer (Actual Neurons to output the result)



Session 2

- Prerequisite concepts
- Feed Forward network
- Back propagation step by step
- Optimization

Prerequisites for neural networks training

- Linear combination of weights, input and biases
  - Vector dot product
- Activation functions
- Loss functions
  - Concept of derivatives
  - Learning rate

Loss functions used in neural networks

- Regression
  - MSE

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

- Binary Classification
  - Binary cross Entropy

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log(1 - \hat{Y}_i))$$

- Multiclass Classification
  - Categorical cross entropy
  - Sparse categorical cross entropy

$$CCE = -\sum_{i=1}^{n} y_i \cdot \log(f(X_i))$$

Forward propagation and Back propagation

1. Select the input neurons
2. Initialize weights(W) and biases(b) (random, specific values)
3. Forward propagation with weights and biases and activation functions
4. Predict the outcome
5. Compute loss
6. Update weights and biases using optimization method
    a. Gradient descent

Step 1: Input Neurons

To predict the strength of cement based on the composition

1st Input

cement	slag	ash	water
141.3	212.0	0.0	203.5
168.9	42.2	124.3	158.3
250.0	0.0	95.7	187.4
266.0	114.0	0.0	228.0
154.8	183.4	0.0	193.3

Input dimension = 4

So 4 neurons in the input layer

Step 2 : Weight initialization

Let’s initialize the weights with value 1

Input dimension = 4

4 weight values need to be initialized.

Let’s initialize bias with 0. (for the class example !!)

Step 3: Forward propagation and predict the outcome

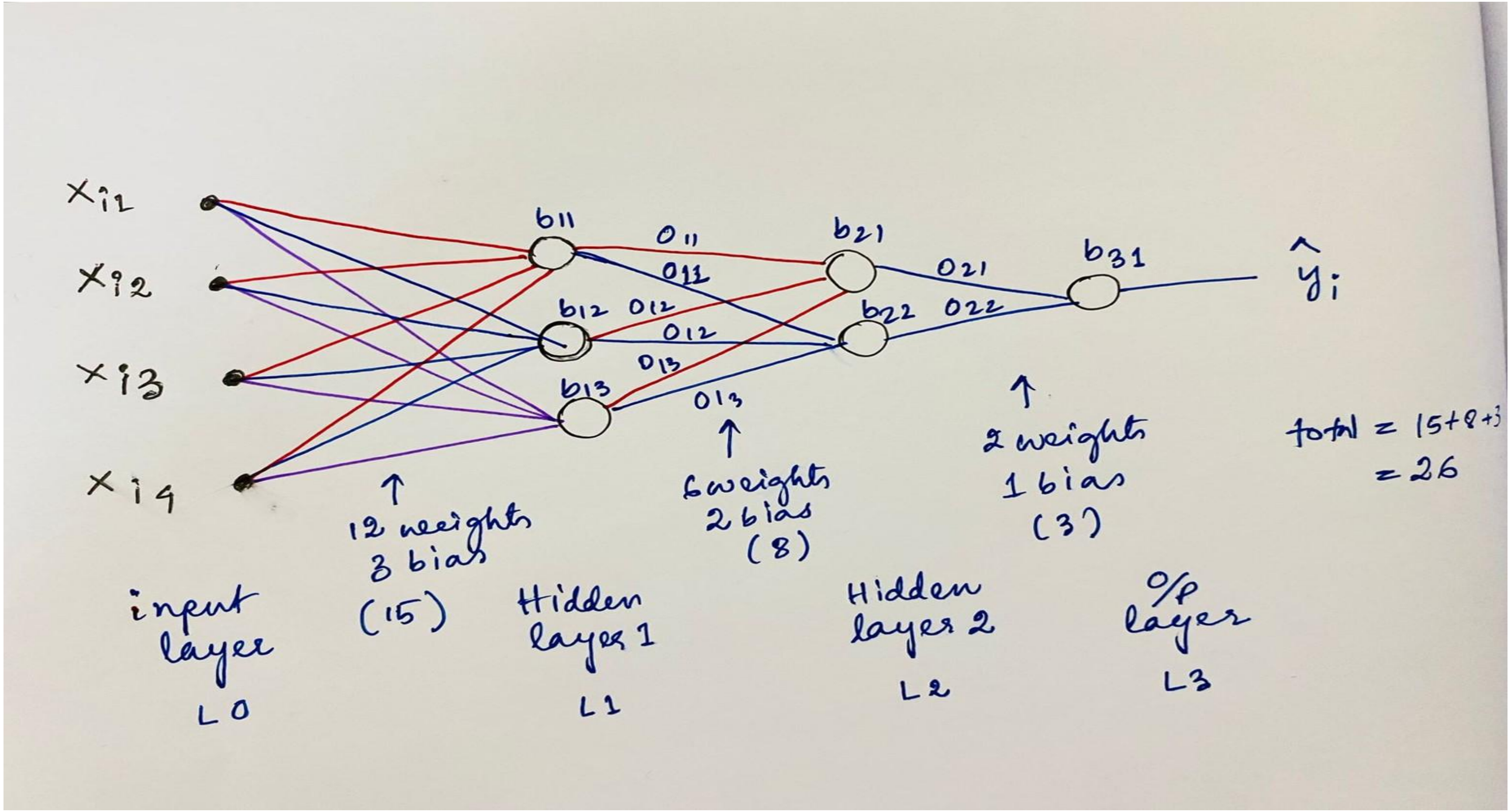
- Vector dot product
- Activation function
    - Hidden layers (Linear)
    - Output layer (Linear)

    (for our example Let's take linear activation function in all the Hidden layers)

    $f(x)=x$

- Number of learnable parameters

Feed forward propagation (example)



Notations

b_{ij} = bias for layer i node j

O_{ij} = Output for layer i node j

w_{ij}^{k} = Weight for node i of previous layer to node j of next layer

k = layer number of next layer.

Calculations

<u>Layer 1</u>

$$\begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ w^1_{21} & w^1_{22} & w^1_{23} \\ w^1_{31} & w^1_{32} & w^1_{33} \\ w^1_{41} & w^1_{42} & w^1_{43} \end{bmatrix}^T \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}$$

$$= \sigma\left(\begin{bmatrix} w^1_{11} x_{i1} + w^1_{21} x_{i2} + w^1_{31} x_{i3} + w^1_{41} x_{i4} \\ w^1_{12} x_{i1} + w^1_{22} x_{i2} + w^1_{32} x_{i3} + w^1_{42} x_{i4} \\ w^1_{13} x_{i1} + w^1_{23} x_{i2} + w^1_{33} x_{i3} + w^1_{43} x_{i4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix} \right)$$

$$= \begin{bmatrix} o_{11} \\ o_{12} \\ o_{13} \end{bmatrix}$$

Layer 2

$$\begin{bmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \\ w^2_{31} & w^2_{32} \end{bmatrix}^T \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix} + \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix}$$

$$3 \times 2 \xrightarrow{T} 2 \times 3 \qquad 3 \times 1$$

$$2 \times 1 \qquad + 2 \times 1$$

$$= \sigma\left(\begin{bmatrix} w^2_{11} O_{11} + w^2_{21} O_{12} + w^2_{31} O_{13} + b_{21} \\ w^2_{12} O_{11} + w^2_{22} O_{12} + w^2_{32} O_{13} + b_{22} \end{bmatrix} \right)$$

$$= \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix}$$

Layer 3

$$\begin{bmatrix} w^3_{11} \\ w^3_{21} \end{bmatrix}^T \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix} + \begin{bmatrix} b_{31} \end{bmatrix}$$

$$(2 \times 1)^T \longrightarrow 1 \times 2 \quad 2 \times 1 \cdot \quad 1 \times 1$$

$$1 \times 1$$

$$= \sigma\left(\left[w^3_{11} O_{21} + w^3_{21} O_{22} + b_{31} \right]\right) = \hat{Y}_i$$

Step4: Compute loss

- Regression
  - MSE

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

Step 5 : Update weights and biases using optimization method (Back propagation)

- Compute derivative of loss with respect to weights and biases
- Chain rule
- Update the weights and biases

$$w_{new} = w_{old} - \eta \frac{dLoss}{dw_{old}}$$

A major problem faced by back propagation !!

Vanishing gradient problem

As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.

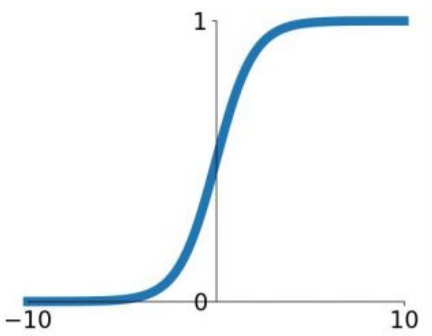
By the chain rule, the derivatives of each layer are multiplied down the network (from the final layer to the initial) to compute the derivatives of the initial layers.

Requirements of an Activation function

- Non-linear
- Differentiable
- Computationally less expensive
- Zero centered
- Non saturating

Activation functions

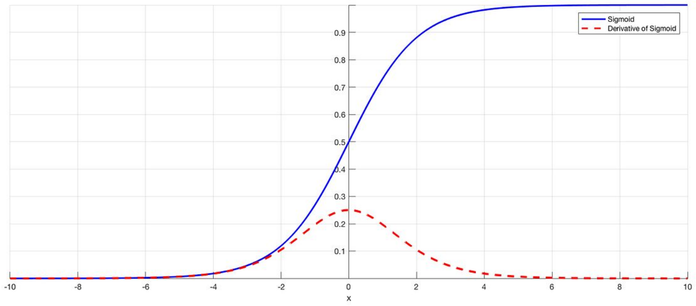
Sigmoid



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

It squashes the input of any range to (0,1)

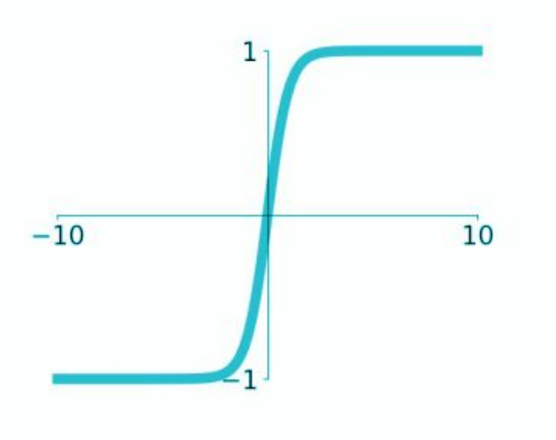
Derivative of sigmoid



Problems with sigmoid:

1. Vanishing gradient
2. Output is not zero centered
3. Exponential computation

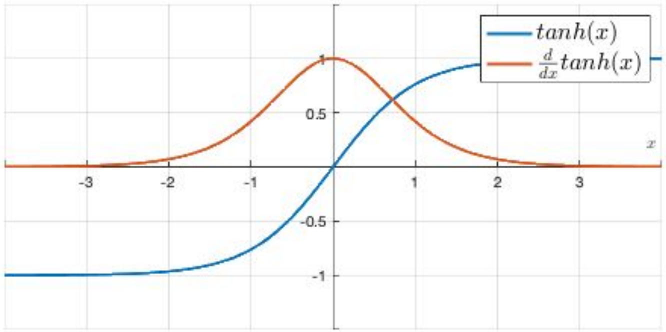
Tanh (Hyperbolic tangent)



$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

It squashes the input of any range to (-1,1)

Derivative of tanh

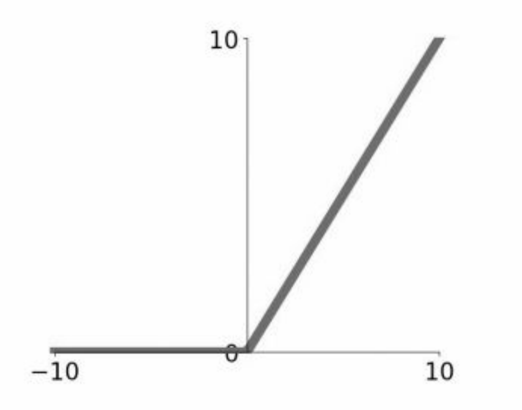


Output is zero centered

Problems with Tanh:

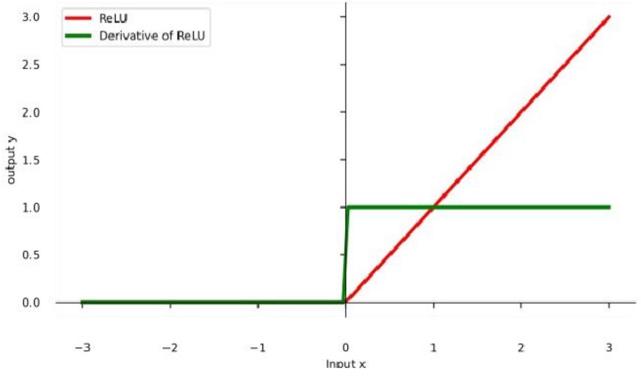
1. Vanishing gradient
2. Exponential computation

Relu



$$max(0, x)$$

Derivative of Relu

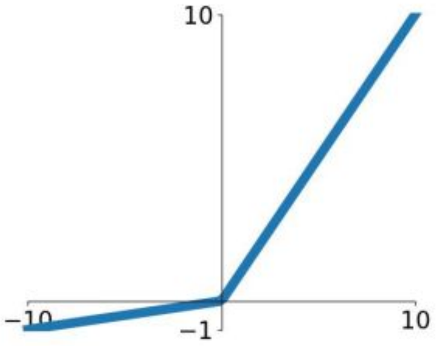


1. Non linear
2. Non saturating (vanishing gradient problem is resolved)
3. Computationally inexpensive
4. Converges faster

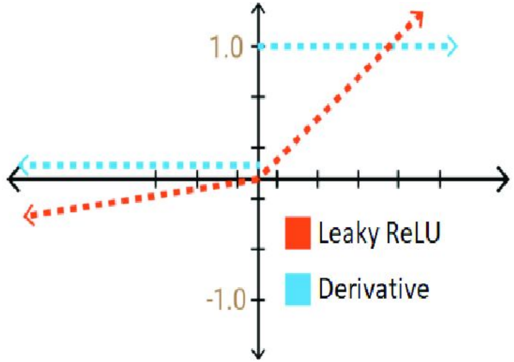
Problems

1. Not fully differentiable
2. Dying Relu problem
3. Not zero centered

Leaky Relu



$$max(0.01x, x)$$



1. Non linear
2. Non saturating (vanishing gradient problem is resolved)
3. Computationally inexpensive
4. Converges faster
5. Dying Relu problem is resolved

Softmax

- Used in the output layer for multiclass classification problems
- It converts the raw values into vector probabilities
- Softmax outputs values ranged between (0,1)

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Visualize

https://playground.tensorflow.org/#activation=sigmoid&batchSize=10&dataset=gauss&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=1&seed=0.97158&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

Case study

- Regression (Admission prediction/ concrete strength)
- Binary classification (cat vs dog)
- Mnist digit classification