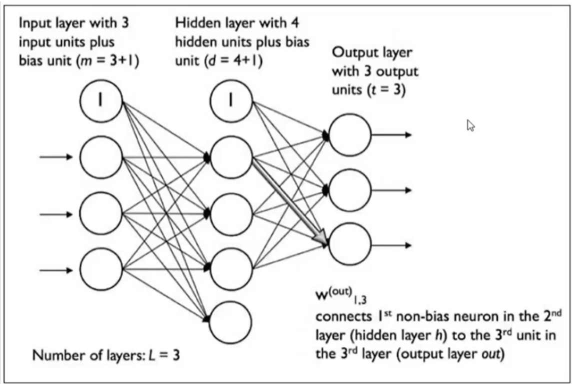
Multi-Layer Perception (MLP)

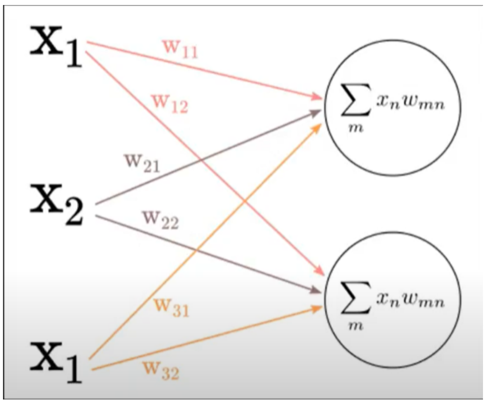
Junaid Mahmud

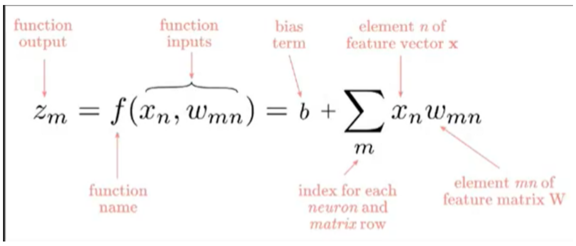


⇒ When multiple perceptrons are stacked together one after another, it is called a multi-layer perceptron.

Components of MLP

⇒ Linear Function





$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$$

$$z = W^T x$$

$$= \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$= \begin{bmatrix} x_1 w_{11} + x_2 w_{12} + x_3 w_{13} \\ x_1 w_{21} + x_2 w_{22} + x_3 w_{23} \end{bmatrix}$$

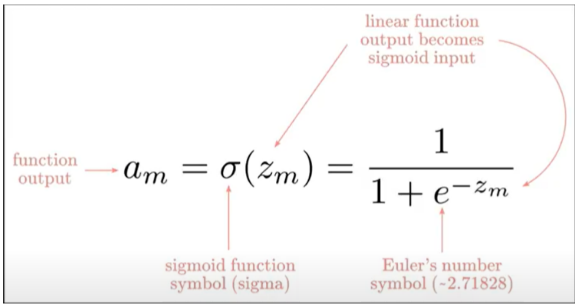
- Linear Function/Aggregation: Input will be given, weights will be associated, and we will get the sum.

- W is a matrix here.
- X is a vector.

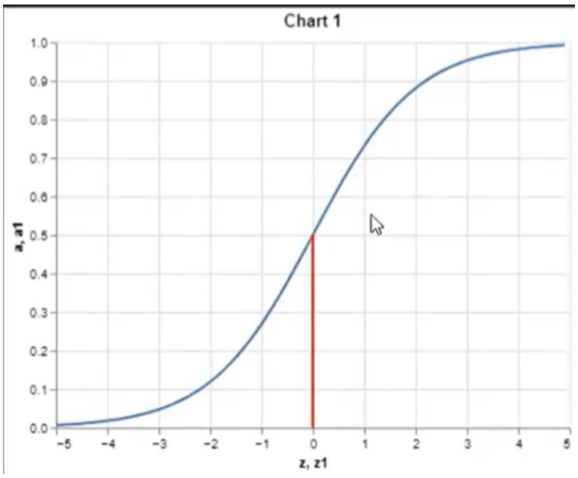
- Z is the output of a linear function.

$$= \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

Sigmoid Function



$$a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$



- The mid-portion is "kind of" linear.
- But the whole function has non-linearity. It's getting saturated on both ends.
- Linear models can not do well with the data that is not linearly separable.

- Requirements for AN
  1. Aggregation Function
  2. Activation function
- If the input is not strong, the activation function will not fire. The value of the activation function will not be **activated.**

Output Function

$$\hat{y} = f(a_m) \begin{cases} +1, & \text{if } a > 0.5 \\ -1, & \text{otherwise} \end{cases}$$

- Threshold function
- Binary Classification

$$\hat{y} = f(a_m) = a_m$$

- Identity Function
- Regression Problem

$$\hat{y} = \sigma(a)_i = \frac{e^{\beta a_i}}{\sum_{j=1}^{k} e^{\beta z_j}}$$

- Softmax Function
- Multiclass Classification

Cost Function

added to simplify gradient computation

predicted value for training example k

sum of squared errors for the entire dataset

$$E = \frac{1}{2} \sum_{k} (a_k - y_k)^2$$

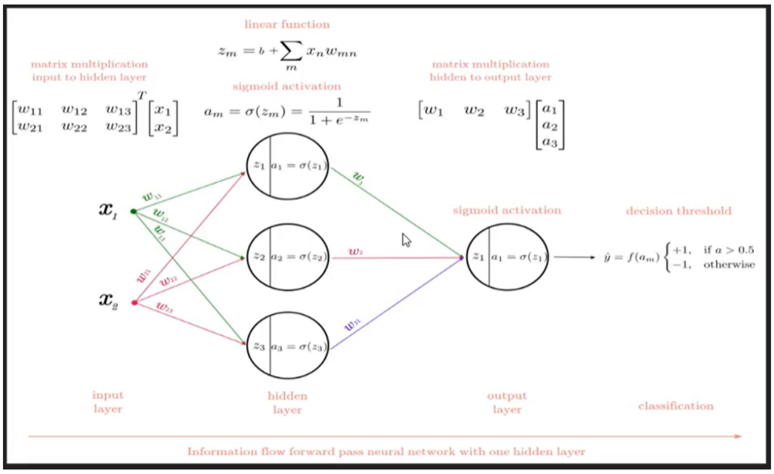
index for input-output pair

expected value for training example k

- Loss Function
- Cost Function
- Objective Function

i. mean squared error
ii. sum of squared error
iii. binary cross entropy

Forward Propagation

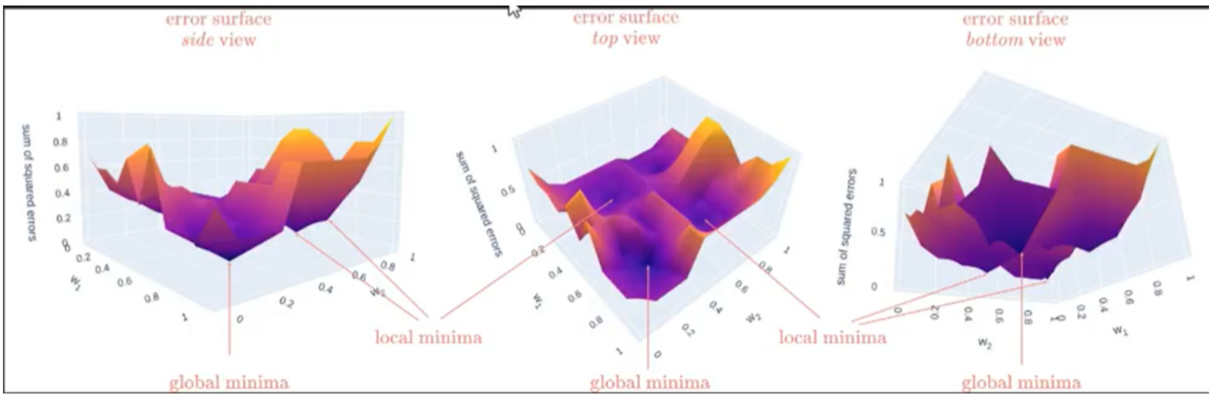


Forward Propagation/Forward Pass

- Each layer has an output and an activation function.
- Weight matrix(**w,** transposed) and feature matrix(**x**).
- z value is passed through the activation function. Then prediction is based on threshold value.

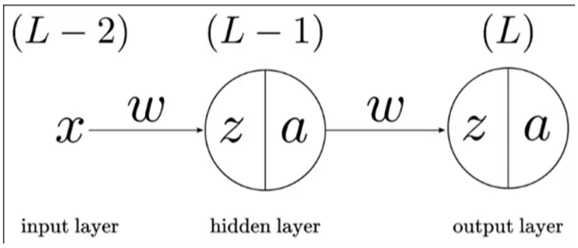
Backpropagation Algorithm

- Minimize the error via gradient descent
- Convex and non-convex optimization, Introducing nonlinearities.
- Multiple "**valleys" with "local minima".**



- We have to find the global minima/minimum from multiple local minima.

Backprop for multilayer single-unit perceptron



$$E = \frac{1}{2}\sum_k (a_k - y_k)^2$$

- How the error E change when we change the activation by **a** tiny amount.
- How the activation **a** changes when we change the a0ctivation z by a tiny amount.
- How **z** changes when we change the weights w by a tiny amount.

partial derivative symbol (stylized d)

partial derivative of the error w.r.t. the weight

$$\frac{\partial E}{\partial w^{(L)}} = \overbrace{\frac{\partial E}{\partial a^{(L)}}}^{\text{partial derivative of the error w.r.t. sigmoid activation}} \times \underbrace{\frac{\partial a^{(L)}}{\partial z^{(L)}}}_{\text{partial derivative of sigmoid activation w.r.t. linear function}} \times \overbrace{\frac{\partial z^{(L)}}{\partial w^{(L)}}}^{\text{partial derivative of linear function w.r.t. the weight}}$$

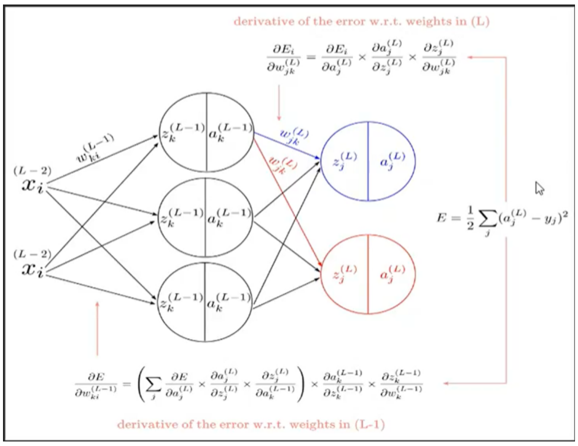
index for the current layer

$$\frac{\partial E}{\partial w^{(L-1)}} = \frac{\partial E}{\partial a^{(L)}} \times \frac{\partial a^{(L)}}{\partial z^{(L)}} \times \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \times \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \times \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}$$

$$\frac{\partial E}{\partial w^{(L-1)}} = \frac{\partial E}{\partial a^{(L)}} \times \frac{\partial a^{(L)}}{\partial z^{(L)}} \times \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \times \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \times \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}$$

- How do I learn the **bias** value?

Backpropagation for multi-layer multi-unit perceptron



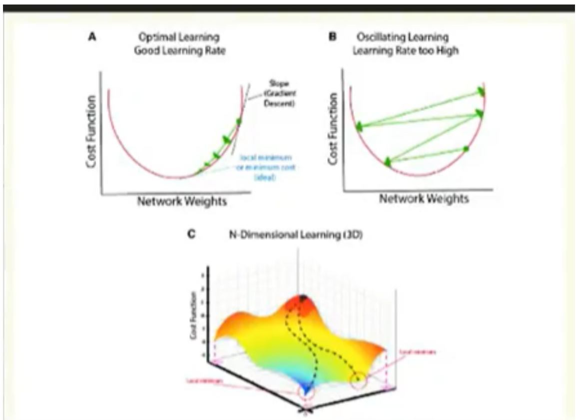
- Modern Neural Nets
- Shallow networks
- Feed forward neural network
- Artificial Neural Networks
- ANN
- Fully Connected Neural Networks
- FCNN
- FC

MLP Learning Rule

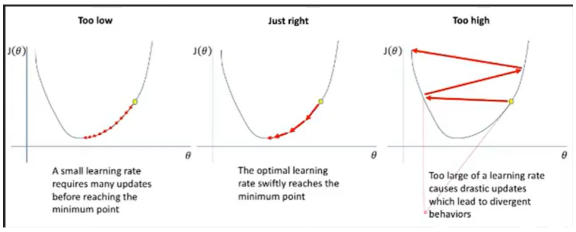
$$w_{jk}^{L} = w_{jk}^{L} - \eta \times \frac{\partial E}{\partial w_{jk}^{L}}$$

$$b^{(L)} = b^{(L)} - \eta \times \frac{\partial E}{\partial b^{(L)}}$$

- We will get derivatives from the backpropagation.
- Weights will be updated based on derivatives.
- There is a learning rate multiplied by the derivative.



Finding the global minima

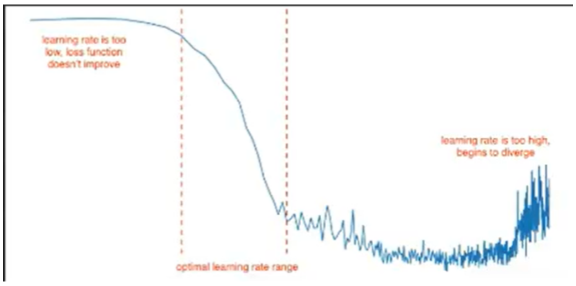


How learning rate affect the weights

- The optimal point / global minimum will be found only when the conditions are met. So, **learning rate** affects the weight.

- **We may miss the expected weight or global minima if the learning rate is high.**
- If the **learning rate is small, more time will take to converge.**

⇒ Optimal **Learning Rate:** Start with a high value, which will get lower with the steps.



Plot of learning rate

MP neuron vs. Perceptron vs. ADALINE vs. MLP

MP neuron, 1943	Perceptron, 1958	ADALINE, 1960	MLP, 1986
Linear aggregation f	Linear aggregation f	Linear aggregation f	Linear aggregation f
No weight	Weighted sum	Weighted sum	Weighted sum
No bias	No bias	No bias	Bias
Manual threshold f	Heaviside step f	Heaviside step f	Sigmoid f
Linear units	Linear units	Linear units	Non-linear units
Cannot solve XOR	Cannot solve XOR	Sub-optimal solution	Can solve
No Learning rule	Simple learning rule	Learning rule on GD	Learning rule on BP
No LR	No LR	LR	LR
Slowest	Faster	More Faster	Fastest