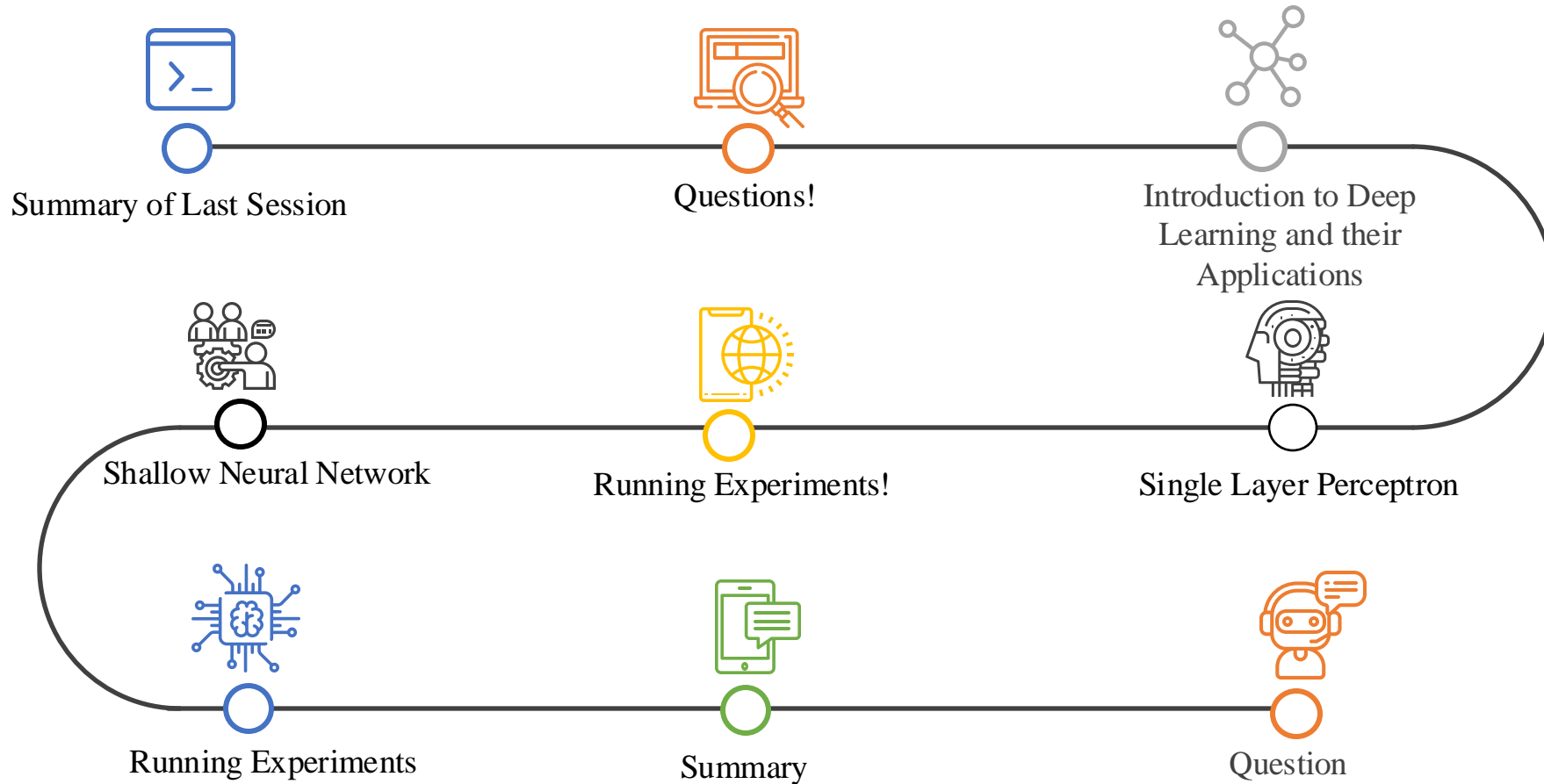





Today Outline







Summary of Python

Keywords


Keyword	Description	Code Examples
False, True	Boolean data type	False == (1 > 2) True == (2 > 1) 
and, or, not	Logical operators → Both are true → Either is true → Flips Boolean	True and True # True True or False # True not False # True
break	Ends loop prematurely	while True: break # finite loop
continue	Finishes current loop iteration	while True: continue print("42") # dead code
class	Defines new class	class Coffee: # Define your class
def	Defines a new function or class method.	def say_hi(): print('hi')
if, elif, else	Conditional execution: - "if" condition == True? - "elif" condition == True? - Fallback: else branch	x = int(input("ur val:")) if x > 3: print("Big") elif x == 3: print("3") else: print("Small")
for, while	# For loop for i in [0,1,2]: print(i)	# While loop does same j = 0 while j < 3: print(j); j = j + 1 
in	Sequence membership	42 in [2, 39, 42] # True
is	Same object memory location	y = x = 3 x is y # True [3] is [3] # False
None	Empty value constant	print() is None # True
lambda	Anonymous function	(lambda x: x+3)(3) # 6 
return	Terminates function. Optional return value defines function result.	def increment(x): return x + 1 increment(4) # returns 5

Basic Data Structures

Type	Description	Code Examples
Boolean	The Boolean data type is either True or False . Boolean operators are ordered by priority: not → and → or { } →  {1, 2, 3} → 	## Evaluates to True: 1<2 and 0<=1 and 3>2 and 2>=2 and 1==1 and 1!=0 ## Evaluates to False: bool(None or 0 or 0.0 or '' or [] or {} or set()) Rule: None, 0, 0.0, empty strings, or empty container types evaluate to False
Integer, Float	An integer is a positive or negative number without decimal point such as 3. A float is a positive or negative number with floating point precision such as 3.1415926. Integer division rounds toward the smaller integer (example: 3//2==1).	## Arithmetic Operations x, y = 3, 2 print(x + y) # = 5 print(x - y) # = 1 print(x * y) # = 6 print(x / y) # = 1.5 print(x // y) # = 1 print(x % y) # = 1 print(-x) # = -3 print(abs(-x)) # = 3 print(int(3.9)) # = 3 print(float(3)) # = 3.0 print(x ** y) # = 9 
String	Python Strings are sequences of characters. String Creation Methods: 1. Single quotes >>> 'Yes' >>> "Yes" 2. Double quotes >>> "Yes" 3. Triple quotes (multi-line) >>> """Yes We Can""" 4. String method >>> str(5) == '5' True 5. Concatenation >>> "Ma" + "hatma" 'Mahatma' Whitespace chars: Newline \n, Space \s, Tab \t	## Indexing and Slicing s = "The youngest pope was 11 years" s[0] # 'T' s[1:3] # 'he' s[-3:-1] # 'ar' s[-3:] # 'ars' x = s.split() x[-2] + " " + x[2] + "s" # '11 popes' ## String Methods y = " Hello world\t\n " y.strip() # Remove Whitespace "HI".lower() # Lowercase: 'hi' "hi".upper() # Uppercase: 'HI' "hello".startswith("he") # True "hello".endswith("lo") # True "hello".find("ll") # Match at 2 "cheat".replace("ch", "m") # 'meat' ''.join(["F", "B", "I"]) # 'FBI' len("hello world") # Length: 15 "ear" in "earth" # True 

Summary of Python

Complex Data Structures

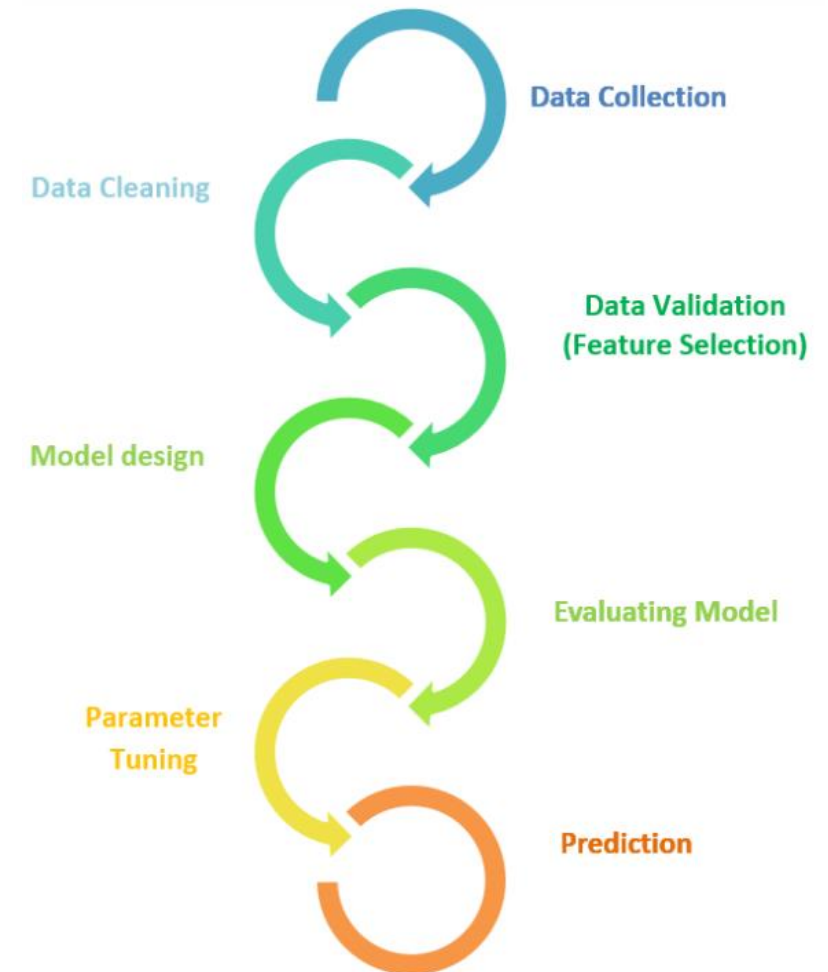
Type	Description	Example
List	Stores a sequence of elements. Unlike strings, you can modify list objects (they're mutable).	<pre>l = [1, 2, 2] print(len(l)) # 3</pre> 
Adding elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation.	<pre>[1, 2].append(4) # [1, 2, 4] [1, 4].insert(1, 9) # [1, 9, 4] [1, 2] + [4] # [1, 2, 4]</pre>
Removal	Slow for lists	<pre>[1, 2, 2, 4].remove(1) # [2, 2, 4]</pre>
Reversing	Reverses list order	<pre>[1, 2, 3].reverse() # [3, 2, 1]</pre>
Sorting	Sorts list using fast Timsort	<pre>[2, 4, 2].sort() # [2, 2, 4]</pre>
Indexing	Finds the first occurrence of an element & returns index. Slow worst case for whole list traversal.	<pre>[2, 2, 4].index(2) # index of item 2 is 0 [2, 2, 4].index(2, 1) # index of item 2 after pos 1 is 1</pre>
Stack	Use Python lists via the list operations append() and pop()	<pre>stack = [] stack.append(42) # [3, 42] stack.pop() # 42 (stack: [3]) stack.pop() # 3 (stack: [])</pre>
Set	An unordered collection of unique elements (at-most-once) → fast membership $O(1)$	<pre>basket = {'apple', 'eggs', 'banana', 'orange'} same = set(['apple', 'eggs', 'banana', 'orange'])</pre>

Type	Description	Example
Dictionary	Useful data structure for storing (key, value) pairs	<pre>cal = {'apple' : 52, 'banana' : 89, 'choco' : 546} # calories</pre>
Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the <code>keys()</code> and <code>values()</code> functions to access all keys and values of the dictionary	<pre>print(cal['apple'] < cal['choco']) # True cal['cappu'] = 74 print(cal['banana'] < cal['cappu']) # False print('apple' in cal.keys()) # True print(52 in cal.values()) # True</pre>
Dictionary iteration	You can access the (key, value) pairs of a dictionary with the <code>items()</code> method.	<pre>for k, v in cal.items(): print(k if v > 500 else '') # 'choco'</pre>
Membership operator	Check with the <code>in</code> keyword if set, list, or dictionary contains an element. Set membership is faster than list membership.	<pre>basket = {'apple', 'eggs', 'banana', 'orange'} print('eggs' in basket) # True print('mushroom' in basket) # False</pre>
List & set comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a for clause. Close with zero or more for or if clauses. Set comprehension works similar to list comprehension.	<pre>l = ['hi ' + x for x in ['Alice', 'Bob', 'Pete']] # ['Hi Alice', 'Hi Bob', 'Hi Pete'] l2 = [x * y for x in range(3) for y in range(3) if x > y] # [0, 0, 2] squares = { x**2 for x in [0, 2, 4] if x < 4 } # {0, 4}</pre>

Summary of Machine Learning

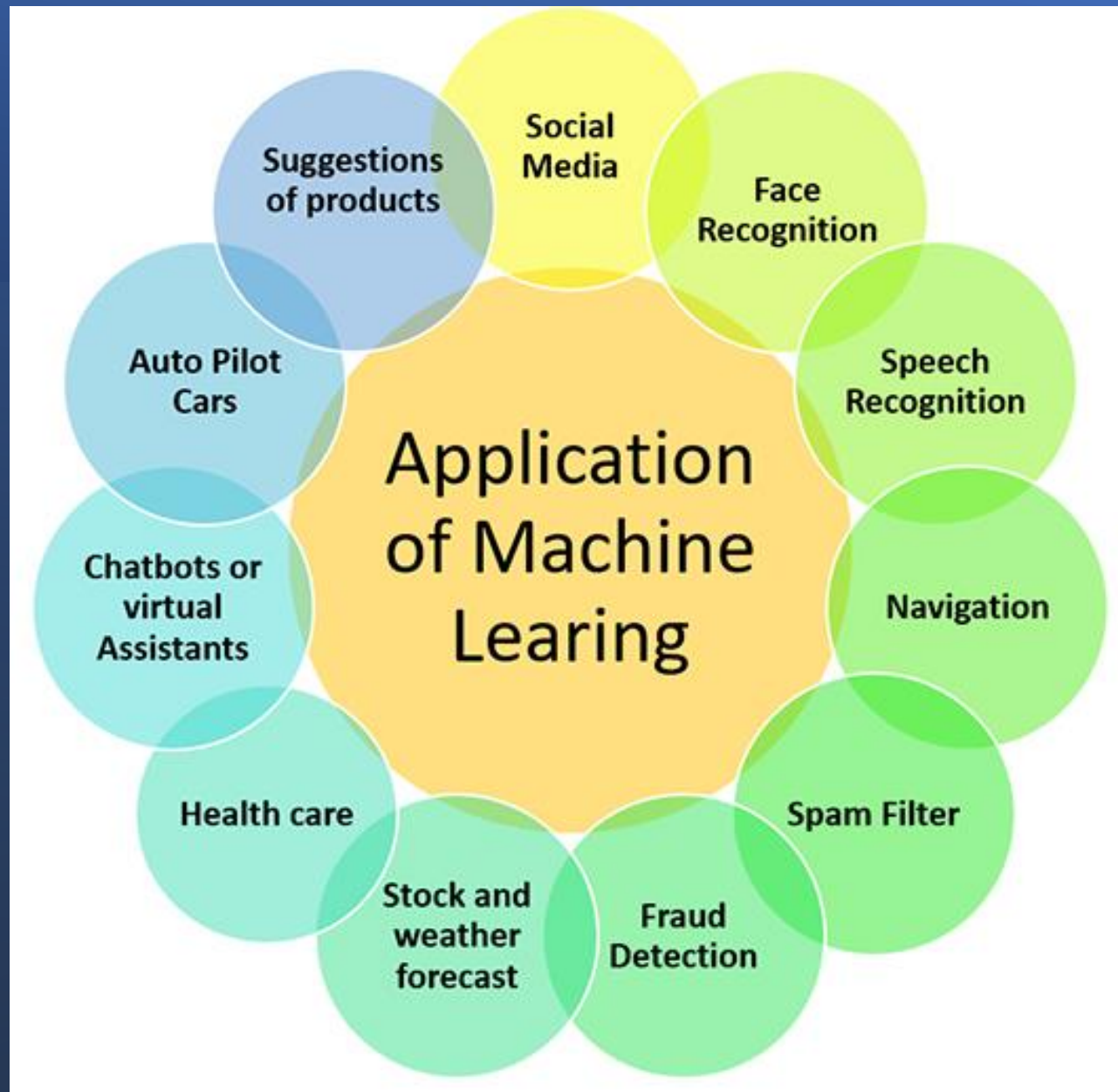
- Machine learning is itself a subset of Artificial Intelligence (AI) that enables computers to learn from data and make decisions without explicit programming. It encompasses various techniques and algorithms that allow systems to recognize patterns, make predictions, and improve performance over time.

A I VERSUS M A C H I N E L E A R N I N G	
AI	MACHINE LEARNING
Intelligence demonstrated by machines, similar to the natural intelligence displayed by humans	Scientific study of algorithms and statistical methods that computer systems use to perform a specific task effectively without using explicit instructions
Focuses on mimicking the intelligent behavior similar to a human	Focuses on identifying the hidden patterns and to improve the accuracy of prediction



Machine learning models and their training algorithms

Supervised learning	Unsupervised learning	Semi-supervised learning	Reinforcement learning
<p>Data scientists provide input, output and feedback to build model (as the definition).</p> <p>EXAMPLE ALGORITHMS:</p> <p>Linear regressions</p> <ul style="list-style-type: none">■ Sales forecasting.■ Risk assessment. <p>Support vector machines</p> <ul style="list-style-type: none">■ Image classification.■ Financial performance comparison. <p>Decision trees</p> <ul style="list-style-type: none">■ Predictive analytics.■ Pricing.	<p>Use deep learning to arrive at conclusions and patterns through unlabeled training data.</p> <p>EXAMPLE ALGORITHMS:</p> <p>Apriori</p> <ul style="list-style-type: none">■ Sales functions.■ Word associations.■ Searcher. <p>K-means clustering</p> <ul style="list-style-type: none">■ Performance monitoring.■ Searcher intent. <p>Artificial neural networks</p> <ul style="list-style-type: none">■ Generate new, synthetic data.■ Data mining and pattern recognition.	<p>Builds a model through a mix of labeled and unlabeled data, a set of categories, suggestions and example labels.</p> <p>EXAMPLE ALGORITHMS:</p> <p>Generative adversarial networks</p> <ul style="list-style-type: none">■ Audio and video manipulation.■ Data creation. <p>Self-trained Naïve Bayes classifier</p> <ul style="list-style-type: none">■ Natural language processing.	<p>Self-interpreting but based on a system of rewards and punishments learned through trial and error, seeking maximum reward.</p> <p>EXAMPLE ALGORITHMS:</p> <p>Q-learning</p> <ul style="list-style-type: none">■ Policy creation.■ Consumption reduction. <p>Model-based value estimation</p> <ul style="list-style-type: none">■ Linear tasks.■ Estimating parameters.

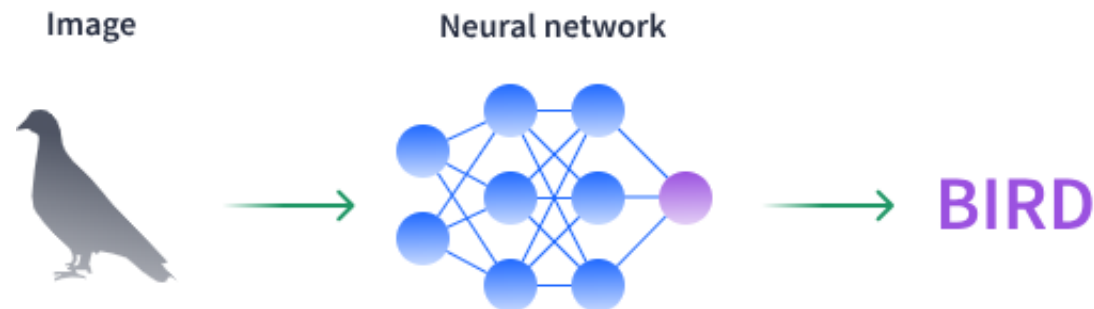


Please Refer to Google Colab Code
(Instagram Reach Analysis)
The dataset is uploaded on Canvas



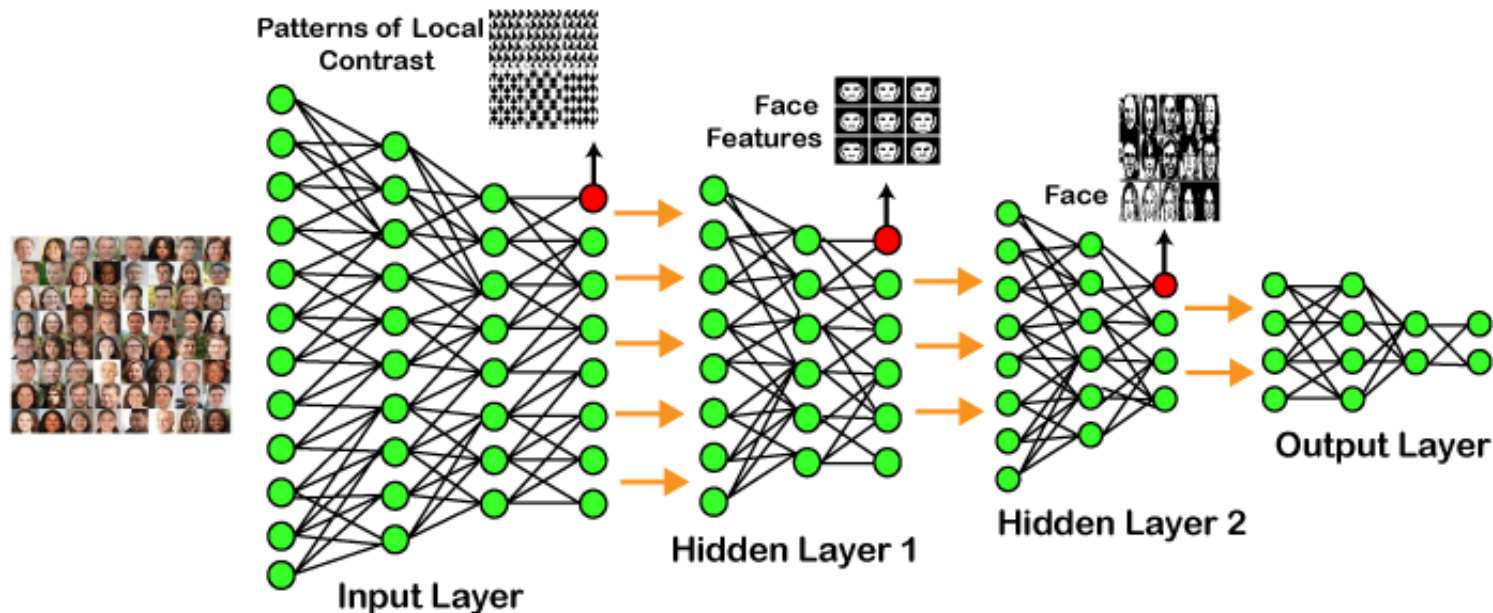
Introduction to Deep Learning

- **Deep learning** is an approach to machine learning completely based on Artificial Neural Networks.
- Basically, it is a machine learning class that makes use of numerous nonlinear processing units so as to perform feature extraction as well as transformation. The output from each preceding layer is taken as input by each one of the successive layers.
- Deep learning models are capable enough to focus on the accurate features themselves by requiring a little guidance from the programmer and are very helpful in solving out the problem of dimensionality. Deep learning algorithms are used, especially when we have a huge no of inputs and outputs.
- Since deep learning has been evolved by the machine learning, which itself is a subset of artificial intelligence and as the idea behind the Artificial Intelligence is to mimic the human behavior, so same is "the idea of deep learning to build such algorithm that can mimic the brain".
- Deep learning is implemented with the help of Neural Networks, and the idea behind the motivation of Neural Network is the biological neurons, which is nothing but a brain cell.



Example of Deep Learning

We provide the raw data of images to the first layer of the input layer. After then, these input layer will determine the patterns of local contrast that means it will differentiate on the basis of colors, luminosity, etc. Then the 1st hidden layer will determine the face feature, i.e., it will fixate on eyes, nose, and lips, etc. And then, it will fixate those face features on the correct face template. So, in the 2nd hidden layer, it will actually determine the correct face here as it can be seen in the above image, after which it will be sent to the output layer. Likewise, more hidden layers can be added to solve more complex problems, for example, if you want to find out a particular kind of face having large or light complexions. So, as and when the hidden layers increase, we are able to solve complex problems.



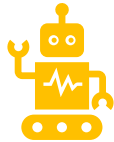
Why Deep Learning is Important?



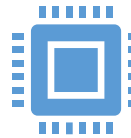
It can solve many problems that are difficult or impossible for traditional algorithms or human experts.



It can handle large and complex data sets, such as images, videos, audio, text, and more.



It drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention.



It can improve image recognition and natural language processing, where the traditional learning algorithms' limitations are surpassed.



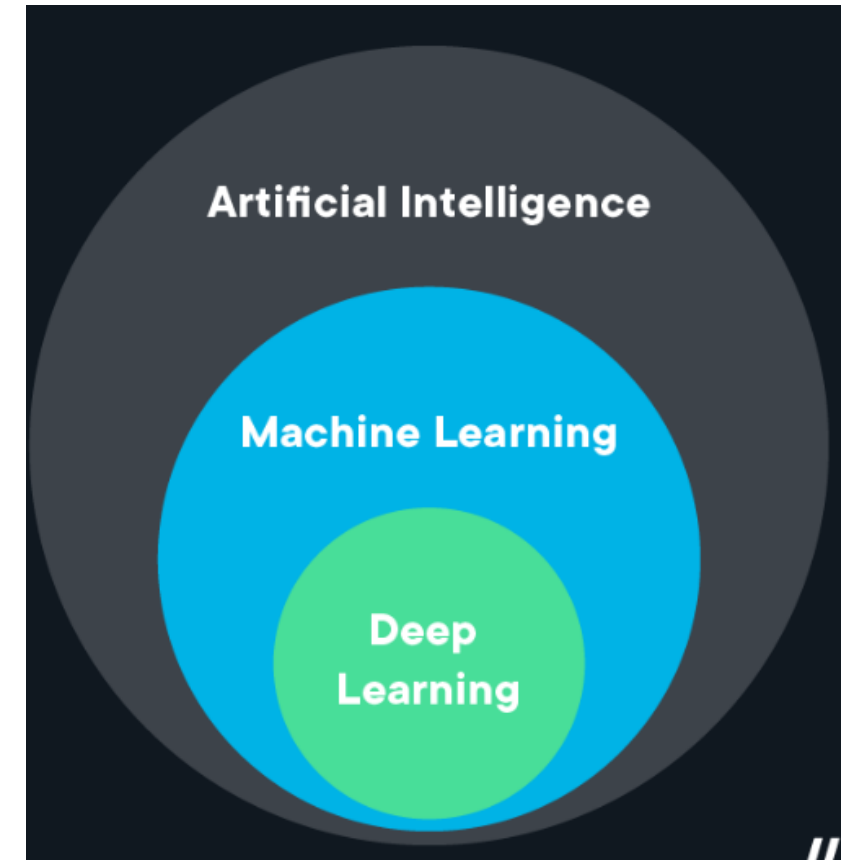
It can identify complex patterns and relationships in data with high accuracy, making fraud detection and medical diagnosis possible.



It can process raw data and generate patterns for decision-making.

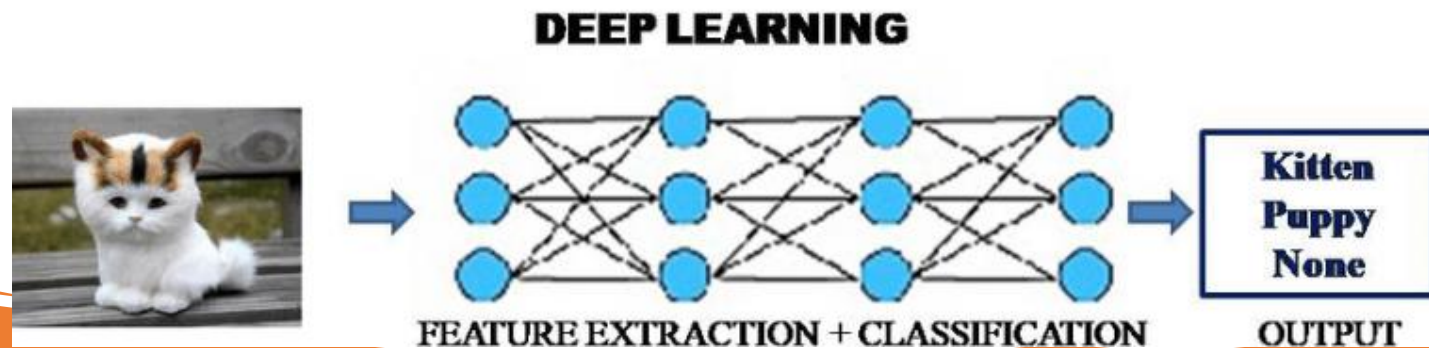
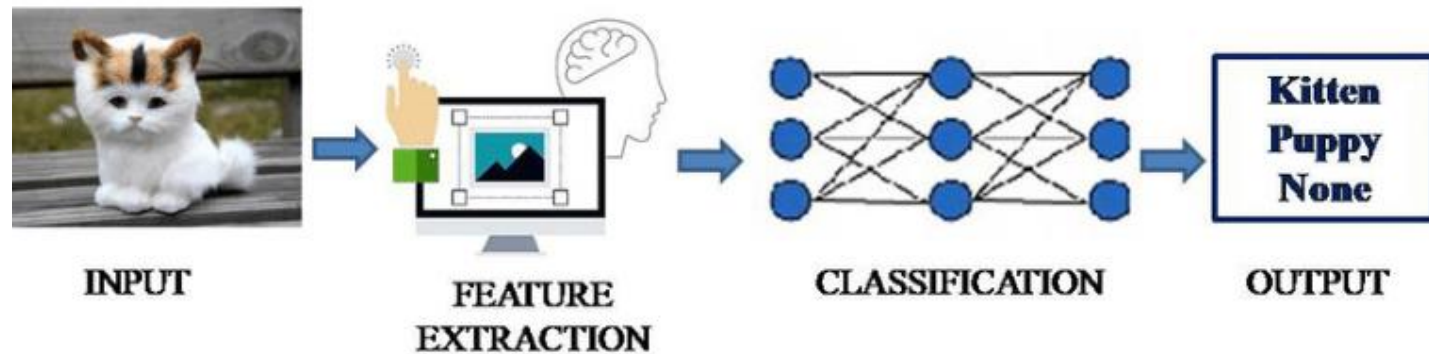
Difference between Artificial Intelligence, Machine Learning and Deep Learning

- Deep Learning is basically a sub-part of the broader family of Machine Learning which makes use of Neural Networks to mimic human brain-like behavior.
- Deep Learning algorithms focus on information processing patterns mechanism to possibly identify the patterns just like our human brain does and classifies the information accordingly.
- Deep Learning works on larger sets of data when compared to Machine Learning and the prediction mechanism is self-administered by machines.



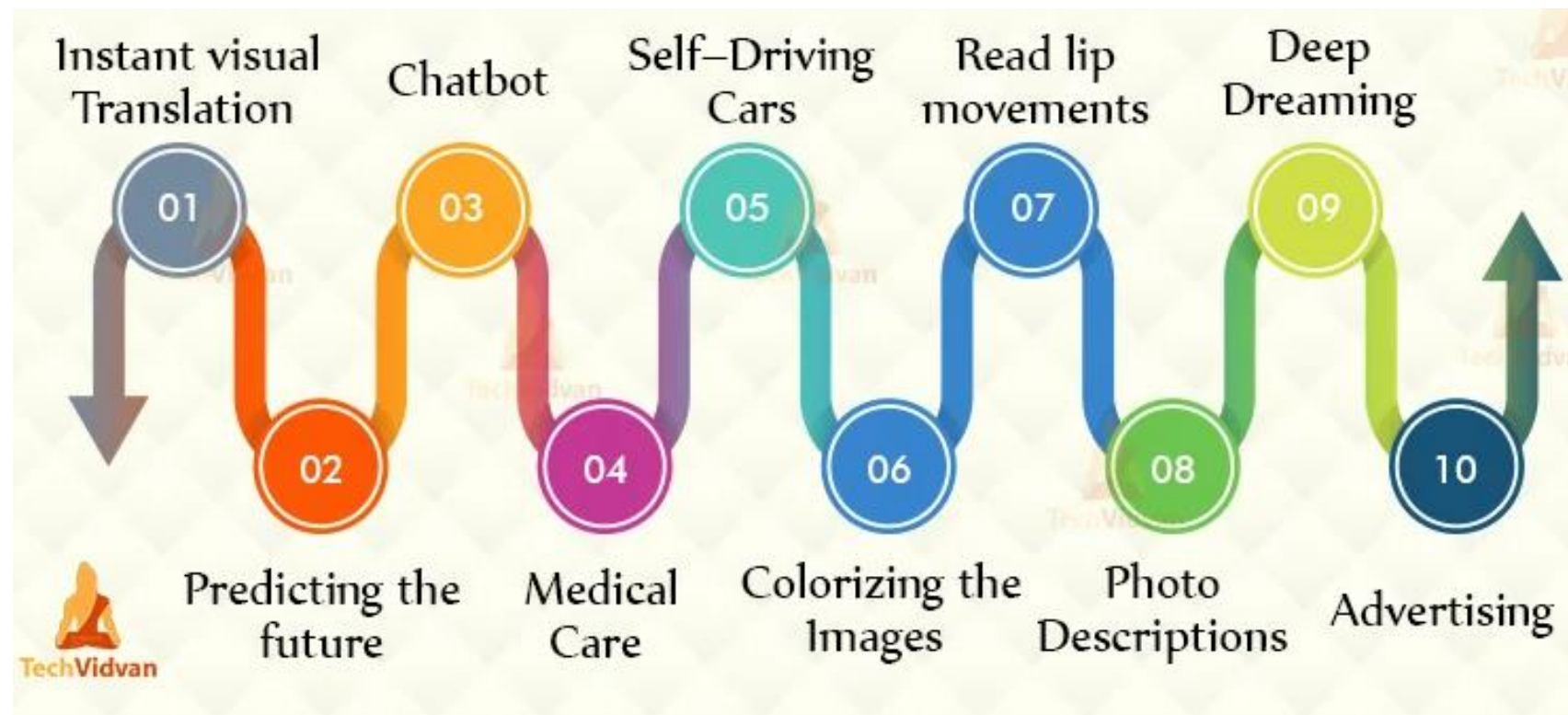
If you want to use a machine learning model to determine if a particular image is showing a Cat or not, we humans first need to identify the unique features of a Cat (weight, breed, color, age), then extract the feature and give it to the algorithm as input data. In this way, the algorithm would perform a classification of the images. That is, in machine learning, a programmer must intervene directly in the action for the model to come to a conclusion.

In the case of a deep learning model, the feature extraction step is completely unnecessary. The model would recognize these unique characteristics of a Cat and make correct predictions without human intervention.

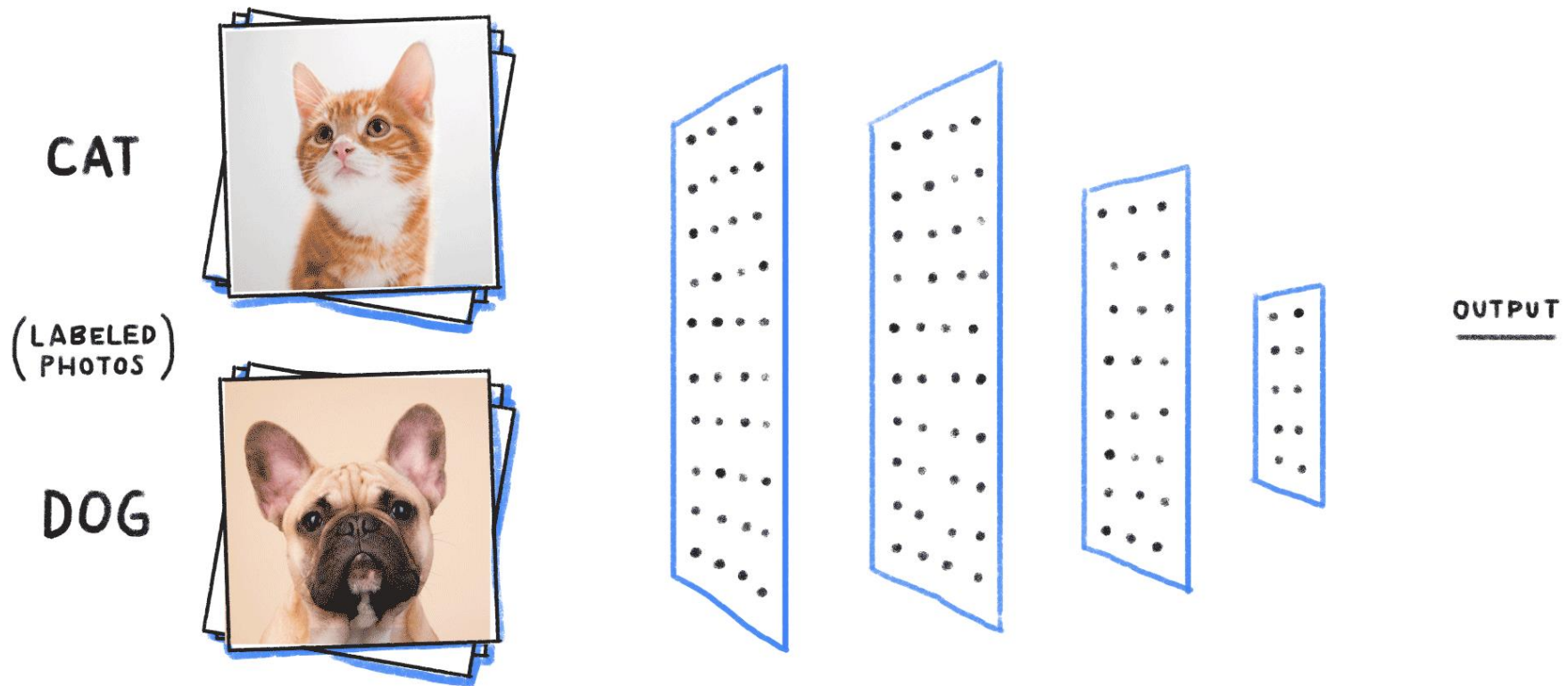


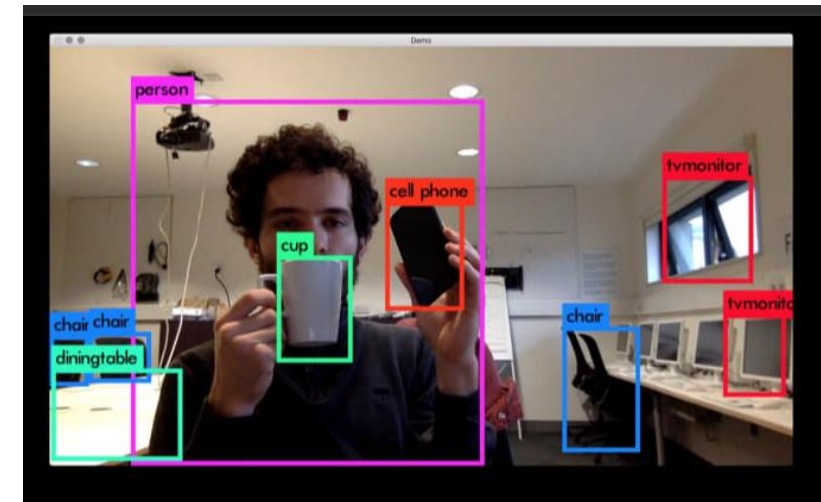
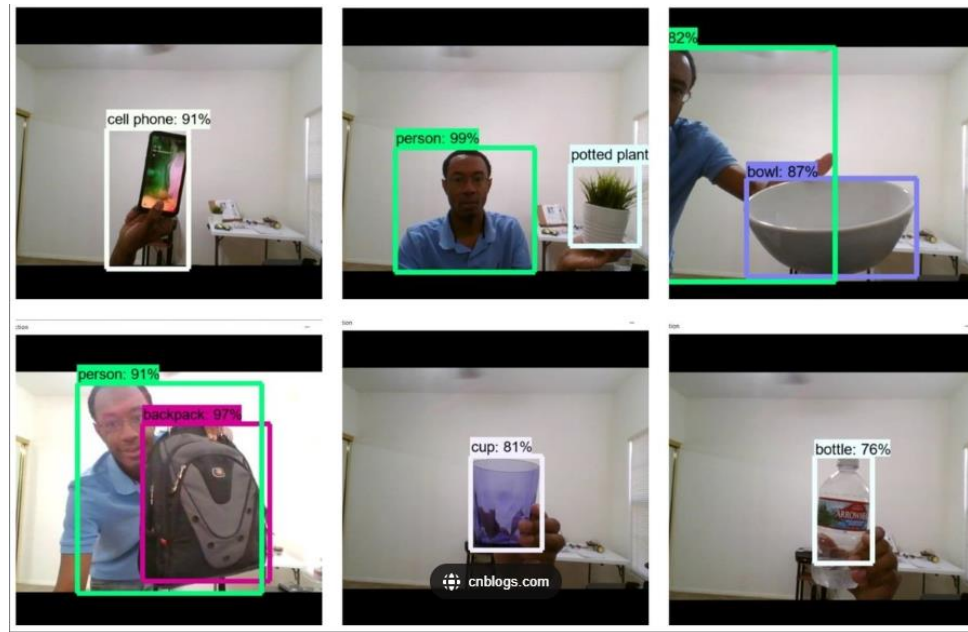
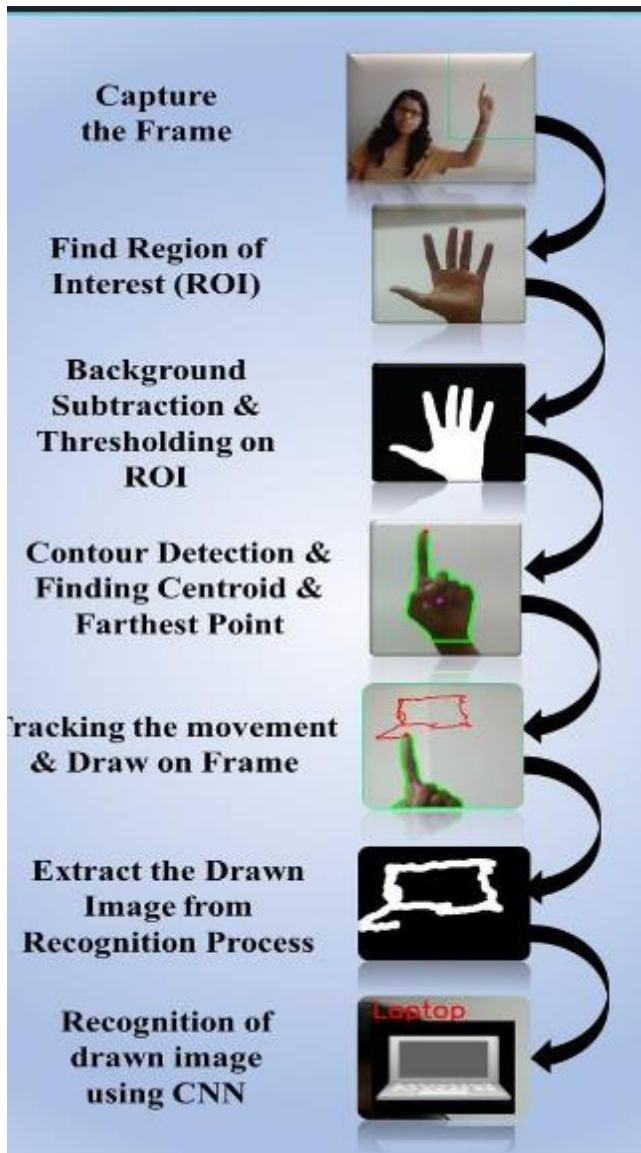
Machine Learning	Deep Learning
Apply statistical algorithms to learn the hidden patterns and relationships in the dataset.	Uses artificial neural network architecture to learn the hidden patterns and relationships in the dataset.
Can work on the smaller amount of dataset	Requires the larger volume of dataset compared to machine learning
Better for the low-label task.	Better for complex task like image processing, natural language processing, etc.
Takes less time to train the model.	Takes more time to train the model.
A model is created by relevant features which are manually extracted from images to detect an object in the image.	Relevant features are automatically extracted from images. It is an end-to-end learning process.
	More complex, it works like the black box

Deep Learning Applications?



Detecting Dog or Cat?



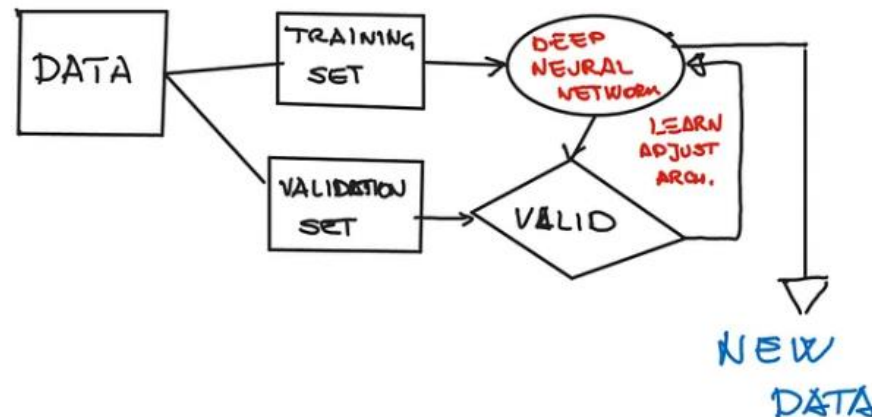


Object Detection

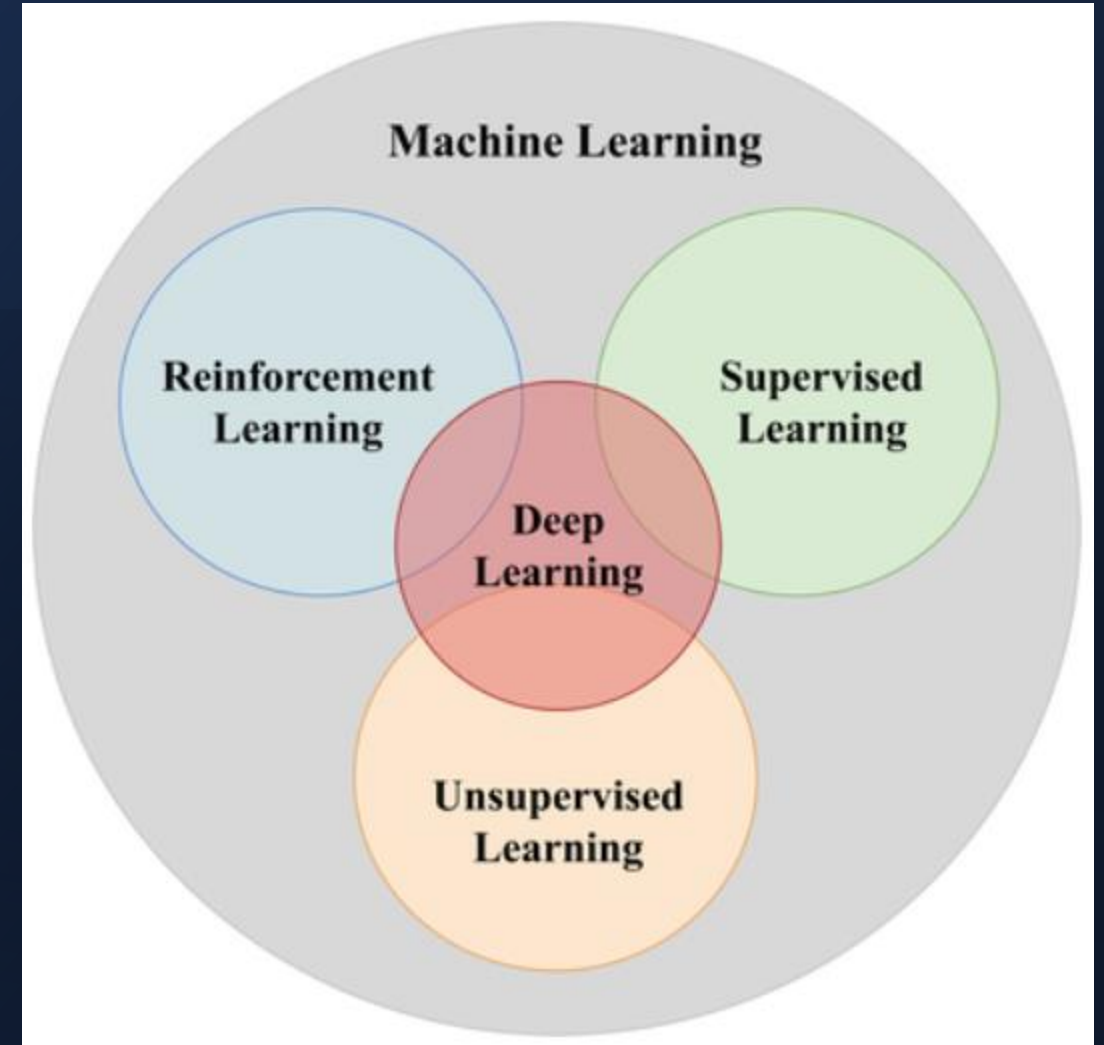
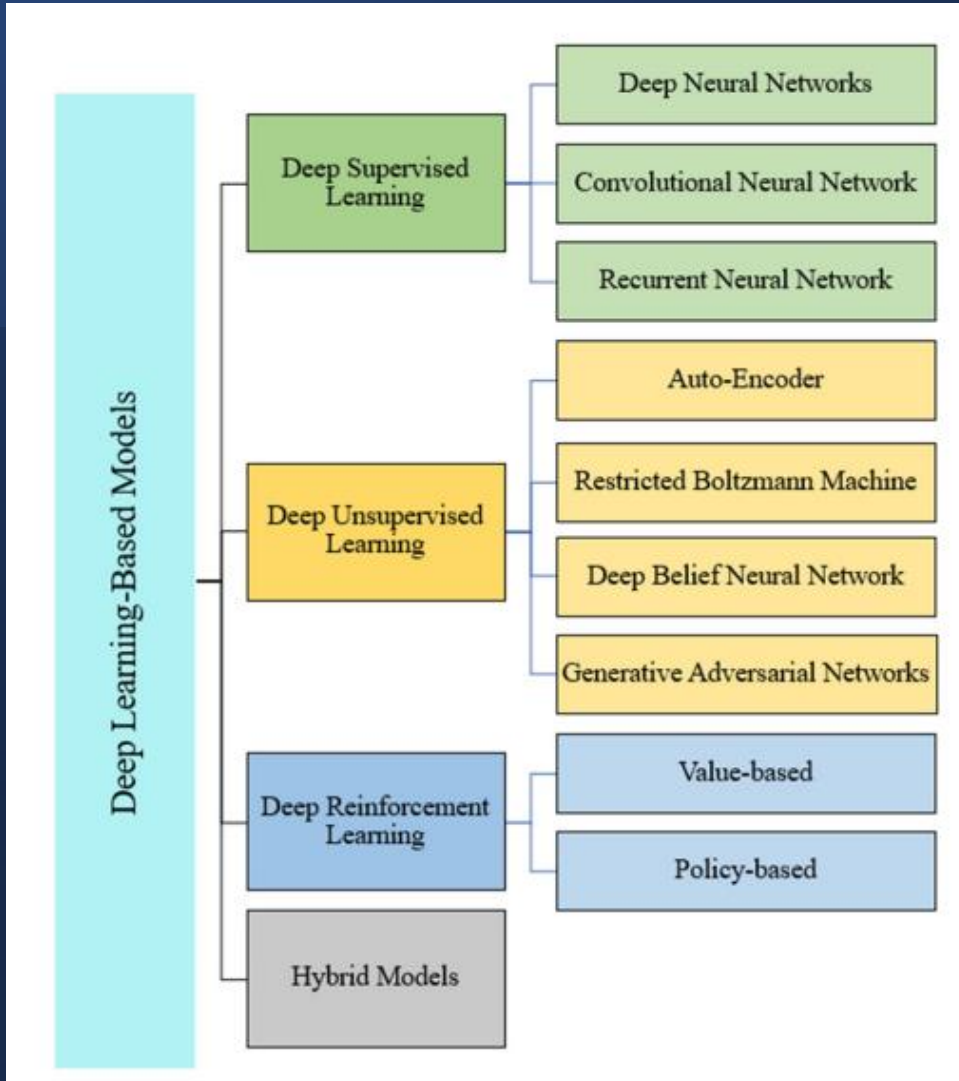
How Deep Learning Works?

NO FEATURE EXTRACTION

- The result of feature extraction is a representation of the given raw data that these classic machine learning algorithms can use to perform a task.
- For example, we can now classify the data into several categories or classes. Feature extraction is usually quite complex and requires detailed knowledge of the problem domain.
- This preprocessing layer must be adapted, tested and refined over several iterations for optimal results.
- Deep learning's artificial neural networks don't need the feature extraction step. The layers are able to learn an implicit representation of the raw data directly and on their own.



Types of Deep Learning



A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

-  Backfed Input Cell
-  Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Different Memory Cell
-  Kernel
-  Convolution or Pool

Perceptron (P)



Feed Forward (FF)



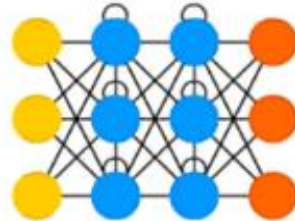
Radial Basis Network (RBF)



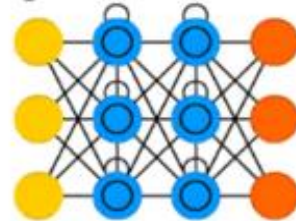
Deep Feed Forward (DFF)



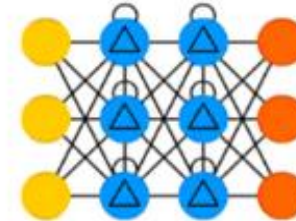
Recurrent Neural Network (RNN)



Long / Short Term Memory (LSTM)



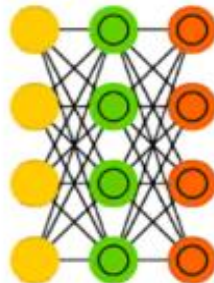
Gated Recurrent Unit (GRU)



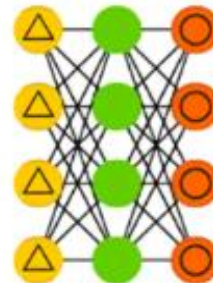
Auto Encoder (AE)



Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)



Artificial Neural Network

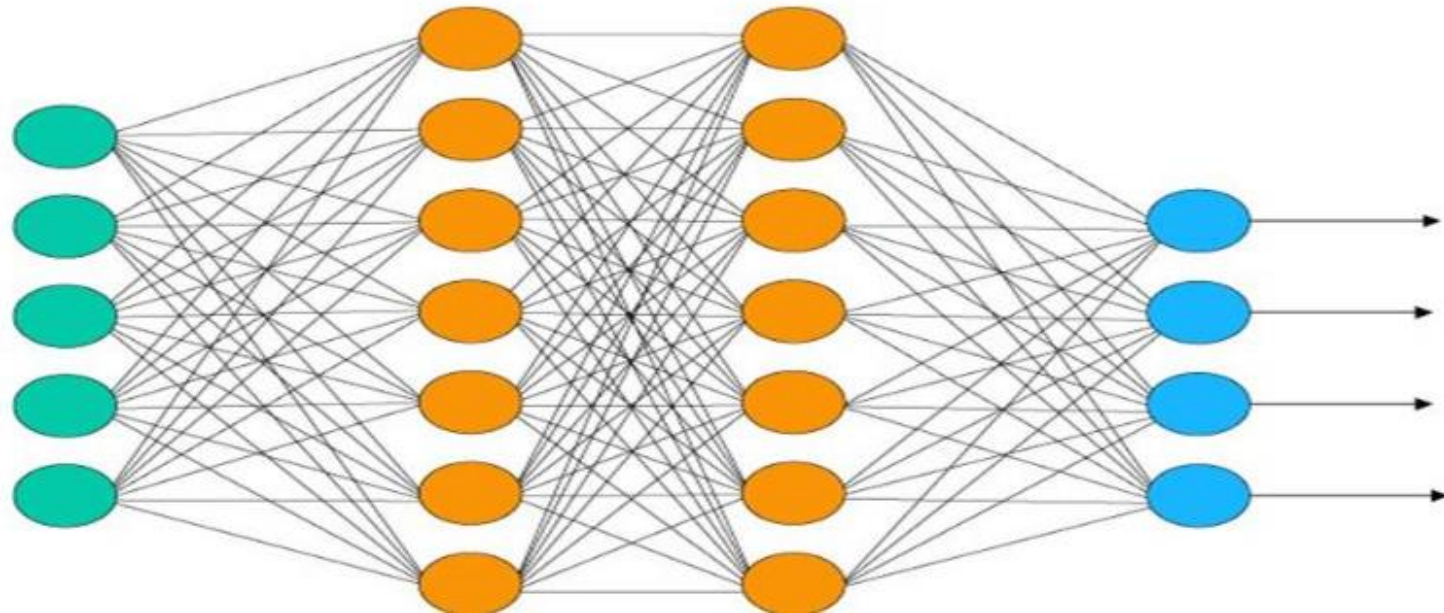
- Artificial Neural Network (ANN) is a type of neural network that is based on a Feed-Forward strategy. It is called this because they pass information through the nodes continuously till it reaches the output node. This is also known as the simplest type of neural network.

Some advantages of ANN :

- Ability to learn irrespective of the type of data (Linear or Non-Linear).
- ANN is highly volatile and serves best in fi time series forecasting.

Some disadvantages of ANN :

- The simplest architecture makes it difficult explain the behavior of the network.
- This network is dependent on hardware.



Biological Neural Network

- The *biological neural network* is also composed of several processing pieces known as **neurons** that are linked together via **synapses**. These neurons accept either external input or the results of other neurons. The generated output from the individual neurons propagates its effect on the entire network to the last layer, where the results can be displayed to the outside world.

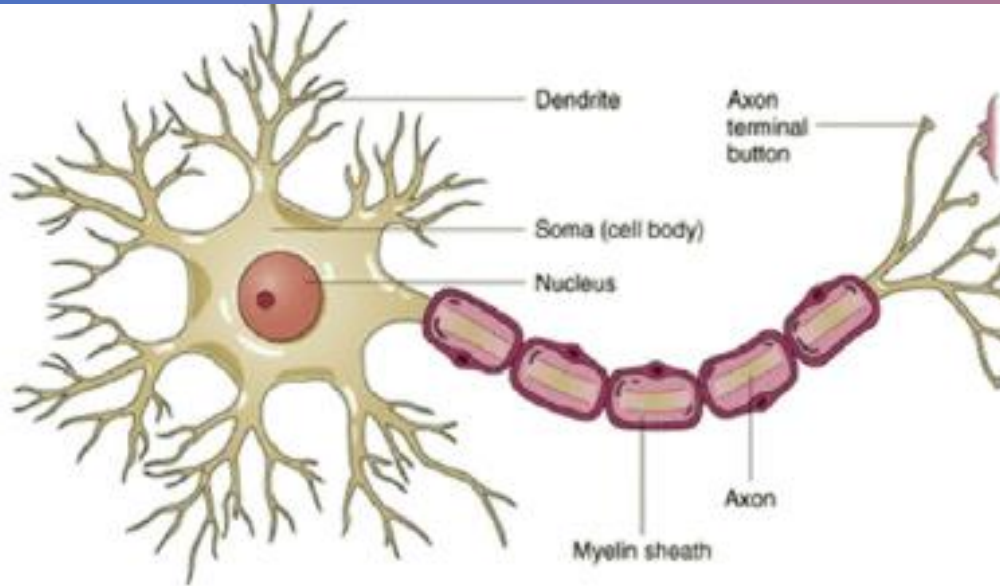
- Every synapse has a processing value and weight recognized during network training. The performance and potency of the network fully depend on the neuron numbers in the network, how they are connected to each other (i.e., topology), and the weights assigned to every synapse

- **Advantages**

- It can handle extremely complex parallel inputs.
- The input processing element is the synapses.

- **Disadvantages**

- As it is complex, the processing speed is slow.
- There is no controlling mechanism in this network.



Difference between Artificial Neural Network and Biological Neural Network

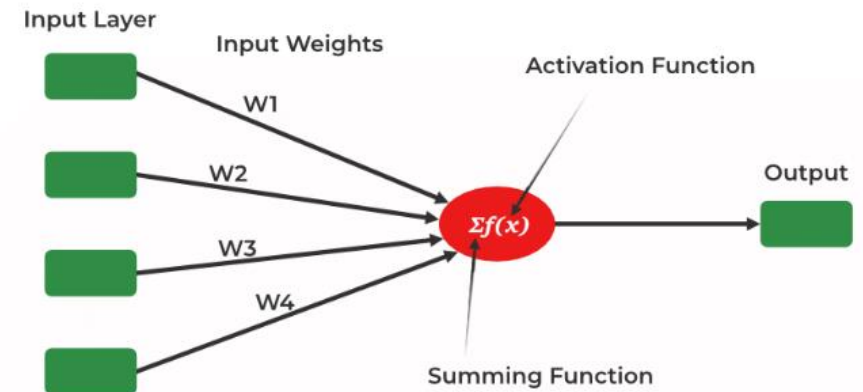
Features	Artificial Neural Network	Biological Neural Network
Definition	It is the mathematical model which is mainly inspired by the biological neuron system in the human brain.	It is also composed of several processing pieces known as neurons that are linked together via synapses.
Processing	Its processing was sequential and centralized.	It processes the information in a parallel and distributive manner.
Size	It is small in size.	It is large in size.
Control Mechanism	Its control unit keeps track of all computer-related operations.	All processing is managed centrally.
Rate	It processes the information at a faster speed.	It processes the information at a slow speed.
Complexity	It cannot perform complex pattern recognition.	The large quantity and complexity of the connections allow the brain to perform complicated tasks.
Feedback	It doesn't provide any feedback.	It provides feedback.
Fault tolerance	There is no fault tolerance.	It has fault tolerance.
Operating Environment	Its operating environment is well-defined and well-constrained	Its operating environment is poorly defined and unconstrained.
Memory	Its memory is separate from a processor, localized, and non-content addressable.	Its memory is integrated into the processor, distributed, and content-addressable.
Reliability	It is very vulnerable.	It is robust.
Learning	It has very accurate structures and formatted data.	They are tolerant to ambiguity.
Response time	Its response time is measured in milliseconds.	Its response time is measured in nanoseconds.

Basic Neural Network: Single Layer Perceptron

- It is one of the oldest and first introduced neural networks. It was proposed by Frank Rosenblatt in 1958. Perceptron is mainly used to compute the logical gate like AND, OR, and NOR which has binary input and binary output. The main functionality of the perceptron is:
 1. Takes input from the input layer
 2. Weight them up and sum it up.
 3. Pass the sum to the nonlinear function to produce the output.

In single layer perceptron, there are :

1. *Input Layer*
2. *Input weights*
3. *Activation Function*
4. *Output*



Working of Single Layer Perceptron

The algorithm behind the Single Layer Perceptron is:

- In a single layer perceptron, the weights to each input node are assigned randomly since there is no a priori knowledge associated with the nodes.
- Also, a threshold value is assigned randomly.
- Now it sums all the weights which are inputted and if the sums are above the threshold then the network is activated.
- If the calculated value is matched with the desired value, then the model is successful.
- If it is not, then since there is no back-propagation technique involved in this the error needs to be calculated using the below formula and the weights need to be adjusted again.

Weight Adjustment:

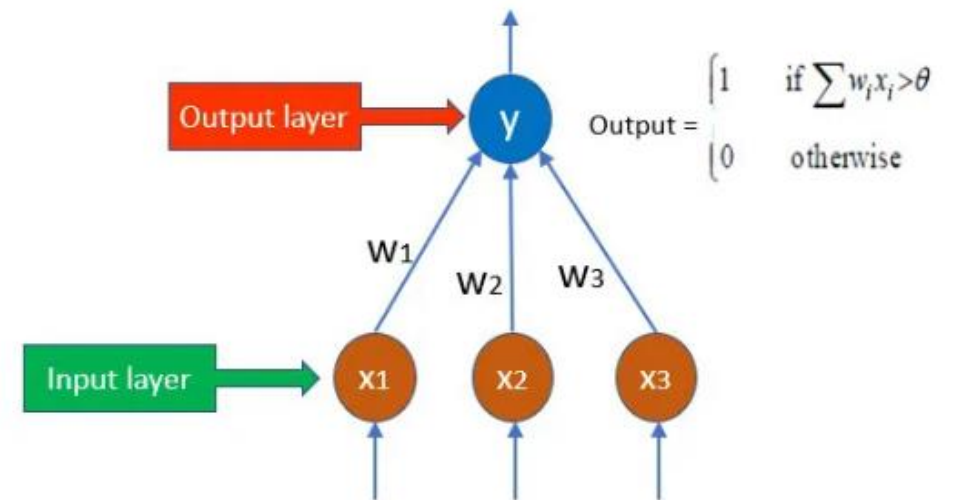
$$\Delta w = \eta * d * x$$

Learning Rate

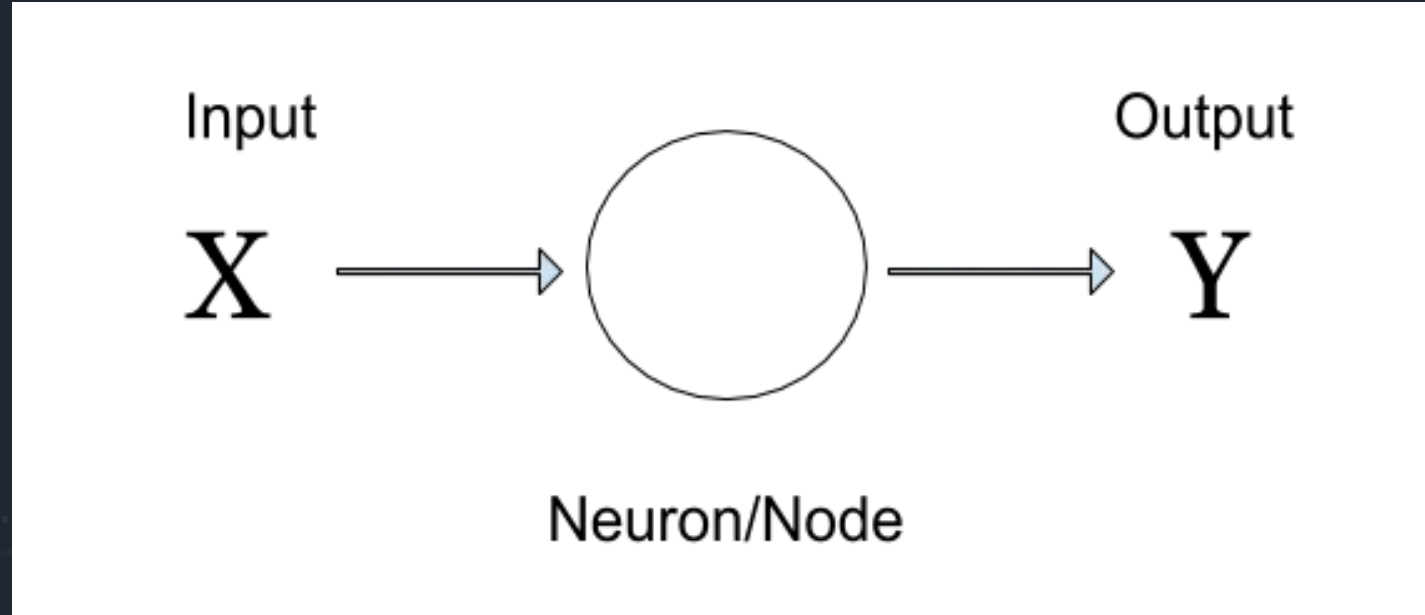
Input Data

Perceptron weight adjustment

d : Predicted Output – Desired Output



What is Neuron?



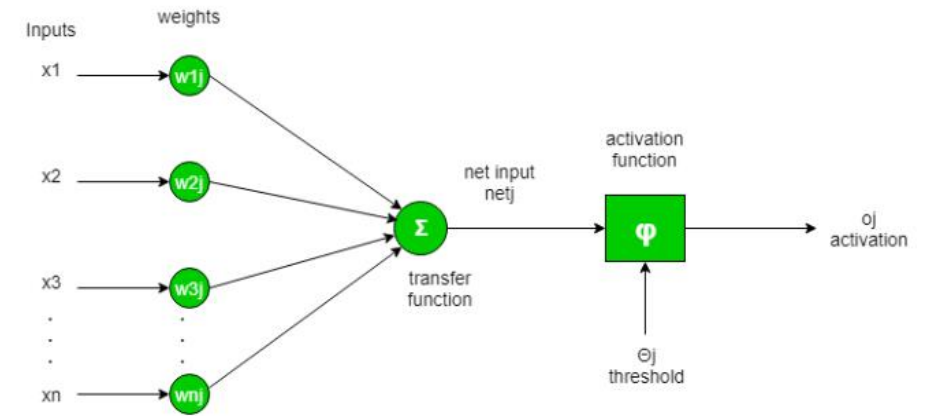
- An artificial “neuron” is modeled using the architecture of the human biological neuron, forming the driving unit of a neural network structure.
- It receives inputs(X). It then processes those inputs using a transformation and activation function . We have our desired output(y).

What is Activation Function?

To put in simple terms, an artificial neuron calculates the 'weighted sum' of its inputs and adds a bias:


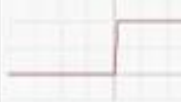







$$\text{net input} = \sum (\text{weight} * \text{input}) + \text{bias}$$

- The value of *net input* can be any anything from $-\infty$ to $+\infty$.
- The neuron doesn't really know how to bound to value and thus is not able to decide the firing pattern.
- Thus the activation function is an important part of an artificial neural network. They basically decide whether a neuron should be activated or not. Thus it bounds the value of the net input.
- The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output.



Types of Activation Function

[We will go through this in near future].

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Please Refer to Google Colab Code (SingleLayerPerceptron.ipynb)

```
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

def __init__(self, path):
    self.file = None
    self.fingerprints = set()
    self.logdupes = True
    self.debug = debug
    self.logger = logging.getLogger(__name__)
    if path:
        self.file = open(os.path.join(path, "requests.log"),
                        "a")
        self.file.seek(0)
        self.fingerprints.update(self.request_fingerprint(request) for request in self.requests)

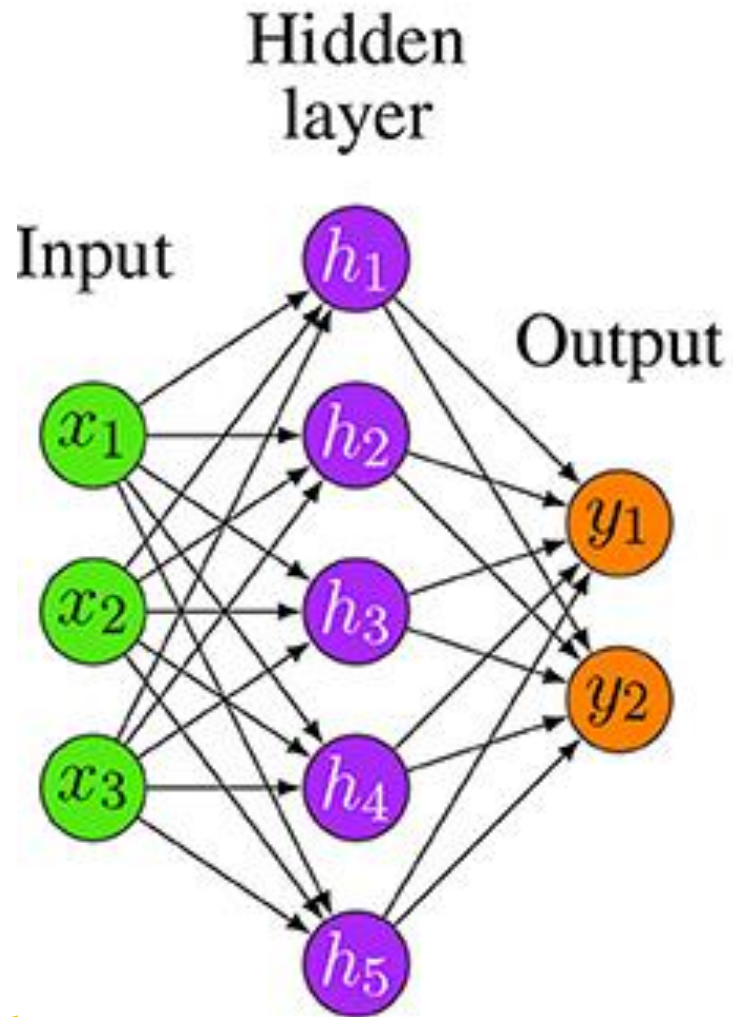
    @classmethod
    def from_settings(cls, settings):
        debug = settings.getbool("debug")
        return cls(job_dir(settings), debug)

    def request_seen(self, request):
        fp = self.request_fingerprint(request)
        if fp in self.fingerprints:
            return True
        self.fingerprints.add(fp)
        if self.file:
            self.file.write(fp + os.linesep)

    def request_fingerprint(self, request):
        return request_fingerprint(request)
```

dashtech.org

Shallow Neural Network



- A shallow neural network has only one hidden layer between the input and output layers, while a deep neural network has multiple hidden layers.
- A shallow network might be used for simple tasks like image classification, while a deep network might be used for more complex tasks like image segmentation or natural language processing.
- The main advantage of a shallow network is that it is computationally less expensive to train, and can be sufficient for simple tasks. However, it may not be powerful enough to capture complex patterns in the data.

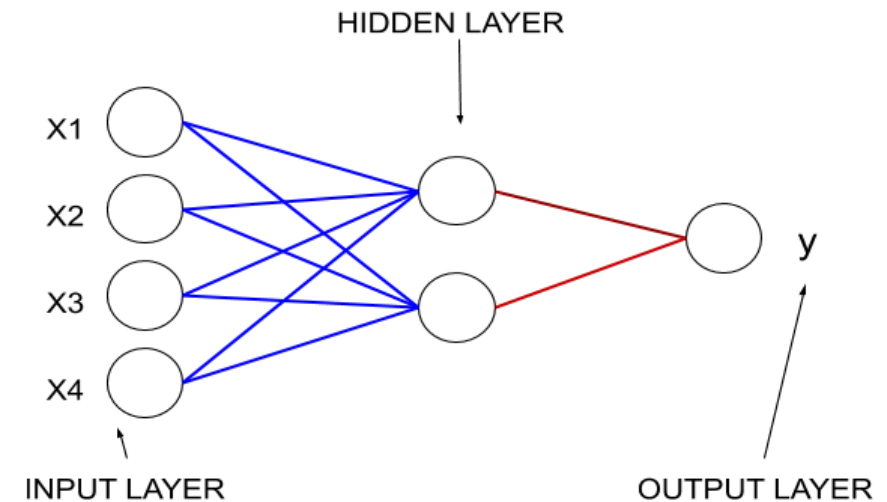
Structure of Shallow Neural Networks

Each neural network consists of 3 types of layers that are responsible for its functioning. These 3 layers are:

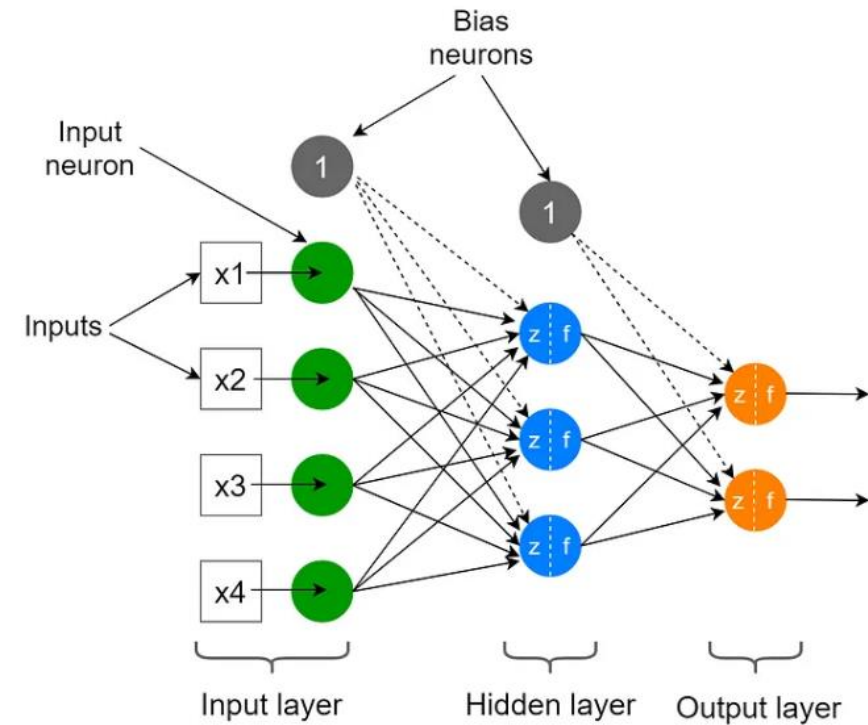
1. Input layer: All the inputs of the model are fed into this layer. In fact, this is where the training observations are fed through the independent variables.

2. Hidden layer: This is where all the action takes place, inputs are fed, processed with the help of linear and nonlinear functions, and are passed to the output layer. These are the intermediate layers between the input and output layers. This is where the neural network learns about the relationships and interactions of the variables fed in the input layer.

3. Output layer: After the processing, the data is sent to the output layer. This is the layer where the final output is extracted as a result of all the processing which takes place within the hidden layers.



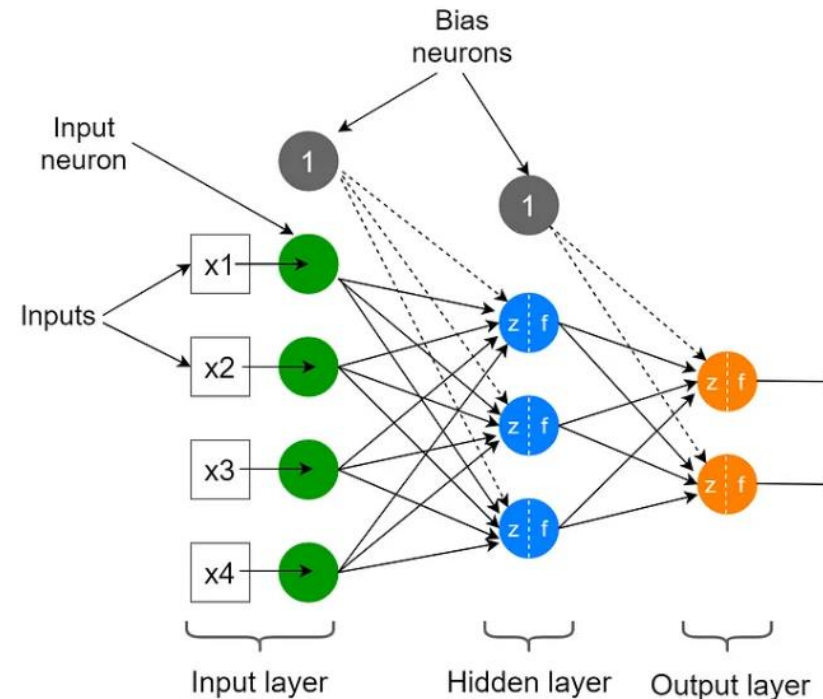
Shallow Neural Network Architecture



- Also known as Multi Layer Perceptron (MLP) or Feed Forward Neural Network (FFNN)
- The term “feed-forward” means that data moves from the input to the output through layers in *one (forward) direction*. In a feed-forward neural network, the only feedback happens once the input data has arrived at the output layer and there is no feedback while data is moving through the intermediate layers.
- **The input layer**
 - The input layer consists of input neurons that take inputs, **x1**, **x2**, etc. The input neurons of an MLP (FCNN) do not perform any calculations and just output whatever they are given.
 - A bias neuron is also there, and it always outputs the value 1.
 - There are different types of input layer:
 1. **Dense → In shallow Neural Network, we usually use Dense**
 2. Convolutional and
 3. Recurrent
 - we should reshape our input data depending on the input layer type.
- The input neurons always take numbers. If we provide images, videos, texts or speech for input data, they will be converted into numbers.

Shallow Neural Network Architecture: Input Layer

- In a Dense (FC) input layer type, the number of input neurons, that is the size of the input layer, is equal to the number of features (columns) of the input data.
- These features are denoted by x_1 , x_2 , x_3 , etc.
- These are the inputs to the first hidden layer and are fully connected to the neurons in the hidden layer through the weights that measure the level of importance.
- The size of the input layer is a hyperparameter that we need to specify in our model before training.
- Its optimal value does not learn from data.
- In contrast, parameters learn their optimal values from data during the training process.



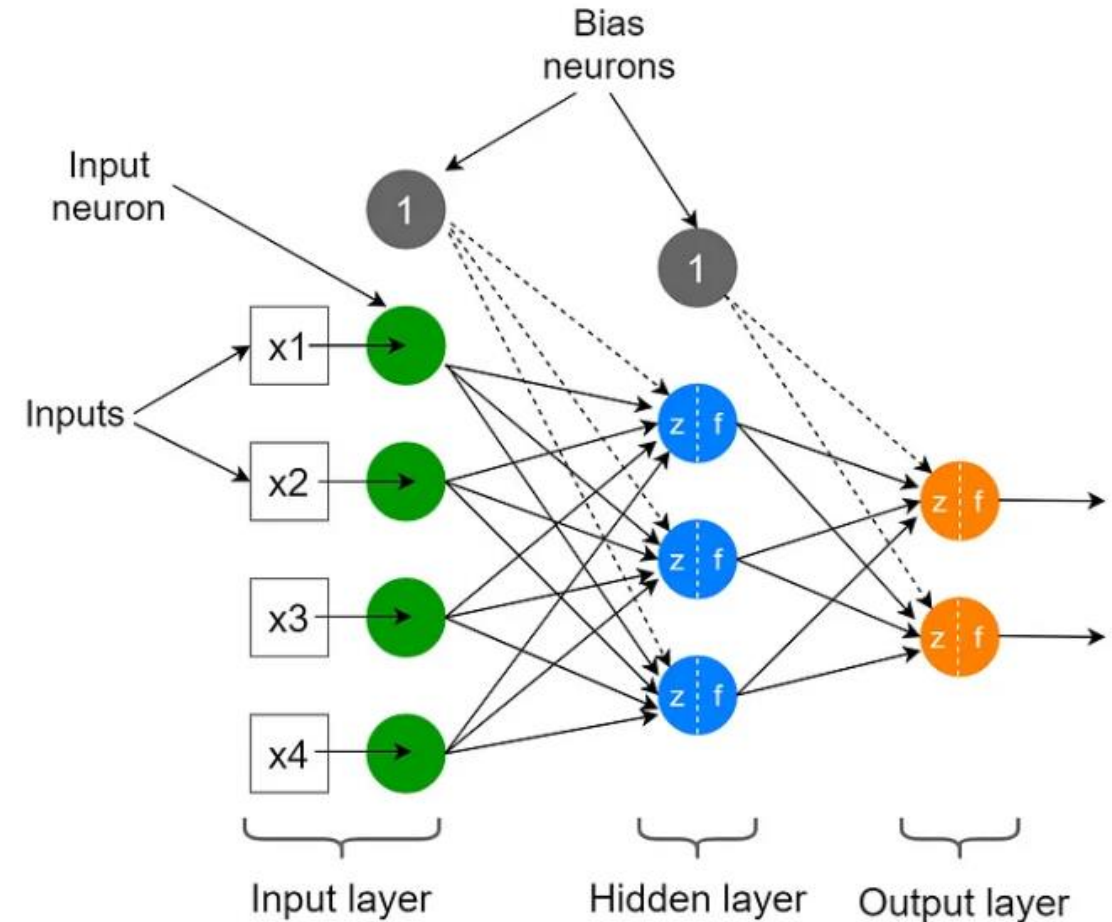
Shallow Neural Network Architecture: Hidden Layer

- *The hidden (invisible) layer:*

- ✓ This is the layer between the input and output layers.
- ✓ In a shallow neural network, there is only one or two hidden layer (s).

- There are several actions/functions happens in this layer:

- ✓ Initialize Parameters
- ✓ Forward propagation
- ✓ Compute cost
- ✓ Backward propagation
- ✓ Update parameters





Shallow Neural Network Architecture: Initialize Parameters

- There are different approaches to initialize parameters. The most common of these techniques are:
 - Zero Initialization
 - Random Initialization
 - Fixup Initialization
 - LSUV Initialization
 - Xavier
 - Kaiming He



Shallow Neural Network Architecture: Initialize Parameters

- *Zero Initialization*

Here, we'll initialize all weight matrices and biases to zeros .
And, the model usually does not learn anything.

- *Random Initialization*

The random values usually are initialized from standard normal distribution or uniform distribution.

Random initialization is helping but still the loss function has high value and may take long time to converge and achieve a significantly low value.



Shallow Neural Network Architecture: Initialize Parameters

- *Fixup Initialization*

Used on Residual Neural Network :

1. Initialize the classification layer and the last layer of each residual branch to 0.
2. Initialize every other layer using a standard method (e.g., He et al. (2015)), and scale only the weight layers inside residual branches by $L^{-\frac{1}{2m-2}}$.
3. Add a scalar multiplier (initialized at 1) in every branch and a scalar bias (initialized at 0) before each convolution, linear, and element-wise activation layer.

- *LSUV Initialization*

A data-driven approach that has minimal calculations and very low computational overhead. The initialization is a 2 part process, first initializing weights to orthonormal matrices (as opposed to Gaussian noise, which is only approximately orthogonal). The next part is to iterate with a mini-batch and scale the weights so that the variance of activations is.

LSUV Init can be thought of as a combination of orthonormal initialization and, BatchNorm performed only on the first mini-batch.

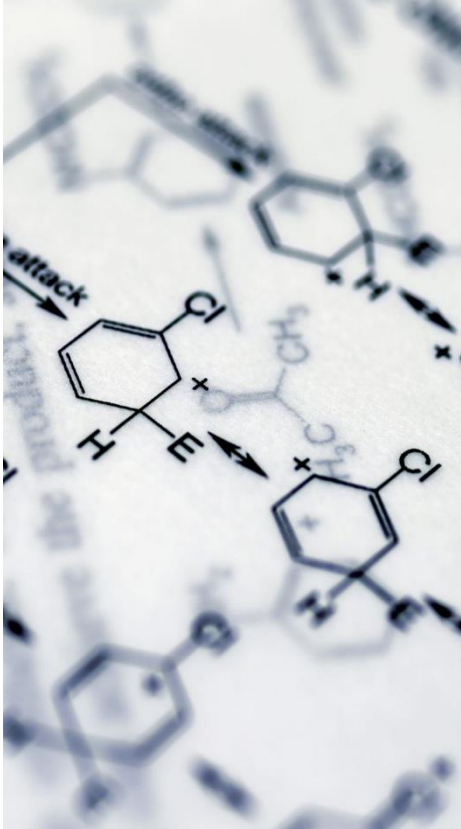
Shallow Neural Network Architecture: Initialize Parameters

- Xavier Initialization:
- The Xavier initialization (or Glorot initialization) is a popular technique for initializing weights in a neural network.
- The main idea is to set the initial weights of the network in a way that allows the activations and gradients to flow effectively during both forward and backpropagation.
- It considers the number of input and output units of each layer to determine the scale of the random initialization.
- The method used for randomization isn't so important, but rather the number of outputs in the following layer matters.
- With the passing of each layer, the Xavier initialization maintains the variance in some bounds so that we can take full advantage of the activation functions.
- Two formulas are included for this strategy:

Uniform Xavier initialization: draw each weight, w , from in $[-x, x]$ for $x = \sqrt{\frac{6}{inputs + outputs}}$

Normal Xavier initialization: draw each weight, w , from 0, and a standard deviation $\sigma = \sqrt{\frac{2}{inputs + outputs}}$

Initialize Parameters: Kaiming He



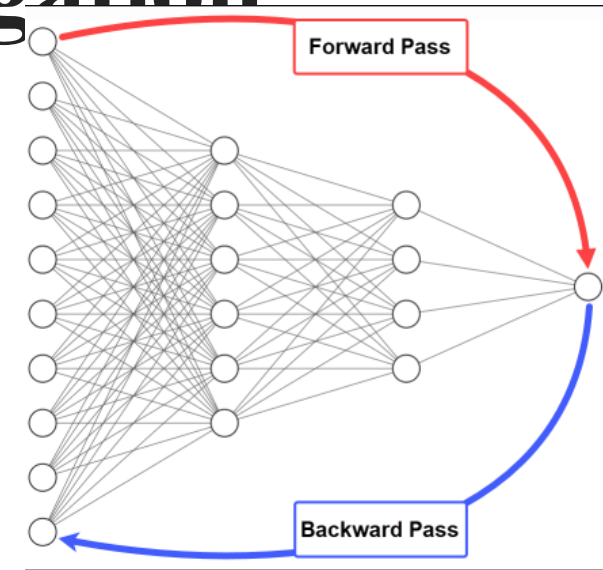
- The Kaiming He initialization method, also known as He initialization or He normal initialization, is a technique used to initialize the weights of a neural network in deep learning.
- This method is particularly effective for rectified linear units (ReLUs), which are commonly used activation functions in deep neural networks.
- The key idea behind Kaiming He initialization is to set the initial weights of the neurons in a way that prevents the vanishing or exploding gradient problem during the training process.
- It calculates the initial weights by drawing random values from a Gaussian distribution with a mean of 0 and a standard deviation that is proportional to the square root of 2 divided by the number of input units in the neuron's layer.
- This initialization strategy helps facilitate stable and efficient training of deep neural networks, contributing to improved convergence and performance.

Shallow Neural Network

Architecture: propagation

To train a neural network, there are 2 phases:

- Forward
- Backward



In the *forward pass*, we start by propagating the data inputs to the input layer, go through the hidden layer(s), measure the network's predictions from the output layer, and finally calculate the network error based on the predictions the network made. This network error measures how far the network is from making the correct prediction.

For example, if the correct output is 4 and the network's prediction is 1.3, then the absolute error of the network is $4 - 1.3 = 2.7$.

Note that the process of propagating the inputs from the input layer to the output layer is called **forward propagation**.

Once the network error is calculated, then the forward propagation phase has ended, and backward pass start

Shallow Neural Network Architecture: Compute Cost

1. *Quadratic Cost function: regression*

$$C = \frac{1}{2} \sum_j (\hat{y}_j - y_j^{pred})^2$$

where \hat{y}_j and y_j^{pred} are the true target value of point j , and the predicted target value respectively.

2. *Mean Error (ME): Regression*

- In this cost function, the error for each training data is calculated and then the mean value of all these errors is derived.
- Calculating the mean of the errors is the simplest and most intuitive way possible.
- The errors can be both negative and positive. So, they can cancel each other out during summation giving zero mean error for the model.
- Thus, this is not a recommended cost function, but it does lay the foundation for other cost functions of regression models.

3. *Mean Squared Error (MSE): Regression*

$$MSE = \frac{\sum_{i=0}^n (y - y')^2}{n}$$

4. *Mean Absolute Error (MAE): Regression*

$$MAE = \frac{\sum_{i=0}^n |y - y'|}{n}$$

Shallow Neural Network Architecture: Compute Cost

- Once you are done with calculating Activation functions, it's time to check how accurate your model is.
- After training your model, you need to see how well your model is performing. While accuracy functions tell you how well the model is performing, they do not provide you with an insight on how to better improve them.
- Hence, you need a correctional function that can help you compute when the model is the most accurate, as you need to hit that small spot between an undertrained model and an overtrained model.
- A Cost Function is used to measure just how wrong the model is in finding a relation between the input and output. It tells you how badly your model is behaving/predicting.

Regression Cost Function: Regression models deal with predicting a continuous value for example salary of an employee, price of a car, loan prediction, etc. A cost function used in the regression problem is called “Regression Cost Function”.

Classification Cost Function: Classification models are used to make predictions of categorical variables, such as predictions for 0 or 1, Cat or dog, etc. The cost function used in the classification problem is known as the Classification cost function. However, the classification cost function is different from the Regression cost function.

Shallow Neural Network Architecture: Compute Cost

2. Cross Entropy Cost: Classification

$$C = - \sum_j (\hat{y}_j \log(y_j) + (1 - \hat{y}_j) \log(1 - y_j))$$

3. Exponential Cost

$$C = \tau \exp \left[\frac{1}{\tau} \sum_j (\hat{y}_j - y_j^{pred})^2 \right]$$

where τ is a hyper-parameter.

Shallow Neural Network Architecture: Compute Cost

4. Hellinger Distance

$$C = \frac{1}{\sqrt{2}} \sum_j \left(\sqrt{\hat{y}_j} - \sqrt{y_j^{pred}} \right)^2$$

it needs to have positive values in [0, 1].

5. Kullback-Leibler Divergence

Kullback-Leibler Divergence is also known as : *Information Divergence, Information Gain, Relative entropy, KLIC divergence* or *KL Divergence*, and is defined as:

$$D_{KL}(P||Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}$$

where $D_{KL}(P||Q)$ is a measure of the information lost when Q is used to approximate P .

The **cost function** using *KL Divergence* is:

$$C = \sum_j \hat{y}_j \log \frac{\hat{y}_j}{y_j^{pred}}$$

Shallow Neural Network Architecture: Compute Cost

6. Generalized Kullback-Leibler Divergence

$$C = \sum_j \hat{y}_j \log \frac{\hat{y}_j}{y_j^{pred}} - \sum_j \hat{y}_j - \sum_j y_j^{pred}$$

7. Itakura-Saito Distance

$$C = \sum_j \left(\frac{\hat{y}_j}{y_j^{pred}} - \log \frac{\hat{y}_j}{y_j^{pred}} - 1 \right)$$

Shallow Neural Network Architecture: propagation

In the backward pass, the flow is reversed so that we start by propagating the error to the output layer until reaching the input layer passing through the hidden layer(s).

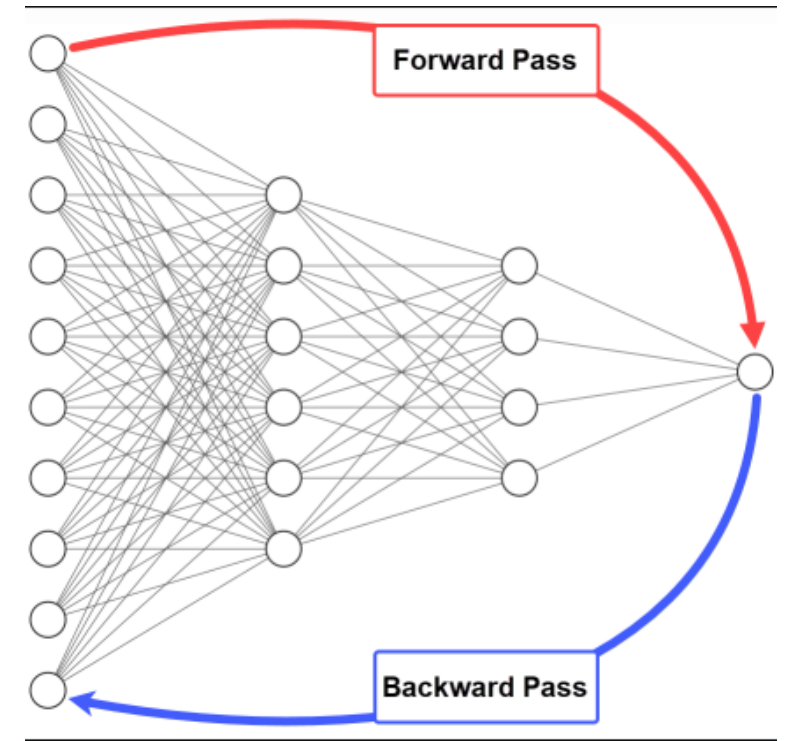
The process of propagating the network error from the output layer to the input layer is called **backward propagation**, or simple **backpropagation**.

The backpropagation algorithm is the set of steps used to update network weights to reduce the network error.

The forward and backward phases are repeated from some epochs.

In each epoch, the following occurs:

- 1.The inputs are propagated from the input to the output layer.
- 2.The network error is calculated.
- 3.The error is propagated from the output layer to the input layer.



Shallow Neural Network Architecture: Update Parameters

- We know that the weights of a neural network are initialized randomly.
- In order to use the neural network for correct predictions, we need to update these weights.
- To update parameters, we can use several techniques, even more:
 - Multiply all parameters by a scalar value, e.g. 0.9.
 - Fit the model on new data with a lower learning rate in the optimizer.
 - Use gradient descent and back-propagation to compute the gradient of the loss function with respect to each of the parameters.
 - Compute the loss function by summing over all time steps, then compute the gradient of this loss function with respect to the weights

Why use the backpropagation algorithm?

- The backpropagation algorithm is one of the algorithms responsible for updating network weights with the objective of reducing the network error. It's quite important.

Here are some of the advantages of the backpropagation algorithm:

- It's memory-efficient in calculating the derivatives, as it uses less memory compared to other optimization algorithms, like the genetic algorithm. This is a very important feature, especially with large networks.
- The backpropagation algorithm is fast, especially for small and medium-sized networks. As more layers and neurons are added, it starts to get slower as more derivatives are calculated.
- This algorithm is generic enough to work with different network architectures, like convolutional neural networks, generative adversarial networks, fully-connected networks, and more.
- There are no parameters to tune the backpropagation algorithm, so there's less overhead. The only parameters in the process are related to the gradient descent algorithm, like learning rate



Shallow Neural Network Architecture

The Output layer

- The output layer consists of output neurons that make the final prediction. We must determine the size of the output layer, that is the number of neurons in the output layer. This is also a hyperparameter. Its value depends on the type of problem we want to solve. Here are some of the popular types of problems and the number of nodes we should specify for them.
- **Binary classification (two classes):** One node
- **Multi-class classification (more than two classes):** The number of classes (mutually exclusive) is equal to the number of neurons in the output layer.
- **Multi-label classification:** The number of classes (mutually inclusive) is equal to the number of neurons in the output layer.
- **Regression:** One node

**Step by Step
Developing Shallow Neural
Network (Example!)**



How it Works?

1- Input Layer:

- ✓ Training data and features

2- Hidden Layer

- ✓ Initialize Parameters
- ✓ Forward propagation
- ✓ Compute cost
- ✓ Backward propagation
- ✓ Update parameters
- ✓ Optimal Values

3- Output Layer

- ✓ Parameters learn and Predictions



Step 1: import libraries



```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.datasets import make_moons
import seaborn as sns
sns.set()
```

Step 2: Generate a Dataset with 3000 samples



```
X,y = make_circles(n_samples = 3000, noise = 0.08, factor=0.3)
#X,y = make_moons(n_samples = 3000, noise = 0.08)
X0 = []
X1 = []
for i in range(len(y)):
    if y[i] == 0:
        X0.append(X[i,:])
    else:
        X1.append(X[i,:])

X0_np = np.array(X0)
X1_np = np.array(X1)

X0_train = X0_np[:1000,:].T # we want X to be made of examples which are stacked horizontally
X0_test = X0_np[1000:,:].T
X1_train = X1_np[:1000,:].T
X1_test = X1_np[1000:,:].T

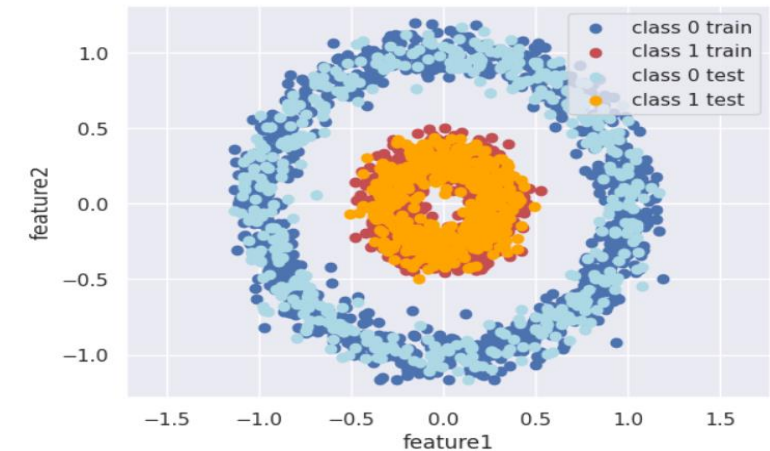
X_train = np.hstack([X0_train,X1_train]) # all training examples
y_train=np.zeros((1,2000))
y_train[0, 1000:] = 1
X_test = np.hstack([X0_test,X1_test]) # all test examples
y_test=np.zeros((1,1000))
y_test[0, 500:] = 1
```

```
plt.scatter(X0_train[0,:],X0_train[1:], color = 'b', label = 'class 0 train')
plt.scatter(X1_train[0,:],X1_train[1:], color = 'r', label = 'class 1 train')
plt.scatter(X0_test[0,:],X0_test[1:], color = 'LightBlue', label = 'class 0 test')
plt.scatter(X1_test[0,:],X1_test[1:], color = 'Orange', label = 'class 1 test')
plt.xlabel('feature1')
plt.ylabel('feature2')
plt.legend()
plt.axis('equal')
plt.show()
```

```
# we will plot shapes of training and test set to make sure that they were made by stacking examples horizontally
# so in every column of these matrices there is one examples
# in two rows there are features (feature1 is plotted on the x-axis, and feature2 is plotted on the y-axis)
print('Shape of X_train set is %i x %i.'%X_train.shape)
print('Shape of X_test set is %i x %i.'%X_test.shape)
# labels for these examples are in y_train and y_test
print('Shape of y_train set is %i x %i.'%y_train.shape)
print('Shape of y_test set is %i x %i.'%y_test.shape)
```

More details on Data:

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_circles.html



```
Shape of X_train set is 2 x 2000.
Shape of X_test set is 2 x 1000.
Shape of y_train set is 1 x 2000.
Shape of y_test set is 1 x 1000.
```

Step 3: Initialize Parameters



```
def initialize_parameters(n_x, n_h, n_y):  
  
    np.random.seed(2)  
  
    W1 = np.random.randn(n_h, n_x) * 0.01  
    b1 = np.zeros(shape=(n_h, 1))  
    W2 = np.random.randn(n_y, n_h) * 0.01  
    b2 = np.zeros(shape=(n_y, 1))  
  
    assert (W1.shape == (n_h, n_x))  
    assert (b1.shape == (n_h, 1))  
    assert (W2.shape == (n_y, n_h))  
    assert (b2.shape == (n_y, 1))  
  
    parameters = {"W1": W1,  
                  "b1": b1,  
                  "W2": W2,  
                  "b2": b2}  
  
    return parameters
```

Function `initialize_parameters(n_x, n_h, n_y)` takes as an input the size of an input layer, actually the number of features in the single example from a dataset. In our case $n_x = 2$ because we have two features (drawn on x-axis and y-axis). Parameters that we are going to use are: W_1, W_2, b_1, b_2 . Matrices W_1 and W_2 are initialized with small random values, and biases are initialized with zeros. The output of this function is a Python dictionary which contains values of all these parameters.

Step 4: Activation Function



```
[24] #def relu(x):  
      # return x*(x>0)  
def sigmoid(x):  
    return 1/(1+np.exp(-x))  
def tanh(x):  
    return (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
```

Activation functions. We will use *sigmoid* function in the output layer and *tanh* function in the hidden layer.

Step 5: Forward propagation

```
def forward_pass(X, parameters):  
  
    # to make forward pass calculations we need W1 and W2 so we will extract them from dictionary parameters  
    W1 = parameters['W1']  
    W2 = parameters['W2']  
    b1 = parameters['b1']  
    b2 = parameters['b2']  
  
    # first layer calculations - hidden layer calculations  
    Z1 = np.dot(W1, X) + b1  
    A1 = tanh(Z1) # activation in the first layer is tanh  
  
    # output layer calculations  
    Z2 = np.dot(W2, A1) + b2  
    A2 = sigmoid(Z2) # A2 are predictions, y_hat  
  
    # cache values for backpropagation calculations  
    cache = {'Z1': Z1,  
            'A1': A1,  
            'Z2': Z2,  
            'A2': A2  
            }  
  
    return A2, cache
```

Function `forward_pass(X, parameters)` makes calculations for the forward pass through the neural network. It takes \mathbf{X} and `parameters` as arguments and outputs A_2 (the output of a neural network, it is matrix of predictions) and dictionary cache. This dictionary contains cached values which we will use in backward propagation calculations.

```
def cost_calculation(A2,Y):  
  
    m = Y.shape[1]  
    cost = (- 1 / m) * np.sum(Y * np.log(A2) + (1 - Y) * (np.log(1 - A2))) # compute cost  
    return cost
```



Step 6: Cost Function

Function cost_calculation computes a cost when ground truth label is Y and the neural network outputs A2.

Step 7: Backward propagation



```
def backward_pass(parameters, cache, X, Y):

    # unpack parameters and cache to get values for backpropagation calculations
    W1 = parameters['W1']
    W2 = parameters['W2']

    Z1 = cache['Z1']
    A1 = cache['A1']
    Z2 = cache['Z2']
    A2 = cache['A2']

    m = X.shape[1] # number of examples in a training set

    dZ2= A2 - Y
    dW2 = (1 / m) * np.dot(dZ2, A1.T)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True) # keepdims - prevents python to output rank 1 array (n,)

    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2)) # we use tanh activation function
    dW1 = (1 / m) * np.dot(dZ1, X.T)
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

    grads = {"dw1": dW1,
             "db1": db1,
             "dw2": dW2,
             "db2": db2}

    return grads
```

Function `backward_pass` which takes parameters, cached values from forward propagation step and dataset **X** which has labels **Y**.

Step 8: Update Parameters



```
def update_parameters(parameters, learning_rate, grads):
```

```
    w1 = parameters['w1']  
    b1 = parameters['b1']  
    w2 = parameters['w2']  
    b2 = parameters['b2']
```

```
    dw1 = grads['dw1']  
    db1 = grads['db1']  
    dw2 = grads['dw2']  
    db2 = grads['db2']
```

```
    w1 = w1 - learning_rate * dw1  
    b1 = b1 - learning_rate * db1  
    w2 = w2 - learning_rate * dw2  
    b2 = b2 - learning_rate * db2
```

```
    # updated parameters
```

```
    parameters = {"w1": w1,  
                  "b1": b1,  
                  "w2": w2,  
                  "b2": b2}
```

```
    return parameters
```

Function `update_parameters` updates parameters in every iteration.

Step 9: Optimal Values



```
def NN_model(X,Y,n_h, num_iterations, learning_rate):  
  
    n_x = X.shape[0] # size of an input layer = number of features  
    n_y = Y.shape[0] # size of an output layer  
    parameters = initialize_parameters(n_x, n_h, n_y)  
  
    #unpack parameters  
    W1 = parameters["W1"]  
    b1 = parameters["b1"]  
    W2 = parameters["W2"]  
    b2 = parameters["b2"]  
  
    for i in range(num_iterations):  
        A2, cache = forward_pass(X, parameters)  
        cost = cost_calculation(A2, Y)  
        grads = backward_pass(parameters, cache, X, Y)  
        parameters = update_parameters(parameters, learning_rate, grads)  
  
    return parameters
```

Function NN_model calculates optimal values for parameters. Those are parameters that are learned through num_iterations iterations.

Step 10: Prediction



```
def predict(parameters, X):  
  
    A2, cache = forward_pass(X, parameters)  
    predictions = np.round(A2)  
  
    return predictions
```

Step 11: Parameters to Learn



```
| num_iterations = 20000  
| learning_rate = 0.1  
| n_h = 2  
| parameters_final = NN_model(X_train,y_train,n_h, num_iterations, learning_rate)
```

Step 12: Results



```
Y_predictions_test = predict(parameters_final, X_test)
Y_predictions_train = predict(parameters_final, X_train)
```

```
print("train accuracy: {} %".format(100 - np.mean(np.abs(Y_predictions_train - y_train)) * 100))
print("test accuracy: {} %".format(100 - np.mean(np.abs(Y_predictions_test - y_test)) * 100))
```

```
train accuracy: 86.75 %
test accuracy: 83.0 %
```