upGrad

#LifeKoKaroLift

GLOBAL PROGRAM

IN

MACHINE LEARNING

AND

ARTIFICIAL INTELLIGENCE

upGrad

SESSION : INTRODUCTION TO NEURAL NETWORKS

Instructor : SACHIN KUMAR
Master Trainer (UpGrad)

Agenda

upGrad

- Introduction
  - Machine Learning & AI
  - Problems faced by traditional ML algorithms
  - Application on Neural Networks
- Perceptron – The building block
  - What is a Perceptron?
  - Activation functions
- Structure of an ANN & Feedforward in Neural Networks
  - How does an ANN look like?
  - Notations used for layers, input, outputs etc
  - Different inputs & outputs to a Neural
  - Feedforward in an Artificial Neural Network
  - Types of Neural Networks
- Backpropagation in an ANN
  - Training an ANN
  - Cost Function for Regression/Classification
  - Gradient Descent
  - Feedforward and Backprop equations
- TensorFlow playground
  - Building an ANN
  - Training an ANN
  - Changes in Parameters
- Doubt resolution

Artificial Intelligence/Machine Learning /Deep Learning

upGrad

ARTIFICIAL INTELLIEGENCE

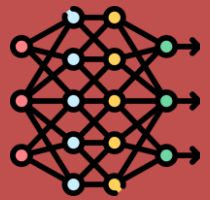
Techniques that help computer mimic "cognitive" functions associated with the human mind,

MACHINE LEARNING

Ability to learn pattern from data without being explicitly programmed

DEEP LEARNING

Extract pattern from data using Deep Neural Networks



Quest to Build Intelligent Machines

upGrad

Easy for computers to do:

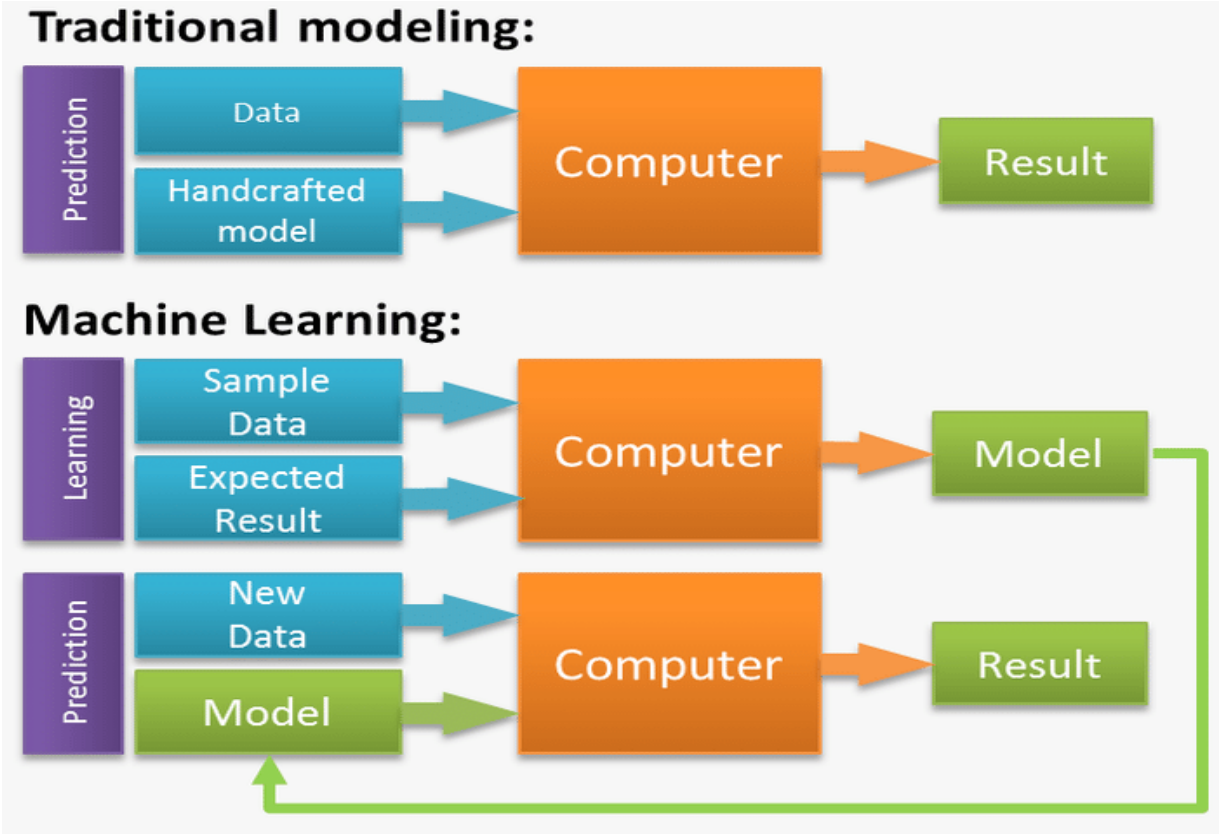
- ✓ Extensive mathematical computations
- ✓ Querying from a large corpus of text
- ✓ Combinatorial like Chess playing

Hard for computers to do:

- Understand spoken language
- Recognize images and faces
- Recognize feelings
- Learning from few mistakes
- Generate New Ideas

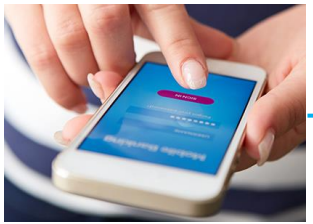
Machine Learning was a Paradigm Shift

upGrad



Machine Learning can Solve Many Problems

upGrad



Demographics
Usage, Billing,
Complaints,
Marketing

Train a
Classification
Model

High Risk

Moderate
Risk

Low Risk

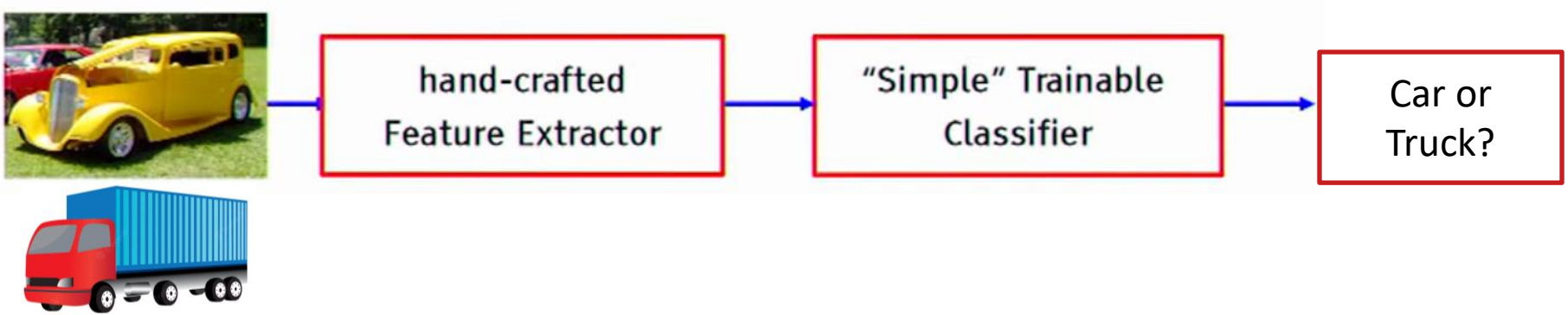
Location, Area(Sq Ft),
Rooms,
Distance to School,
Villa/Apartment

Train a
Regression
Model

Price of House

But Not for All Class of Problems

upGrad

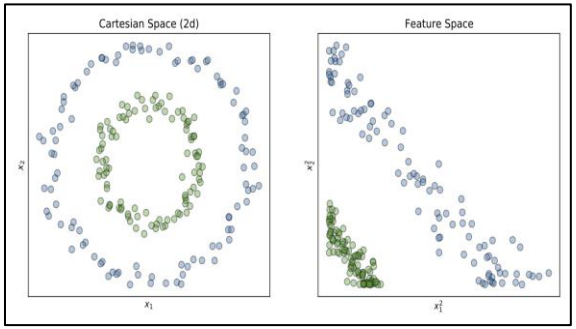


The challenge to Machine Learning proved to be solving the tasks that are easy for people to ***perform***, but hard for people to ***describe*** formally.

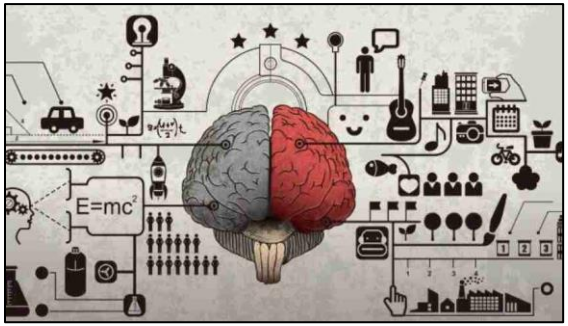
Challenges with Traditional Machine Learning

upGrad

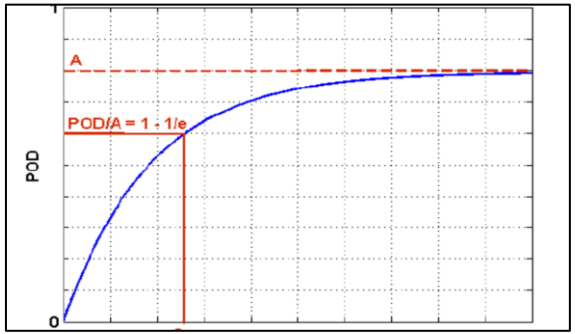
Require domain expertise to handcraft useful features



Results heavily depend on Feature Representation



Cannot solve extremely complex problems



Has an upper limit to Performance

Deep Learning (Using ANN) as an Answer

upGrad

Learning hard-to-describe problem:

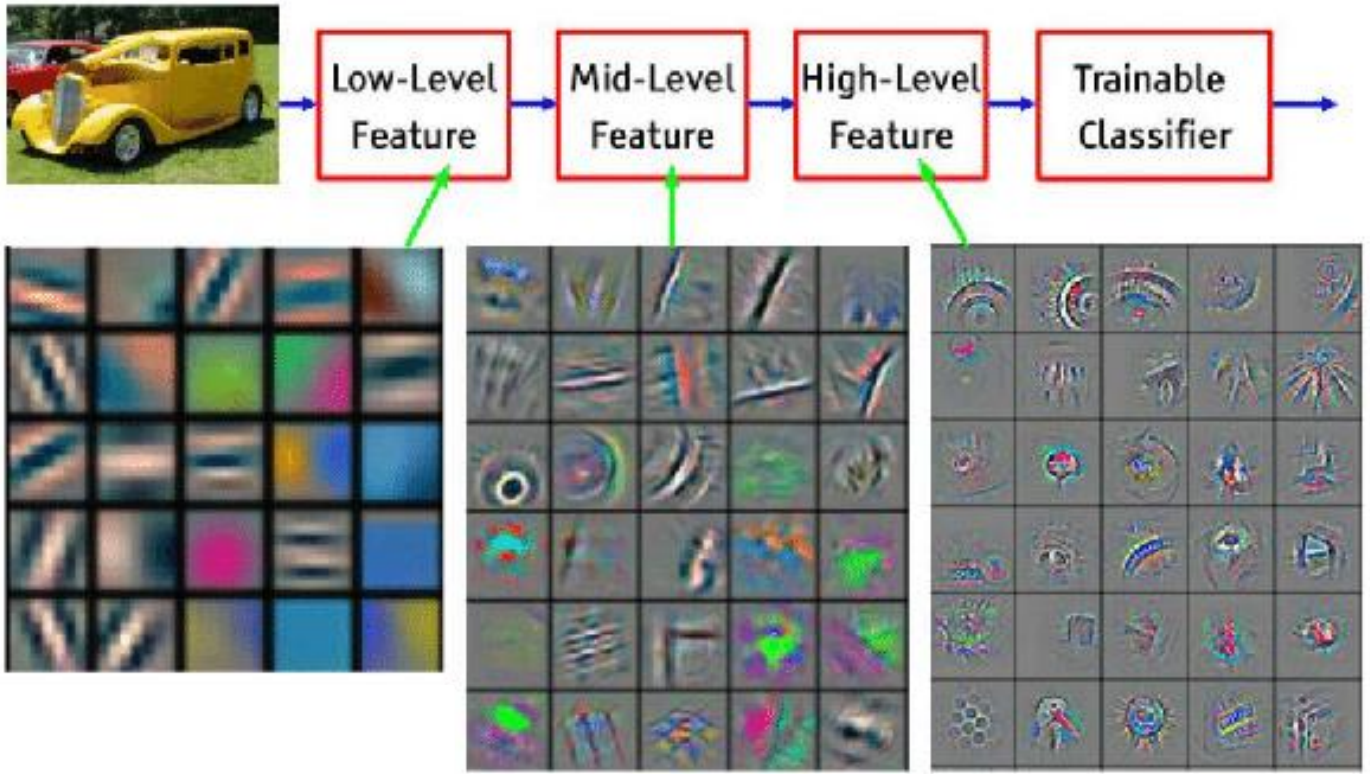
- Allow computers to learn from experience
- Understand the world in terms of a hierarchy of concepts
- Each concept defined through its relation to simpler concepts.

Advantages:

- No need for human operators to formally specify all the knowledge that the computer needs
- The hierarchy of concepts enables the computer to learn complicated concepts by building them out of simpler ones.

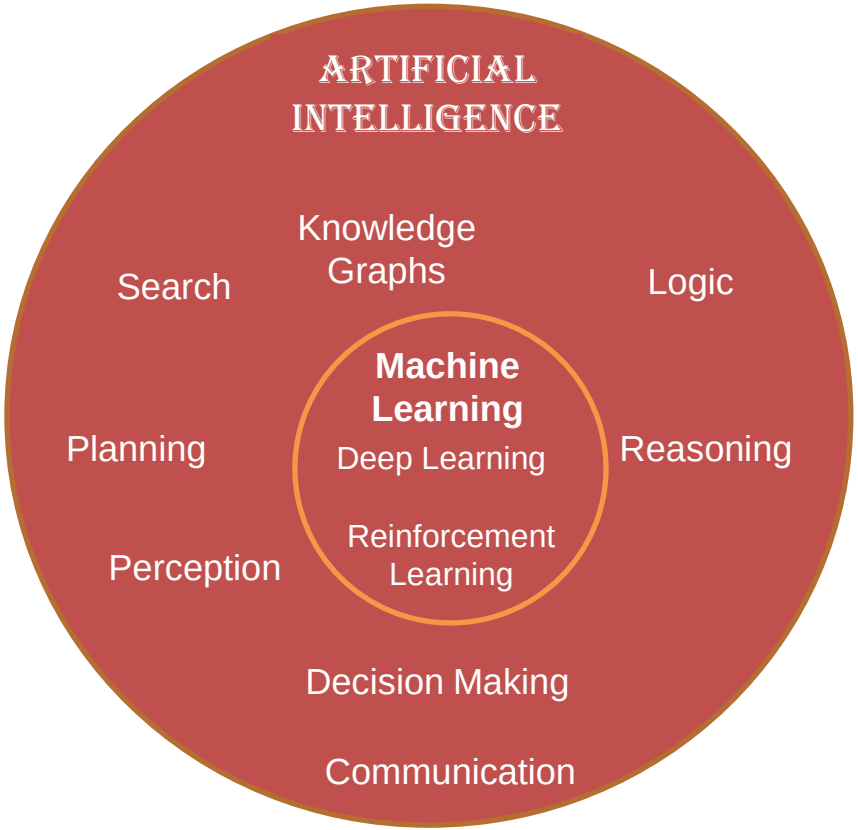
Deep Learning aka Hierarchical Feature Learning

upGrad



Artificial Intelligence is a broad set of Technology

upGrad



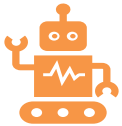
Ability of Machines to mimic "cognitive" functions that we associate with the human mind, such as "learning, seeing, interpreting and problem solving"

Examples of Artificial Intelligence Applications

upGrad

Computer Vision

Speech Recognition



Personal Assistant

Medical Diagnosis

Chatbots

Assisted driving Cars

Applications of (Narrow)AI

upGrad

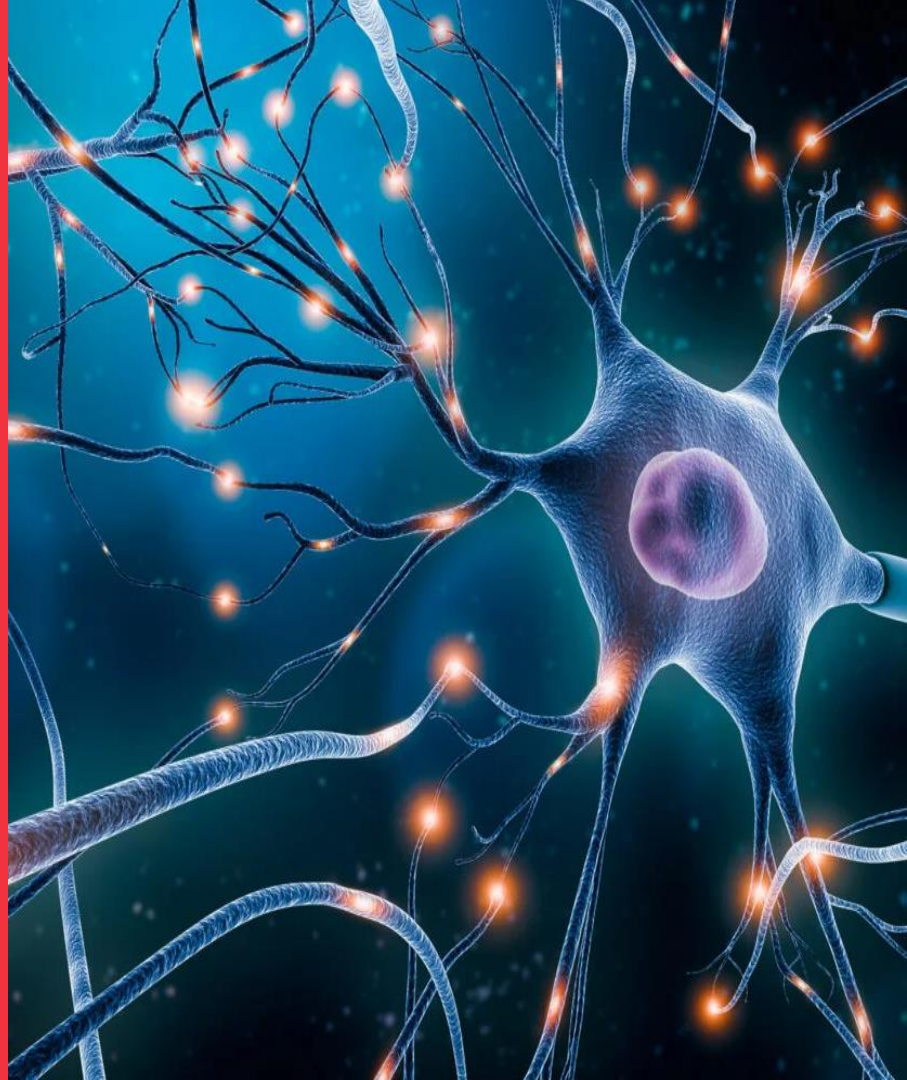
- Text Data
  - Text Categorization
  - POS/NER/Semantics
  - Text Summarization
  - Text Generation
  - Language Translation

- Image Data
  - Categorization
  - Character Recognition
  - Object Detection
  - Face Recognition
  - Image Generation

- Voice Data
  - Speech to Text
  - Speech Synthesis
  - Music Recognition

upGrad

Perceptron and Activation Functions

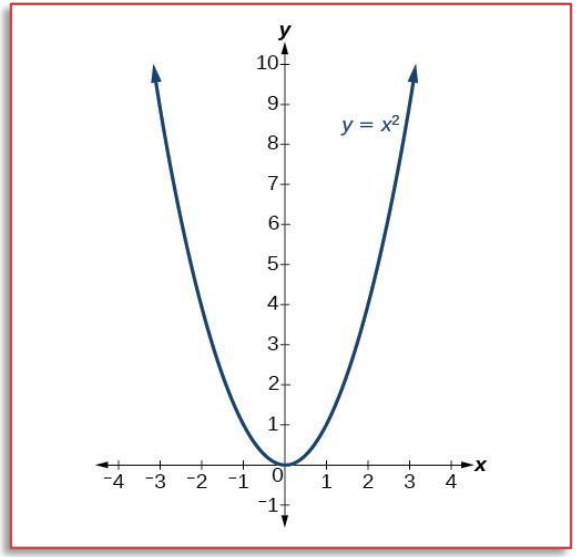


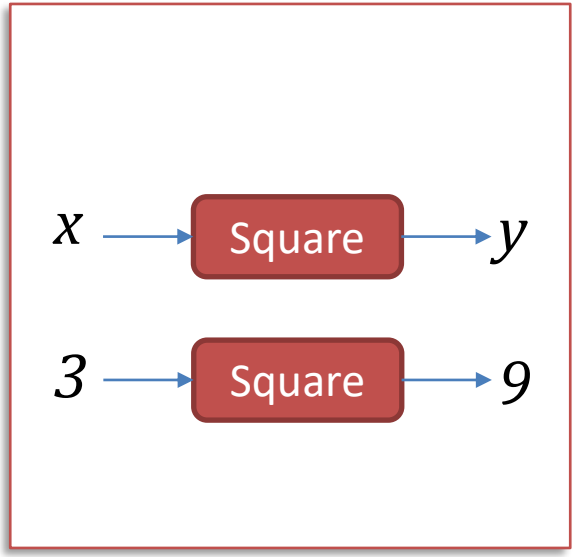
Representation of a Function

upGrad

$y = f(x)$

$f(x) = x^2$



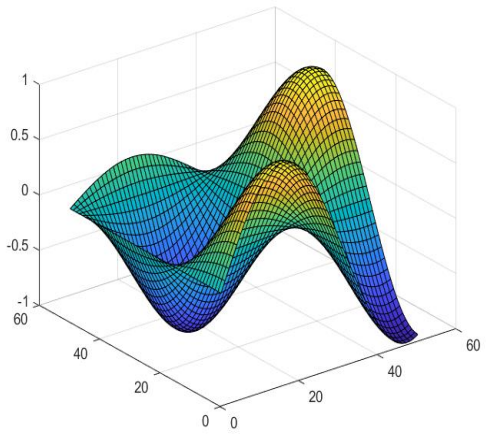


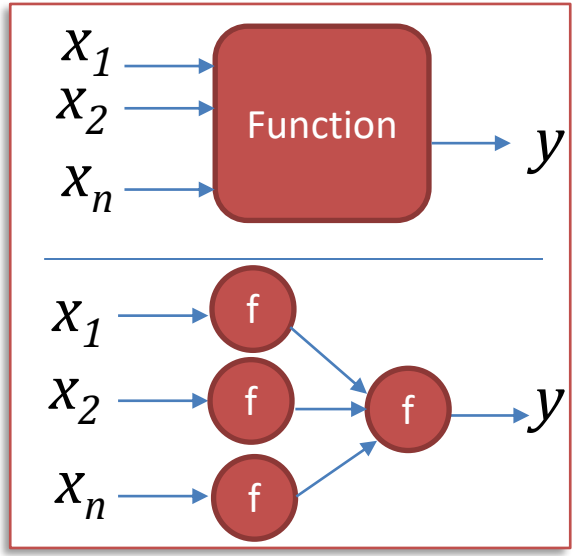
Representation of a Function – in Higher Dimension

upGrad

$$Y = f(W, X)$$

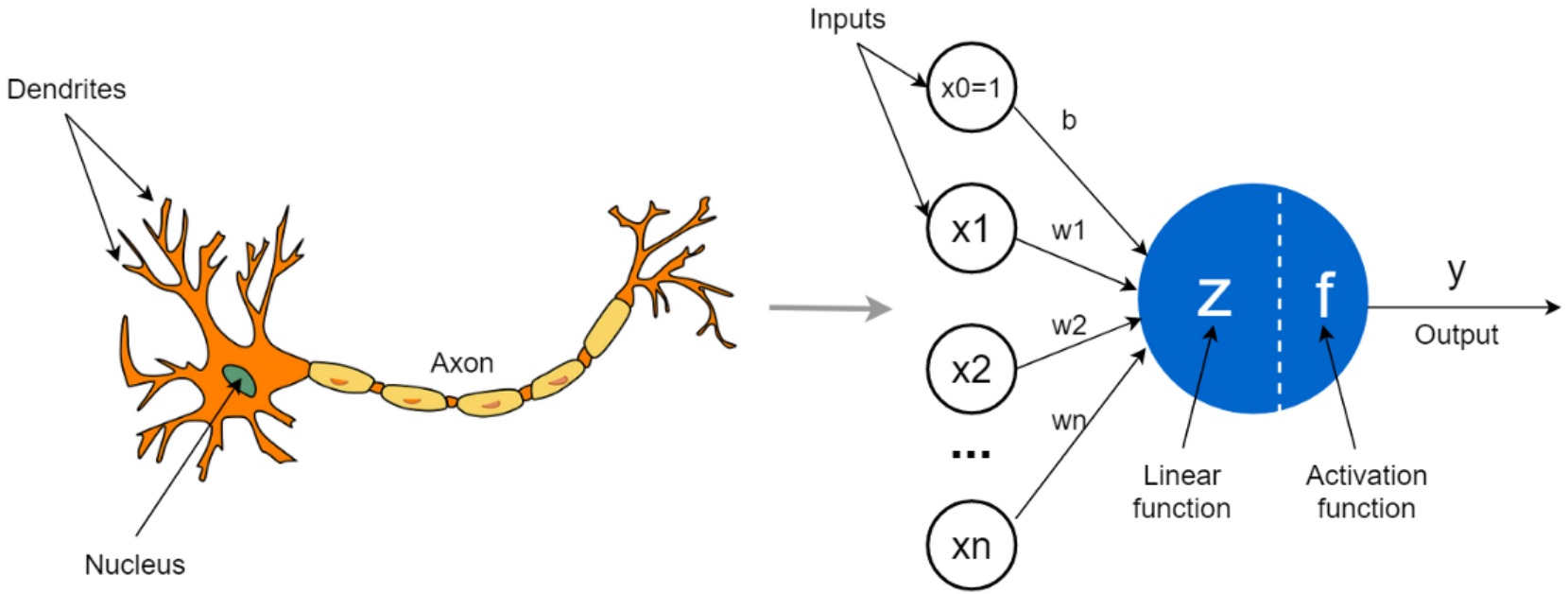
$$y_1 = w_1x_1 + w_2x_2 + \ldots + w_nx_n$$





Artificial Neural Network – Inspired by Nature

upGrad

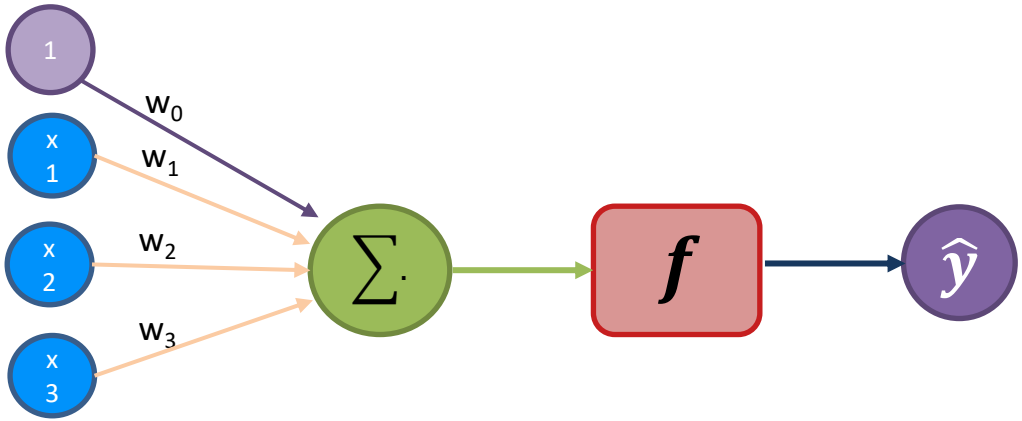


Biological Neuron

Artificial Neuron

A Closer Look At The Perceptron

upGrad



Inputs Weights *Summation* *Activation Function* *Predicted value*

Non-linear Function

$$\hat{y} = f(\sum x_i . wi)$$

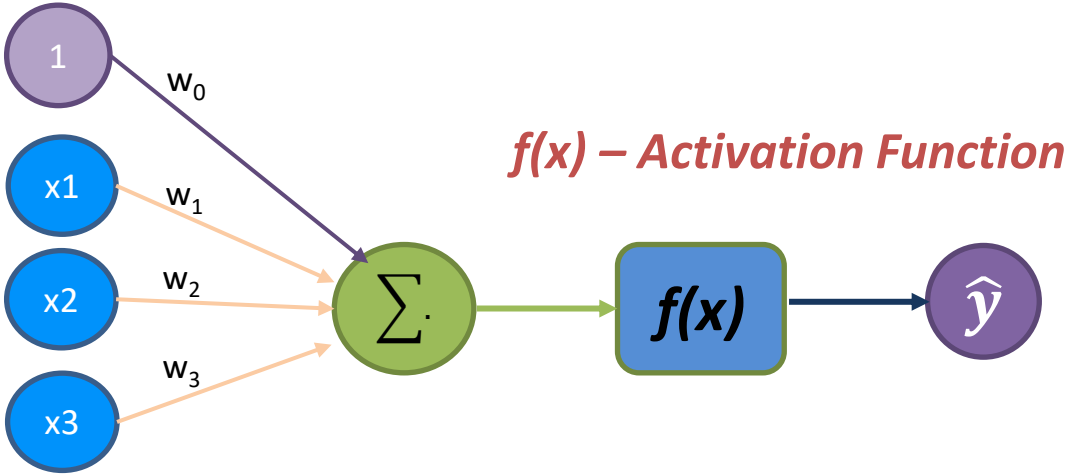
$$\hat{y} = f(w_0 + \sum x_i . wi)$$

$$\hat{y} = f(W^T . X) \qquad X = 1, x_1, x_{2,.}$$

FORWARD PROPOGATION

The Perceptron Can Take Many Forms

upGrad



f(x) – Activation Function

$$\hat{y} = f(w_0 + \sum x_i . wi)$$

Let, f(x) = 1 [Linear]

$Y = w_0 + w_1x_1 + w_2x_2 + w_3x_3$ [Linear Regression]

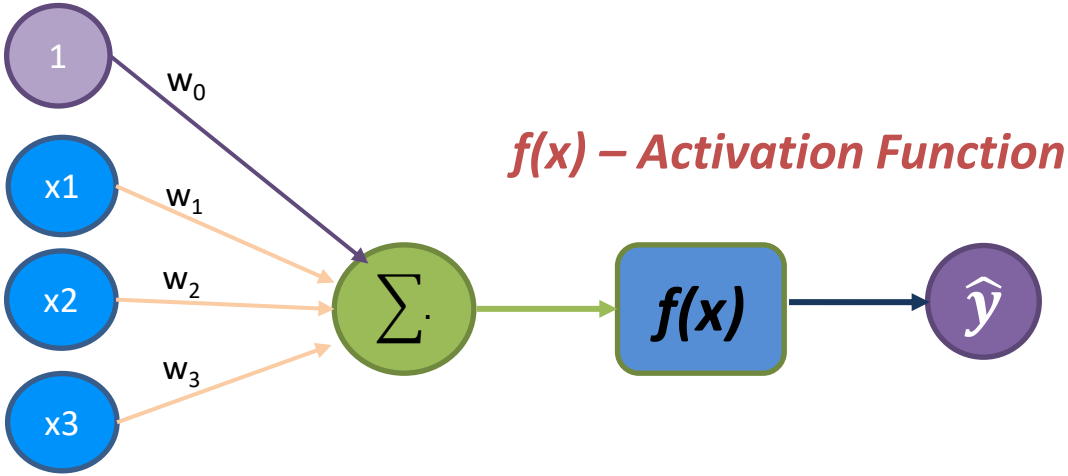
w_0= 1.2 w_1 = 1.0 w_2 = 1.25 w_3 = 2.5

x_1 = 0.5, x_2 = 0.3; x_3 = 0.7

Y = 1.2 + 1*0.5 + 1.25*0.3 + 2.5*0.7
= 3.825

The Perceptron Can Take Many Forms

upGrad



f(x) – Activation Function

$$\hat{y} = f(w_0 + \sum x_i . wi)$$

Let, f(x) = 1/(1+ e^{-x}) [Sigmoid]

$$Y = \frac{e^{(w0 + w1x1 + w2x2 + w3x3)}}{1+e^{(w0 + w1x1 + w2x2 + w3x3)}}$$

[Logistic Regression]

w_0= 1.2 w_1 = 1.0 w_2 = 1.25 w_3 = 2.5

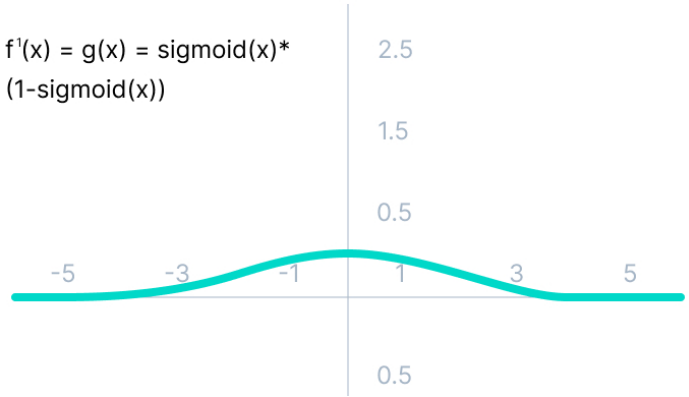
x_1 = 0.5, x_2 = 0.3; x_3 = 0.7

$$Y = \frac{e^{(3.825)}}{1 + e^{(3.825)}} = 0.978$$

Sigmoid Function

upGrad

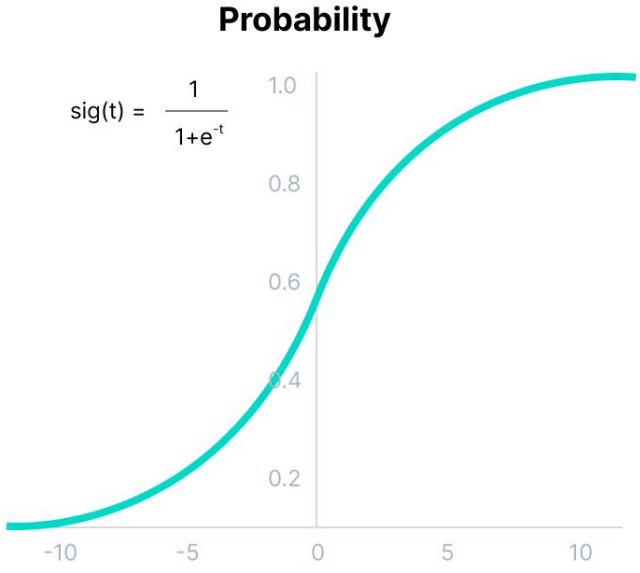
$$f(x) = \frac{1}{1 + e^{-x}}$$

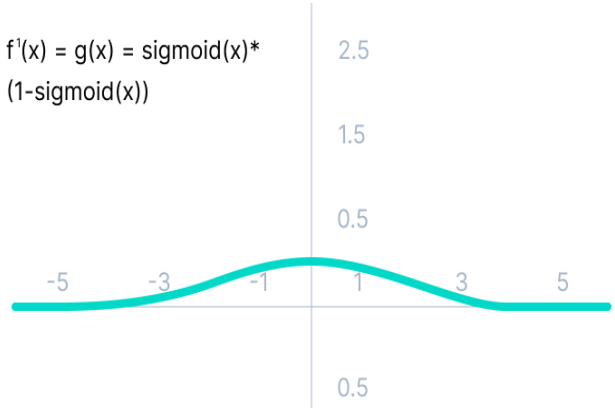


Softmax Activation Function

upGrad

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

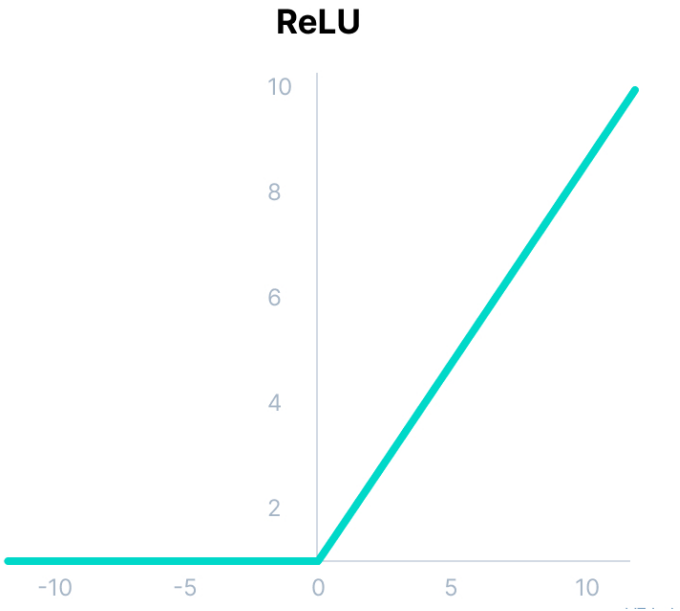




Rectified Linear Unit

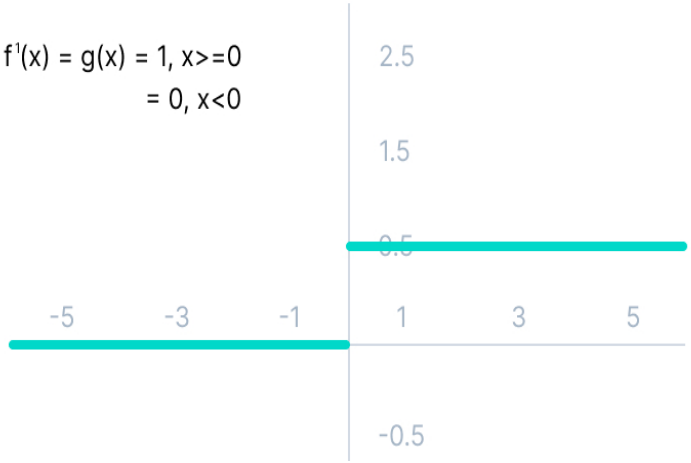
upGrad

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



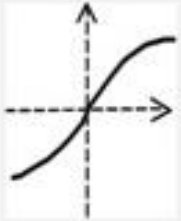
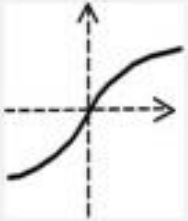
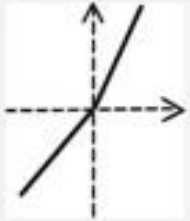
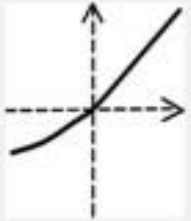
$f'(x) = g(x) = 1, x>=0$

$= 0, x<0$



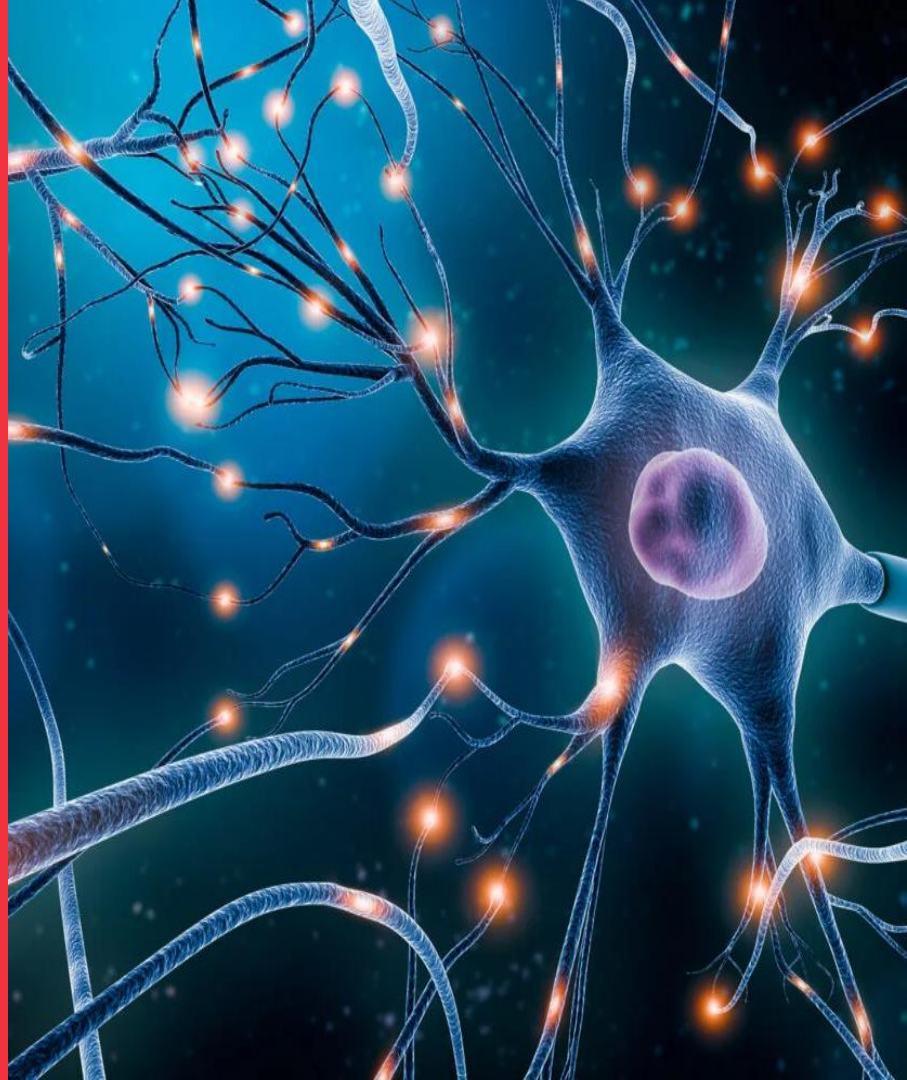
Other Activation Functions

upGrad

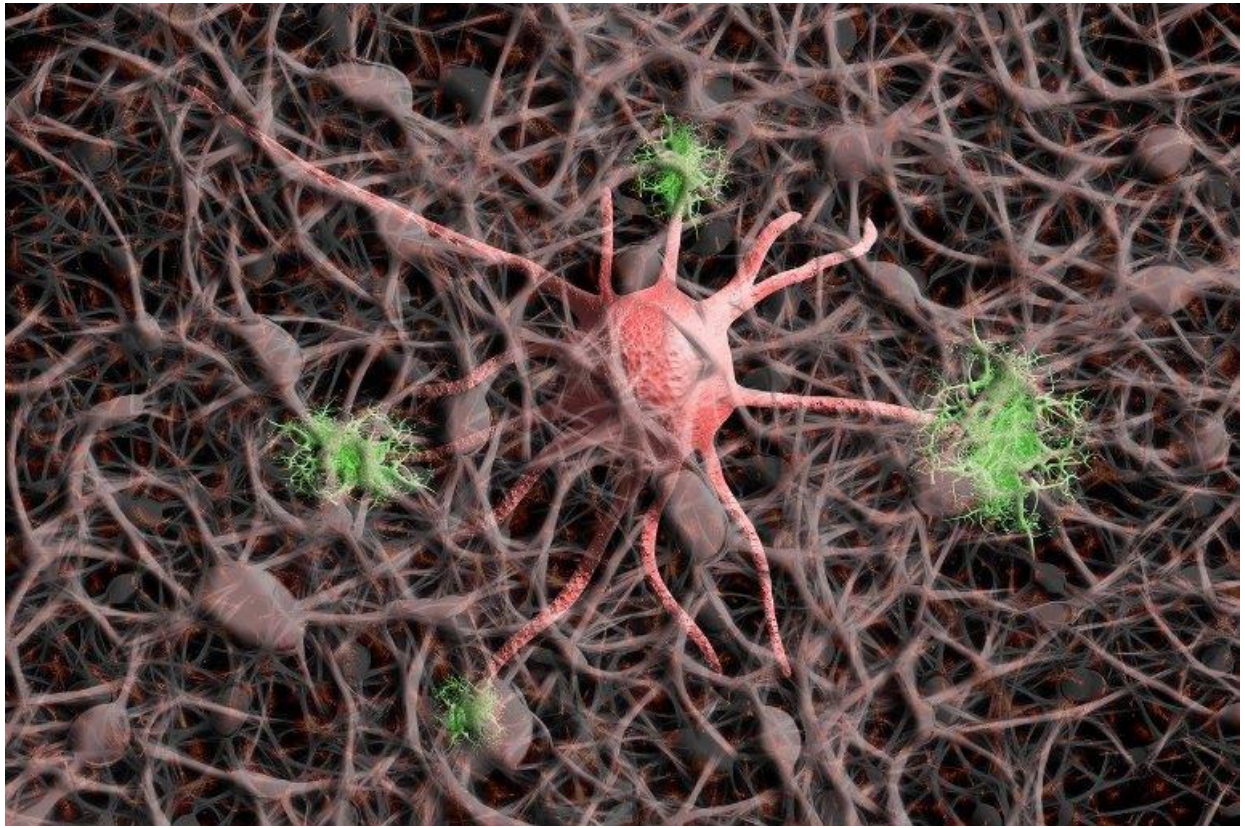
Binary Step	Logistic	TanH	ArcTan	ReLU	PreLU	ELU	SoftPlus
							
$f(x) = \begin{cases} 0 \text{ for } x<0 \\ 1 \text{ for } x\geq 0 \end{cases}$	$f(x) = \frac{1}{1+e^{-x}}$	$f(x) = tangh(x) = \frac{2}{1+e^{-2x}} - 1$	$f(x) = tan^{-1}(x)$	$f(x) = \begin{cases} 0 \text{ for } x<0 \\ x \text{ for } x\geq 0 \end{cases}$	$f(x) = \begin{cases} \propto x \text{ for } x<0 \\ x \text{ for } x\geq 0 \end{cases}$	$f(x) = \begin{cases} \propto (e^{-x}-1) \text{ for } x<0 \\ x \text{ for } x\geq 0 \end{cases}$	$f(x) = log_e(1+e^x)$
$f'(x) = \begin{cases} 0 \text{ for } x\neq 0 \\ ? \text{ for } x=0 \end{cases}$	$f'(x) = f(x)(1-f(x))$	$f'(x) = 1-f(x)^2$	$f'(x) = \frac{1}{x^2+1}$	$f'(x) = \begin{cases} 0 \text{ for } x<0 \\ 1 \text{ for } x\geq 0 \end{cases}$	$f'(x) = \begin{cases} \propto \text{ for } x<0 \\ 1 \text{ for } x\geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x)+\propto \text{ for } x<0 \\ 1 \text{ for } x\geq 0 \end{cases}$	$f'(x) = \frac{1}{1+e^{-x}}$

upGrad

Artificial Neural Networks



Biological Brain

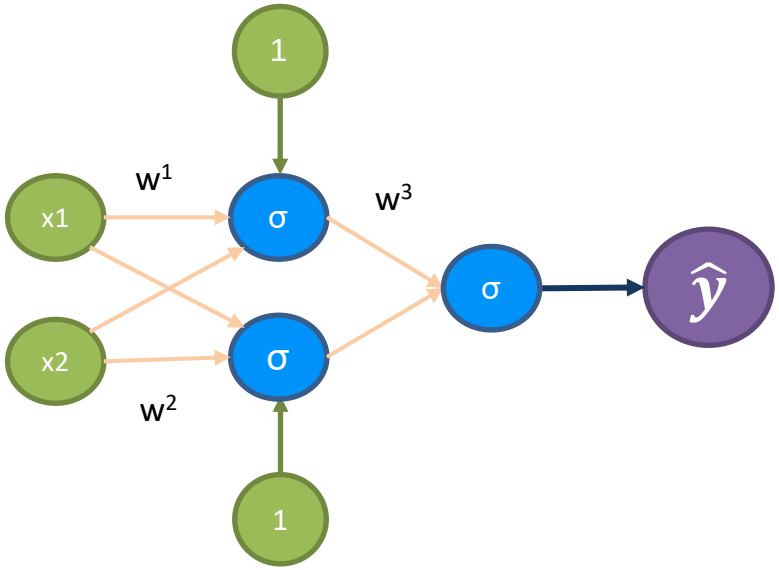


Human Brain:

- 60% Fat
- 100 Billion Neurons
- 1000 trillion Synapses – connections
- Connection store memory and process information.

A Simple Multi-Layered Perceptron

upGrad



x_1, x_2 — Input

$\alpha_1 = \sigma(w_i x_i + b_1)$

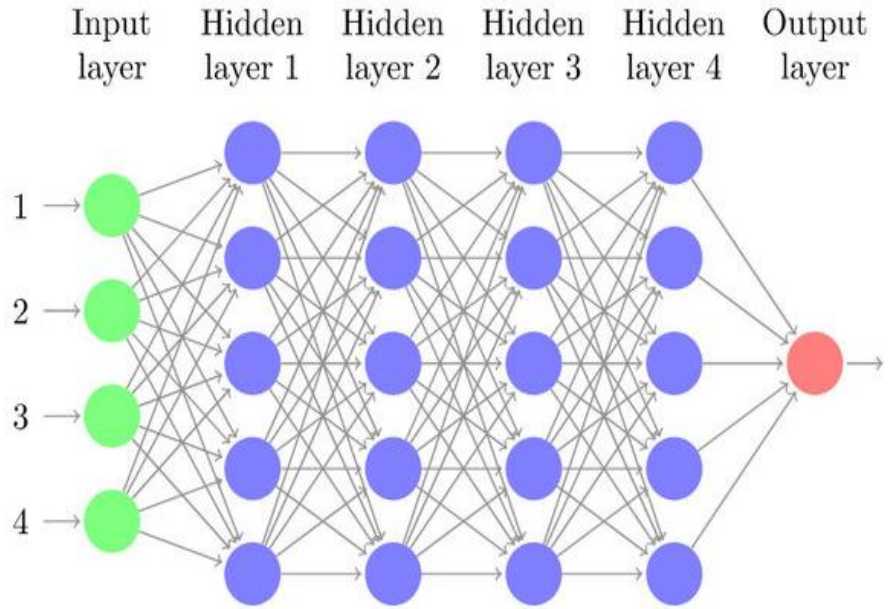
$\alpha_2 = \sigma(w_i x_i + b_2)$ — Intermediate

$\hat{y} = \sigma(w^3{}_1 \alpha_1 + w^3{}_2 \alpha_2)$ — Output

Multi-layered perceptron sum together the simple linear function, and each output a non linear function. The more layers we have in our hidden layer, the more complex non linear models we can find.

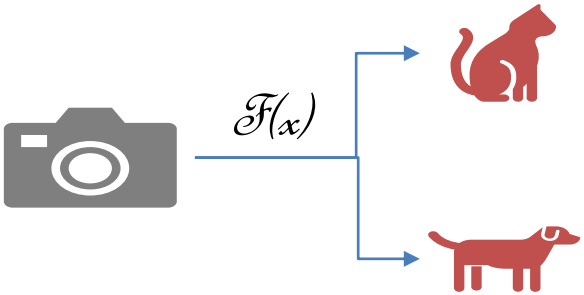
MLP – Universal Function Approximator

upGrad



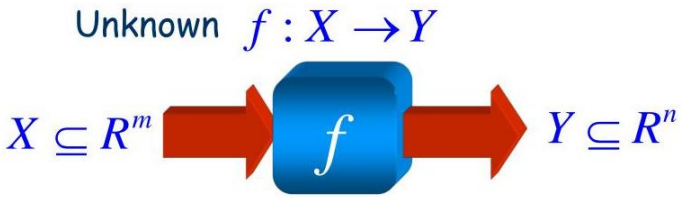
- Multi Layer Perceptron – 3 or more layers of nonlinearly-activating nodes.
- MLPs are fully connected.
  - Node i → w * Node j

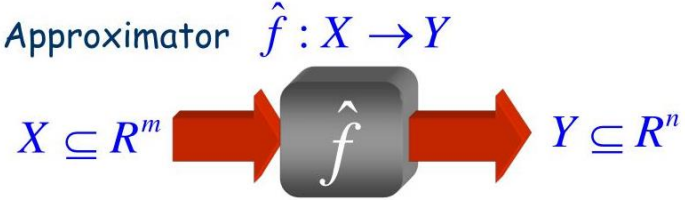
Function Approximator



Task: Find the function $\mathcal{F}(x)$ that can classify the image into Dog or Cat.

Use of Function Approximator: A technique for estimating an unknown underlying function using historical or available observations from the domain. Function approximations are used where theoretical models are unavailable or hard to compute.





Function Approximators

upGrad

Taylor Series: $$f(x) = f(a) + f'(a)\frac{(x-a)}{1!} + f''(a)\frac{(x-a)^2}{2!} + f'''(a)\frac{(x-a)^3}{3!} + ..$$

$$= \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$$

Fourier Series: (Periodic Functions)

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_n \cos(n\omega t) + b_n \sin(n\omega t)\right] \text{ or,}$$

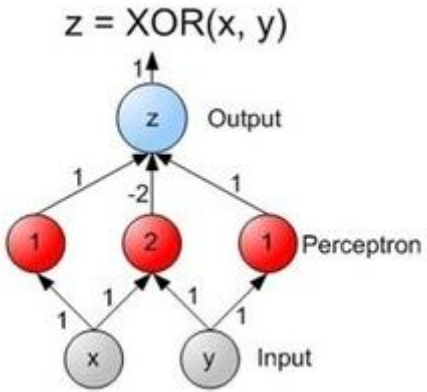
$$f(t) = \frac{1}{2}a_0 + a_1\cos(\omega t) + a_2\cos(2\omega t) + \cdots$$

$$+ b_1\sin(\omega t) + b_2\sin 2(\omega t) + \cdots$$

MLP – Universal Function Approximator

upGrad

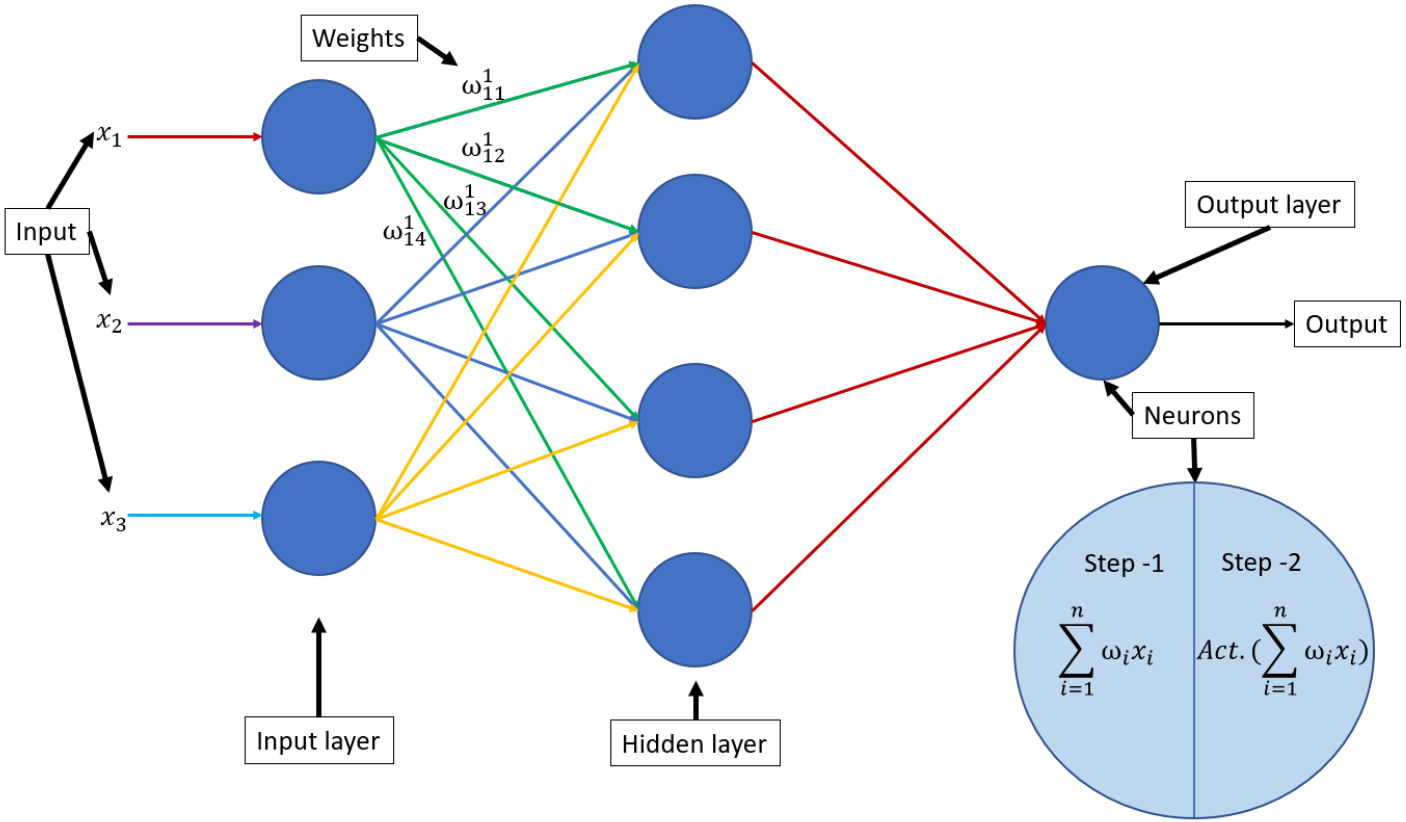
Multi-Layer Perceptron:



- MLPs are also function approximators.
- Cybenko's theorem : Given any known/unknown function, there exists a MLP that can approximate the function to an arbitrary degree of precision.
  - The network can be 1 layered with unbounded no of neurons
  - The network can be multi-layered with bounded no of neurons.
- Learning = δ(connection weights) after each piece of data is processed.
- MLP learn hierarchical feature representations. Do not require feature engineering.
- **MLPs are Universal Function Approximator that can learn from the data !!**

Topology of Artificial Neural Network

upGrad



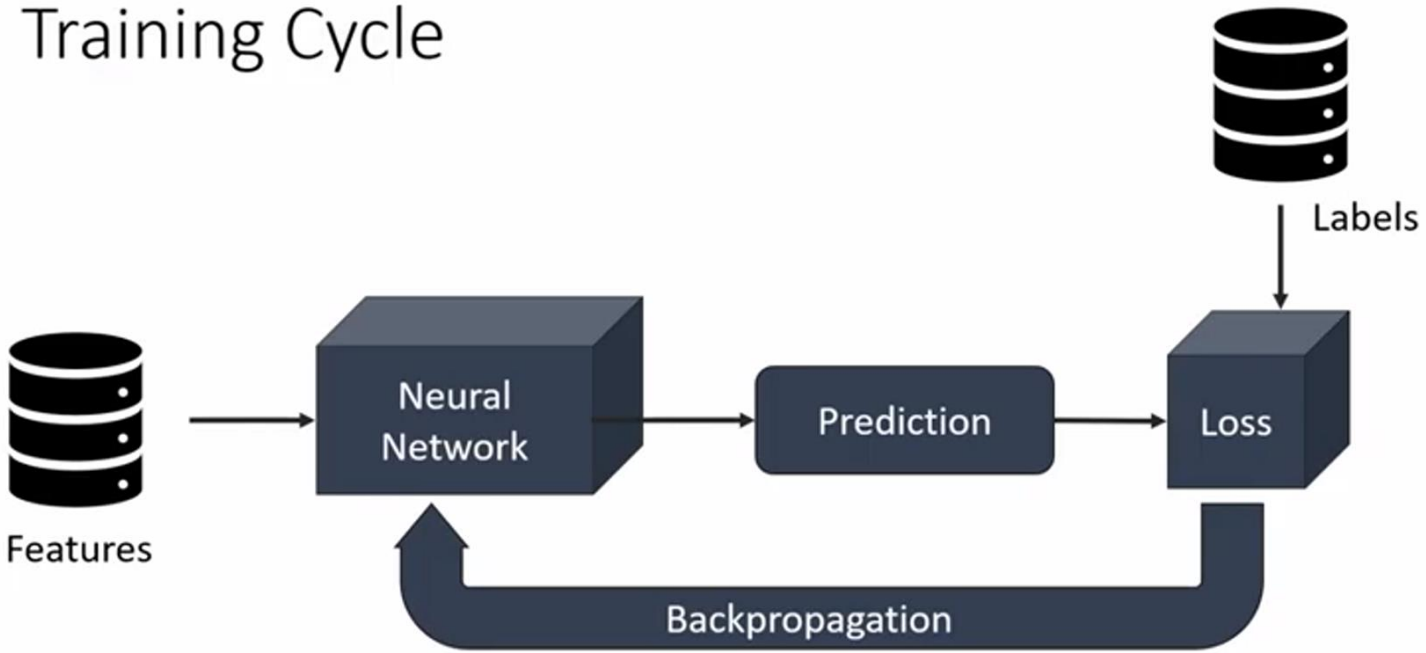
Topology of Artificial Neural Network

upGrad

- The neurons in an ANN are arranged in layers, and these layers are arranged sequentially(from Input to Output).
- Every neuron in the neural network has a bias value associated with it, and each interconnection has a weight associated with it.
- Neurons are densely connected, i.e., all neurons in layer L are connected to all neurons in layer L+1.
- The neurons within the same layer ***do not*** interact with each other.
- All neurons in a particular hidden layer use the same activation function. Different hidden layers can use different activation functions.

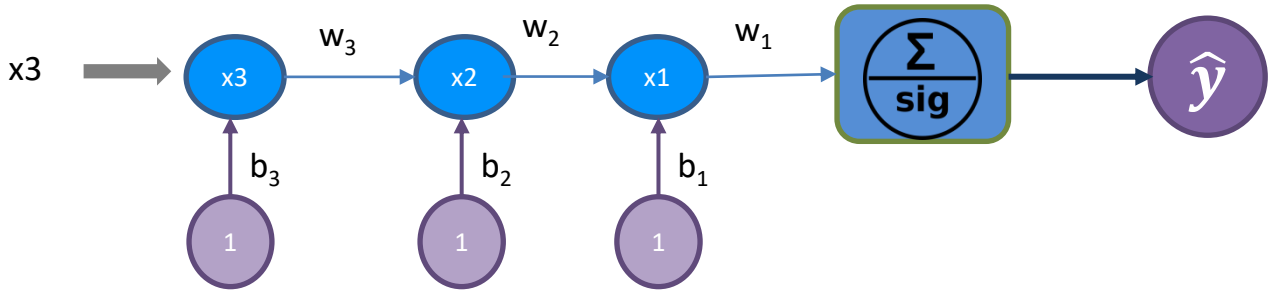
Training a Artificial Neural Network

upGrad



Training a Simple Network

upGrad



1 Define the Network

$$\hat{y} = f(w_1 x_1 + b_1)$$

$$x_1 = (w_2 x_2 + b_2)$$

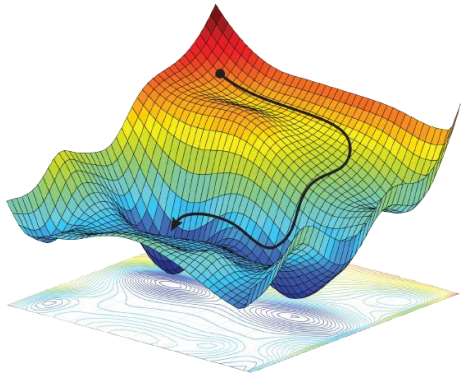
$$x_2 = (w_3 x_3 + b_3)$$

2 Define the Cost Function

$$L(w): \sum (y - \hat{y})^2$$

- *Quantifies model performance*
- *# Continuous and mostly differentiable*

3 Learn the weights/biases



Gradient Descent

upGrad

- Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

- Gradient descent is best used when the parameters cannot be calculated analytically .

- It finds the parameters using an iterative method:
    - The goal is to continue to try different values for the coefficients, evaluate their cost and select new coefficients that have a slightly better (lower) cost.
    - Repeating this process enough times will lead to the coefficients that result in the minimum cost.

Gradient Descent

upGrad

Gradient descent :

1. Estimate the error using current weights (randomly initialized)
2. Find the derivative(gradient) of the cost function.
3. Move in the direction opposite of the gradient, until convergence.

Gradient Descent Update Rule:

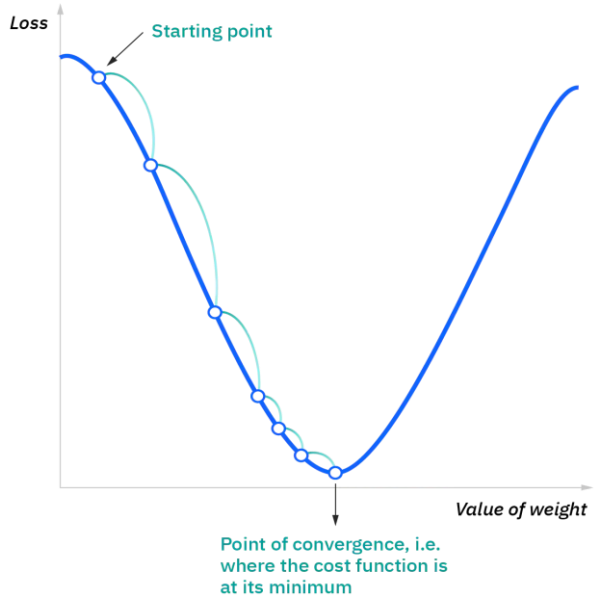
$$\text{repeat until convergence } \{$$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for j=1 to j=k)

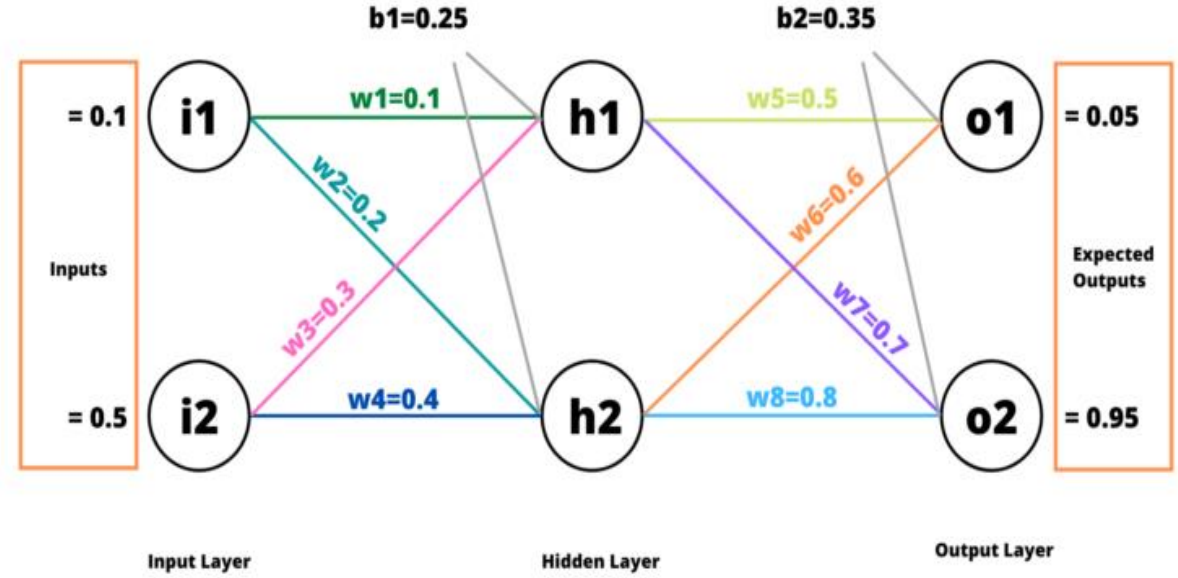
$$\}$$

α – learning rate
j - features



Training a Simple Network – Back Propagation

upGrad

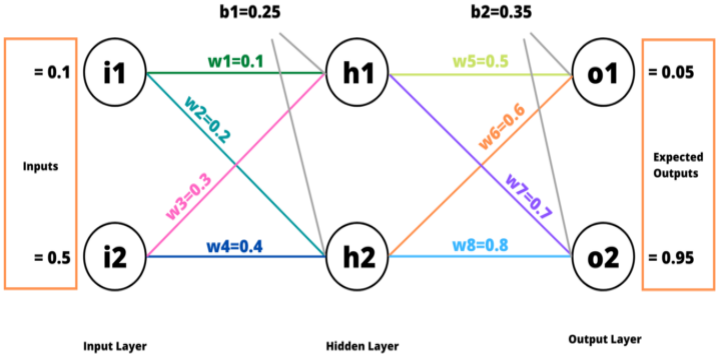


Find the weights (w1, w2, w3…) such that the cost (error) is minimized

$$E(w) or\ L(w): \sum (y - \hat{y})^2$$

Training a Simple Network – Back Propagation

upGrad



Find the weights (w1, w2, w3) such that the cost (error) is minimized

$$L(w): \sum (y - \hat{y})^2$$

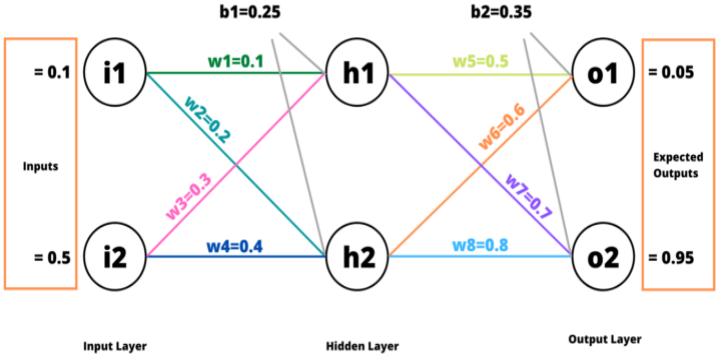
1. Start with random weights (w1, w2, w3,…)
2. Calculate the values at each node and the output.
3. Calculate the Cost Function (Error)
4. Find the Derivative of Cost Function w.r.t each weight.
5. Change w_j such that

$$w_j =: w_j - \alpha * \frac{\partial L}{\partial wj}$$

6. Stop when values converge.

Training a Simple Network – Back Propagation

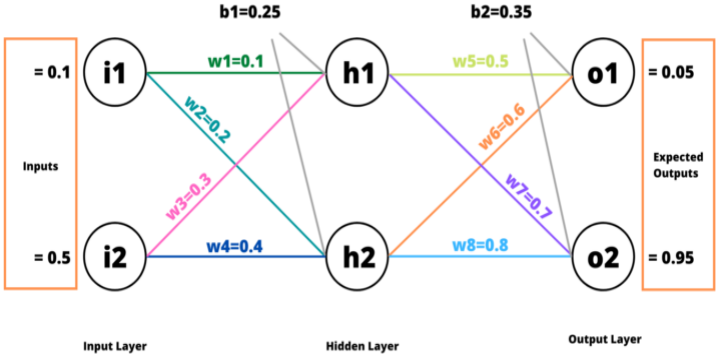
upGrad



1. Start with random weights (w1, w2, w3,…)

w1 = 0.1
w2= 0.2
w3= 0.3
w4= 0.4
w5= 0.5
w6= 0.6
w7= 0.7
w8= 0.8

upGrad



2. Calculate the values at each node and the output. [Forward Pass]

$$sum_{h1} = i_1 * w_1 + i_2 * w_3 + b_1 \qquad output_{h1} = \frac{1}{1 + e^{-sum_{h1}}}$$

$$sum_{h1} = 0.1 * 0.1 + 0.5 * 0.3 + 0.25 = 0.41 \qquad output_{h1} = \frac{1}{1 + e^{-0.41}} = 0.60108$$

$$sum_{h2} = i_1 * w_2 + i_2 * w_4 + b_1 = 0.47$$

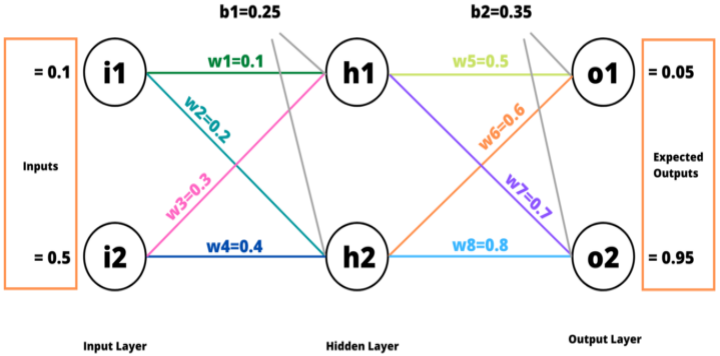
$$output_{h2} = \frac{1}{1 + e^{-sum_{h2}}} = 0.61538$$

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2 = 1.01977$$

$$output_{o1} = \frac{1}{1 + e^{-sum_{o1}}} = 0.73492$$

$$sum_{o2} = output_{h1} * w_7 + output_{h2} * w_8 + b_2 = 1.26306$$

$$output_{o2} = \frac{1}{1 + e^{-sum_{o2}}} = 0.77955$$



3. Calculate the Cost Function (Error)

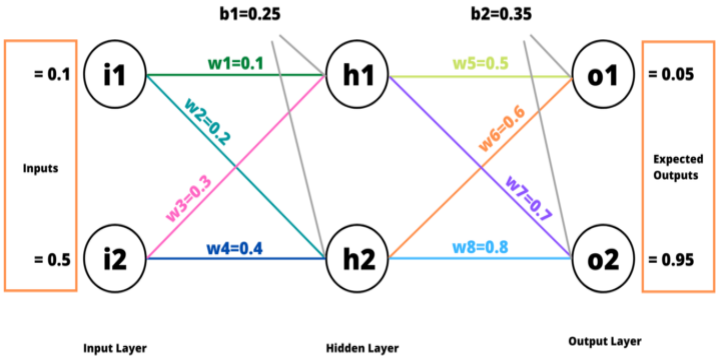
$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_1 = \frac{1}{2}(0.05 - 0.73492)^2 = 0.23456$$

$$E_2 = \frac{1}{2}(0.95 - 0.77955)^2 = 0.01452$$

$$E_{total} = E_1 + E_2 = 0.24908$$

upGrad



4. Find the Derivative of Cost Function w.r.t each weight.

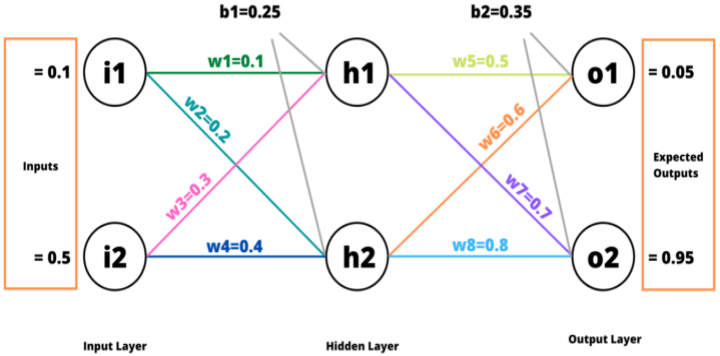
*E*1 is affected by $output_{o1}$, $output_{o1}$ is affected by sum_{o1}, and sum_{o1} is affected by *w*5.

So by Calculus Chain Rule.

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

BACK PROPAGATION

upGrad



4. Find the Derivative of Cost Function w.r.t each weight.

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

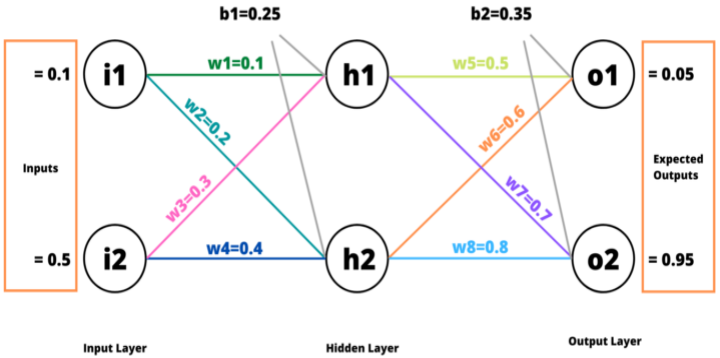
$$E_{total} = \frac{1}{2}(target_1 - output_{o1})^2 + \frac{1}{2}(target_2 - output_{o2})^2$$

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

$$\frac{\partial output_{o1}}{\partial sum_{o1}} = output_{o1}(1 - output_{o1}) \qquad \frac{\partial sum_{o1}}{\partial w5} = output_{h1}$$

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

$$\frac{\partial E_{total}}{\partial w5} = 0.68492 * 0.19480 * 0.60108 \quad = 0.082$$



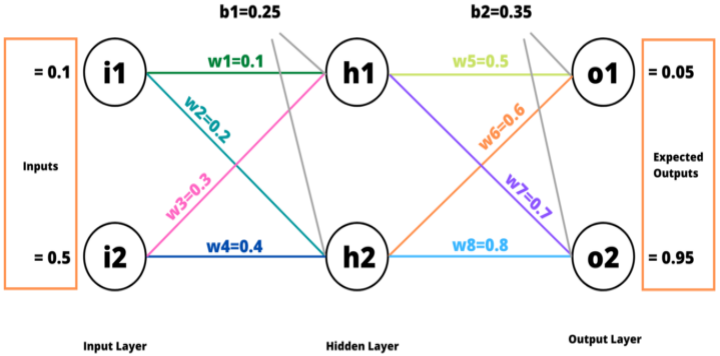
5. Change the weights in negative direction of gradient.

$$new\_w_5 = w5 - n * \frac{\partial E_{total}}{\partial w5}$$, where n is learning rate.

$$new\_w_5 = 0.5 - 0.6 * 0.08020$$

$$new\_w_5 = 0.45187$$

We can proceed similarly for al other weights w6, w7, and w8.



5. Change the weights in negative direction of gradient.

For weights in the hidden layer (w1, w2, w3, w4)

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

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

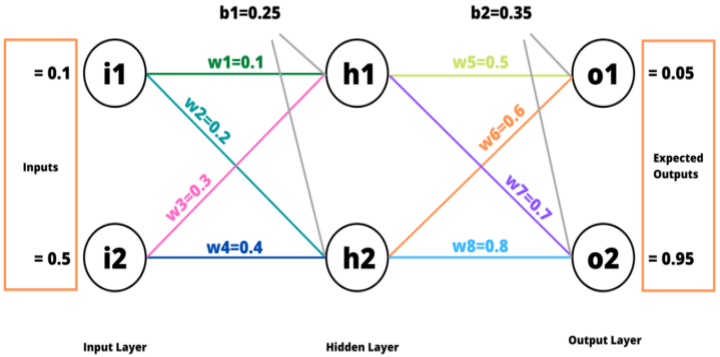
$$\frac{\partial E_2}{\partial w1} = \frac{\partial E_2}{\partial output_{o2}} * \frac{\partial output_{o2}}{\partial sum_{o2}} * \frac{\partial sum_{o2}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

$$\frac{\partial E_{total}}{\partial w1} = 0.00159 + (-0.00049) = 0.00110.$$

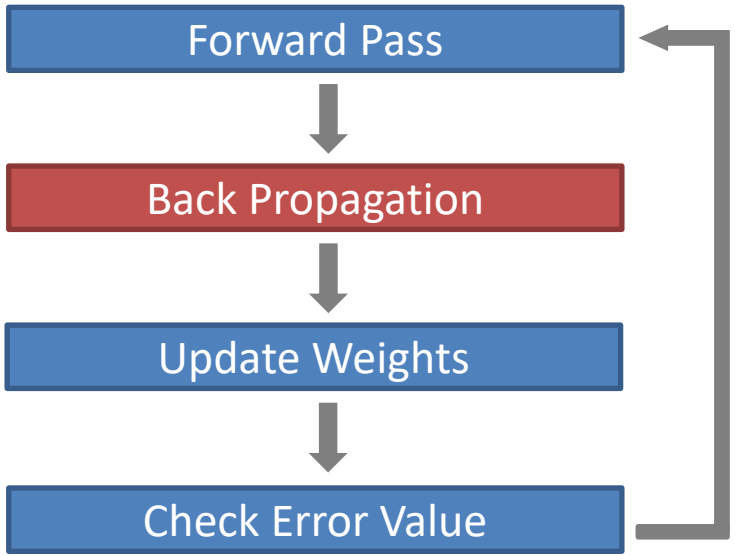
$$new\_w_1 = w1 - n * \frac{\partial E_{total}}{\partial w1} \qquad new\_w_1 = 0.09933$$

Training a Simple Network – Back Propagation

upGrad



6. Stop when values converge.



Cost Functions - Regression

upGrad

Mean Squared Error (MSE)

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

MSE with L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

MSE with L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

Mean Square Log Error

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(\log(Y_i) - \log(\hat{Y}_i))^2$$

Huber Loss Function

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Cost Functions - Classification

upGrad

Binary Cross Entropy – Binary Classification

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

Cross Entropy – Multi Class Classification

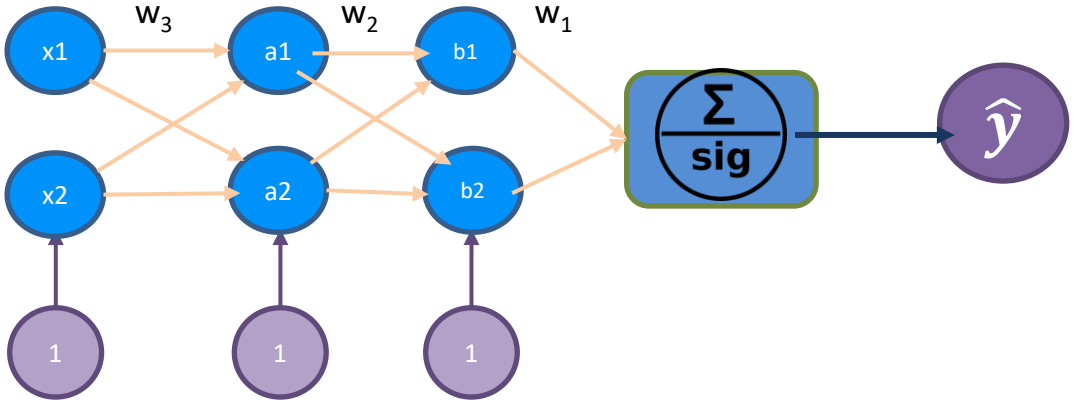
$$\text{cross-entropy} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{k} t_{i,j} \log(p_{i,j})$$

Kullback Leibler Divergence Loss (KL Loss)

$$D_{KL}(P||Q) = \begin{cases} -\sum_x P(x).log\frac{Q(x)}{P(x)} = \sum_x P(x).log\frac{P(x)}{Q(x)}, & for\ discrete\ distributions \\ -\int P(x).log\frac{Q(x)}{P(x)}.dx = \int P(x).log\frac{P(x)}{Q(x)}.dx, & for\ continuous\ distributions \end{cases}$$

Training more Complex Networks

upGrad



$$\frac{\partial L}{\partial w} = \left[\frac{\partial L}{\partial b}\right] * \left[\frac{\partial b}{\partial a}\right] * \left[\frac{\partial a}{\partial x}\right]$$

Derivative of a vector w.r.t another vector yields a Matrix – **Jacobian**

Linear Algebra & Matrix Calculus

Types of Gradient Descent

upGrad

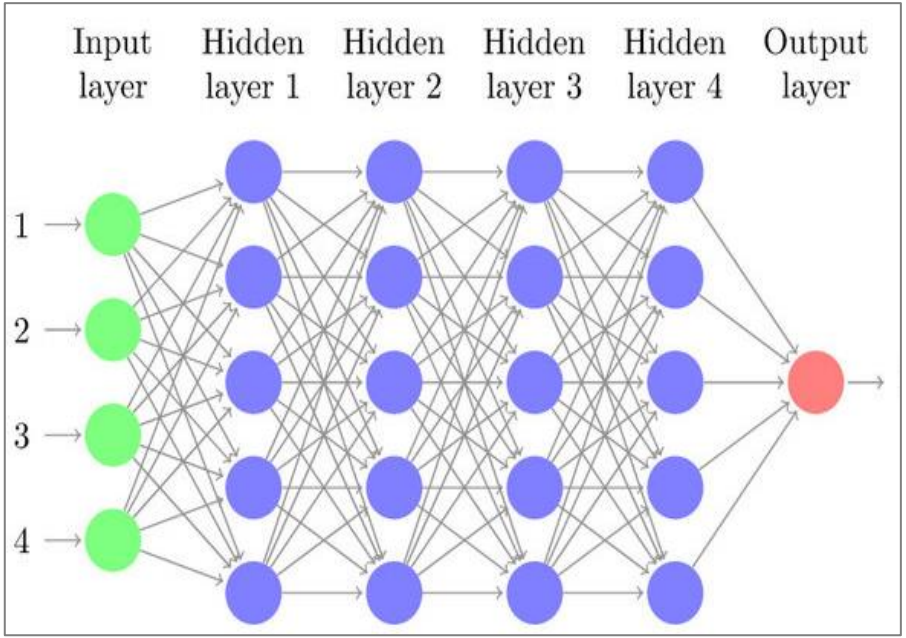
Batch Gradient Descent	Mini-Batch Gradient Descent	Stochastic Gradient Descent
Uses ALL training samples to calculate Error, for adjusting weights	Uses A random sample to calculate Error, for adjusting weights	Uses ONE training Sample to calculate Error, for adjusting weights. Also called “Online” training.
Good for small datasets. Very time consuming with large datasets, large feature set.	Compromise between the two and works well in both cases. Efficient use of matrix multiplication, especially using GPUs.	Good for very big dataset with large no of features and redundant data.

Parameters and Hyper-Parameters of Neural Network

upGrad

Parameters:

- Weights
- Biases

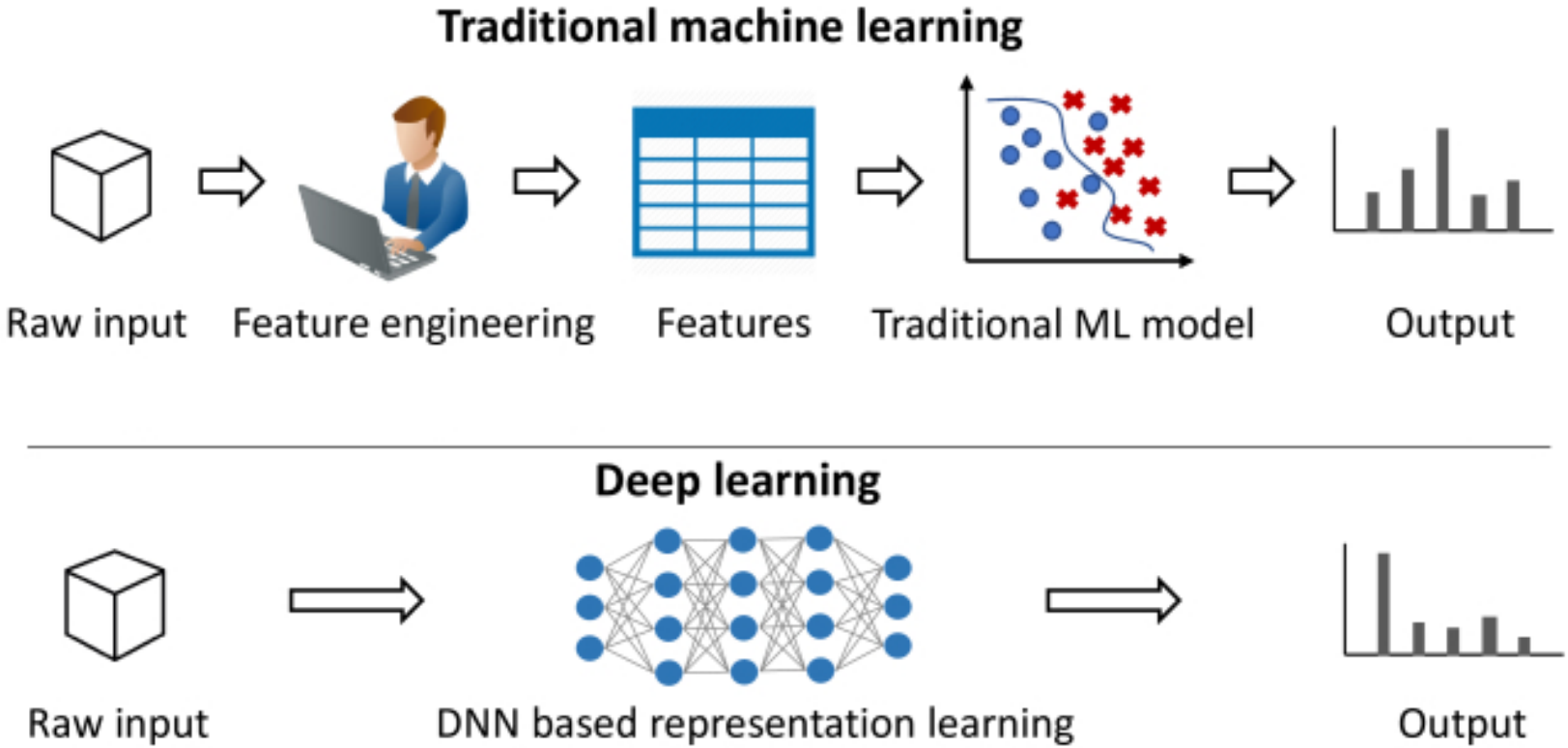


Hyper Parameters:

- Number of layers
- Number of neurons in the input, hidden and output layers
- Activation Function
- Regularization
- Learning rate
- Batch Size
- Number of epochs

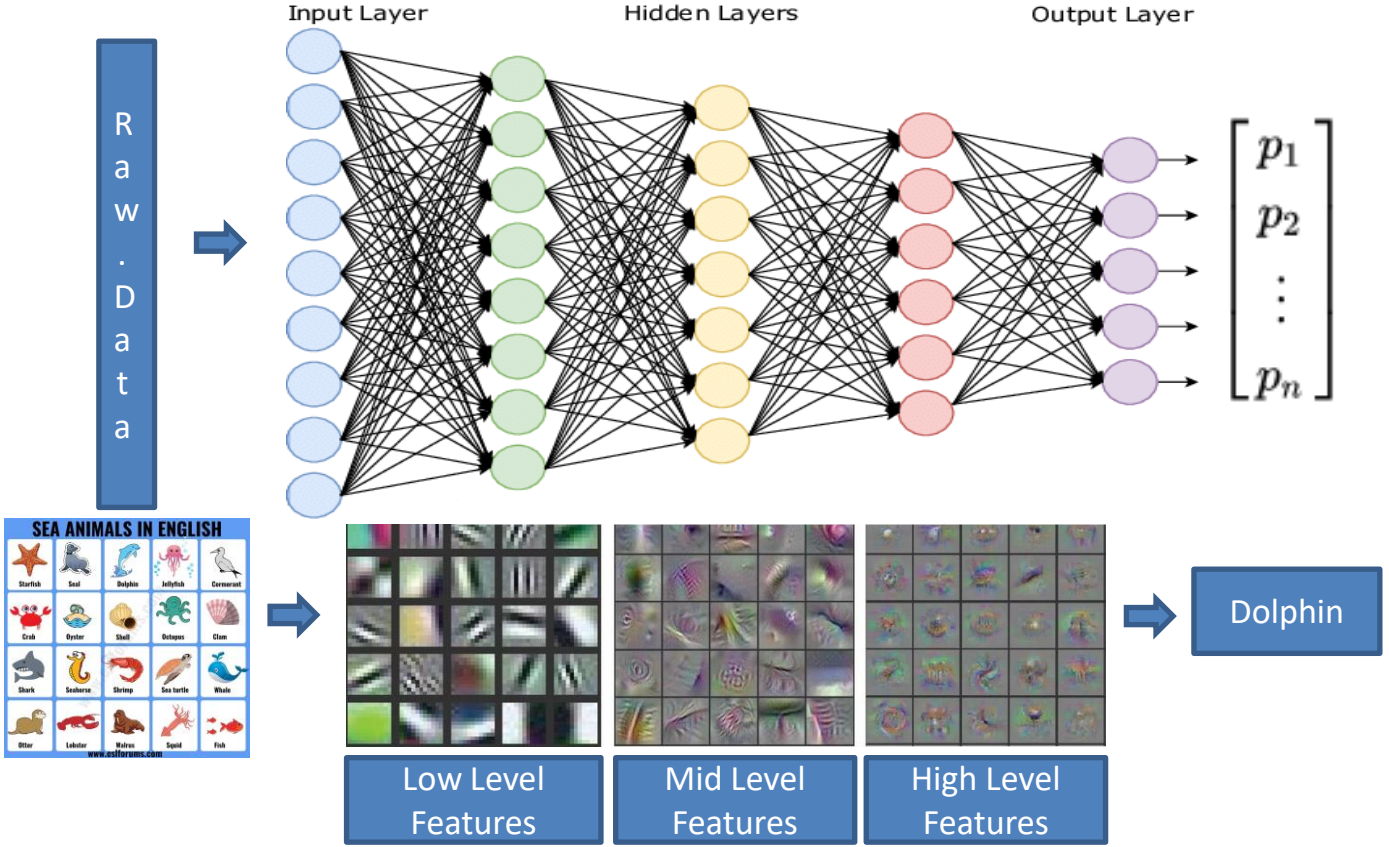
Deep Learning automated Feature Extraction

upGrad



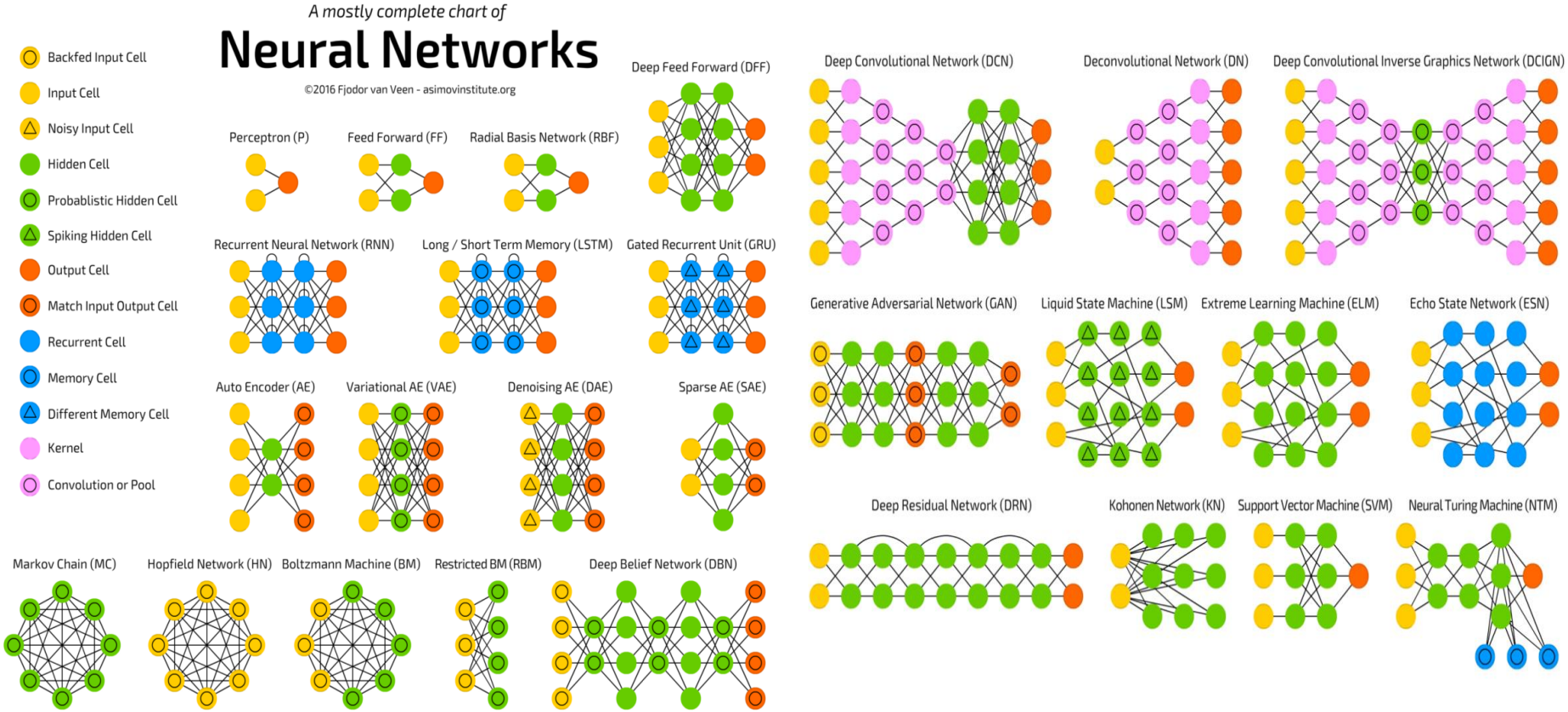
CNN- Illustration of Hierarchical Feature Extraction

upGrad



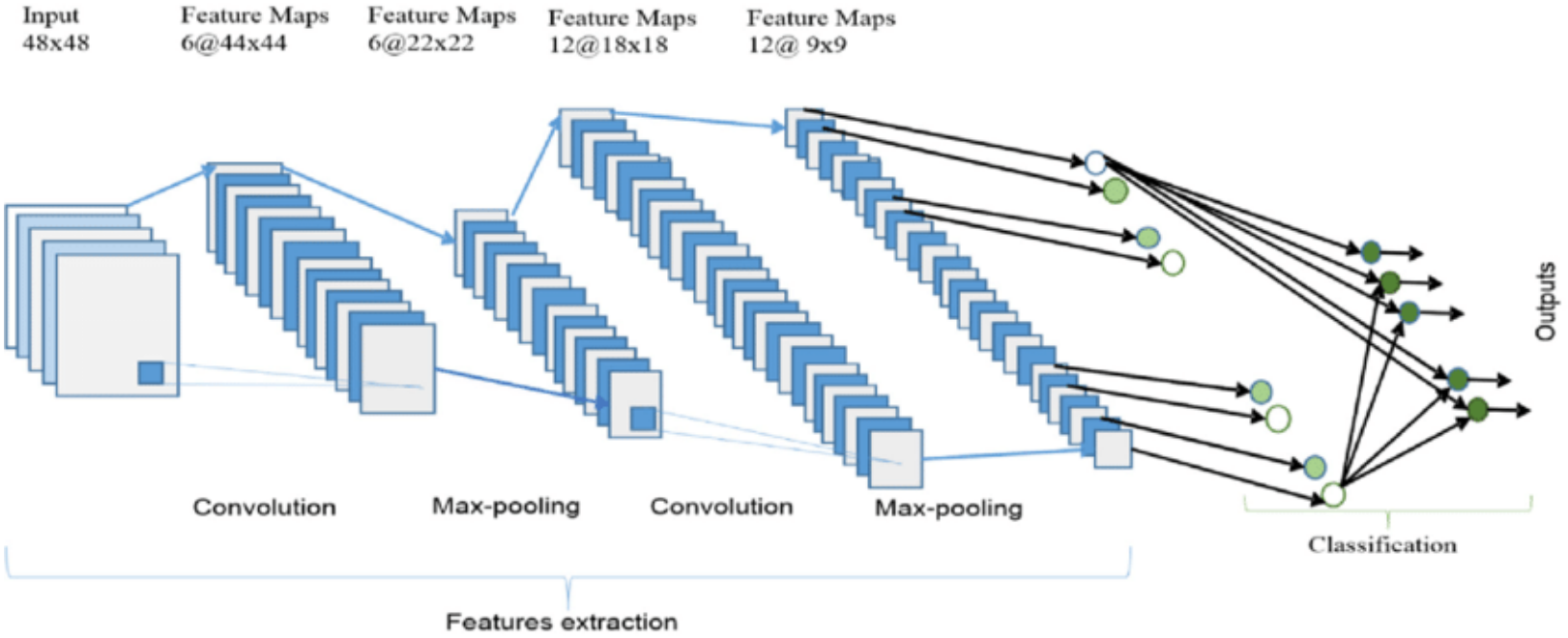
Neural Network Architectures

upGrad



Convolutional Neural Network – Network the See

upGrad

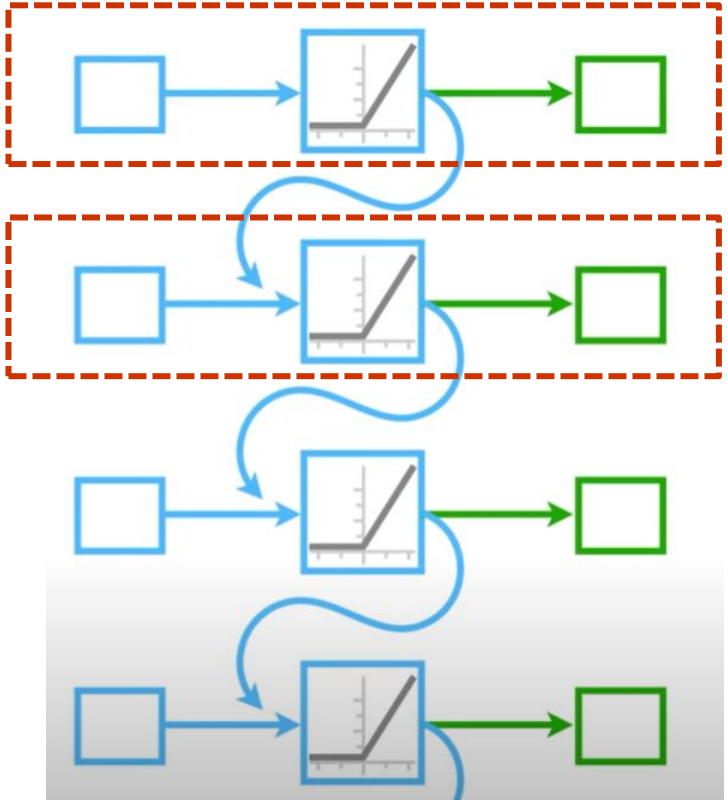


- Sequence of ***convolution*** (of filters) and ***pooling*** of pixel values exploit spatial locality- enforce a local connectivity pattern between neurons of adjacent layers. The learned "filters" produce the strongest response to a spatially local input pattern.
- Each filter is replicated across the entire visual field – leading to translation invariance

Recurrent Neural Network – Feedback Loop

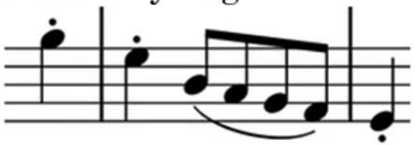
upGrad

- RNN is a class of ANN with connections between nodes (cycle) - Output from some nodes to affect subsequent input to the same nodes.
- Exhibit temporal dynamic behavior.
- Process variable length sequences of inputs and generate variable sequence output.
- RNN can take a series of inputs with no predetermined limit on size and generate variable sequence output:
  1. Vector-Sequence Models — Image caption:
  2. Sequence-Vector Model — Sentiment analysis of a movie
  3. Sequence-to-Sequence Model - Language translation



Recurrent Neural Network – Applications

upGrad

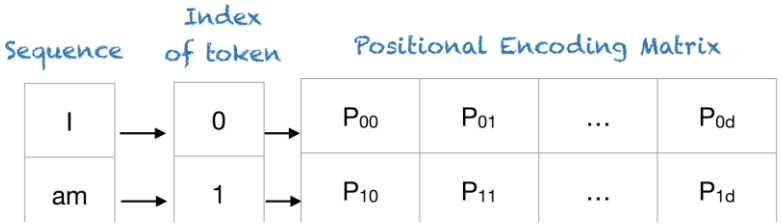
Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation	Ø	→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.

Transformer – Attention Mechanism

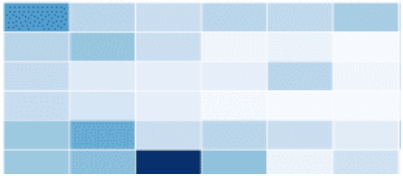
upGrad

The transformer neural network is a novel architecture that aims to solve sequence-to-sequence tasks while handling long-range dependencies.

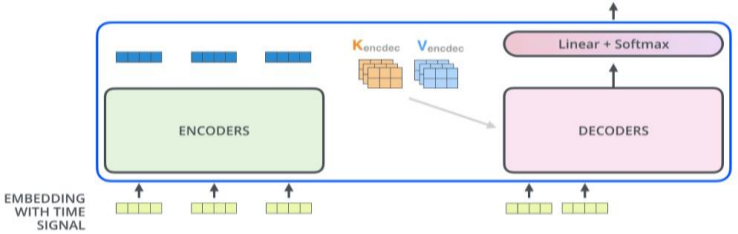
1. Positional encoding : Word embeddings are combined with their position in the sentence. This allows to input the entire sentence at once, and parallelize the process (using GPU)



2. Self – Attention : Derive the relationship between each word with all other words in the input (using Query, Key and Value mechanism)

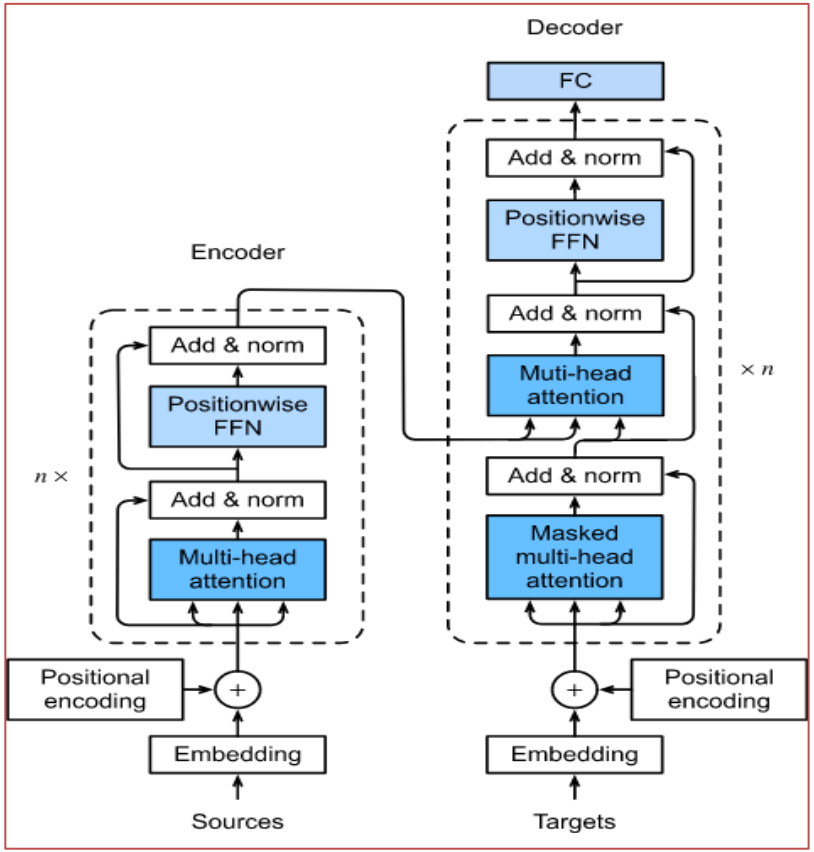


3. Decoding: The decoder generates the output sentence by defining the conditional probability distribution of a target sequence given a contextualized encoding sequence.



Transformer Archictecture

upGrad



Famous Transformer Examples

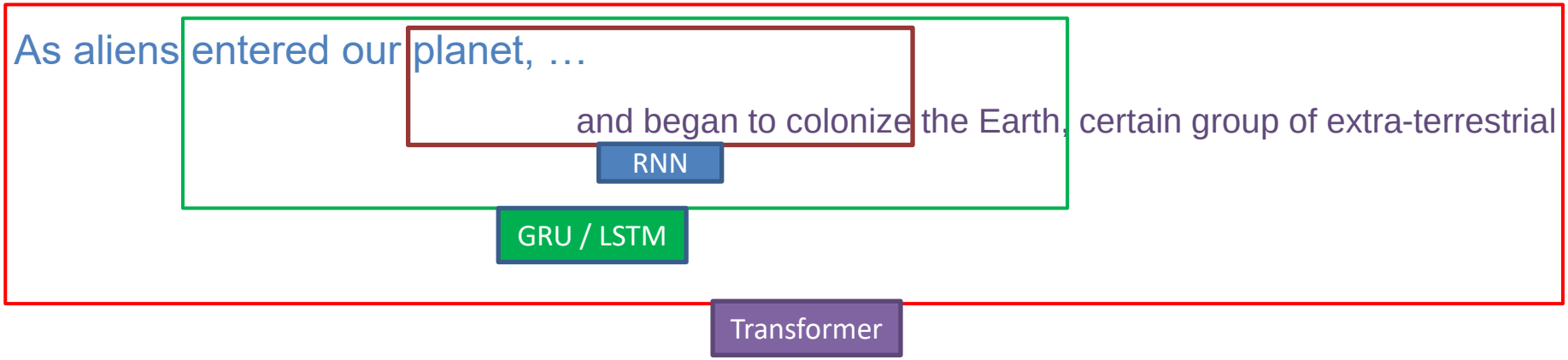
upGrad

- BERT :
  - Bidirectional Encoder Representations from Transformers is a transformer-based machine learning technique for natural language processing.
  - BERT was created and published in 2018 by Google.
  - BERT is a deeply bidirectional, unsupervised language representation, pre-trained using only a plain text corpus.

- GPT-3 :
  - **Generative Pre-trained Transformer 3** is a language model that uses deep learning to produce human-like text.
  - GPT-3, which was introduced in May 2020, by OpenAI.
  - The architecture is a standard transformer network (with a few tweaks) with 175 billion parameters.

Attention – RNN vs LSTM vs Transformer

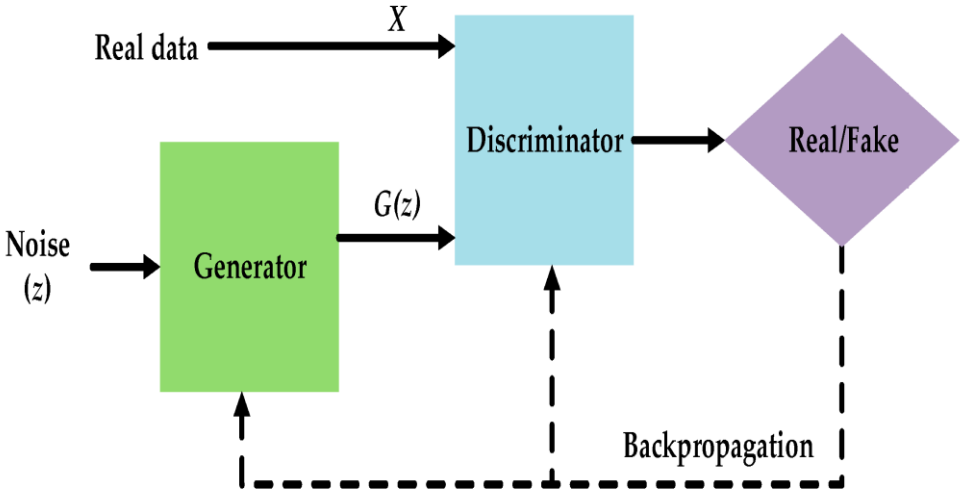
upGrad



Generative Adversarial Networks

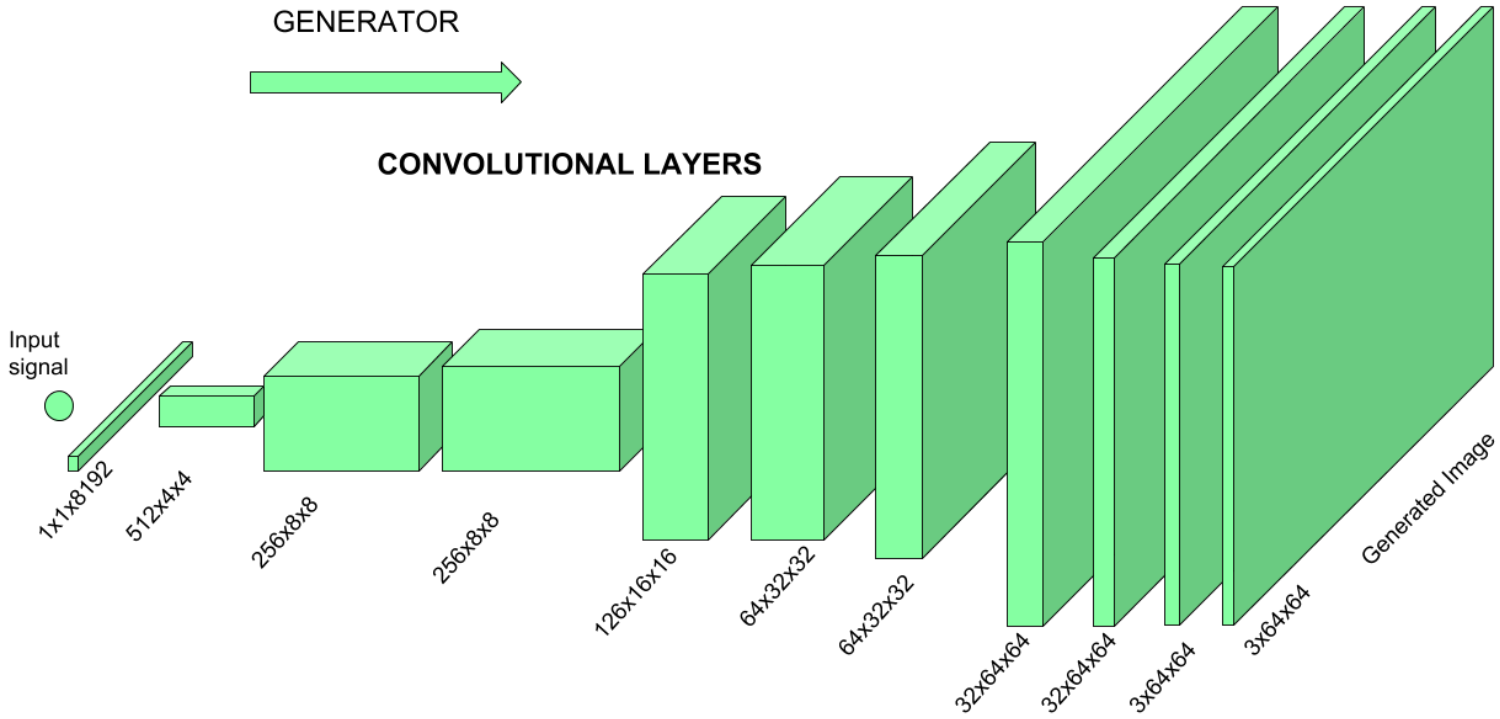
upGrad

- GAN is a class of ML frameworks where two neural networks contest with each other in the form of a zero-sum game.
- Given a training set, this technique learns to generate new data with the same statistics as the training set.
- For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics.



Generator of GAN

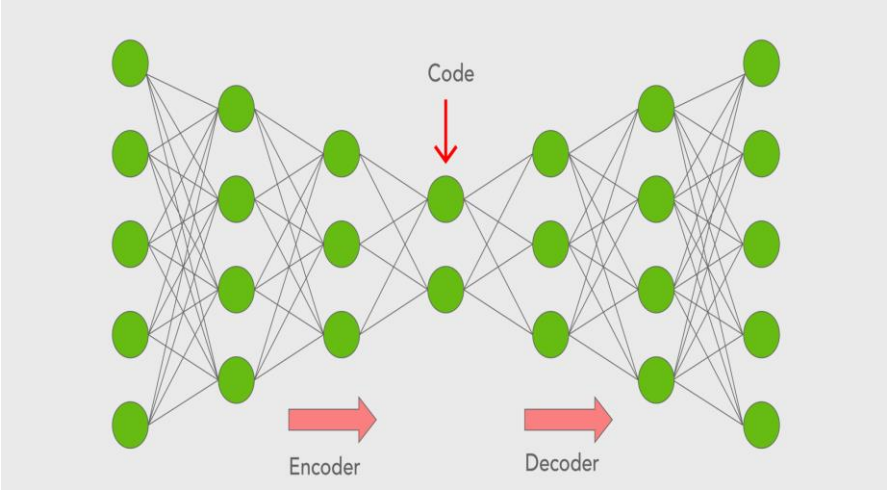
upGrad



Auto Encoders

upGrad

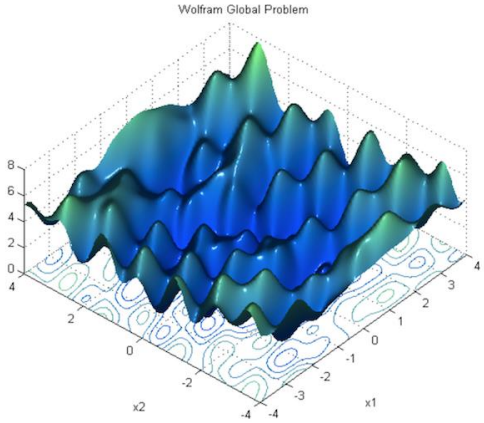
- Autoencoders operate by taking in data, compressing and encoding the data, and then reconstructing the data from the encoding representation.
- The model is trained until the loss is minimized and the data is reproduced as closely as possible.
- Through this process, an autoencoder can learn the important features of the data
- Used for dimensionality reduction, data denoising, feature extraction, image generation, sequence to sequence prediction, and recommendation systems.



Challenges of Deep Learning

upGrad

Large Dataset



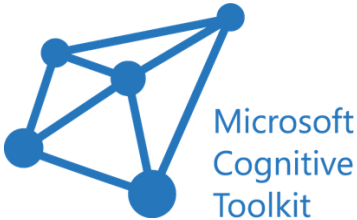
Global Minima

Resource & Time

Framework for Deep Learning

upGrad

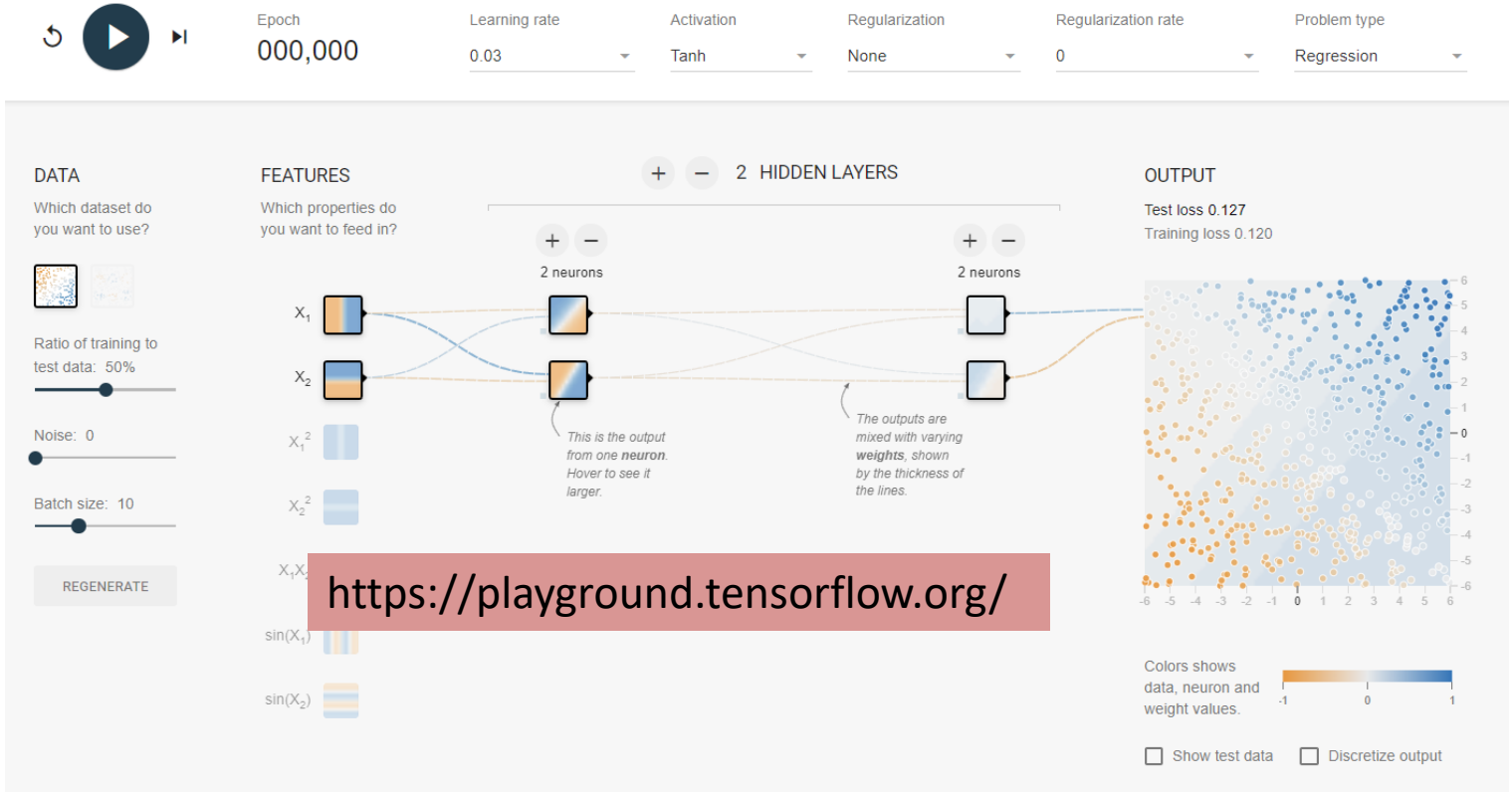
Caffe



theano

TensorFlow Playground

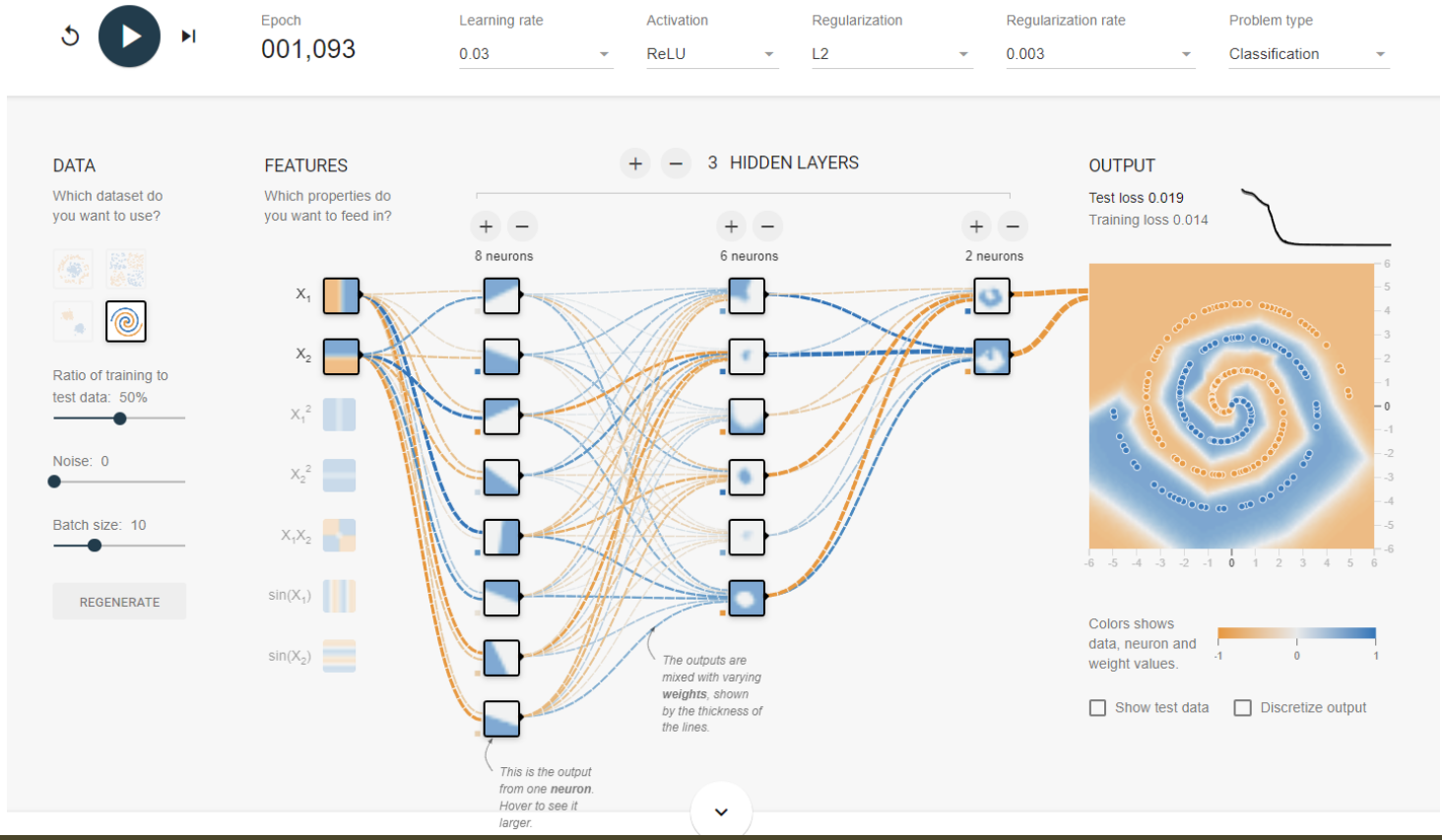
upGrad



https://playground.tensorflow.org/

Tensorflow Playground

upGrad



Discussion

upGrad

upGrad
#LifeKoKaroLift

Thank You!