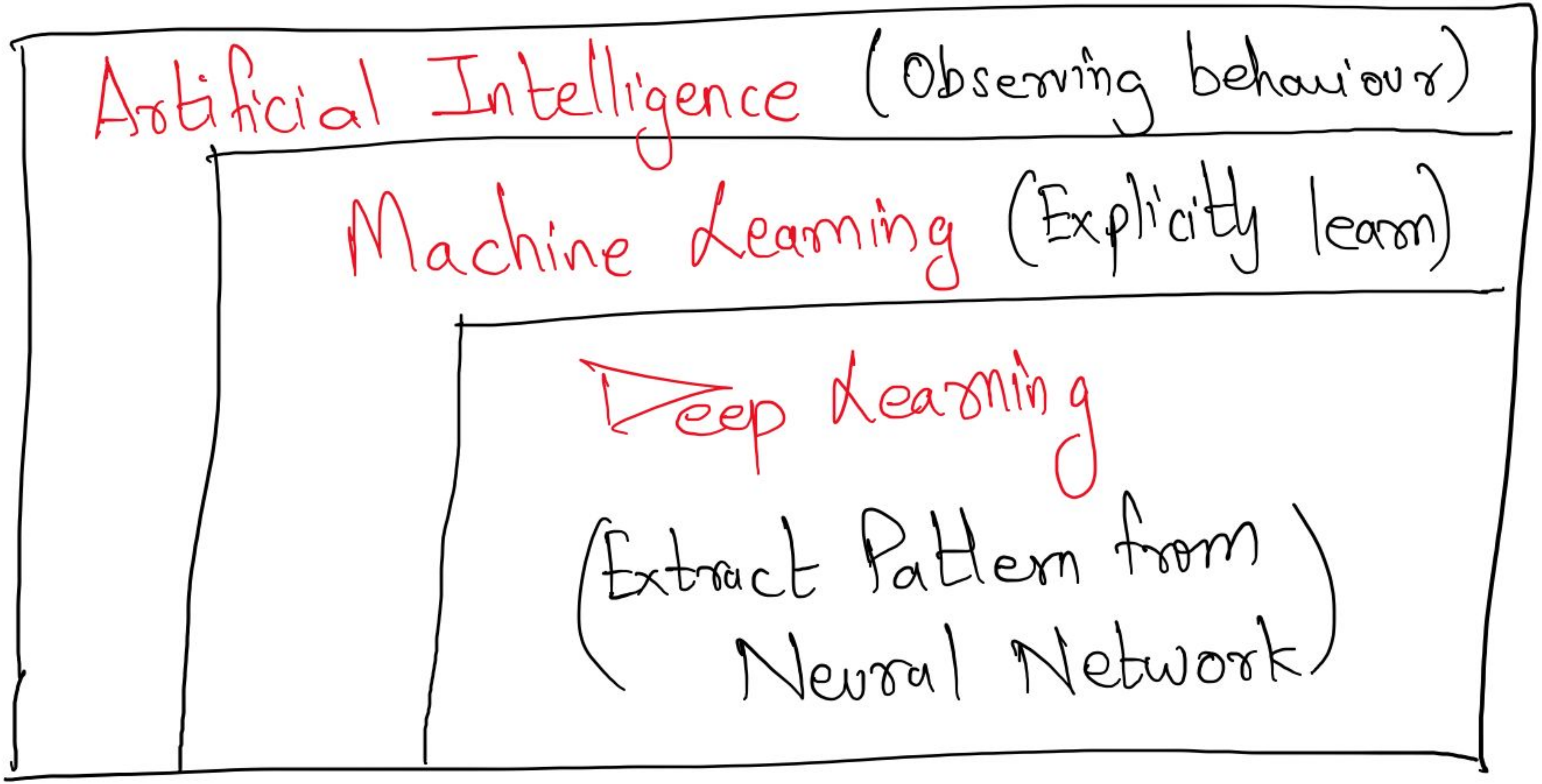
Neural Network and Deep Learning

B.Tech. Data Science, NMIMS

By,

Bilal Hungund, Data Scientist, Halliburton


Artificial Intelligence (Observing behaviour)
Machine Learning (Explicitly learn)
Deep Learning
(Extract Pattern from Neural Network)

Why Deep Learning

- → Time Consuming and brittle in Unstructured data
- → Hard engineered features are not scalable
- → Ways to learn underlying features for unstructured data

Evolution of Deep Learning

1952 Gradient Descent

1958 Perceptron

⋮

1986 Backpropagation

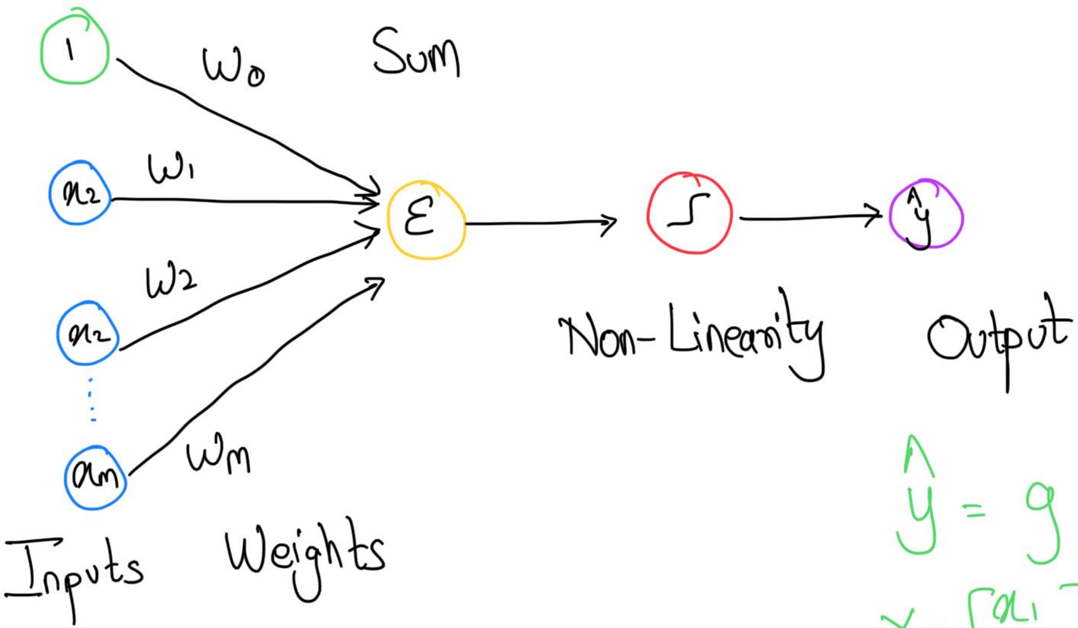
1995 Deep Convolution NN

⋮

Why now?

1) Big Data
2) Hardware (GPUs, TPUs)
3) Software (Pytorch, Tensorflow)

Perceptron

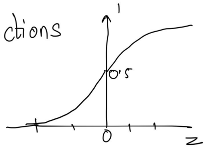


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

$g(z) = \sigma(z)$

Activation Functions

$$\frac{1}{1+e^{-z}}$$



Simplify to Matrix

$$\hat{y} = g\left(w_0 + X^T W\right)$$

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

Hyperbolic Tangent (tanh)

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

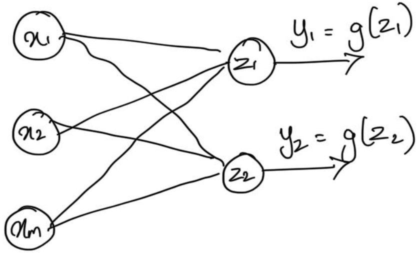
$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)

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

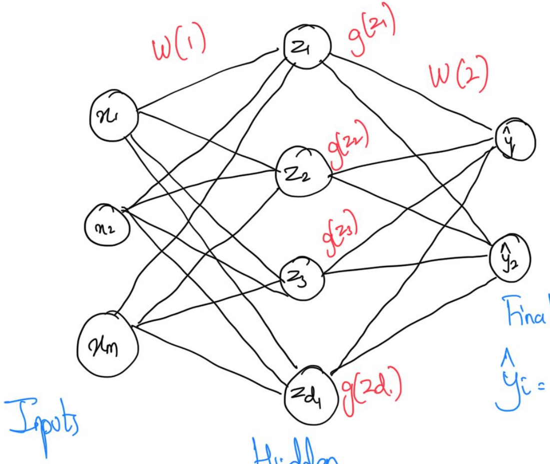
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Multi Output Perceptron



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$$

Single Layer Neural Network



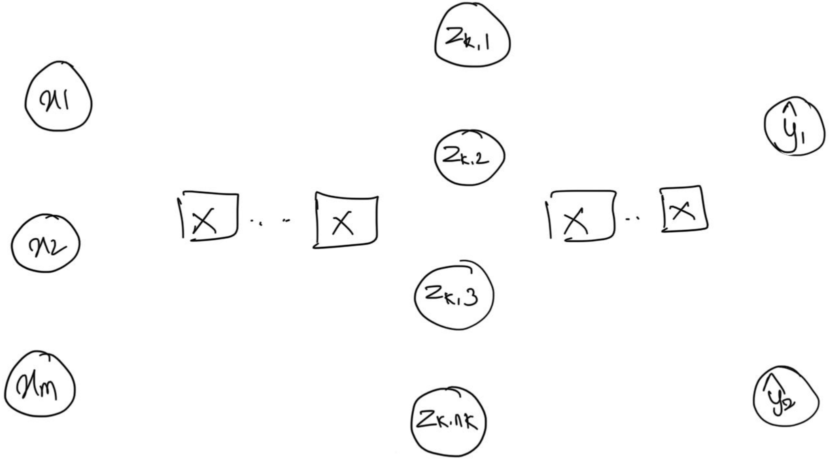
Final Output

$$\hat{y}_i = g\left(w_{0,i} + \sum_{j=1}^{d_1} g(z_j)\, w_{j,i}^{(2)}\right)$$

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j\, w_{j,i}^{(1)}$$

Deep Neural Network

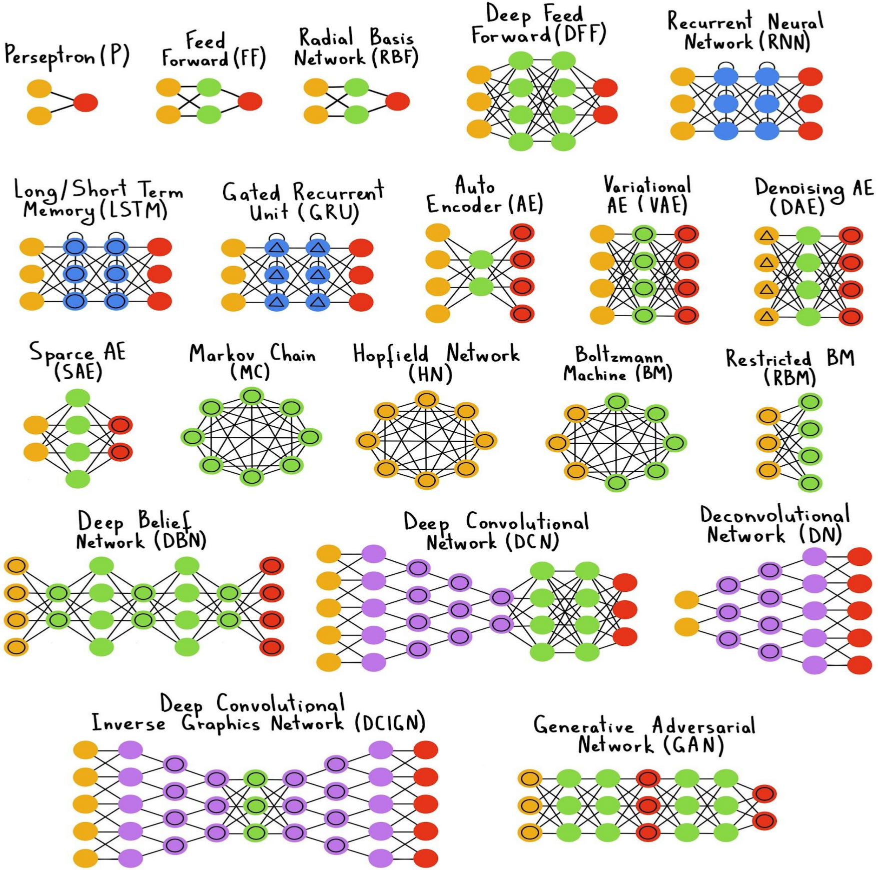
k # hidden layer



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Keep stacking hidden layers

Neural Networks



ABACUS.AI

Quantifying Loss

It measures the cost incurred from incorrect predictions

$$\underbrace{\mathcal{L}\Big(\underbrace{f\big(x^{(i)};\, w\big)}_{\text{Predicted}},\ \underbrace{y^{(i)}}_{\text{Actual}}\Big)}_{J(w) \rightarrow \text{Empirical Loss}}$$

Binary Cross Entropy Loss

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actuals}} \underbrace{\log\big(\hat{y}^{(i)}\big)}_{\text{Predicted}} + \underbrace{\big(1 - y^{(i)}\big)}_{\text{Actuals}} \underbrace{\log\big(1 - \hat{y}^{(i)}\big)}_{\text{Predicted}}$$

Mean Squared Error Loss

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} \big(\underbrace{y^{(i)}}_{\text{Actuals}} - \underbrace{\hat{y}^{(i)}}_{\text{Predicted}}\big)$$

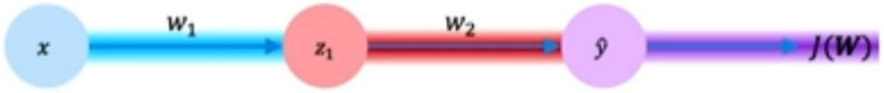
$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

Optimization

- Learning Rate
- Regularization
- Dropout
- Early Checkpoint