DS JULY 2022 Batch
Module 18 – Deep Learning part 1

Topics

- Introduction
- Biological and Artificial Neuron
- Perceptron and its learning rule and drawbacks
- Multilayer Perceptron, loss function
- Activation Functions
- Training MLP: Backpropagation
- Introduction to Tensorflow and Keras
- Vanishing and Exploding Gradient Problem

Introduction

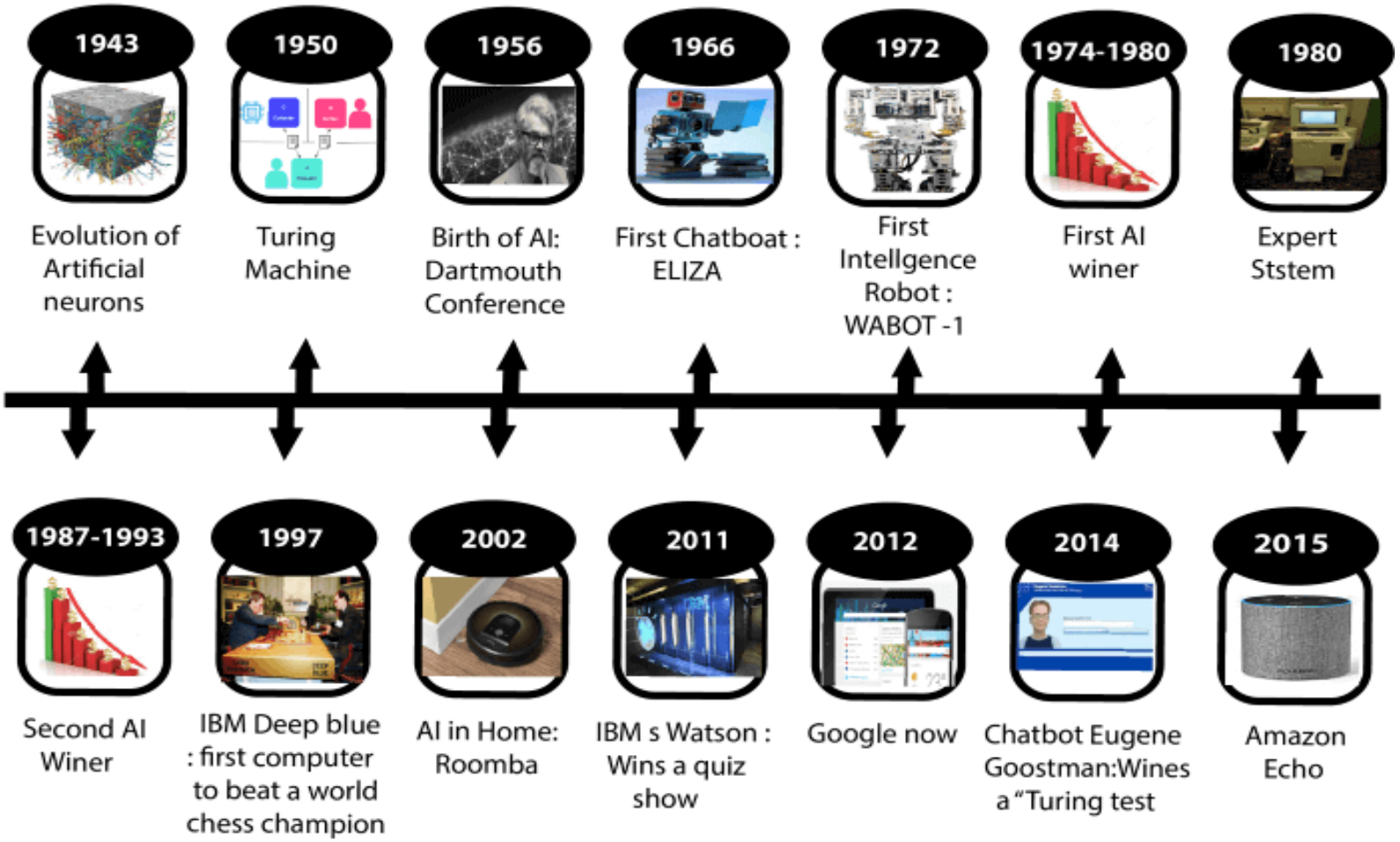
Artificial Intelligence

- Artificial intelligence (AI) is a branch of computer science that attempts to simulate human intelligence in machines.

Artificial Intelligence : History

- **Birth of AI: 1950-1956**
- This range of time was when the interest in AI really came to a head. Alan Turing published his work “Computer Machinery and Intelligence” which eventually became The Turing Test, which experts used to measure computer intelligence. The term “artificial intelligence” was coined and came into popular use.
- Dates of note:
  - 1950: Alan Turing published “Computer Machinery and Intelligence” which proposed a test of machine intelligence called The Imitation Game.
  - 1952: A computer scientist named Arthur Samuel developed a program to play checkers, which is the first to ever learn the game independently.
  - 1955: John McCarthy held a workshop at Dartmouth on “artificial intelligence” which is the first use of the word, and how it came into popular usage.

Artificial Intelligence : History



Artificial Intelligence : Application

- **AI in healthcare :** Companies are applying machine learning to make better and faster diagnoses than humans. One of the best-known healthcare technologies is **IBM Watson**. It understands natural language and can respond to questions asked of it. The system mines patient data and other available data sources to form a hypothesis, which it then presents with a confidence scoring schema. Other AI applications include using **online virtual health assistants and chatbots** to help patients and healthcare customers find medical information, schedule appointments, understand the billing process and complete other administrative processes. **An array of AI technologies is also being used to predict, fight and understand pandemics such as COVID-19.**
- **AI in business :** Machine learning algorithms are being integrated into analytics and **customer relationship management (CRM)** platforms to uncover information on how to better serve customers. Chatbots have been incorporated into websites to provide immediate service to customers. Automation of job positions has also become a talking point among academics and IT analysts.

Artificial Intelligence : Application

- **AI in finance:** AI in **personal finance applications**, such as Intuit Mint or TurboTax, is disrupting financial institutions. Applications such as these collect personal data and provide financial advice. Other programs, such as IBM Watson, have been applied to the process of buying a home. Today, artificial intelligence software performs much of the trading on Wall Street.
- **AI in manufacturing:** Industrial robots that were at one time programmed to perform single tasks and separated from human workers, increasingly function as **cobots** : Smaller, multitasking robots that collaborate with humans and take on responsibility for more parts of the job in warehouses, factory floors and other workspaces.
- **AI in banking:** Banks are successfully employing **chatbots to make their customers aware of services** and offerings and to handle transactions that don't require human intervention. **AI virtual assistants** are being used to improve and cut the costs of compliance with banking regulations. Banking organizations are also using AI to improve their decision-making for loans, and to set credit limits and identify investment opportunities.
- **AI in transportation:** In addition to AI's fundamental role in operating autonomous vehicles, AI technologies are used in transportation to **manage traffic, predict flight delays**, and make ocean shipping safer and more efficient.

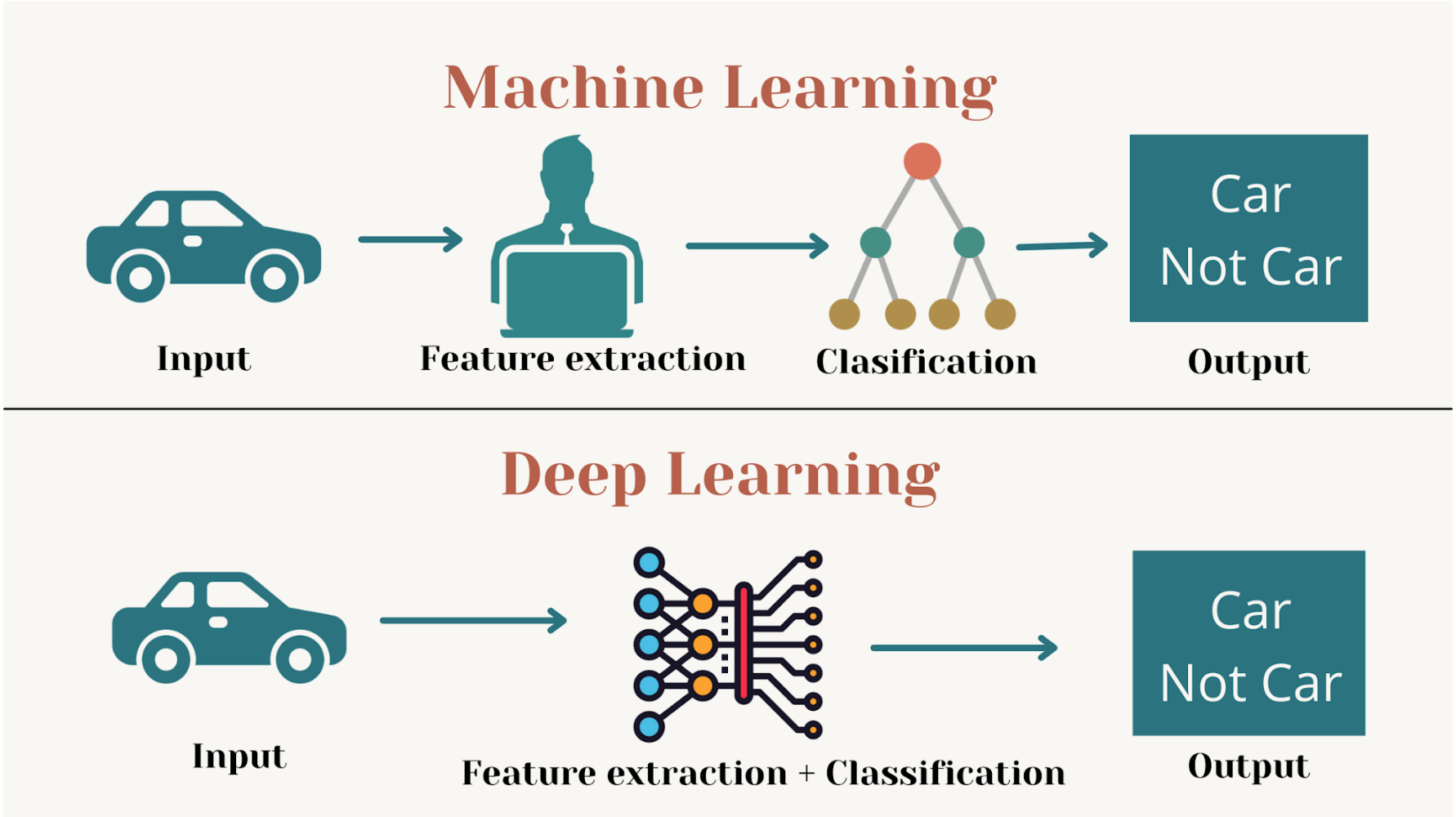
Artificial Intelligence : Application

- **AI in manufacturing:** Industrial robots that were at one time programmed to perform single tasks and separated from human workers, increasingly function as **cobots** : Smaller, multitasking robots that collaborate with humans and take on responsibility for more parts of the job in warehouses, factory floors and other workspaces.
- **AI in banking:** Banks are successfully employing **chatbots to make their customers aware of services** and offerings and to handle transactions that don't require human intervention. **AI virtual assistants** are being used to improve and cut the costs of compliance with banking regulations. Banking organizations are also using AI to improve their decision-making for loans, and to set credit limits and identify investment opportunities.
- **AI in transportation:** In addition to AI's fundamental role in operating autonomous vehicles, AI technologies are used in transportation to **manage traffic, predict flight delays**, and make ocean shipping safer and more efficient.

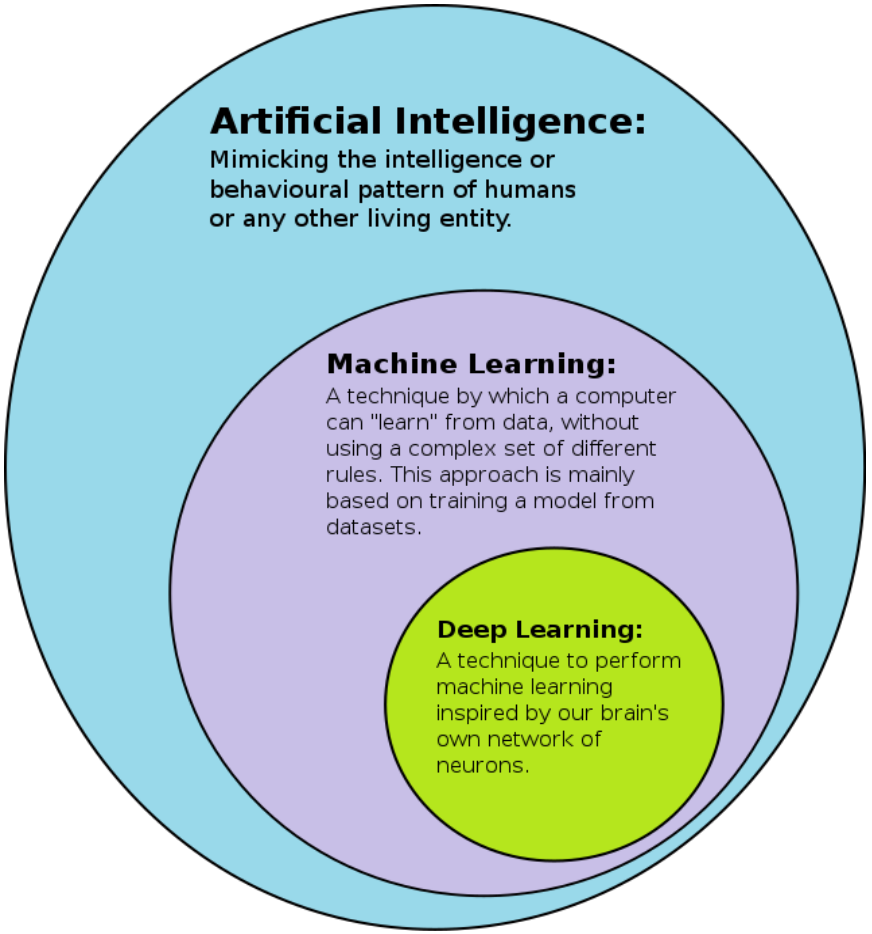
And Many more .....

Deep Learning

ML vs. DL


Machine Learning
Car
Not Car
Input
Feature extraction
Clasification
Output
Deep Learning
Car
Not Car
Input
Feature extraction + Classification
Output

AI vs. ML vs. DL


Artificial Intelligence:
Mimicking the intelligence or behavioural pattern of humans or any other living entity.
Machine Learning:
A technique by which a computer can "learn" from data, without using a complex set of different rules. This approach is mainly based on training a model from datasets.
Deep Learning:
A technique to perform machine learning inspired by our brain's own network of neurons.

Biological and Artificial Neuron

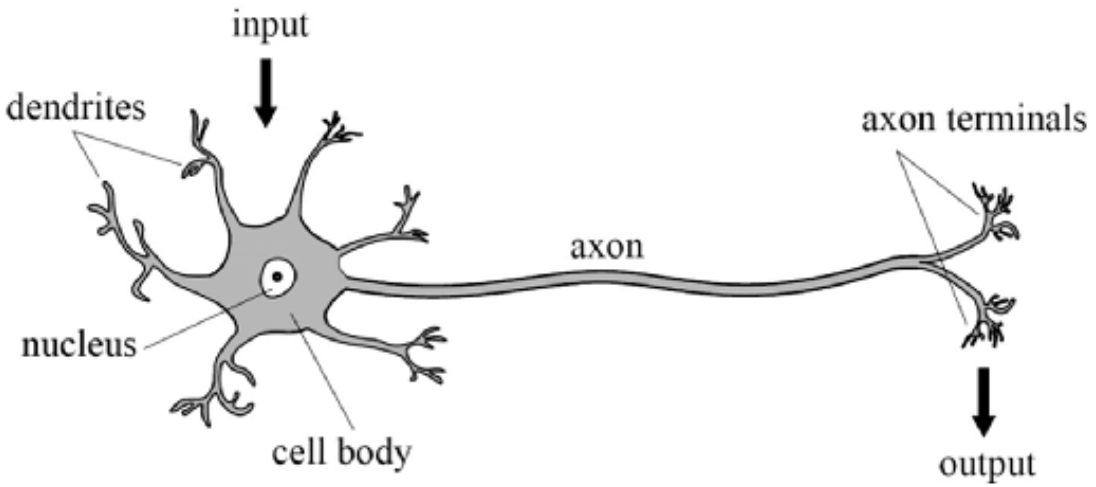
Neural Networks

- Deep Learning models utilize artificial neural networks (ANN) that are inspired by the biological neural network of the human brain.

- ANNs try to imitate neuronal activity in the brain in order to learn to recognize data patterns.

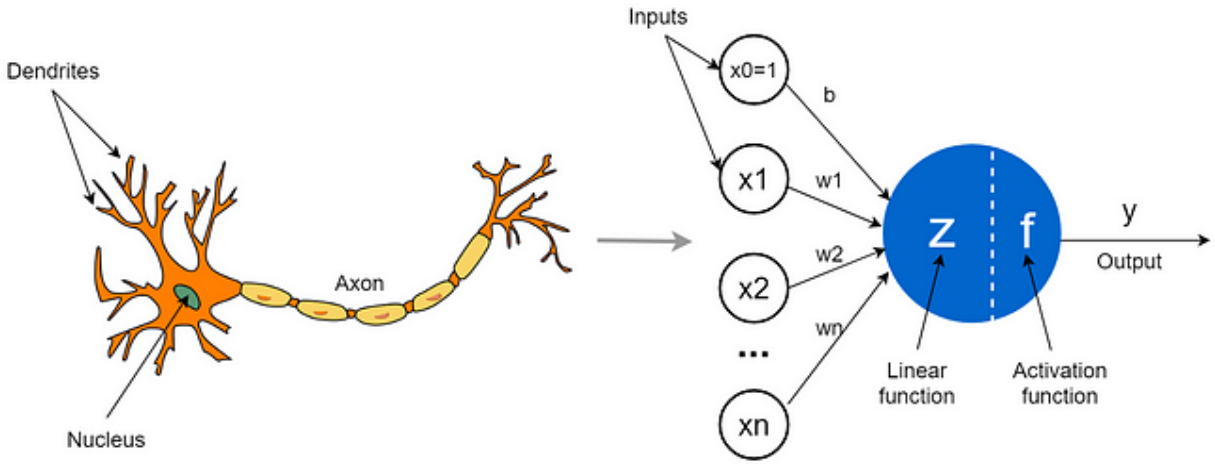
Biological Neuron

- **Neurons** are interconnected nerve cells that build the nervous system and transmit information throughout the body.
- **Dendrites** are extension of a nerve cell that receive impulses from other neurons.
- **Cell nucleus** stores cell's hereditary material and coordinates cell's activities.
- **Axon** is a nerve fiber that is used by neurons to transmit impulses.
- **Synapse** is the connection between two nerve cells.



Biological Neuron: Artificial Neuron

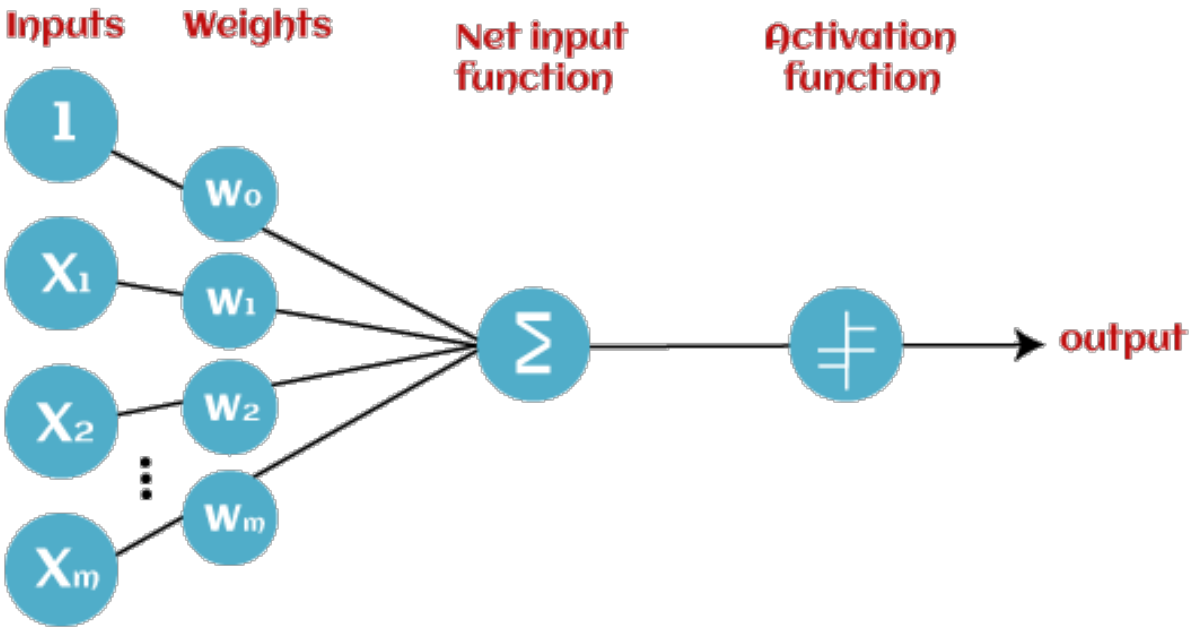
- An artificial neuron is analogous to biological neurons, where each neuron takes inputs, adds weights to them separately, sums them up, and passes this sum through a transfer function to produce a nonlinear output.
- Researchers Warren McCullock and Walter Pitts published their first concept of simplified brain cell in 1943.
- Nerve cell was considered similar to a simple logic gate with binary outputs.
- Dendrites can be assumed to process the input signal with a certain threshold such that if the signal exceeds the threshold, the output signal is generated.



Perceptron

Perceptron

- Perceptron is a building block of an Artificial Neural Network.
- Single layer neural network
- Consists of weights, the summation processor, and an activation function



Perceptron

- **Input Nodes or Input Layer:** This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

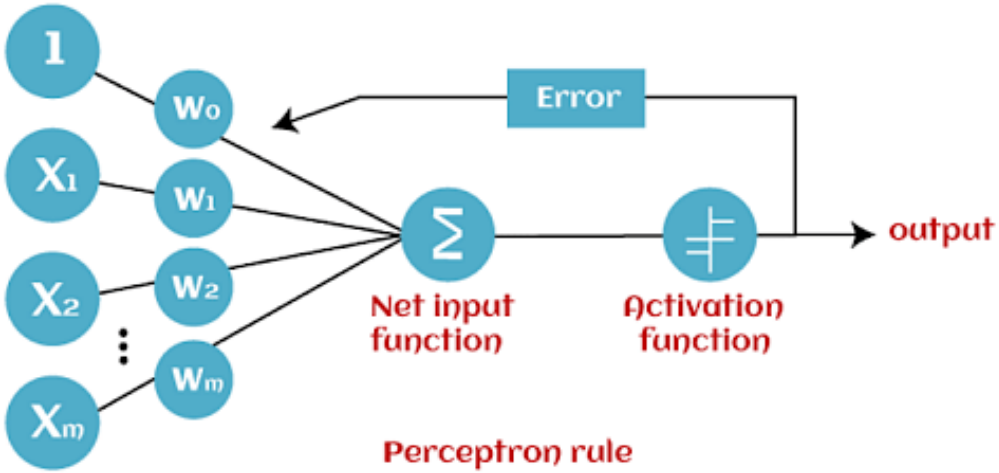
- **Weight and Bias:** Weight parameter represents the strength of the connection between units. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function:** These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

- The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

Perceptron : working

- Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not.



- The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output.

Perceptron : working

- **Perceptron Function**
- Perceptron is a function that maps its input “x,” which is multiplied with the learned weight coefficient; an output value ”f(x)”is generated.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

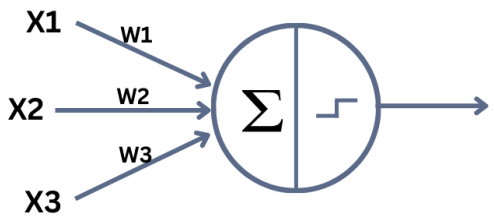
- In the equation given above:
  - “w” = vector of real-valued weights
  - “b” = bias (an element that adjusts the boundary away from origin without any dependence on the input value)
  - “x” = vector of input x values

- The output can be represented as “1” or “0.” It can also be represented as “1” or “-1” depending on which activation function is used.

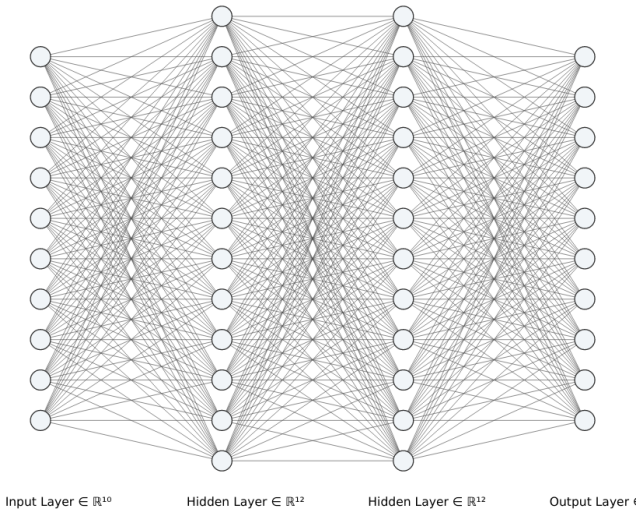
- **Error in Perceptron :** In the Perceptron Learning Rule, the predicted output is compared with the known output. If it does not match, the error is propagated backward to allow weight adjustment to happen.

Perceptron : Types

- **Single Layer Perceptron model:** One of the easiest ANN(Artificial Neural Networks) types consists of a feed-forward network and includes a threshold transfer inside the model. The main objective of the single-layer perceptron model is to analyse the linearly separable objects with binary outcomes. A Single-layer perceptron can learn only linearly separable patterns.
- **Multi-Layered Perceptron model:** It is mainly similar to a single-layer perceptron model but has more hidden layers.



Single-layer perceptron



Multi-layer perceptron

Feedforward Network

- "The process of receiving an input to produce some kind of output to make some kind of prediction is known as Feed Forward." Feed Forward neural network is the core of many other important neural networks such as convolution neural network.
- In the feed-forward neural network, there are not any feedback loops or connections in the network. Here is simply an input layer, a hidden layer, and an output layer.
- The single-layer perceptron is a popular feed-forward neural network model that is frequently used for classification. Single-layer perceptron can also contain machine learning features.
- During data flow, input nodes receive data, which travel through hidden layers, and exit output nodes. No links exist in the network that could get used to by sending information back from the output node.

Perceptron : limitations

- The following are the limitation of a Perceptron model:
  - The output of a perceptron can only be a binary number (0 or 1) due to the hard-edge transfer function.
  - It can only be used to classify the linearly separable sets of input vectors. If the input vectors are non-linear, it is not easy to classify them correctly.

Multilayer Perceptron

Multilayer Perceptron

- It is a neural network where the mapping between inputs and output is non-linear.

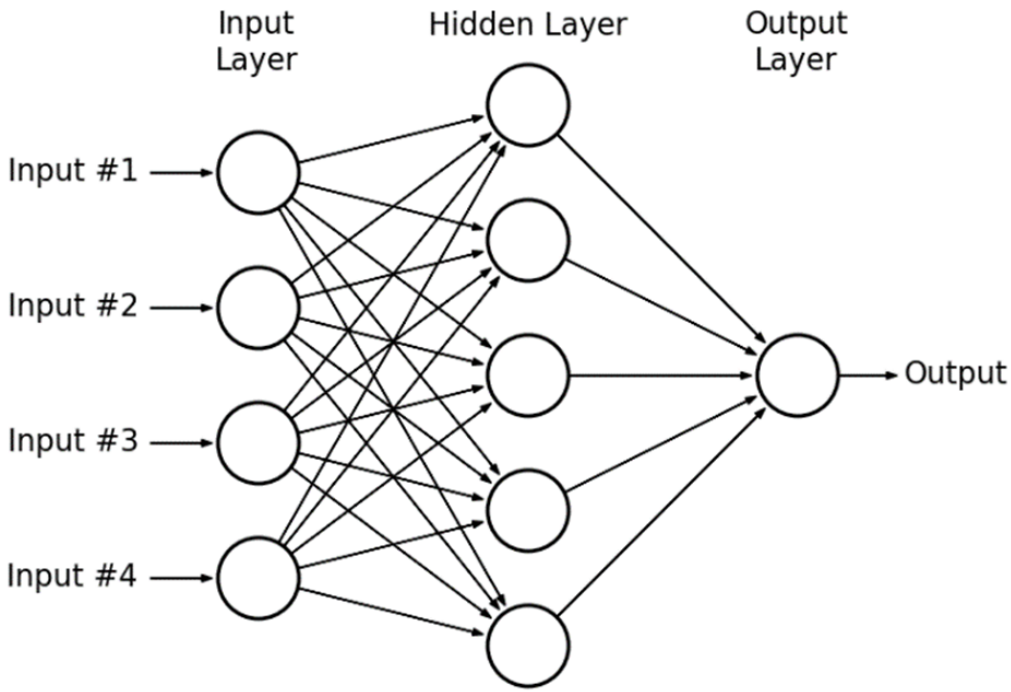
- A Multilayer Perceptron has input and output layers, and one or more hidden layers with many neurons stacked together. And while in the Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a Multilayer Perceptron can use any arbitrary activation function.

- Multilayer Perceptron falls under the category of feedforward algorithms, because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron. But the difference is that each linear combination is propagated to the next layer.

- Each layer is feeding the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer.

Multilayer Perceptron

- A multi-layer perceptron is a type of artificial neural network that is used for supervised learning and which can also be used to study computational neuroscience and parallel distributed processing. Applications include speech recognition, image recognition and machine translation.



Loss Function

What is Loss

- In any model, while prediction, the output deviates from the actual value, the measure of this difference is called loss.

- Here our loss will be, Actual value – Predicted Value, i.e., 10 – 8 = 2.

Loss function

- A loss function is a function that compares the target and predicted output values; measures how well the neural network models the training data. When training, we aim to minimize this loss between the predicted and target outputs.

- The hyperparameters are adjusted to minimize the average loss — we find the weights, wT, and biases, b, that minimize the value of J (average loss).

$$J(w^T, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

Types of Loss function

- There are two main types of loss functions — these correlate to the 2 major types of neural networks: regression and classification loss functions

- Regression Loss Functions — used in regression neural networks; given an input value, the model predicts a corresponding output value (rather than pre-selected labels); Ex. Mean Squared Error, Mean Absolute Error

- Classification Loss Functions — used in classification neural networks; given an input, the neural network produces a vector of probabilities of the input belonging to various pre-set categories — can then select the category with the highest probability of belonging; Ex. Binary Cross-Entropy, Categorical Cross-Entropy

Types of Loss function

- **Mean Squared Error (MSE)**
- One of the most popular loss functions, MSE finds the average of the squared differences between the target and the predicted outputs

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

- However, one disadvantage of this loss function is that it is very sensitive to outliers; if a predicted value is significantly greater than or less than its target value, this will significantly increase the loss

Types of Loss function

- **Mean Absolute Error (MAE)**
- MAE finds the average of the absolute differences between the target and the predicted outputs.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y^{(i)} - \hat{y}^{(i)}|$$

- This loss function is used as an alternative to MSE in some cases. As mentioned previously, MSE is highly sensitive to outliers, which can dramatically affect the loss because the distance is squared. MAE is used in cases when the training data has a large number of outliers to mitigate this.

Types of Loss function

- **Binary Cross-Entropy/Log Loss**
- This is the loss function used in binary classification models — where the model takes in an input and has to classify it into one of two pre-set categories.

$$CE\ Loss = \frac{1}{n}\sum_{i=1}^{N} -\ (y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)$$

- In binary classification, there are only two possible actual values of y — 0 or 1. Thus, to accurately determine loss between the actual and predicted values, it needs to compare the actual value (0 or 1) with the probability that the input aligns with that category (p(i) = probability that the category is 1; 1 — p(i) = probability that the category is 0)

Types of Loss function

- **Categorical Cross-Entropy Loss**
- In cases where the number of classes is greater than two, we utilize categorical cross-entropy — this follows a very similar process to binary cross-entropy.

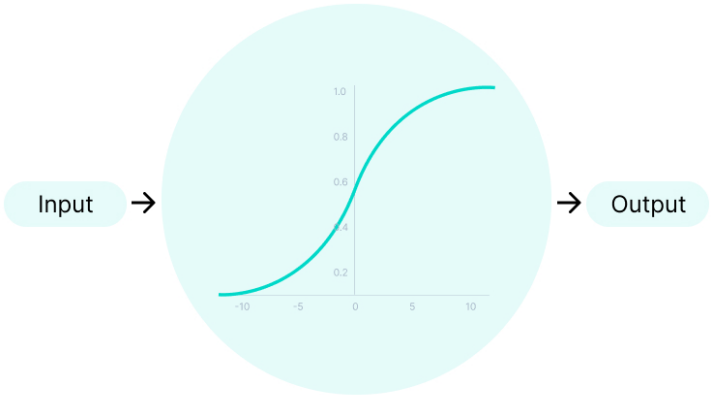
$$CE\ Loss = -\frac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot log(p_{ij})$$

- Binary cross-entropy is a special case of categorical cross-entropy, where M = 2 — the number of categories is 2.

Activation Function

Activation Function

- Activation functions are mathematical equations that determine the output of a neural network model. Activation functions also have a major effect on the neural network's ability to converge and the convergence speed, or in some cases, activation functions might prevent neural networks from converging in the first place. Activation function also helps to normalize the output of any input in the range between 1 to -1 or 0 to 1.
- An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations.



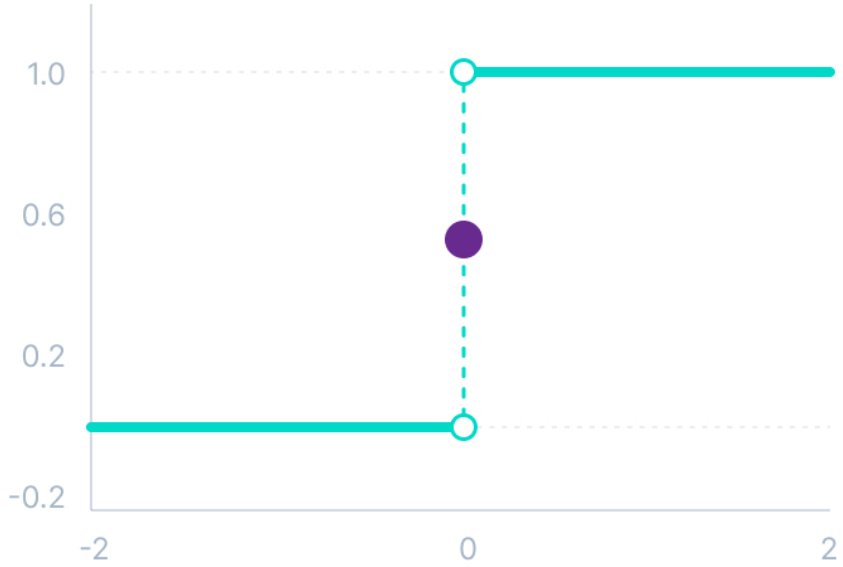
- The role of the Activation Function is to derive output from a set of input values fed to a node (or a layer).

Activation Function : Types : Step

- The Activation Functions can be basically divided into 3 types-

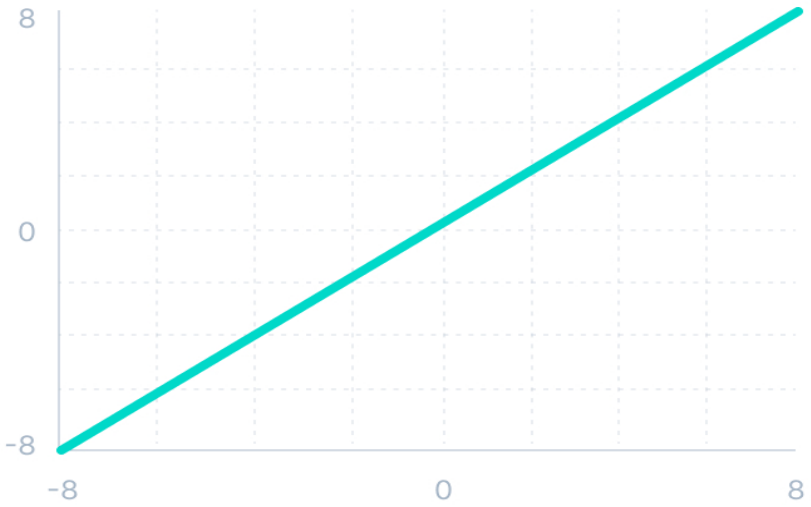
- **Binary Step Function**
  - Binary step function depends on a threshold value that decides whether a neuron should be activated or not.
  - The input fed to the activation function is compared to a certain threshold; if the input is greater than it, then the neuron is activated, else it is deactivated, meaning that its output is not passed on to the next hidden layer.

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$



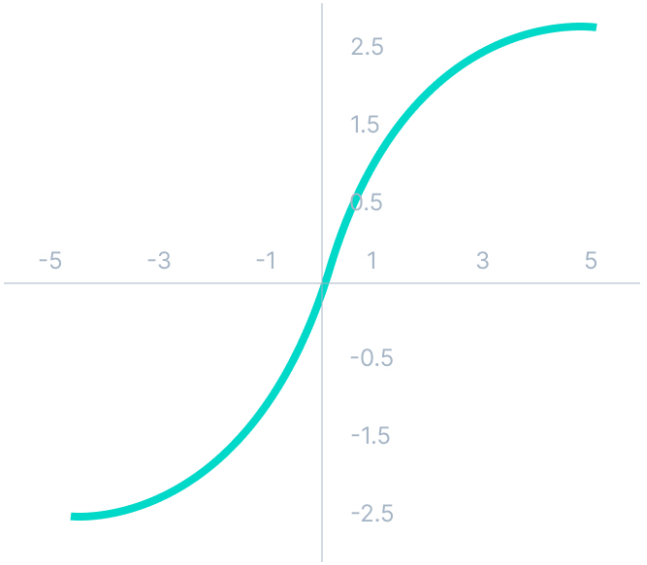
Activation Function : Types : Linear

- **Linear Activation Function**
  - As you can see the function is a line or linear.
  - Therefore, the output of the functions will not be confined between any range.
  - The linear activation function, also known as "no activation," or "identity function" (multiplied x1.0), is where the activation is proportional to the input.
  - The function doesn't do anything to the weighted sum of the input, it simply spits out the value it was given.



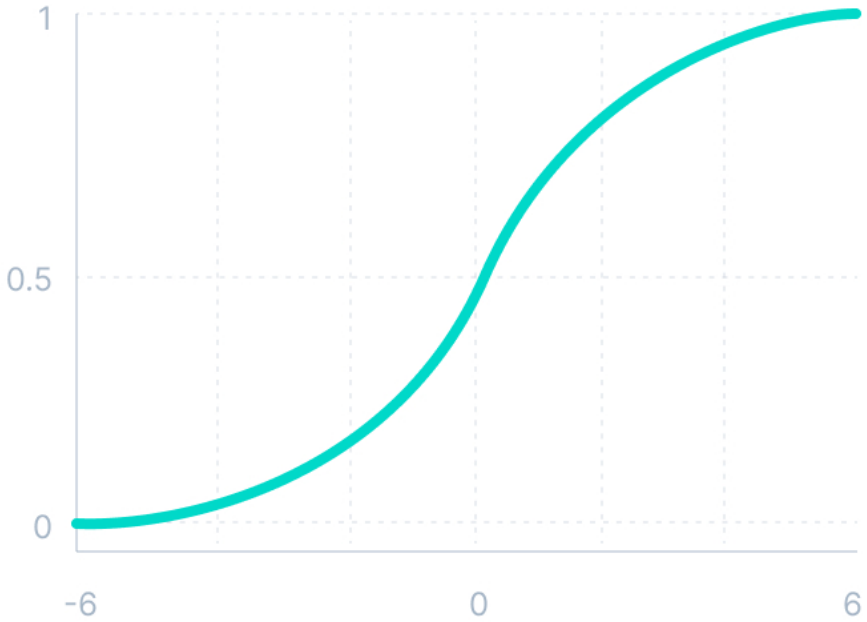
Activation Function : Types : Non-linear

- **Non-linear Activation Functions**
- Non-linear activation functions solve the following limitations of linear activation functions:
  - They allow backpropagation because now the derivative function would be related to the input, and it's possible to go back and understand which weights in the input neurons can provide a better prediction.
  - They allow the stacking of multiple layers of neurons as the output would now be a non-linear combination of input passed through multiple layers. Any output can be represented as a functional computation in a neural network.



Activation Function : Sigmoid

- **Sigmoid / Logistic Activation Function**
  - This function takes any real value as input and outputs values in the range of 0 to 1.
  - The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below.
  - It is commonly used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.
  - The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values. This is represented by an S-shape of the sigmoid activation function.

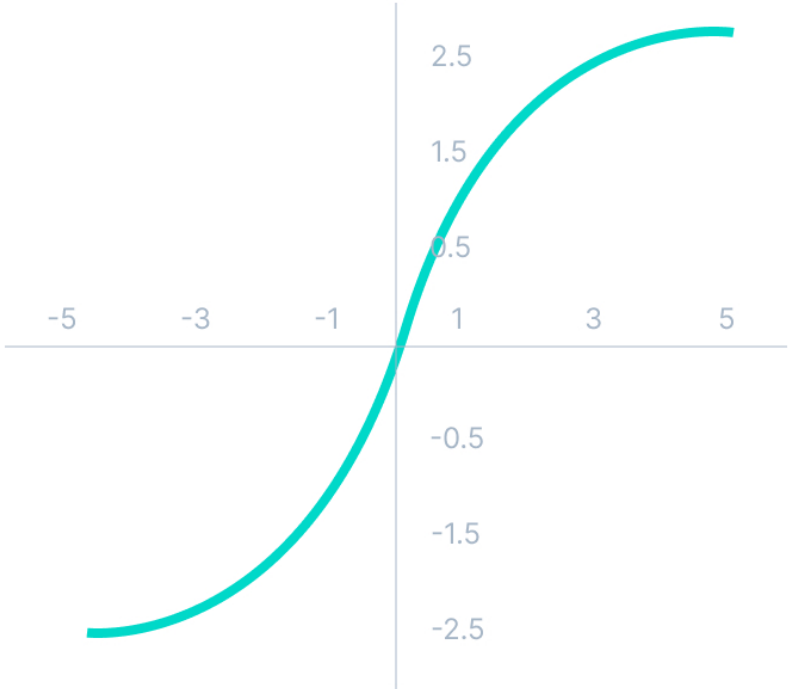


$$f(x) = \frac{1}{1 + e^{-x}}$$

Activation Function : Tanh

Tanh Function (Hyperbolic Tangent)

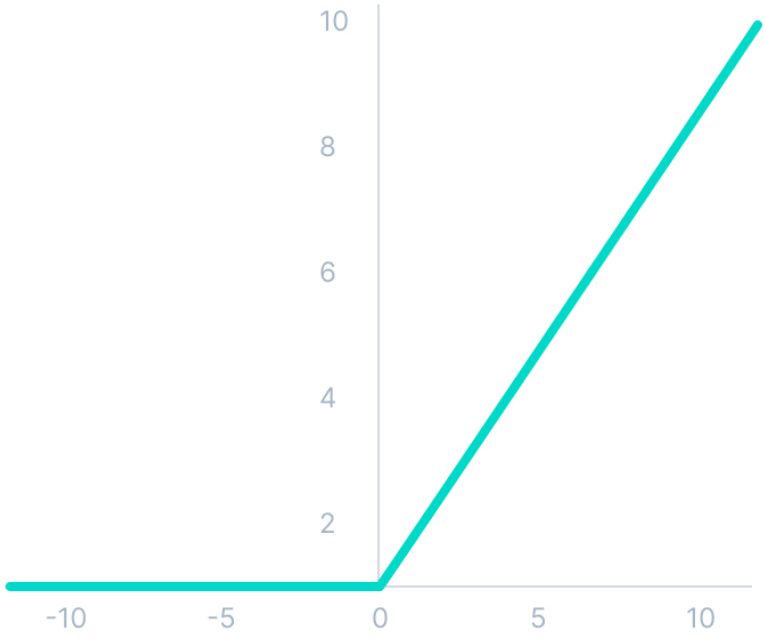
- Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1. In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.
- The output of the tanh activation function is Zero centered; hence we can easily map the output values as strongly negative, neutral, or strongly positive.
- Usually used in hidden layers of a neural network as its values lie between -1 to; therefore, the mean for the hidden layer comes out to be 0 or very close to it. It helps in centering the data and makes learning for the next layer much easier.



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

Activation Function : ReLU

- **ReLU Function**
  - ReLU stands for Rectified Linear Unit.
  - Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient.
  - The main catch here is that the ReLU function does not activate all the neurons at the same time.
  - The neurons will only be deactivated if the output of the linear transformation is less than 0.
  - Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh functions.
  - ReLU accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear, non-saturating property.



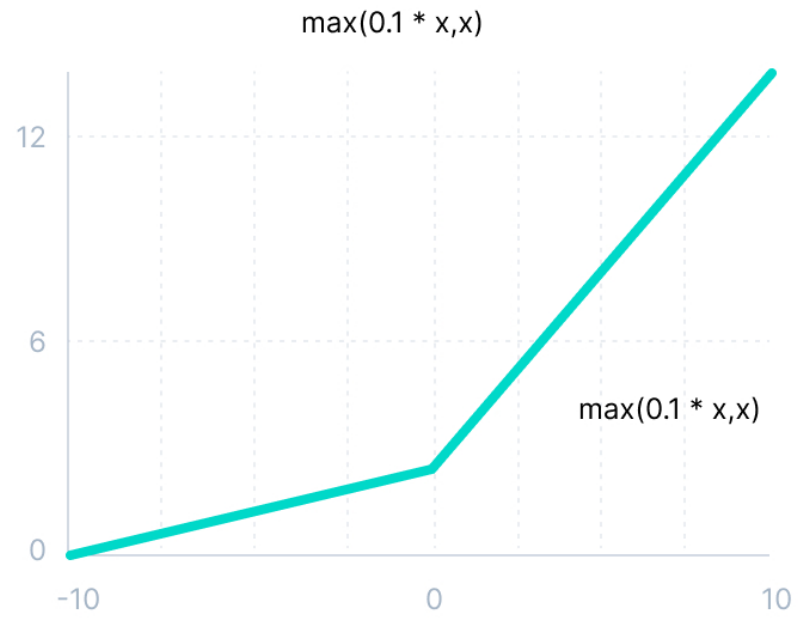
$$f(x) = max\,(0, x)$$

Activation Function : Limitation of ReLU

- **The Dying ReLU problem**
  - The negative side of the graph makes the gradient value zero. Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated.
  - All the negative input values become zero immediately, which decreases the model's ability to fit or train from the data properly.
  - Leaky ReLU is an improved version of ReLU function to solve the Dying ReLU problem as it has a small positive slope in the negative area.

Activation Function : Leaky ReLU

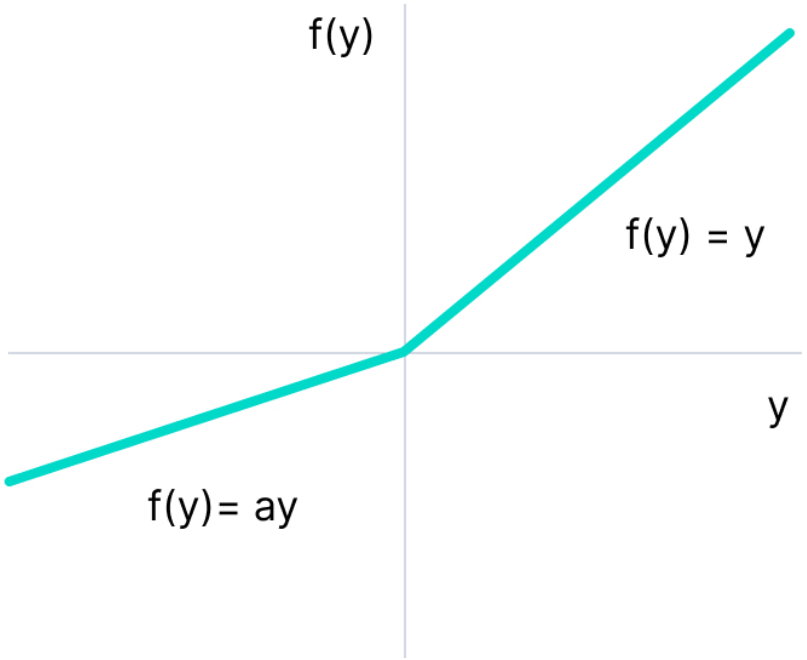
- **Leakey ReLU**
  - The advantages of Leaky ReLU are same as that of ReLU, in addition to the fact that it does enable backpropagation, even for negative input values.
  - By making this minor modification for negative input values, the gradient of the left side of the graph comes out to be a non-zero value. Therefore, we would no longer encounter dead neurons in that region.
  - The limitations that this function faces include:
    - The predictions may not be consistent for negative input values.
    - The gradient for negative values is a small value that makes the learning of model parameters time-consuming.



$$f(x) = max\ (0.1x, x)$$

Activation Function : Parametric ReLU

- Parametric ReLU Function
  - Parametric ReLU is another variant of ReLU that aims to solve the problem of gradient's becoming zero for the left half of the axis.
  - This function provides the slope of the negative part of the function as an argument a. By performing backpropagation, the most appropriate value of a is learnt.
  - Where "a" is the slope parameter for negative values.
  - The parameterized ReLU function is used when the leaky ReLU function still fails at solving the problem of dead neurons, and the relevant information is not successfully passed to the next layer.



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

Activation Function : How to choose right one

- You need to match your activation function for your output layer based on the type of prediction problem that you are solving—specifically, the type of predicted variable.
- Here's what you should keep in mind.
- As a rule of thumb, you can begin with using the ReLU activation function and then move over to other activation functions if ReLU doesn't provide optimum results.
- And here are a few other guidelines to help you out.
    - ReLU activation function should only be used in the hidden layers.
    - Sigmoid/Logistic and Tanh functions should not be used in hidden layers as they make the model more susceptible to problems during training (due to vanishing gradients).
    - Swish function is used in neural networks having a depth greater than 40 layers.

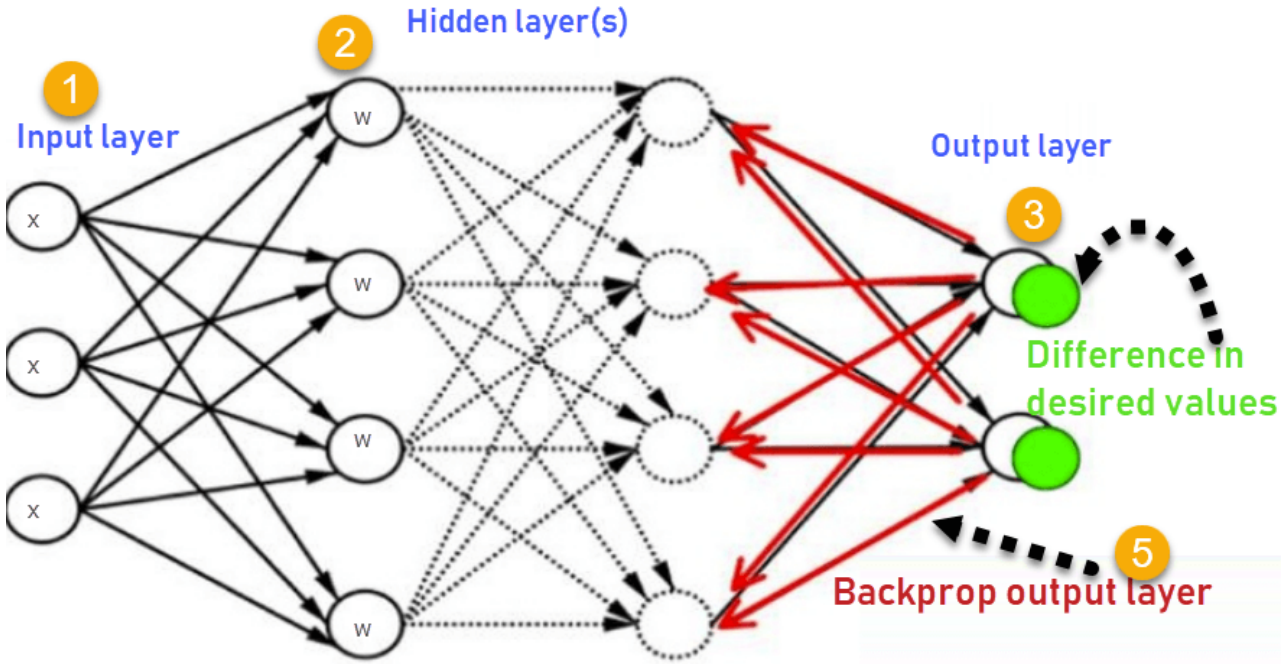
Activation Function : How to choose right one

- A few rules for choosing the activation function for your output layer based on the type of prediction problem that you are solving:
- Regression - Linear Activation Function
- Binary Classification—Sigmoid/Logistic Activation Function
- Multiclass Classification—Softmax
- Multilabel Classification—Sigmoid
- The activation function used in hidden layers is typically chosen based on the type of neural network architecture.
- Convolutional Neural Network (CNN): ReLU activation function.
- Recurrent Neural Network: Tanh and/or Sigmoid activation function.

Back Propagation

Back Propagation

- Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.
- In other words, backpropagation aims to minimize the cost function by adjusting network's weights and biases. The level of adjustment is determined by the gradients of the cost function with respect to those parameters.



TensorFlow
Keras

Tensorflow

TensorFlow

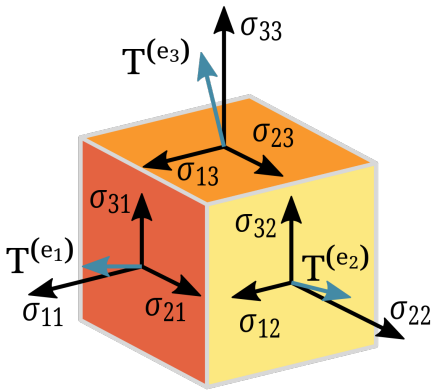
- **Created by the Google Brain team** and initially released to the public in **2015,** TensorFlow is an **open source library** for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning models and algorithms (aka neural networks) and makes them useful by way of common programmatic metaphors. **It uses Python or JavaScript** to provide a convenient front-end API for building applications, while executing those applications in high-performance C++.
- TensorFlow, can train and run deep neural networks for **handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE** (partial differential equation)-based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.
- TensorFlow also has a **broad library of pre-trained models** that can be used in your own projects.

Tensorflow: How it works

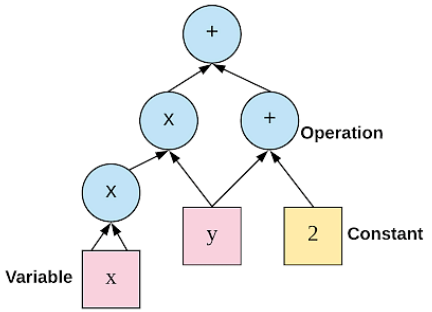
- TensorFlow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.
- It takes inputs as a multi-dimensional array where you can construct a flowchart of operations that can be performed on these inputs.
- Tensorflow architecture works in three significant steps:
  - Data pre-processing - structure the data and brings it under one limiting value
  - Building the model - build the model for the data
  - Training and estimating the model - use the data to train the model and test it with unknown data
- TensorFlow requirements can be classified into the development phase (training the model) and run phase (running the model on different platforms). The model can be trained and used on GPUs as well as CPUs. Once the model has been trained, you can run it on:
  - Desktop (Linux, Windows, macOS)
  - Mobile devices (iOS and Android)
  - Cloud as a web service

Tensorflow: Components

- **Tensor :** Tensor forms the core framework of TensorFlow. All the computations in TensorFlow involve tensors. It is a matrix of n-dimensions that represents multiple types of data. A tensor can be the result of a computation or it can originate from the input data

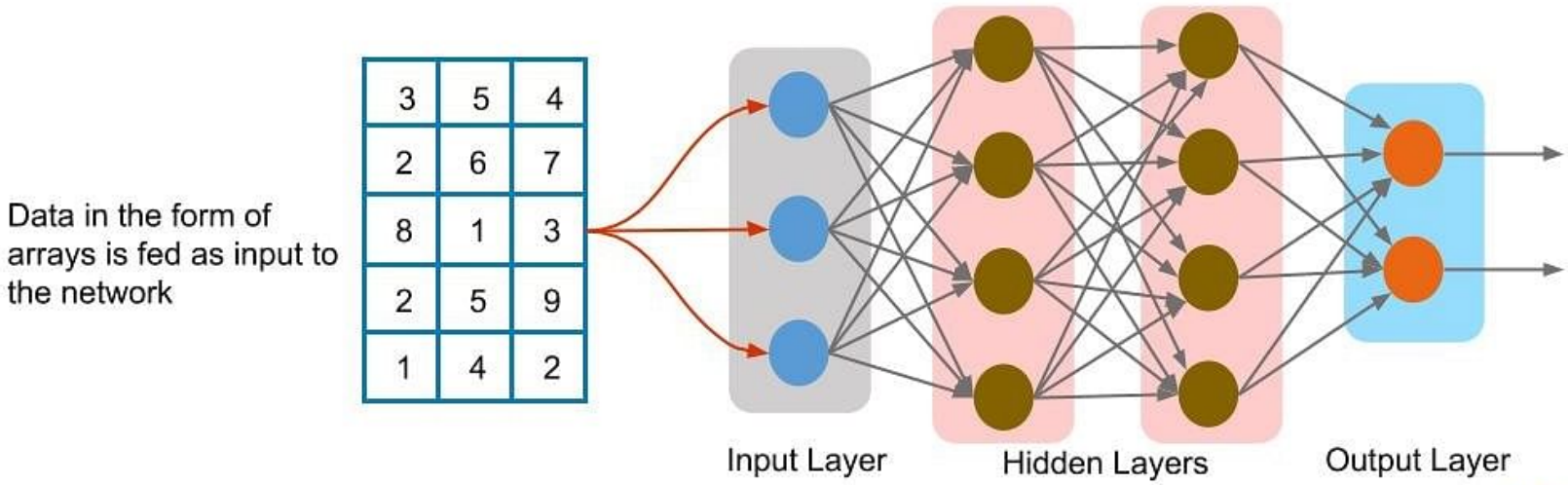


- **Graphs :** Graphs describe all the operations that take place during the training. Each operation is called an op node and is connected to the other. The graph shows the op nodes and the connections between the nodes, but it does not display values.



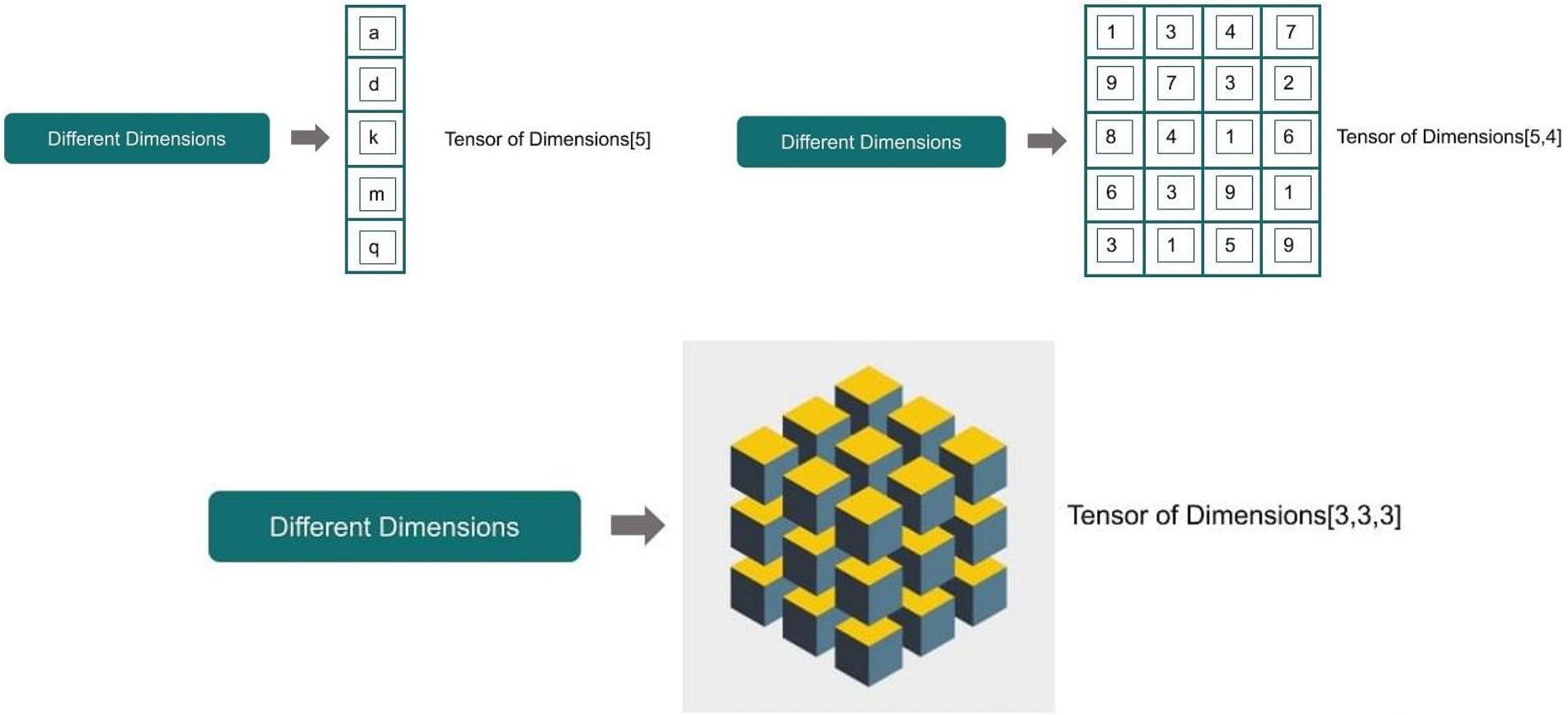
Tensorflow: Tensor

- **What are Tensors ?**
- Tensor is a generalization of vectors and matrices of potentially higher dimensions. Arrays of data with varying dimensions and ranks that are fed as input to the neural network are called tensors.
- For deep learning, especially in the training process, you will have large amounts of data that exist in a very complicated format. It helps when you are able to put, use, or store it in a compact way, which tensors provide, even if they appear in multi-dimensional arrays. When the data is stored in tensors and fed into the neural network, the output we get is as shown below:



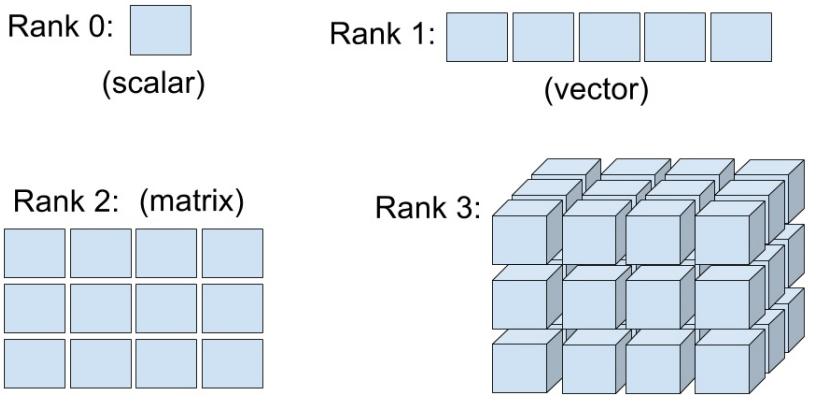
Tensorflow: Tensor

- **Dimension :** Dimension is the size of the array elements. Below you can take a look at various types of dimensions:



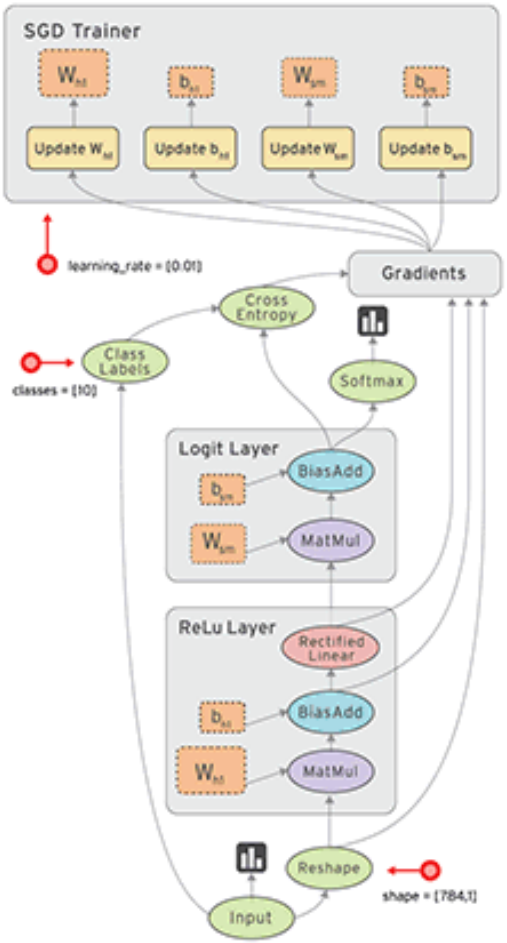
Tensorflow: Tensor

- **Ranks :** Tensor ranks are the number of dimensions used to represent the data. For example:
- Rank 0 - When there is only one element. We also call this as a scalar.
  - Example: s = [2000]
- Rank 1 - This basically refers to a one-dimensional array called a vector.
  - Example: v = [10, 11, 12]
- Rank 2 - This is traditionally known as a two-dimensional array or a matrix.
  - Example: m = [1,2,3],[4,5,6]
- Rank 3 - It refers to a multidimensional array, generally referred to as tensor.
  - Example: t = [[[1],[2],[3]],[[4],[5],[6]],[[7],[8],[9]]]
- Ranks can then be four or five, and so on.



Tensorflow: Data Flow Graph

- **What is a Data Flow Graph?**
  - When we have the data stored in tensors, there are computations that need to be completed, which happens in the form of graphs.
  - Unlike traditional programming, where written code gets executed in sequence, here we build data flow graphs that consist of nodes. The graphs are then executed in the form of a session. It is important to remember that we first have to create a graph. When we do so, none of the code is actually getting executed. You execute that graph only by creating a session.
  - Each computation in TensorFlow is represented as a data flow graph below.



Tensorflow : Algorithms Supported

- **Here is the list of algorithms currently supported by TensorFlow:**
  - Classification - tf.estimator.LinearClassifier
  - Linear regression - tf.estimator.LinearRegressor
  - Boosted tree classification - tf.estimator.BoostedTreesClassifier
  - Booster tree regression - tf.estimator.BoostedTreesRegressor
  - Deep learning classification - tf.estimator.DNNClassifier
  - Deep learning wipe and deep - tf.estimator.DNNLinearCombinedClassifier

Tensorflow: Program Elements

- **Program Elements in TensorFlow**
  - TensorFlow programs work on two basic concepts:
    - Building a computational graph
    - Executing a computational graph
  - First, you need to start by writing the code for preparing the graph. Following this, you create a session where you execute this graph.
  - TensorFlow programming is slightly different from regular programming. Even if you're familiar with Python programming or machine learning programming in sci-kit-learn, this may be a new concept to you.
  - The way data is handled inside of the program itself is a little different from how it normally is with the regular programming language. For anything that keeps changing in regular programming, a variable needs to be created.
  - In TensorFlow, however, data can be stored and manipulated using three different programming elements:
    - Constants
    - Variables
    - Placeholders

Tensorflow: Program Elements

- **Constants**
  - Constants are parameters with values that do not change. To define a constant, we use tf.constant() command
  - Example:
    - a = tf.constant(2.0, tf.float32)
    - b = tf.constant(3.0)
    - Print(a, b)
  - In the case of constants, you cannot change their values during the computation.
- **Variables**
  - Variables allow us to add new trainable parameters to the graph. To define a variable, we use tf.Variable() command and initialize it before running the graph in a session.
  - Example:
    - W = tf.Variable([.3],dtype=tf.float32)
    - b = tf.Variable([-.3],dtype=tf.float32)
    - x = tf.placeholder(tf.float32)
  - linear_model = W*x+b

Tensorflow: Program Elements

- **Placeholders**
  - Placeholders allow us to feed data to a TensorFlow model from outside a model. It permits value to be assigned later. To define a placeholder, we use the tf.placeholder() command.
  - Example:
    - a = tf.placeholder(tf.float32)
    - b = a*2
    - with tf.Session() as sess:
    - result = sess.run(b,feed_dict={a:3.0})
  - print result

Tensorflow: Program Elements

- Placeholders are a special type of the variable and can be a new concept for many of us. Placeholders are like variables, but they are used for feeding data from outside. Typically, when you are performing computations, you need to load data from a local file or from an image file, CSV file, etc. There is a provision with special types of variables, which can be fed on a regular basis. One of the reasons for having this kind of provision is that if you get the entire input in one shot, it may become very difficult to handle the memory.
- There is a certain way of populating the placeholder called feed_dict, which specifies tensors that provide values to the placeholder.
- In a nutshell, constants, variables, and placeholders handle data within the flow program, after which you have to create a graph and run a session.

Tensorflow: Program Elements

- **Session** : A session is run to evaluate the nodes. This is called as the TensorFlow Runtime.
- Example:
  - a = tf.constant(5.0)
  - b = tf.constant(3.0)
  - c = a*b
  - # Launch Session
  - sess = tf.Session()
  - # Evaluate the tensor c
  - print(sess.run(c))
- When creating a session, you run a particular computation, node, or an operation. Every variable or computation that you perform is like an operation on a node within a graph. Initially, the graph will be the default one. The moment you create a TensorFlow object, there is a default graph that doesn't contain any operations or nodes.

Tensorflow: Program Elements

- **Session** : A session is run to evaluate the nodes. This is called as the TensorFlow Runtime.
- Example:
  - a = tf.constant(5.0)
  - b = tf.constant(3.0)
  - c = a*b
  - # Launch Session
  - sess = tf.Session()
  - # Evaluate the tensor c
  - print(sess.run(c))
- When creating a session, you run a particular computation, node, or an operation. Every variable or computation that you perform is like an operation on a node within a graph. Initially, the graph will be the default one. The moment you create a TensorFlow object, there is a default graph that doesn't contain any operations or nodes.

Keras

Keras

- Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.
- Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. This makes Keras slower than other deep learning frameworks, but extremely beginner-friendly.

Keras

- Keras allows you to switch between different back ends. The frameworks supported by Keras are:
  - Tensorflow
  - Theano
  - PlaidML
  - MXNet
  - CNTK (Microsoft Cognitive Toolkit)
- Out of these five frameworks, TensorFlow has adopted Keras as its official high-level API. Keras is embedded in TensorFlow and can be used to perform deep learning fast as it provides inbuilt modules for all neural network computations. At the same time, computation involving tensors, computation graphs, sessions, etc can be custom made using the Tensorflow Core API, which gives you total flexibility and control over your application and lets you implement your ideas in a relatively short time.

Why we need Keras ?

- Keras is an API that was made to be easy to learn for people. Keras was made to be simple. It offers consistent & simple APIs, reduces the actions required to implement common code, and explains user error clearly.
- Prototyping time in Keras is less. This means that your ideas can be implemented and deployed in a shorter time. Keras also provides a variety of deployment options depending on user needs.
- Languages with a high level of abstraction and inbuilt features are slow and building custom features in then can be hard. But Keras runs on top of TensorFlow and is relatively fast. Keras is also deeply integrated with TensorFlow, so you can create customized workflows with ease.
- The research community for Keras is vast and highly developed. The documentation and help available are far more extensive than other deep learning frameworks.
- Keras is used commercially by many companies like Netflix, Uber, Square, Yelp, etc which have deployed products in the public domain which are built using Keras.

Building model in Keras

- **Define a network:** In this step, you define the different layers in our model and the connections between them. Keras has two main types of models: Sequential and Functional models. You choose which type of model you want and then define the dataflow between them.
- **Compile a network:** To compile code means to convert it in a form suitable for the machine to understand. In Keras, the model.compile() method performs this function. To compile the model, we define the loss function which calculates the losses in our model, the optimizer which reduces the loss, and the metrics which is used to find the accuracy of our model.
- **Fit the network:** Using this, we fit our model to our data after compiling. This is used to train the model on our data.
- **Evaluate the network:** After fitting our model, we need to evaluate the error in our model.
- **Make Predictions:** We use model.predict() to make predictions using our model on new data.

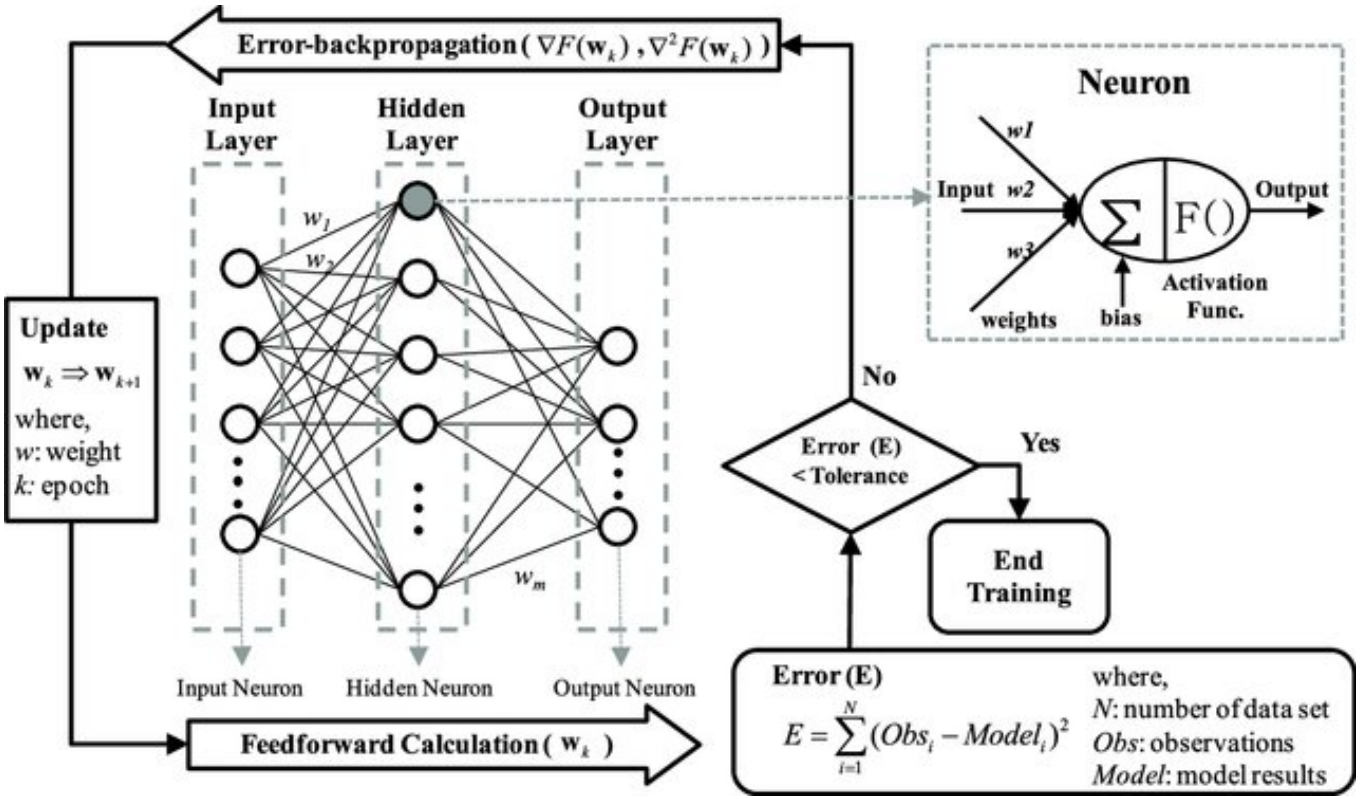
Keras : Application

- Keras is used for creating deep models which can be productized on smartphones.
- Keras is also used for distributed training of deep learning models.
- Keras is used by companies such as Netflix, Yelp, Uber, etc.
- Keras is also extensively used in deep learning competitions to create and deploy working models, which are fast in a short amount of time.

Vanishing & Exploding Gradient Problem

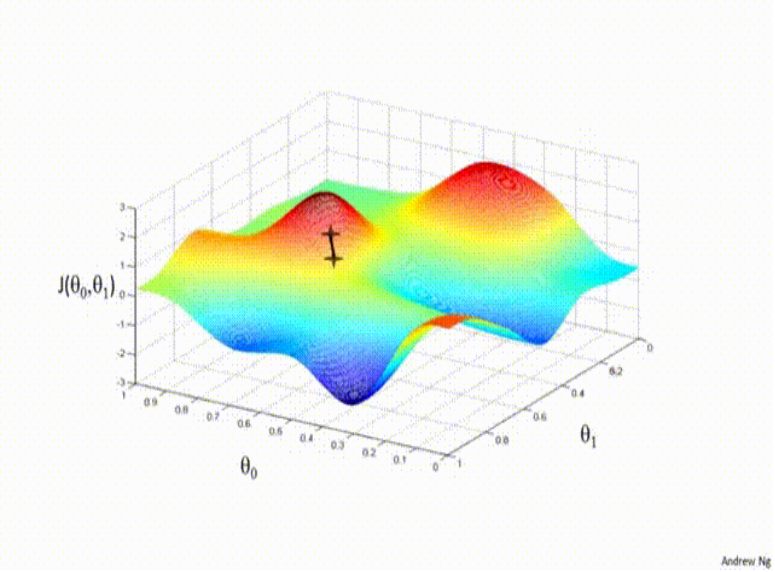
Gradient Problem

- We know that the backpropagation algorithm is the heart of neural network training. Let's have a glimpse over this algorithm that has proved to be a harbinger in the evolution as well as the revolution of Deep Learning.



Gradient Problem

- After propagating the input features forward to the output layer through the various hidden layers consisting of different/same activation functions, we come up with a predicted probability of a sample belonging to the positive class (generally, for classification tasks).
- Now, the backpropagation algorithm propagates backward from the output layer to the input layer calculating the error gradients on the way.
- Once the computation for gradients of the cost function w.r.t each parameter (weights and biases) in the neural network is done, the algorithm takes a gradient descent step towards the minimum to update the value of each parameter in the network using these gradients.



Gradient Problem

- **Vanishing –**
  - As the backpropagation algorithm advances downwards(or backward) from the output layer towards the input layer, the gradients often get smaller and smaller and approach zero which eventually leaves the weights of the initial or lower layers nearly unchanged. As a result, the gradient descent never converges to the optimum. This is known as the vanishing gradients problem.
- **Exploding –**
  - On the contrary, in some cases, the gradients keep on getting larger and larger as the backpropagation algorithm progresses. This, in turn, causes very large weight updates and causes the gradient descent to diverge. This is known as the exploding gradients problem.
- In a network of n hidden layers, n derivatives will be multiplied together. If the derivatives are large then the gradient will increase exponentially as we propagate down the model until they eventually explode, and this is what we call the problem of exploding gradient. Alternatively, if the derivatives are small then the gradient will decrease exponentially as we propagate through the model until it eventually vanishes, and this is the vanishing gradient problem.

Gradient Problem

- In the case of exploding gradients, the accumulation of large derivatives results in the model being very unstable and incapable of effective learning, The large changes in the models weights creates a very unstable network, which at extreme values the weights become so large that is causes overflow resulting in NaN weight values of which can no longer be updated.

- On the other hand, the accumulation of small gradients results in a model that is incapable of learning meaningful insights since the weights and biases of the initial layers, which tends to learn the core features from the input data (X), will not be updated effectively. In the worst case scenario the gradient will be 0 which in turn will stop the network will stop further training.

How to know : Exploding Gradient

- The model is not learning much on the training data therefore resulting in a poor loss.
- The model will have large changes in loss on each update due to the models instability.
- The models loss will be NaN during training.
- When faced with these problems, to confirm whether the problem is due to exploding gradients, there are some much more transparent signs, for instance:
- Model weights grow exponentially and become very large when training the model.
- The model weights become NaN in the training phase.
- The derivatives are constantly

How to know : Vanishing Gradient

- The model will improve very slowly during the training phase and it is also possible that training stops very early, meaning that any further training does not improve the model.
- The weights closer to the output layer of the model would witness more of a change whereas the layers that occur closer to the input layer would not change much (if at all).
- Model weights shrink exponentially and become very small when training the model.
- The model weights become 0 in the training phase.

Gradient Problem : Solution

- **Reducing the amount of Layers**
  - This is the solution could be used in both, scenarios (exploding and vanishing gradient). However, by reducing the amount of layers in our network, we give up some of our models complexity, since having more layers makes the networks more capable of representing complex mappings.
- **Gradient Clipping (Exploding Gradients)**
  - Checking for and limiting the size of the gradients whilst our model trains is another solution.
- **Weight Initialization**
  - A more careful initialization choice of the random initialization for your network tends to be a partial solution, since it does not solve the problem completely.
- **Using Non-saturating Activation Functions**
  - While studying the nature of sigmoid activation function, we observed that its nature of saturating for larger inputs (negative or positive) came out to be a major reason behind the vanishing of gradients thus making it non-recommendable to use in the hidden layers of the network.
  - So to tackle the issue regarding the saturation of activation functions like sigmoid and tanh, we must use some other non-saturating functions like ReLu and its alternatives.


Thank you!