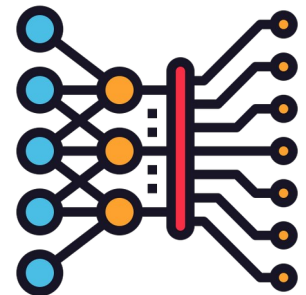


# Deep Learning

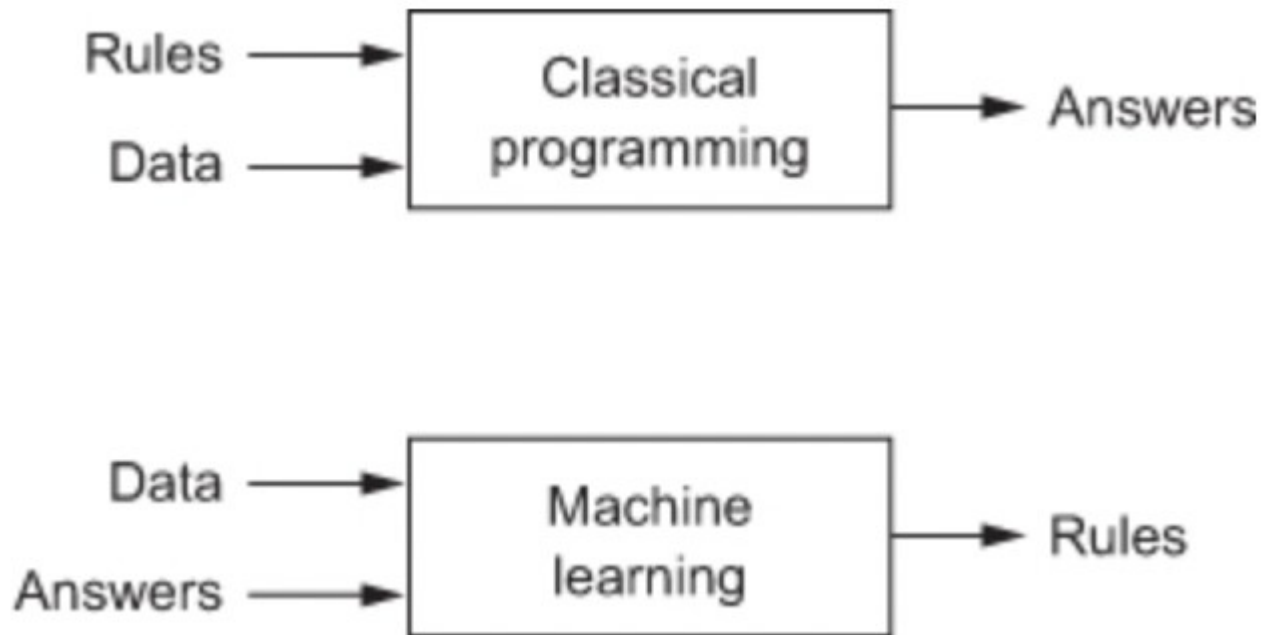
Tushar B. Kute,  
<http://tusharkute.com>



# What is Deep Learning?

- Deep Learning is a subset of Machine Learning that uses mathematical functions to map the input to the output.
- These functions can extract non-redundant information or patterns from the data, which enables them to form a relationship between the input and the output.
- This is known as learning, and the process of learning is called training.

# Programming Patterns

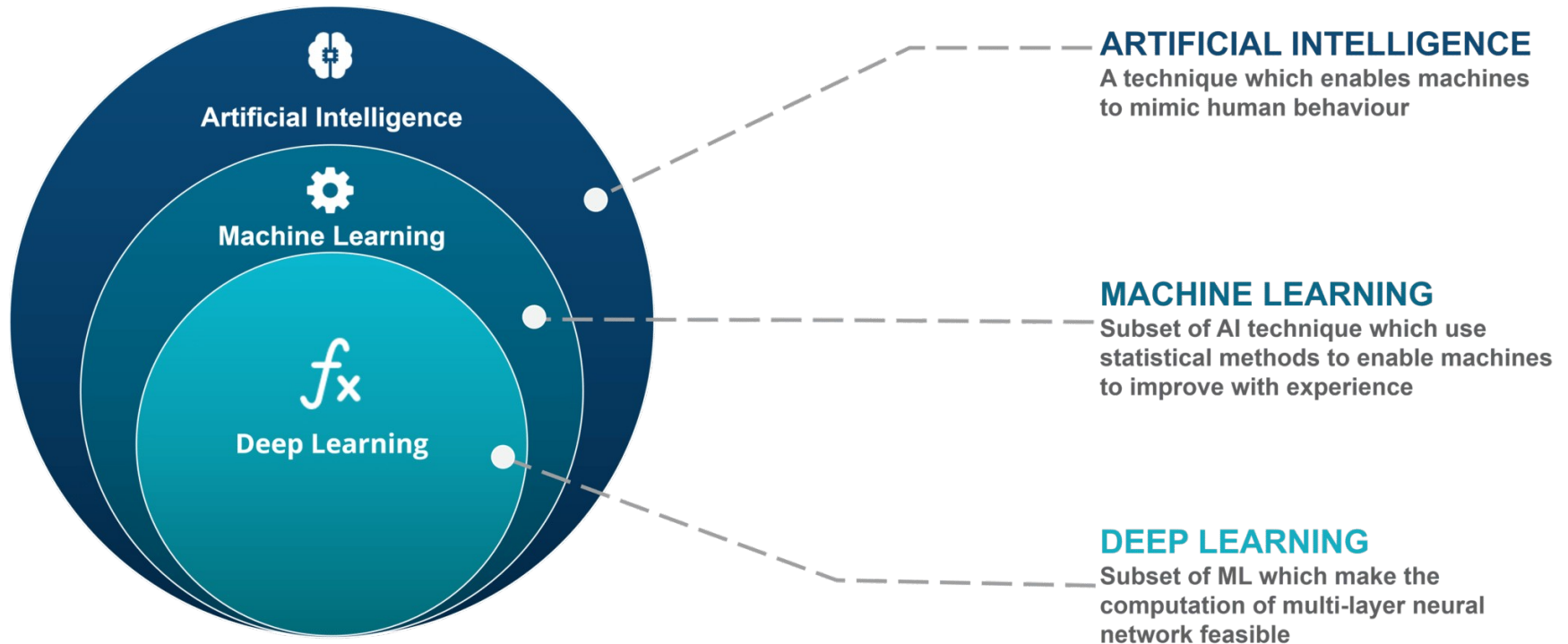


Reference: <https://www.v7labs.com>

# Deep Learning

- Modern deep learning models use artificial neural networks or simply neural networks to extract information.
- These neural networks are made up of a simple mathematical function that can be stacked on top of each other and arranged in the form of layers, giving them a sense of depth, hence the term Deep Learning.
- Deep learning can also be thought of as an approach to Artificial Intelligence, a smart combination of hardware and software to solve tasks requiring human intelligence.

# Deep Learning

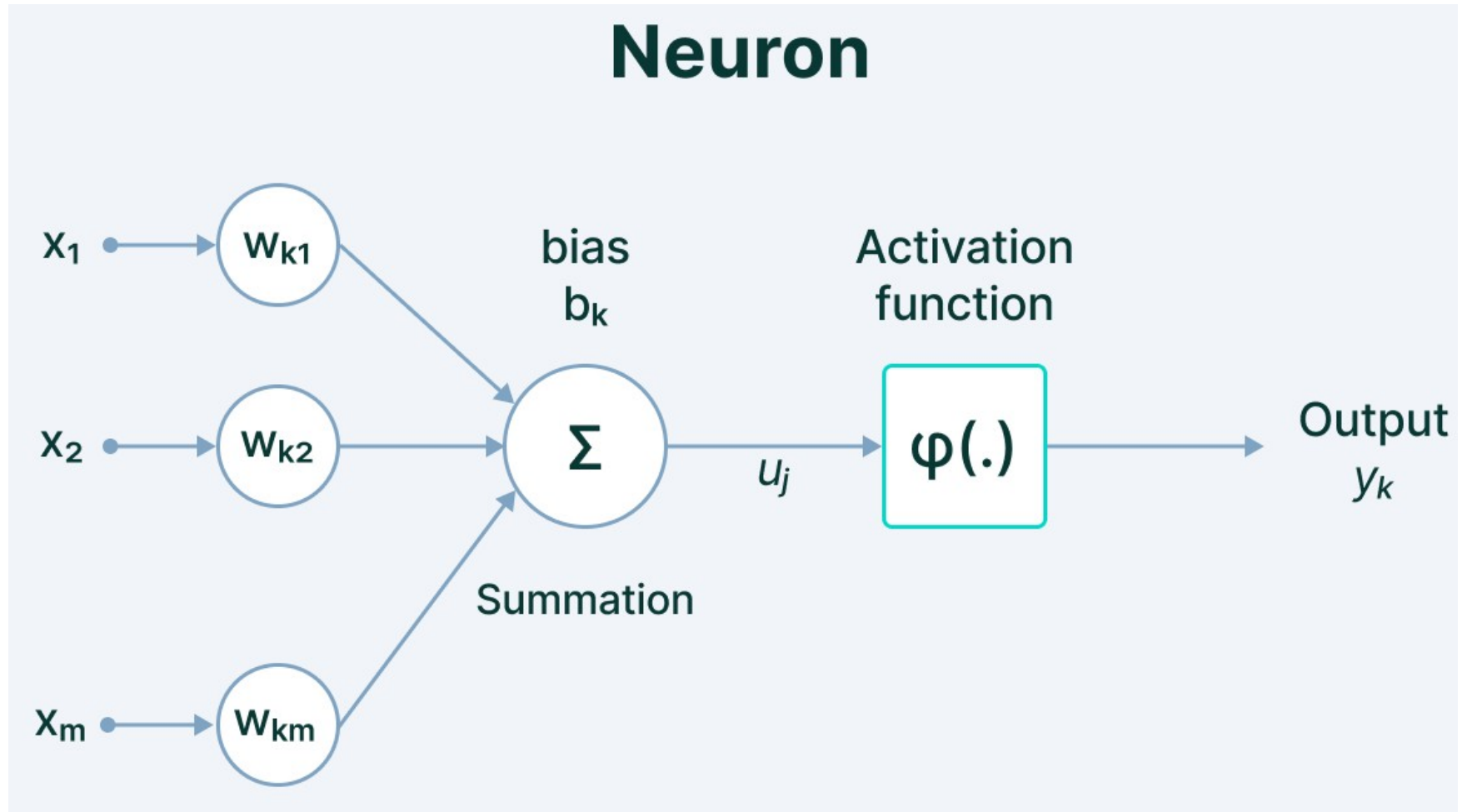


# Deep Learning

- Deep Learning was first theorized in the 1980s, but it has only become useful recently because:
  - It requires large amounts of labeled data
  - It requires significant computational power (high performing GPUs)

# Neuron

## Neuron



# Neuron

- The neuronal perception of deep learning is generally motivated by two main ideas:
  - It is assumed that the human brain proves that intelligent behavior is possible, and—by reverse engineering, it is possible to build an intelligent system
  - Another perspective is that to understand the working of the human brain and the principles that underlie its intelligence is to build a mathematical model that could shed light on the fundamental scientific questions.



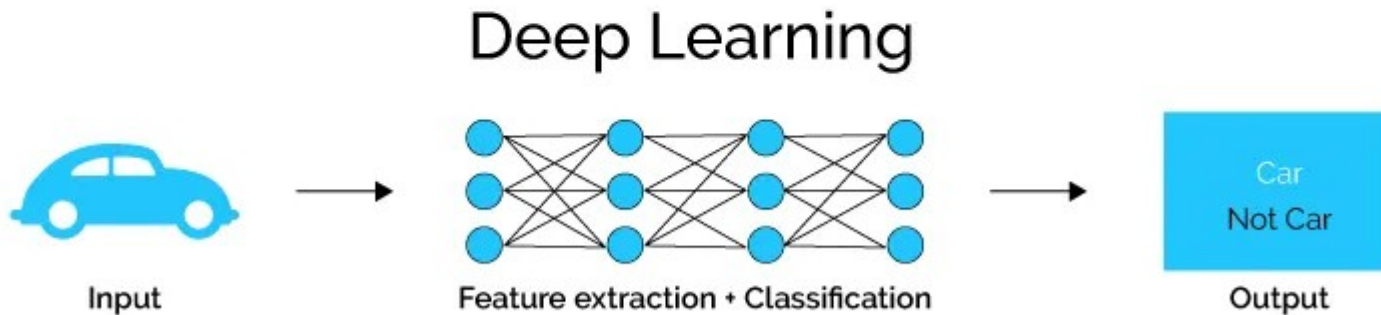
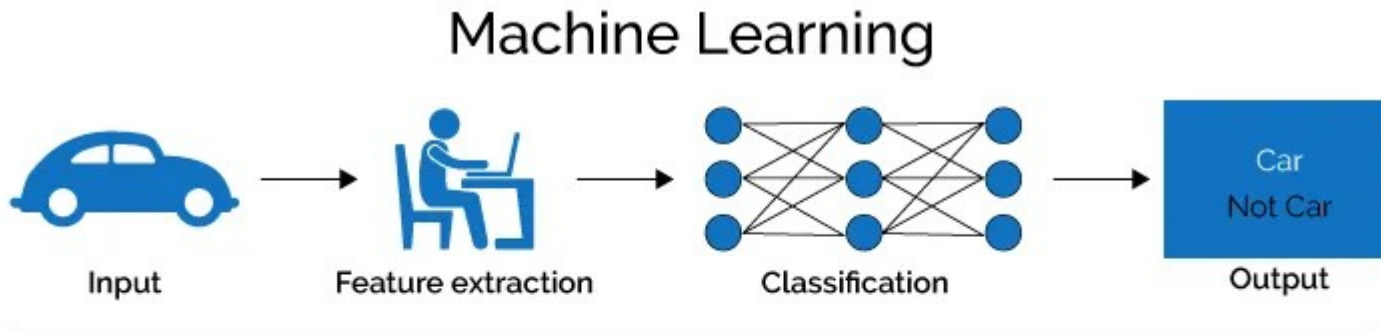
# Deep Learning vs. Machine Learning

- Deep Learning can essentially do everything that machine learning does, but not the other way around.
- For instance, machine learning is useful when the dataset is small and well-curated, which means that the data is carefully preprocessed.
- Data preprocessing requires human intervention. It also means that when the dataset is large and complex, machine learning algorithms will fail to extract information, and it will underfit.

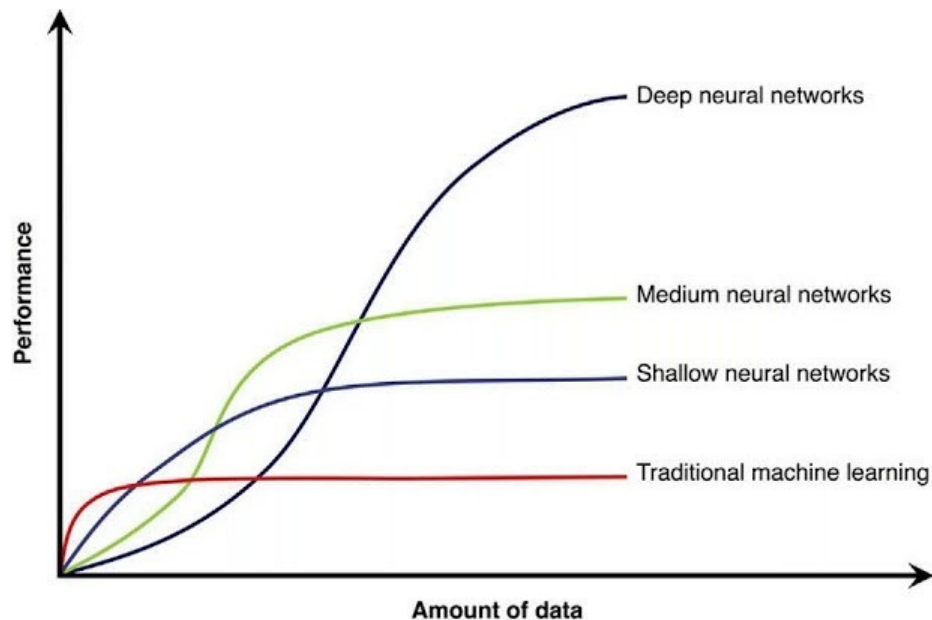
# Deep Learning vs. Machine Learning

- Generally, machine learning is alternatively termed shallow learning because it is very effective for smaller datasets.
- Deep learning, on the other hand, is extremely powerful when the dataset is large.
- It can learn any complex patterns from the data and can draw accurate conclusions on its own. In fact, deep learning is so powerful that it can even process unstructured data - data that is not adequately arranged like text corpus, social media activity, etc.
- Furthermore, it can also generate new data samples and find anomalies that machine learning algorithms and human eyes can miss.

# Deep Learning vs. Machine Learning



# Why Deep Learning ?



## Why Now?

- *Algorithm Advancements*
- *GPU Computing*
- *Availability of Larger Training Data*

# Deep Learning vs. Machine Learning

- On the downside, deep learning is computationally expensive compared to machine learning, which also means that it requires a lot of time to process.
- Deep Learning and Machine Learning are both capable of different types of learning: Supervised Learning (labeled data), Unsupervised Learning (unlabeled data), and Reinforcement Learning.
- But their usefulness is usually determined by the size and complexity of the data.

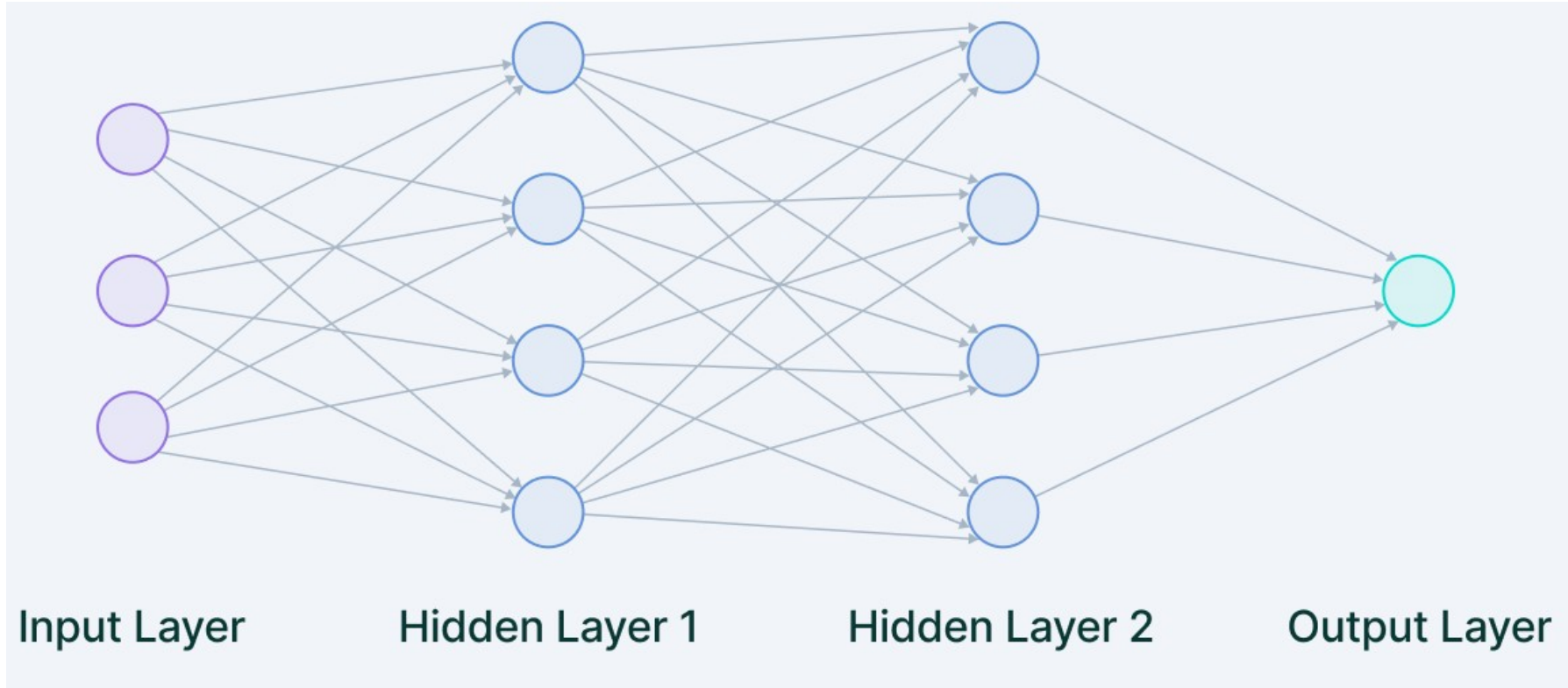
# A Quick Summary

- Machine learning requires data preprocessing, which involves human intervention.
- The neural networks in deep learning are capable of extracting features; hence no human intervention is required.
- Deep Learning can process unstructured data.
- Deep Learning is usually based on representative learning i.e., finding and extracting vital information or patterns that represent the entire dataset.
- Deep learning is computationally expensive and time-consuming.

# How does Deep Learning work?

- Deep Neural Networks have multiple layers of interconnected artificial neurons or nodes that are stacked together.
- Each of these nodes has a simple mathematical function - usually a linear function that performs extraction and mapping of information.
- There are three layers to a deep neural network: the input layer, hidden layers, and the output layer.

# How does Deep Learning work?



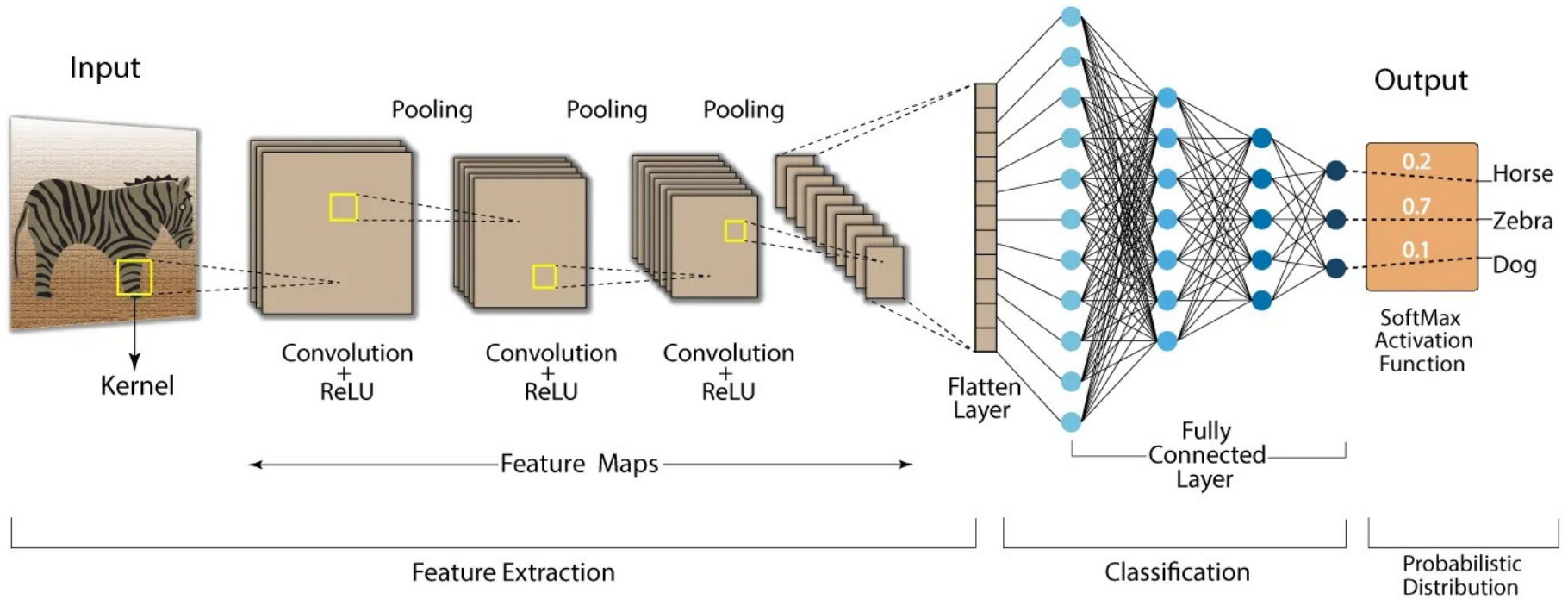


# Types of Neural Network

- Artificial Neural Network
- Convolutional Neural Network
- Recurrent Neural Network
- Generative Adversarial Network

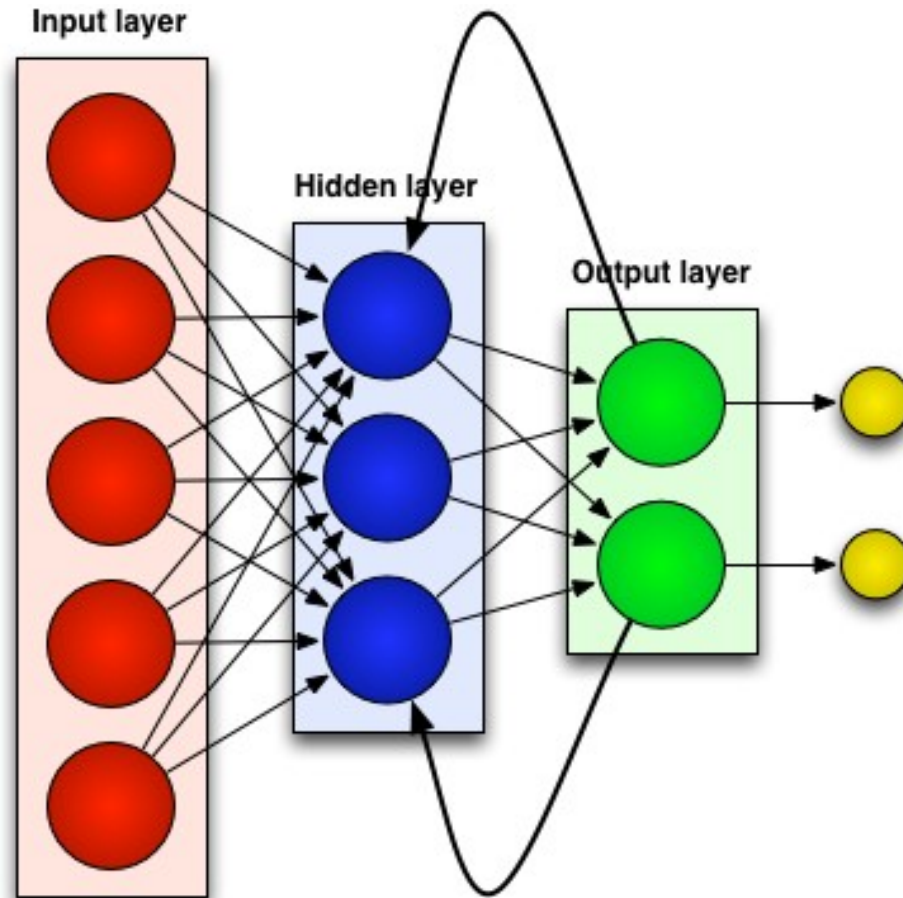
- The Convolutional Neural Networks or CNNs are primarily used for tasks related to computer vision or image processing.
- CNNs are extremely good in modeling spatial data such as 2D or 3D images and videos.
- They can extract features and patterns within an image, enabling tasks such as image classification or object detection.

# CNN



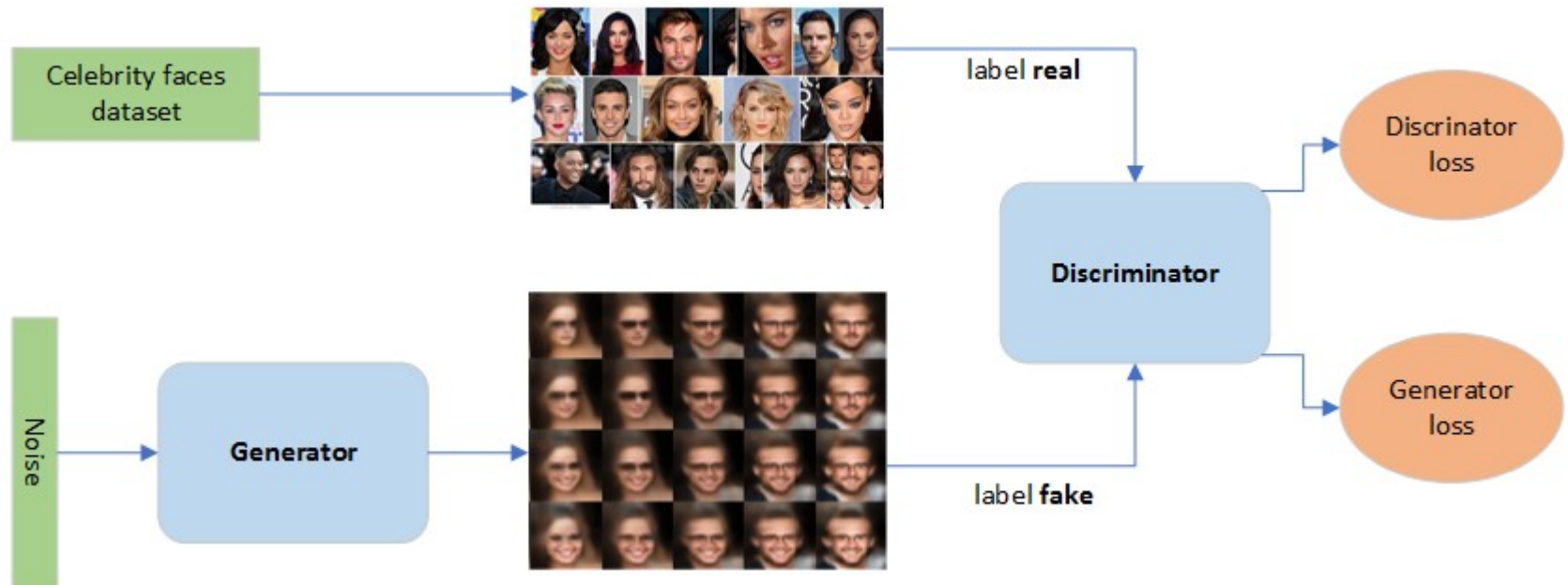
- The Recurrent Neural Networks or RNN are primarily used to model sequential data, such as text, audio, or any type of data that represents sequence or time.
- They are often used in tasks related to natural language processing (NLP).

# RNN



- Generative adversarial networks or GANs are frameworks that are used for the tasks related to unsupervised learning.
- This type of network essentially learns the structure of the data, and patterns in a way that it can be used to generate new examples, similar to that of the original dataset.

# GAN



# Transformers

- Transformers are the new class deep learning model that is used mostly for the tasks related to modeling sequential data, like that in NLP.
- It is much more powerful than RNNs and they are replacing them in every task.
- Recently, transformers are also being applied in computer vision tasks and they are proving to be quite effective than the traditional CNNs.



# Deep Learning: Limitations

- Data availability
- The complexity of the model
- Lacks global generalization
- Incapable of Multitasking
- Hardware dependence

# Deep Learning: Limitations

- Data availability
  - Deep learning models require a lot of data to learn the representation, structure, distribution, and pattern of the data.
  - If there isn't enough varied data available, then the model will not learn well and will lack generalization (it won't perform well on unseen data).
  - The model can only generalize well if it is trained on large amounts of data.

# Deep Learning: Limitations

- The complexity of the model
  - Designing a deep learning model is often a trial and error process.
  - A simple model is most likely to underfit, i.e. not able to extract information from the training set, and a very complex model is most likely to overfit, i.e., not able to generalize well on the test dataset.
  - Deep learning models will perform well when their complexity is appropriate to the complexity of the data.

# Deep Learning: Limitations

- Lacks global generalization
  - A simple neural network can have thousands to tens of thousands of parameters.
  - The idea of global generalization is that all the parameters in the model should cohesively update themselves to reduce the generalization error or test error as much as possible. However, because of the complexity of the model, it is very difficult to achieve zero generalization error on the test set.
  - Hence, the deep learning model will always lack global generalization which can at times yield wrong results.

# Deep Learning: Limitations

- Incapable of Multitasking
  - Deep neural networks are incapable of multitasking.
  - These models can only perform targeted tasks, i.e., process data on which they are trained. For instance, a model trained on classifying cats and dogs will not classify men and women.
  - Furthermore, applications that require reasoning or general intelligence are completely beyond what the current generation's deep learning techniques can do, even with large sets of data.

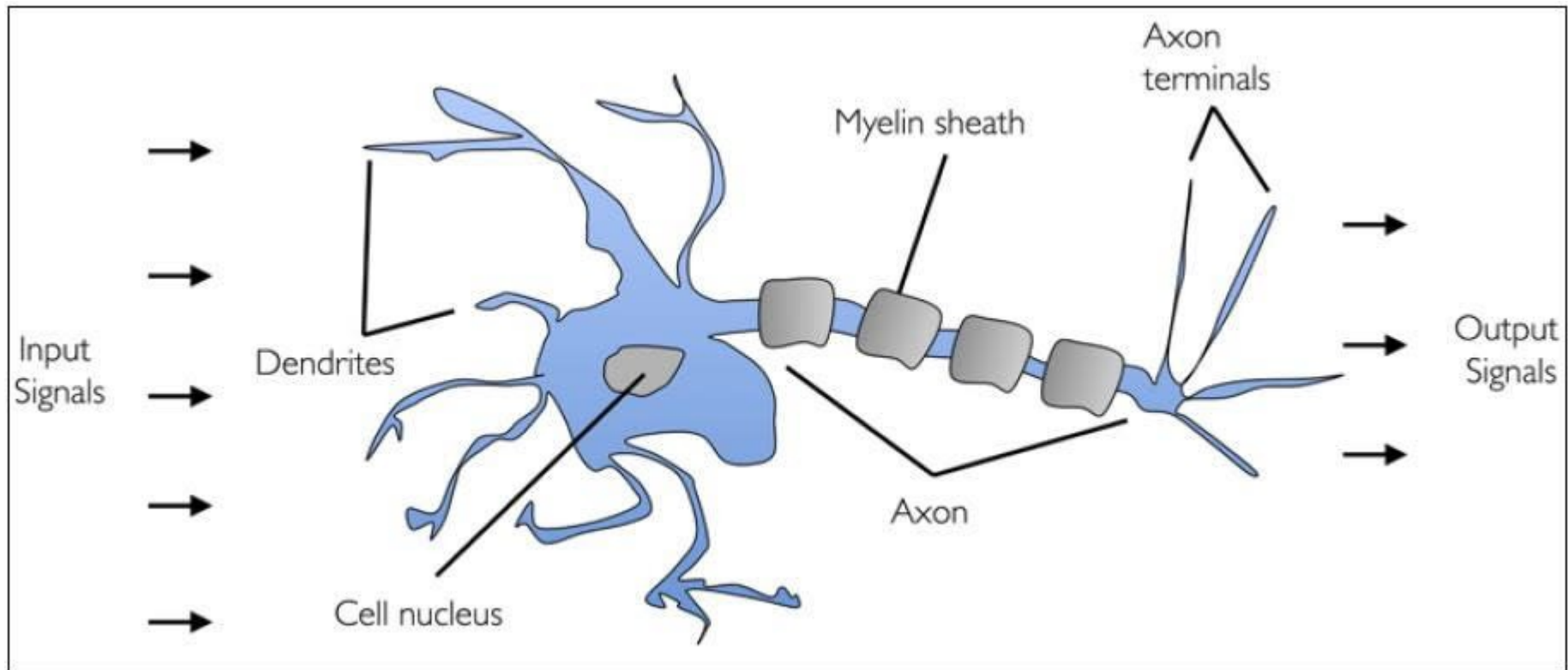
# Deep Learning: Limitations

- Hardware dependence
  - As mentioned before, deep learning models are computationally expensive.
  - These models are so complex that a normal CPU will not be able to withstand the computational complexity.
  - However, multicore high-performing graphics processing units (GPUs) and tensor processing units (TPUs) are required to effectively train these models in a shorter time.
  - Although these processors save time, they are expensive and use large amounts of energy.

# Deep Learning: Applications



# Biological Neuron

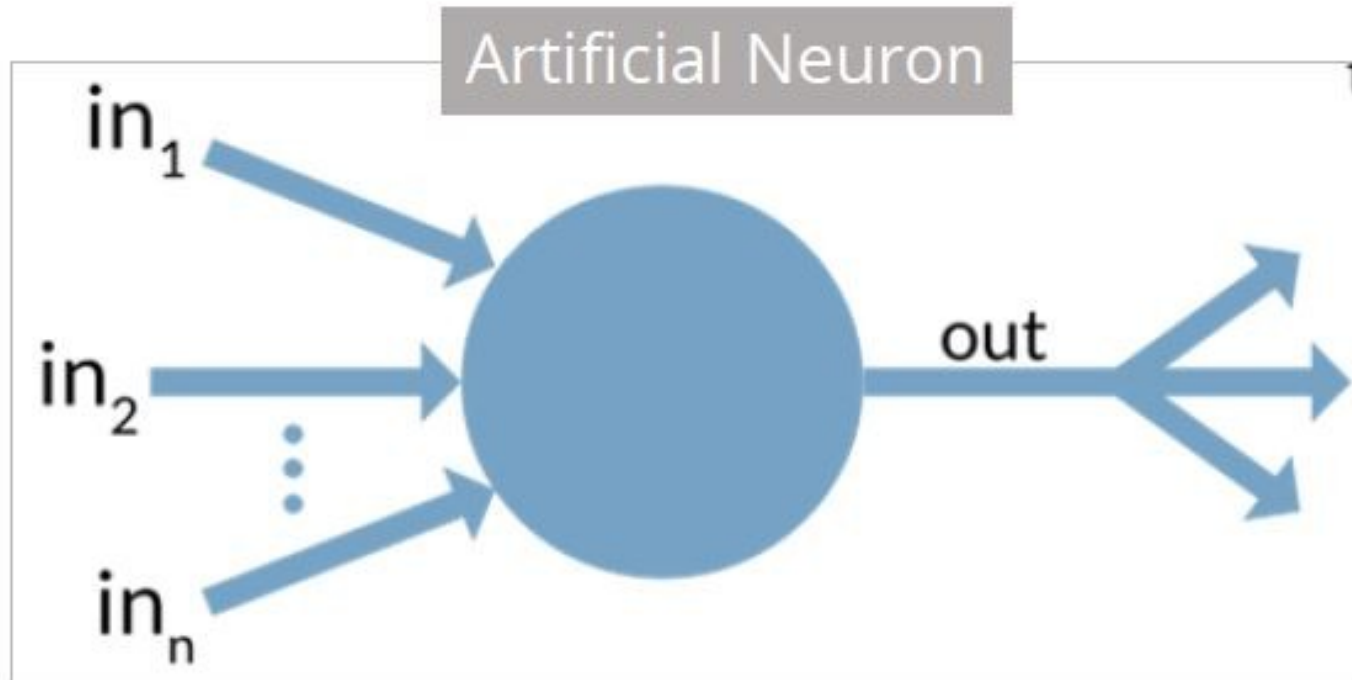




# Artificial Neuron

- Researchers Warren McCullock and Walter Pitts published their first concept of simplified brain cell in 1943.
- This was called McCullock-Pitts (MCP) neuron. They described such a nerve cell as a simple logic gate with binary outputs.
- Multiple signals arrive at the dendrites and are then integrated into the cell body, and, if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon.

# Artificial Neuron



# Biological vs. Artificial Neuron

Biological Neuron	Artificial Neuron
Cell Nucleus (Soma)	Node
Dendrites	Input
Synapse	Weights or interconnections
Axon	Output

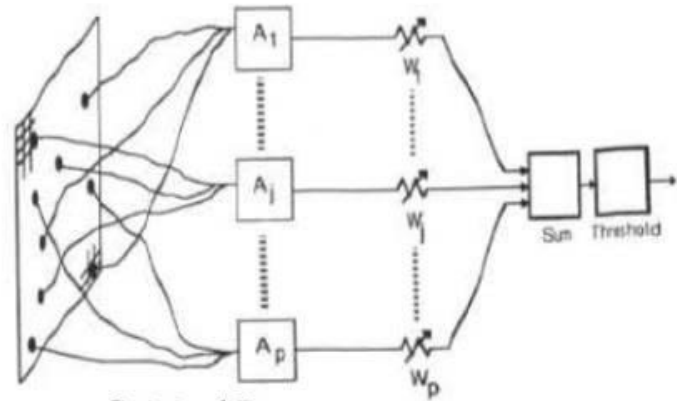
# Artificial Neuron

- The artificial neuron has the following characteristics:
  - A neuron is a mathematical function modeled on the working of biological neurons
  - It is an elementary unit in an artificial neural network
  - One or more inputs are separately weighted
  - Inputs are summed and passed through a nonlinear function to produce output
  - Every neuron holds an internal state called activation signal
  - Each connection link carries information about the input signal
  - Every neuron is connected to another neuron via connection link

# Perceptron

- A perceptron is a neural network unit (an artificial neuron) that does certain computations to detect features or business intelligence in the input data.

## Perceptron (1957)

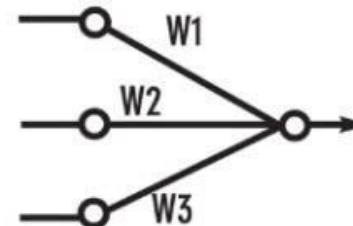


Frank Rosenblatt  
(1928-1971)

Original Perceptron

(From *Perceptrons* by M. L. Minsky and S. Papert,  
1969, Cambridge, MA: MIT Press. Copyright 1969  
by MIT Press.)

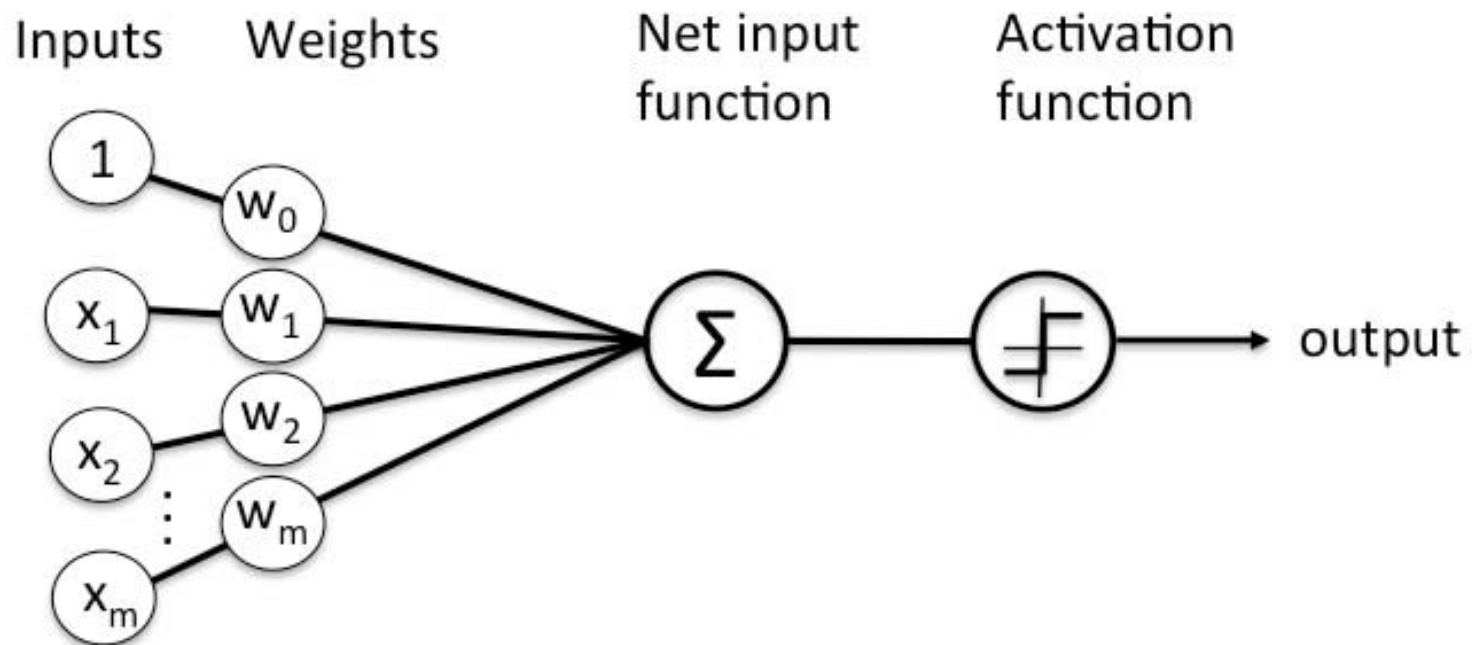
Simplified model:



# Perceptron

- Perceptron was introduced by Frank Rosenblatt in 1957.
- He proposed a Perceptron learning rule based on the original MCP neuron.
- A Perceptron is an algorithm for supervised learning of binary classifiers.
- This algorithm enables neurons to learn and processes elements in the training set one at a time.

# Perceptron



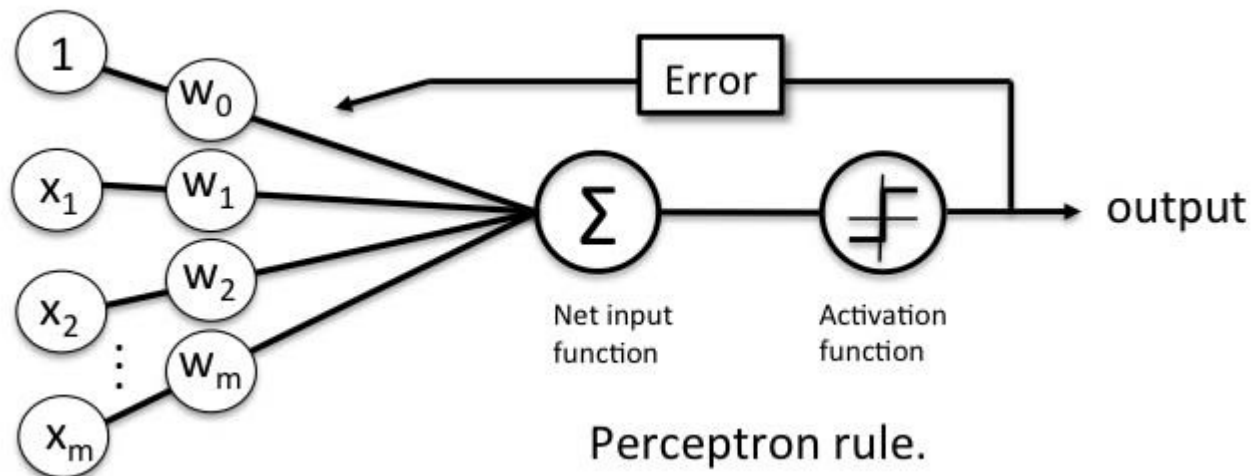
# Perceptron

- There are two types of Perceptrons: Single layer and Multilayer.
- Single layer Perceptrons can learn only linearly separable patterns.
- Multilayer Perceptrons or feedforward neural networks with two or more layers have the greater processing power.
- The Perceptron algorithm learns the weights for the input signals in order to draw a linear decision boundary.
- This enables you to distinguish between the two linearly separable classes +1 and -1.
- Note: Supervised Learning is a type of Machine Learning used to learn models from labeled training data. It enables output prediction for future or unseen data.



# Perceptron Learning Rule

- 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.



# 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

# Perceptron function

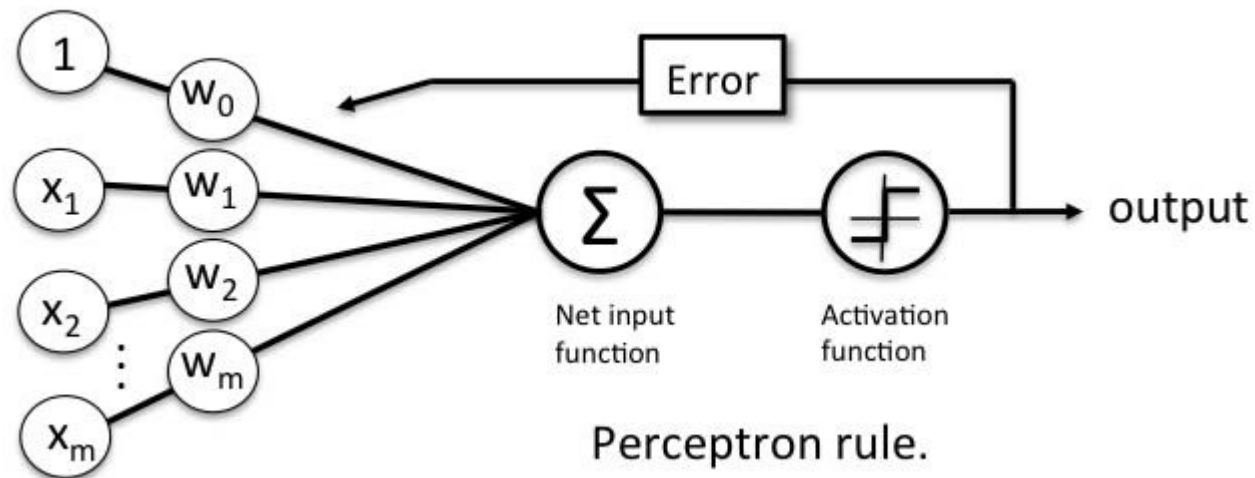
“m” = number of inputs to the Perceptron

$$\sum_{i=1}^m w_i x_i$$

- 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.

# Inputs of Perceptron

- A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result. The above below shows a Perceptron with a Boolean output.



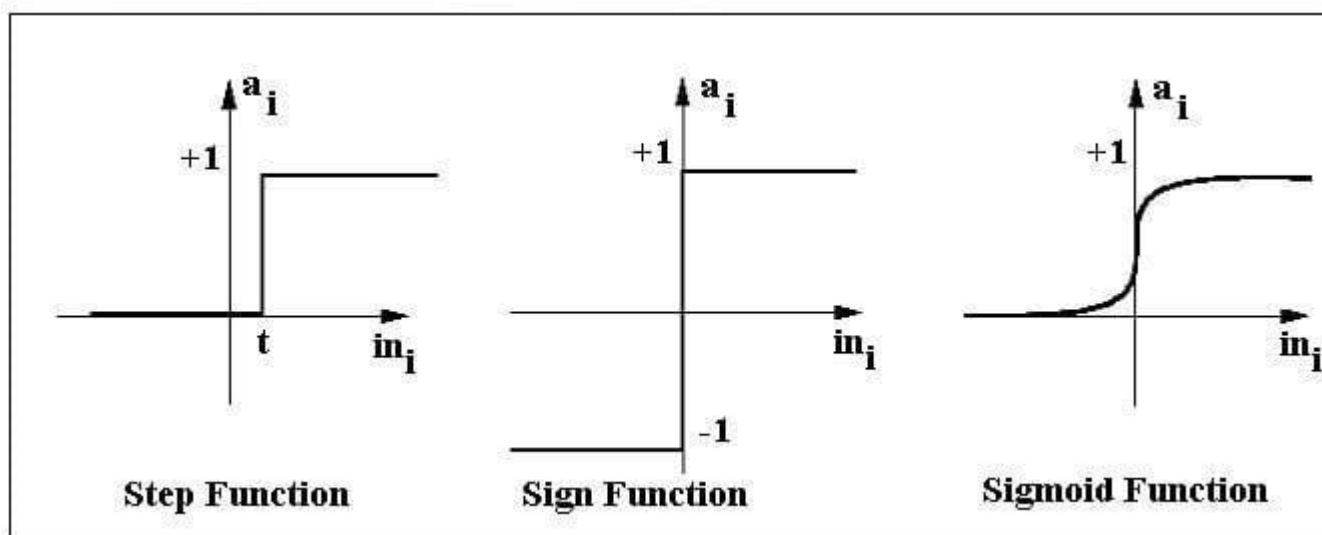
# Inputs of Perceptron

- A Boolean output is based on inputs such as salaried, married, age, past credit profile, etc. It has only two values: Yes and No or True and False.
- The summation function “ $\Sigma$ ” multiplies all inputs of “x” by weights “w” and then adds them up as follows:

$$w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

# Activation function

- The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not.



# Example

- For example:

If  $\sum w_i x_i > 0 \Rightarrow$  then final output “o” = 1 (issue bank loan)

Else, final output “o” = -1 (deny bank loan)

- Step function gets triggered above a certain value of the neuron output; else it outputs zero. Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not. Sigmoid is the S-curve and outputs a value between 0 and 1.

# 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 decision function

- A decision function  $\phi(z)$  of Perceptron is defined to take a linear combination of  $x$  and  $w$  vectors.

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

# Perceptron: Decision Function

- The value  $z$  in the decision function is given by:

$$Z = w_1x_1 + \dots + w_mx_m$$

- The decision function is +1 if  $z$  is greater than a threshold  $\theta$ , and it is -1 otherwise.

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

- This is the Perceptron algorithm.

# Bias Unit

- For simplicity, the threshold  $\theta$  can be brought to the left and represented as  $w_0x_0$ , where  $w_0 = -\theta$  and  $x_0 = 1$ .

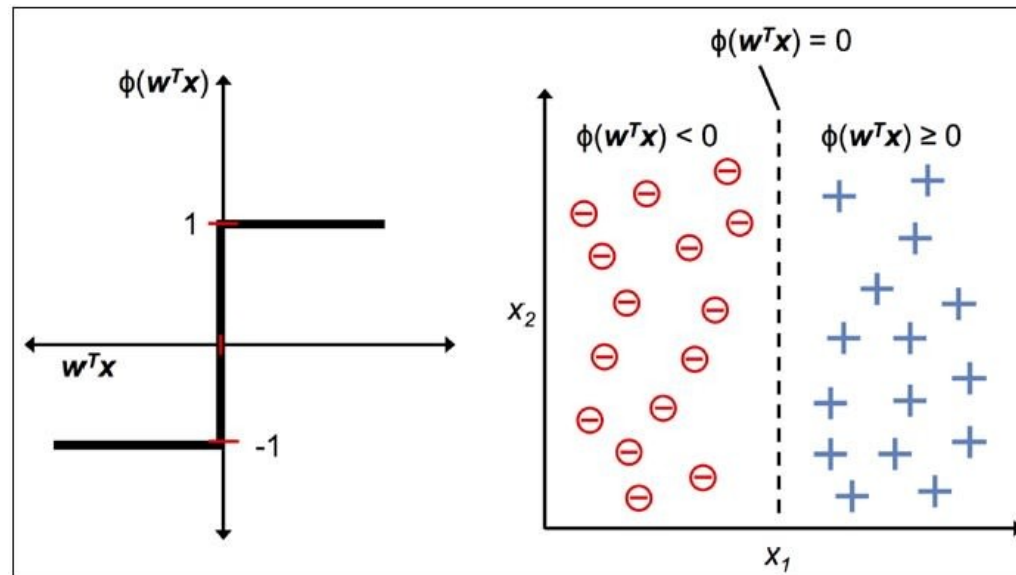
$$Z = w_0x_0 + w_1x_1 + \dots + w_mx_m = \mathbf{w}^T \mathbf{x}$$

- The value  $w_0$  is called the bias unit.
- The decision function then becomes:

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

# Output

- The figure shows how the decision function squashes  $w^T x$  to either +1 or -1 and how it can be used to discriminate between two linearly separable classes.

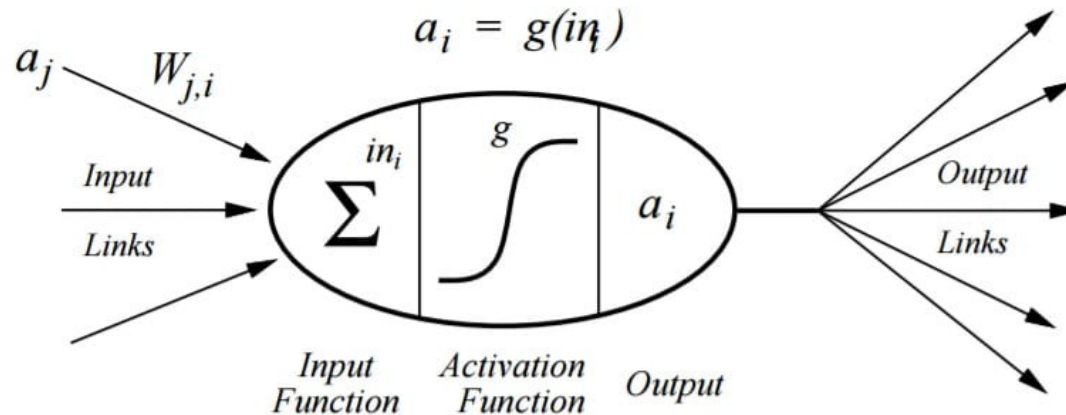


# Perceptron at a Glance

- Perceptron has the following characteristics:
  - Perceptron is an algorithm for Supervised Learning of single layer binary linear classifier.
  - Optimal weight coefficients are automatically learned.
  - Weights are multiplied with the input features and decision is made if the neuron is fired or not.
  - Activation function applies a step rule to check if the output of the weighting function is greater than zero.
  - Linear decision boundary is drawn enabling the distinction between the two linearly separable classes +1 and -1.
  - If the sum of the input signals exceeds a certain threshold, it outputs a signal; otherwise, there is no output.

# Sigmoid Activation Function

- The diagram given here shows a Perceptron with sigmoid activation function. Sigmoid is one of the most popular activation functions.



$$a_i = g\left(\sum_j W_{j,i} a_j\right)$$

# Sigmoid Activation Function

- A Sigmoid Function is a mathematical function with a Sigmoid Curve (“S” Curve). It is a special case of the logistic function and is defined by the function given below:

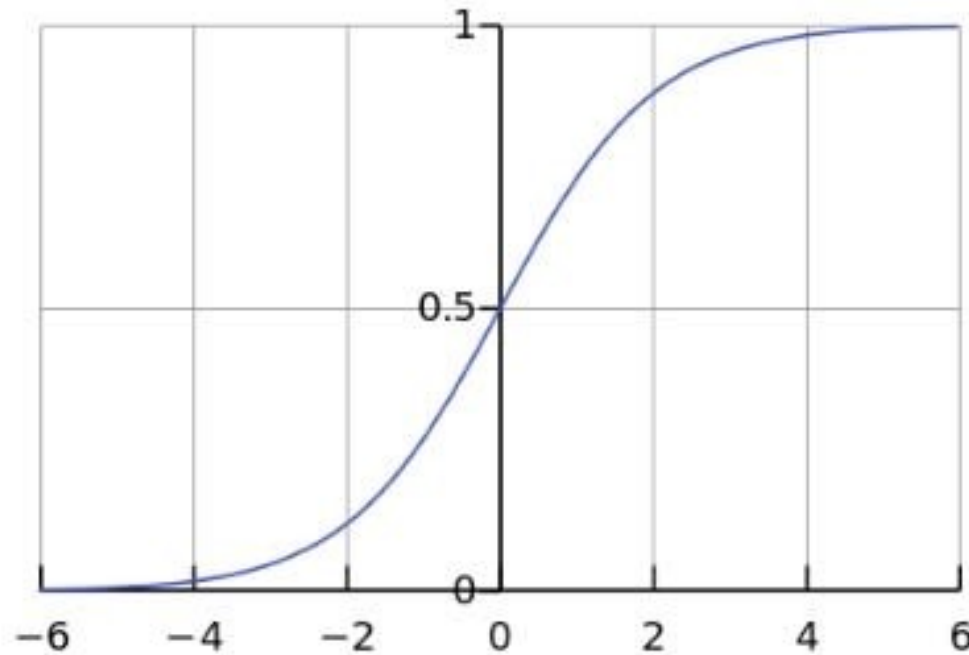
$$\phi_{\text{logistic}}(z) = \frac{1}{1 + e^{-z}}$$

- Here, value of  $z$  is:

$$z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{i=0}^m w_ix_i = w^T x$$

# Sigmoid Curve

- The curve of the Sigmoid function called “S Curve” is shown here.





# Sigmoid Curve

- This is called a logistic sigmoid and leads to a probability of the value between 0 and 1.
- This is useful as an activation function when one is interested in probability mapping rather than precise values of input parameter  $t$ .
- The sigmoid output is close to zero for highly negative input.
- This can be a problem in neural network training and can lead to slow learning and the model getting trapped in local minima during training.
- Hence, hyperbolic tangent is more preferable as an activation function in hidden layers of a neural network.

# Example:

```
>>> import numpy as np

>>> X = np.array([1, 1.4, 2.5]) ## first value must be 1
>>> w = np.array([0.4, 0.3, 0.5])

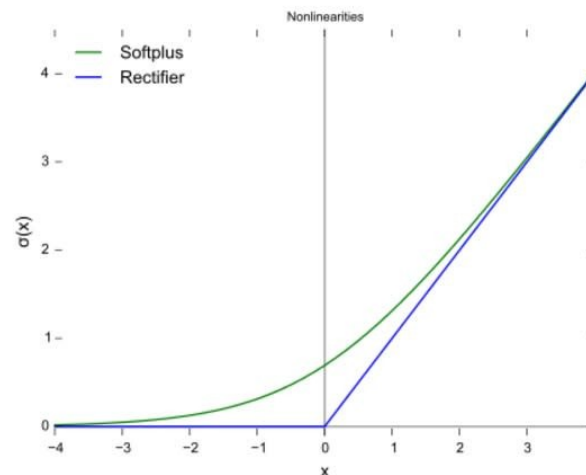
>>> def net_input(X, w):
...     return np.dot(X, w)
...
>>> def logistic(z):
...     return 1.0 / (1.0 + np.exp(-z))
...
>>> def logistic_activation(X, w):
...     z = net_input(X, w)
...     return logistic(z)
...
>>> print('P(y=1|x) = %.3f' % logistic_activation(X, w))
P(y=1|x) = 0.888
```

# Output

- The Perceptron output is 0.888, which indicates the probability of output  $y$  being a 1.
- If the sigmoid outputs a value greater than 0.5, the output is marked as TRUE.
- Since the output here is 0.888, the final output is marked as TRUE.
- In the next section, let us focus on the rectifier and softplus functions.

# Relu and Softplus

- Apart from Sigmoid and Sign activation functions seen earlier, other common activation functions are ReLU and Softplus.
- They eliminate negative units as an output of max function will output 0 for all units 0 or less.



# Relu and Softplus

- A rectifier or ReLU (Rectified Linear Unit) is a commonly used activation function.
- This function allows one to eliminate negative units in an ANN. This is the most popular activation function used in deep neural networks.
- A smooth approximation to the rectifier is the Softplus function:
- The derivative of Softplus is the logistic or sigmoid function:

# Softmax Function

- Another very popular activation function is the Softmax function. The Softmax outputs probability of the result belonging to a certain set of classes. It is akin to a categorization logic at the end of a neural network.
- For example, it may be used at the end of a neural network that is trying to determine if the image of a moving object contains an animal, a car, or an airplane.
- In Mathematics, the Softmax or normalized exponential function is a generalization of the logistic function that squashes a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range (0, 1) that add up to 1.

# Softmax Function

- In probability theory, the output of Softmax function represents a probability distribution over K different outcomes.
- In Softmax, the probability of a particular sample with net input  $z$  belonging to the  $i$ th class can be computed with a normalization term in the denominator, that is, the sum of all  $M$  linear functions:

$$p(y = i|z) = \phi(z) = \frac{e^{z_i}}{\sum_{j=1}^M e^{z_j}}$$

- The Softmax function is used in ANNs and Naïve Bayes classifiers.

# Softmax Function

- For example, if we take an input of  $[1, 2, 3, 4, 1, 2, 3]$ , the Softmax of that is  $[0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]$ .
- The output has most of its weight if the original input is '4'
- This function is normally used for:
  - Highlighting the largest values
  - Suppressing values that are significantly below the maximum value.



# Softmax Function

```
>>> def softmax(z):  
...     return np.exp(z) / np.sum(np.exp(z))  
...  
>>> y_probas = softmax(Z)  
>>> print('Probabilities:\n', y_probas)  
Probabilities:  
[ 0.44668973  0.16107406  0.39223621]  
  
>>> np.sum(y_probas)  
1.0
```

This code implements the softmax formula and prints the probability of belonging to one of the three classes. The sum of probabilities across all classes is 1.

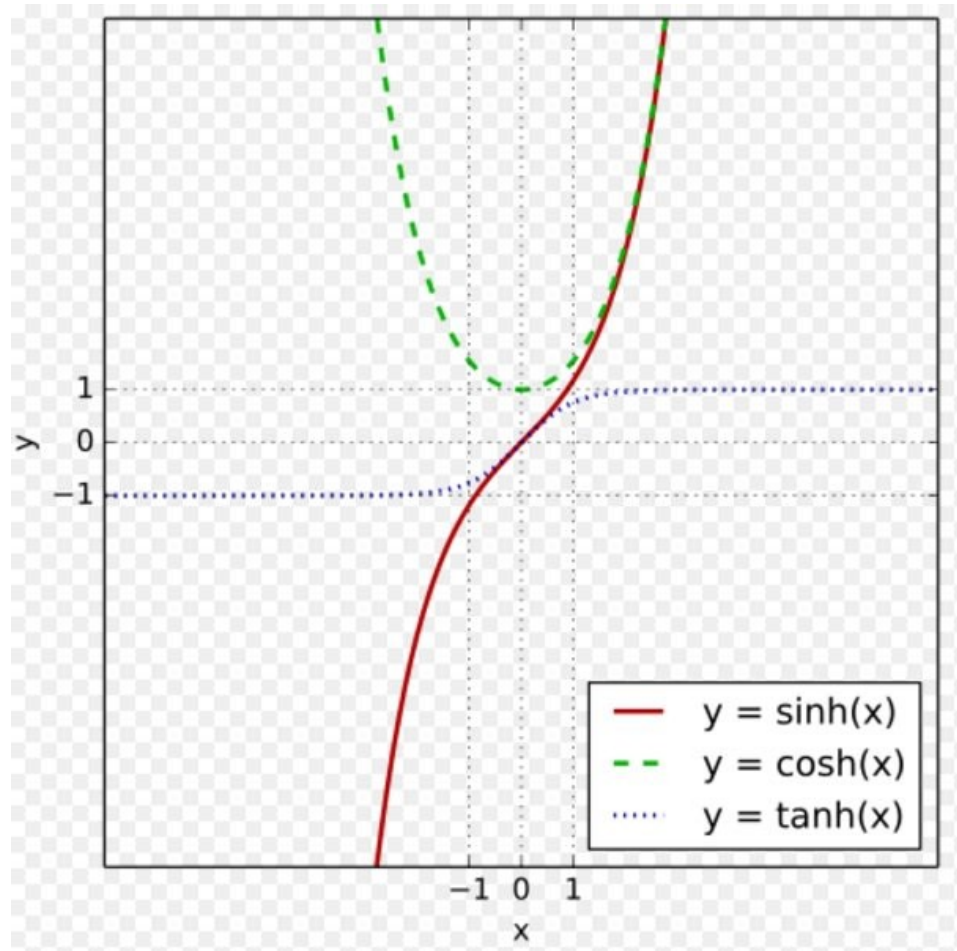
# Hyperbolic Tangent

- Hyperbolic or tanh function is often used in neural networks as an activation function. It provides output between -1 and +1. This is an extension of logistic sigmoid; the difference is that output stretches between -1 and +1 here.

$$\phi_{\tanh}(z) = 2 \times \phi_{\text{logistic}}(2z) - 1 = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- The advantage of the hyperbolic tangent over the logistic function is that it has a broader output spectrum and ranges in the open interval (-1, 1), which can improve the convergence of the backpropagation algorithm.

# Hyperbolic Activation Functions

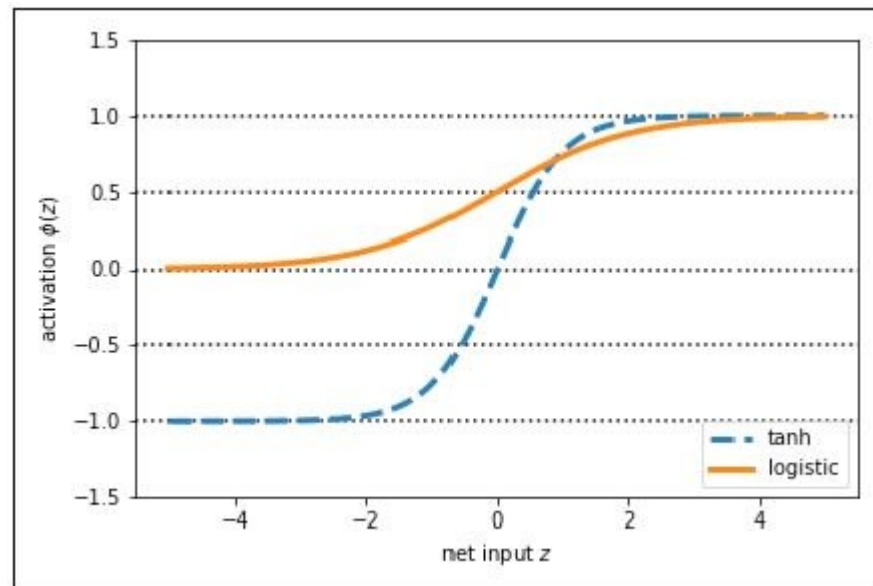


# Hyperbolic Tangent

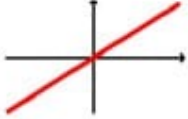

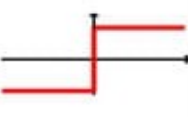


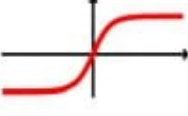

```
>>> def tanh(z):  
...     e_p = np.exp(z)  
...     e_m = np.exp(-z)  
...     return (e_p - e_m) / (e_p + e_m)  
  
>>> z = np.arange(-5, 5, 0.005)  
>>> log_act = logistic(z)  
>>> tanh_act = tanh(z)
```

# Hyperbolic Tangent

- This code implements the tanh formula. Then it calls both logistic and tanh functions on the  $z$  value.
- The tanh function has two times larger output space than the logistic function.



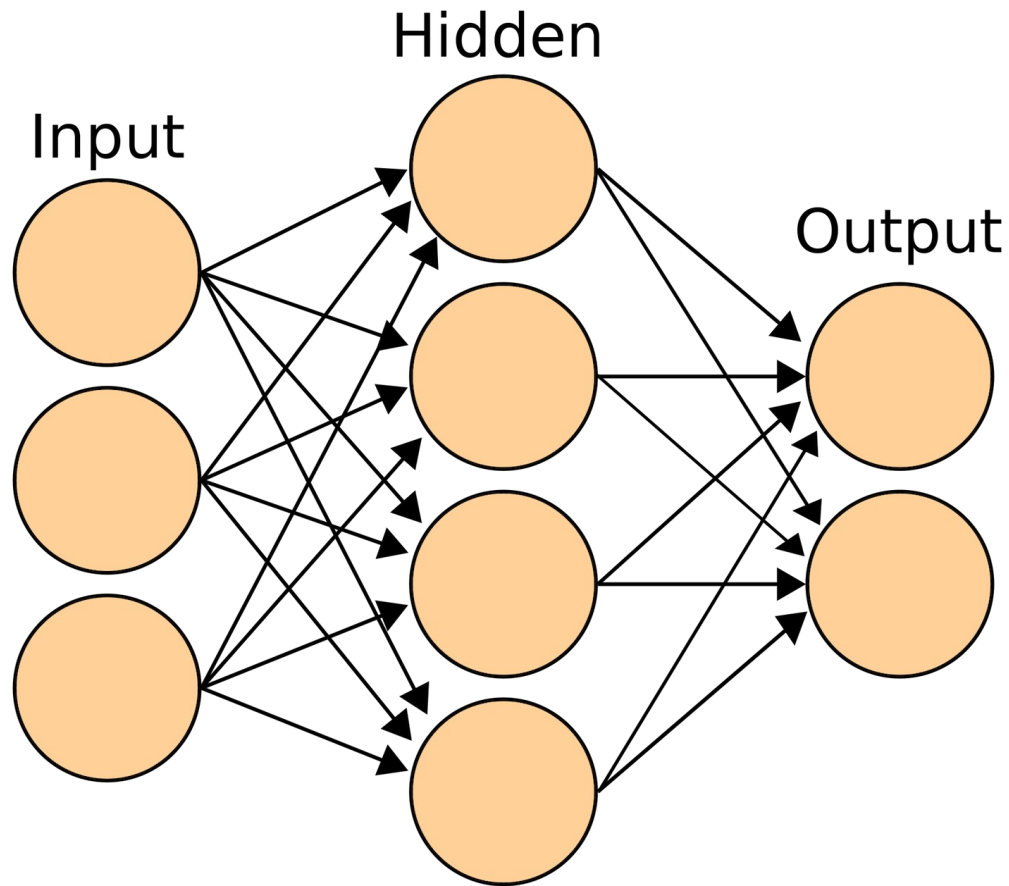
# Summary

Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -1/2 \\ z + 1/2 & -1/2 \leq z \leq 1/2 \\ 1 & z \geq 1/2 \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

# Artificial Neural Network

- A multilayer perceptrons, is commonly known as artificial neural networks.
- A single layer perceptron can solve simple problems where data is linearly separable in to 'n' dimensions, where 'n' is the number of features in the dataset. However, in case of non-linearly separable data, the accuracy of single layer perceptron decreases significantly.
- Multilayer perceptrons, on the other hand, can work efficiently with non-linearly separable data.
- Artificial neural networks, are a combination of multiple neurons connected in the form a network. It has an input layer, one or more hidden layers, and an output layer.

# Artificial Neural Network





# Phases of ANN

- Feed-forward
- Back-propagation

# Feed-forward

- The values received in the input layer are multiplied with the weights. A bias is added to the summation of the inputs and weights in order to avoid null values.
- Each neuron in the first hidden layer receives different values from the input layer depending upon the weights and bias. Neurons have an activation function that operates upon the value received from the input layer. The activation function can be of many types, like a step function, **sigmoid** function, **relu** function, or **tanh** function. As a rule of thumb, relu function is used in the hidden layer neurons and sigmoid function is used for the output layer neuron.
- The outputs from the first hidden layer neurons are multiplied with the weights of the second hidden layer; the results are summed together and passed to the neurons of the proceeding layers. This process continues until the outer layer is reached. The values calculated at the outer layer are the actual outputs of the algorithm.

# Feed-forward

- The feed-forward phase consists of these three steps.
- However, the predicted output is not necessarily correct right away; it can be wrong, and we need to correct it.
- The purpose of a learning algorithm is to make predictions that are as accurate as possible. To improve these predicted results, a neural network will then go through a back propagation phase.
- During back propagation, the weights of different neurons are updated in a way that the difference between the desired and predicted output is as small as possible.

# Back-propagation Step-1

- The error is calculated by quantifying the difference between the predicted output and the desired output.
- This difference is called "loss" and the function used to calculate the difference is called the "loss function".
- Loss functions can be of different types e.g. mean squared error or cross entropy functions.
- Remember, neural networks are supervised learning algorithms that need the desired outputs for a given set of inputs, which is what allows it to learn from the data.

# Back-propagation Step-2

- Once the error is calculated, the next step is to minimize that error.
- To do so, partial derivative of the error function is calculated with respect to all the weights and biases. This is called gradient decent.
- The derivatives can be used to find the slope of the error function.
- If the slop is positive, the value of the weights can be reduced or if the slop is negative the value of weight can be increased. This reduces the overall error.
- The function that is used to reduce this error is called the optimization function.

# Back-propagation

- This one cycle of feed-forward and back propagation is called one "epoch".
- This process continues until a reasonable accuracy is achieved.
- There is no standard for reasonable accuracy, ideally you'd strive for 100% accuracy, but this is extremely difficult to achieve for any non-trivial dataset.
- In many cases 90%+ accuracy is considered acceptable, but it really depends on your use-case.

# Let's Start with an example



## Perfect Roommate



Apple pie



Burger



Chicken

*Reference: Friendly introduction to RNN by Luis Serrano*

# Conditional Outputs

## Weather





## Neural Network



# Let's do some maths

## Vectors



$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

# Now in NN

## Neural Network



# Conceptualizing

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ ☀️ }$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ 🥧 }$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ ☁️⚡️ } = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ 🍔 }$$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ ☁️⚡️ }$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ ☀️ }$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ 🥧 }$$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ ☁️⚡️ }$$

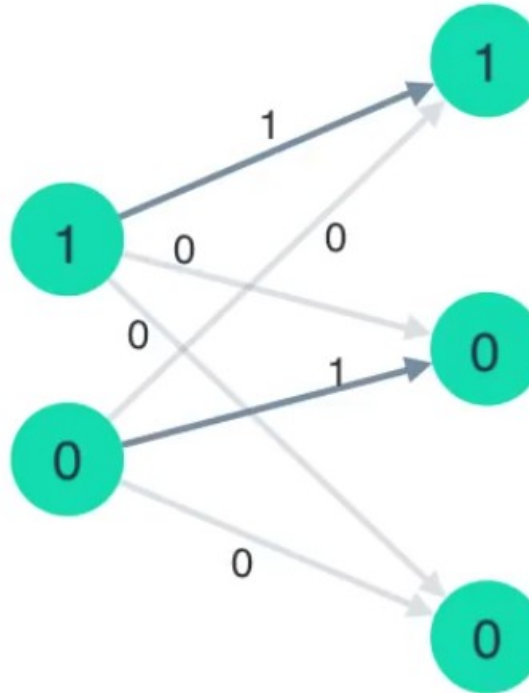
$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ 🍔 }$$

# Adding to NN

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$



$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$



$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$



# Convolutional Neural Network

- Convolutional Neural Networks are a powerful artificial neural network technique.
- These networks preserve the spatial structure of the problem and were developed for object recognition tasks such as handwritten digit recognition.
- They are popular because people are achieving state-of-the-art results on difficult computer vision and natural language processing tasks.

# Convolutional Neural Network

- Given a dataset of gray scale images with the standardized size of  $32 \times 32$  pixels each, a traditional feedforward neural network would require 1,024 input weights (plus one bias).
- This is fair enough, but the flattening of the image matrix of pixels to a long vector of pixel values loses all of the spatial structure in the image.
- Unless all of the images are perfectly resized, the neural network will have great difficulty with the problem.

# Convolutional Neural Network

- Convolutional Neural Networks expect and preserve the spatial relationship between pixels by learning internal feature representations using small squares of input data.
- Feature are learned and used across the whole image, allowing for the objects in the images to be shifted or translated in the scene and still detectable by the network.
- It is this reason why the network is so useful for object recognition in photographs, picking out digits, faces, objects and so on with varying orientation.



# Convolutional Neural Network

- In summary, below are some of the benefits of using convolutional neural networks:
  - They use fewer parameters (weights) to learn than a fully connected network.
  - They are designed to be invariant to object position and distortion in the scene.
  - They automatically learn and generalize features from the input domain.

# Convolutional Neural Network

- There are three types of layers in a Convolutional Neural Network:
  1. Convolutional Layers.
  2. Pooling Layers.
  3. Fully-Connected Layers.

# Convolutional Neural Network

- Convolutional layers are comprised of filters and feature maps.
- Filters
  - The filters are essentially the neurons of the layer. They have both weighted inputs and generate an output value like a neuron.
  - The input size is a fixed square called a patch or a receptive field. If the convolutional layer is an input layer, then the input patch will be pixel values.
  - If they deeper in the network architecture, then the convolutional layer will take input from a feature map from the previous layer.

# Convolutional Neural Network

- Feature Maps
  - The feature map is the output of one filter applied to the previous layer. A given filter is drawn across the entire previous layer, moved one pixel at a time.
  - Each position results in an activation of the neuron and the output is collected in the feature map.
  - You can see that if the receptive field is moved one pixel from activation to activation, then the field will overlap with the previous activation by (field width - 1) input values.

# Convolutional Neural Network

- Feature Maps
  - The distance that filter is moved across the input from the previous layer each activation is referred to as the stride.
  - If the size of the previous layer is not cleanly divisible by the size of the filters receptive field and the size of the stride then it is possible for the receptive field to attempt to read off the edge of the input feature map.
  - In this case, techniques like zero padding can be used to invent mock inputs with zero values for the receptive field to read.

# Convolutional Neural Network

- Pooling Layers
  - The pooling layers down-sample the previous layers feature map.
  - Pooling layers follow a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous layers feature map.
  - As such, pooling may be consider a technique to compress or generalize feature representations and generally reduce the overfitting of the training data by the model.

# Convolutional Neural Network

- Pooling Layers
  - They too have a receptive field, often much smaller than the convolutional layer.
  - Also, the stride or number of inputs that the receptive field is moved for each activation is often equal to the size of the receptive field to avoid any overlap.
  - Pooling layers are often very simple, taking the average or the maximum of the input value in order to create its own feature map.

# Convolutional Neural Network

- Fully Connected Layers
  - Fully connected layers are the normal flat feedforward neural network layer.
  - These layers may have a nonlinear activation function or a softmax activation in order to output probabilities of class predictions.
  - Fully connected layers are used at the end of the network after feature extraction and consolidation has been performed by the convolutional and pooling layers.
  - They are used to create final nonlinear combinations of features and for making predictions by the network.



# Convolutional Neural Network

- Worked Example
  - You now know about convolutional, pooling and fully connected layers.
  - Let's make this more concrete by working through how these three layers may be connected together.

# Convolutional Neural Network

- Worked Example
- Image Input Data
  - Let's assume we have a dataset of gray scale images. Each image has the same size of 32 pixels wide and 32 pixels high, and pixel values are between 0 and 255, e.g. a matrix of  $32 \times 32 \times 1$  or 1,024 pixel values.
  - Image input data is expressed as a 3-dimensional matrix of width  $\times$  height  $\times$  channels.
  - If we were using color images in our example, we would have 3 channels for the red, green and blue pixel values, e.g.  $32 \times 32 \times 3$ .

# Convolutional Neural Network

- Worked Example
- Convolutional Layer
  - We define a convolutional layer with 10 filters and a receptive field 5 pixels wide and 5 pixels high and a stride length of 1.
  - Because each filter can only get input from (i.e. see)  $5 \times 5$  (25) pixels at a time, we can calculate that each will require  $25 + 1$  input weights (plus 1 for the bias input).
  - Dragging the  $5 \times 5$  receptive field across the input image data with a stride width of 1 will result in a feature map of  $28 \times 28$  output values or 784 distinct activations per image.

# Convolutional Neural Network

- Worked Example
- We have 10 filters, so that is 10 different  $28 \times 28$  feature maps or 7,840 outputs that will be created for one image.
- Finally, we know we have 26 inputs per filter, 10 filters and  $28 \times 28$  output values to calculate per filter, therefore we have a total of  $26 \times 10 \times 28 \times 28$  or 203,840 connections in our convolutional layer, we want to phrase it using traditional neural network nomenclature.
- Convolutional layers also make use of a nonlinear transfer function as part of activation and the rectifier activation function is the popular default to use.

# Convolutional Neural Network

- Pool Layer
  - We define a pooling layer with a receptive field with a width of 2 inputs and a height of 2 inputs.
  - We also use a stride of 2 to ensure that there is no overlap. This results in feature maps that are one half the size of the input feature maps.
  - From 10 different  $28 \times 28$  feature maps as input to 10 different  $14 \times 14$  feature maps as output.
  - We will use a `max()` operation for each receptive field so that the activation is the maximum input value.

# Convolutional Neural Network

- Fully Connected Layer
  - Finally, we can flatten out the square feature maps into a traditional flat fully connected layer.
  - We can define the fully connected layer with 200 hidden neurons, each with  $10 \times 14 \times 14$  input connections, or 1,960 + 1 weights per neuron.
  - That is a total of 392,200 connections and weights to learn in this layer.
  - We can use a sigmoid or softmax transfer function to output probabilities of class values directly.

# CNN : Best Practices

- Now that we know about the building blocks for a convolutional neural network and how the layers hang together, we can review some best practices to consider when applying them.
  - Input Receptive Field Dimensions: The default is 2D for images, but could be 1D such as for words in a sentence or 3D for video that adds a time dimension.
  - Receptive Field Size: The patch should be as small as possible, but large enough to see features in the input data. It is common to use  $3 \times 3$  on small images and  $5 \times 5$  or  $7 \times 7$  and more on larger image sizes.

# CNN : Best Practices

- Stride Width: Use the default stride of 1. It is easy to understand and you don't need padding to handle the receptive field falling off the edge of your images. This could be increased to 2 or larger for larger images.
- Number of Filters: Filters are the feature detectors. Generally fewer filters are used at the input layer and increasingly more filters used at deeper layers.
- Padding: Set to zero and called zero padding when reading non-input data. This is useful when you cannot or do not want to standardize input image sizes or when you want to use receptive field and stride sizes that do not neatly divide up the input image size.



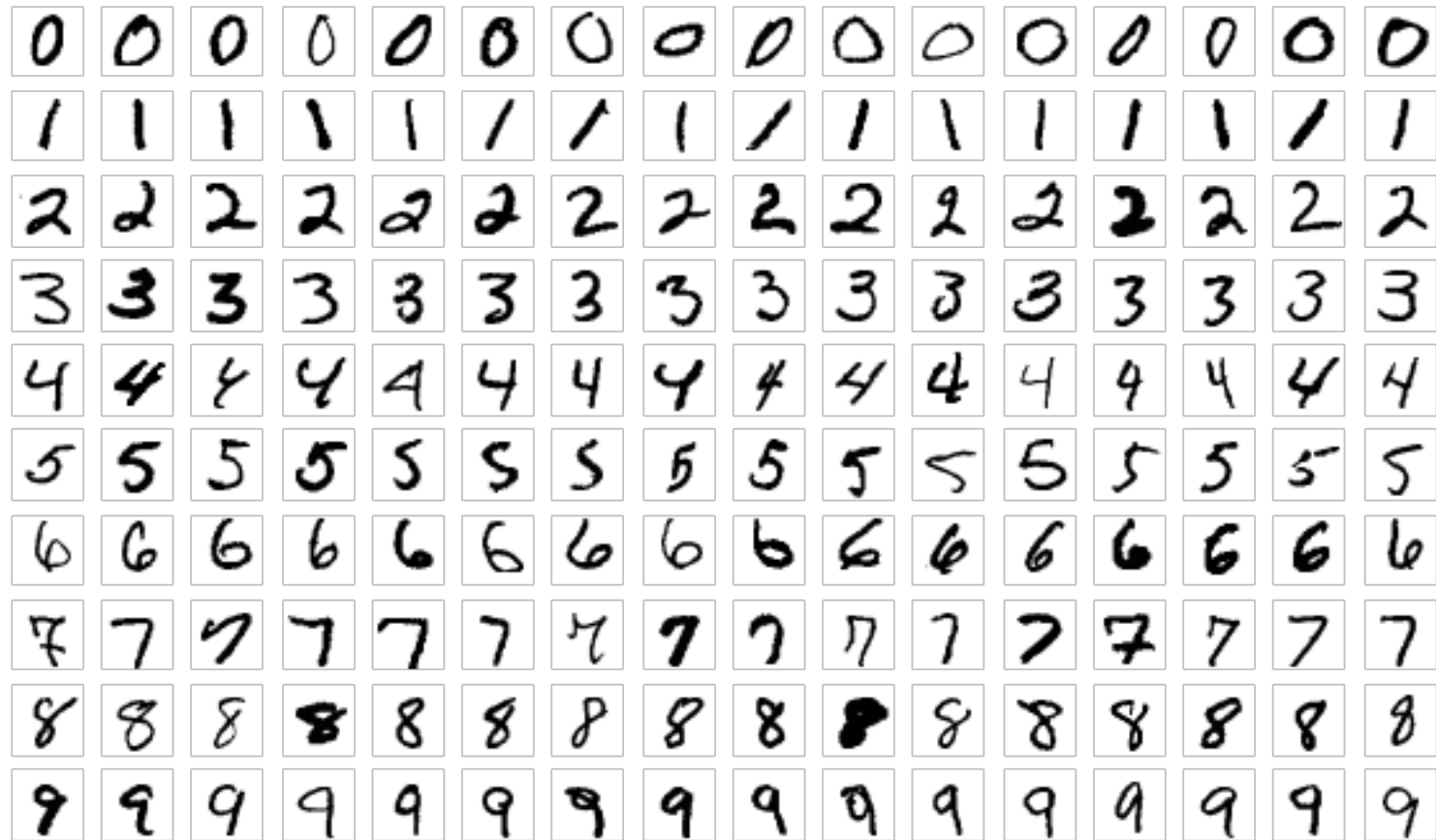
# CNN : Best Practices

- Pooling: Pooling is a destructive or generalization process to reduce overfitting. Receptive field size is almost always set to  $2 \times 2$  with a stride of 2 to discard 75% of the activations from the output of the previous layer.
- Data Preparation: Consider standardizing input data, both the dimensions of the images and pixel values.

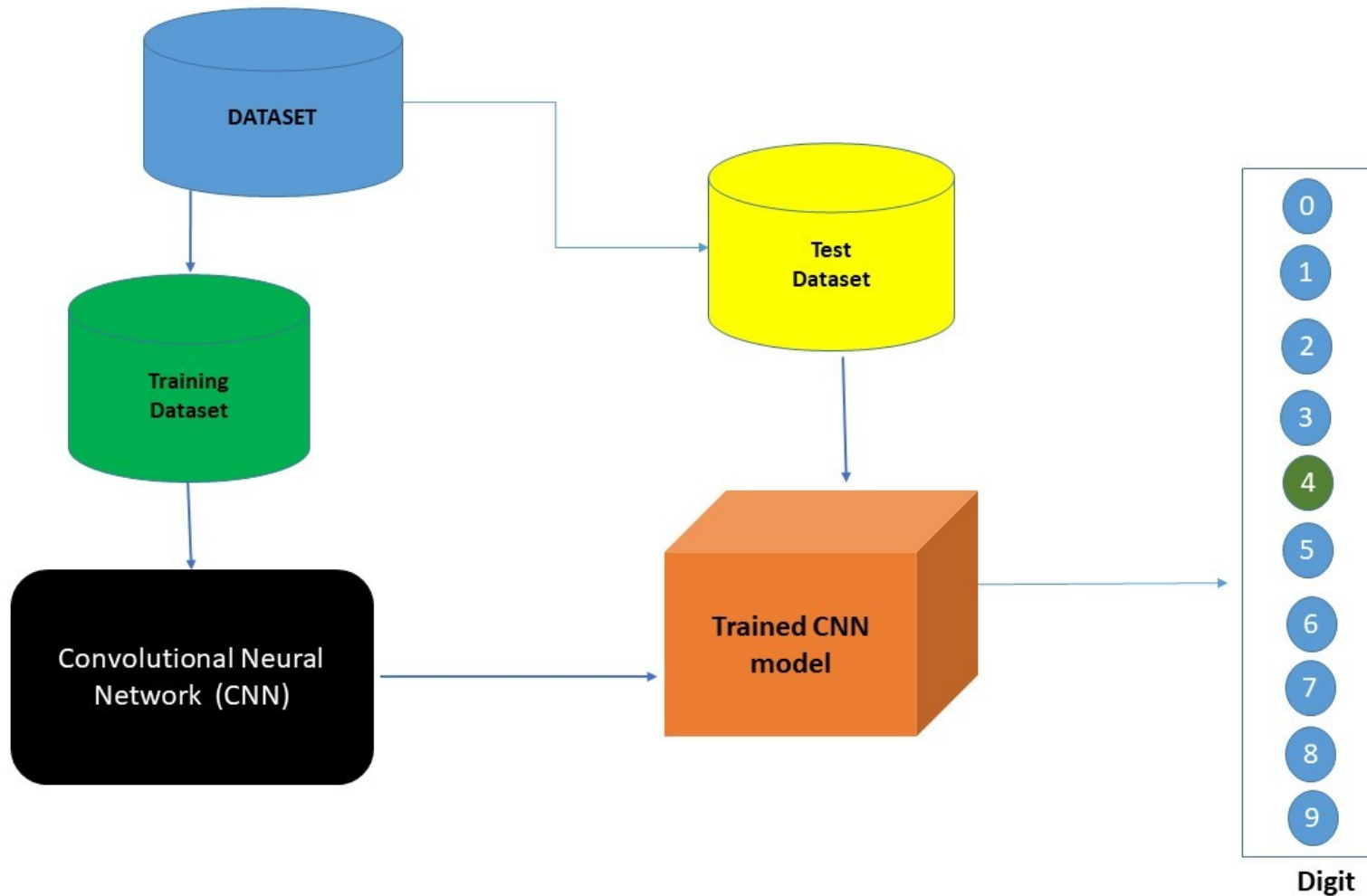
# CNN : Best Practices

- Pattern Architecture: It is common to pattern the layers in your network architecture.
- This might be one, two or some number of convolutional layers followed by a pooling layer.
- This structure can then be repeated one or more times. Finally, fully connected layers are often only used at the output end and may be stacked one, two or more deep.
- Dropout: CNNs have a habit of overfitting, even with pooling layers. Dropout should be used such as between fully connected layers and perhaps after pooling layers.

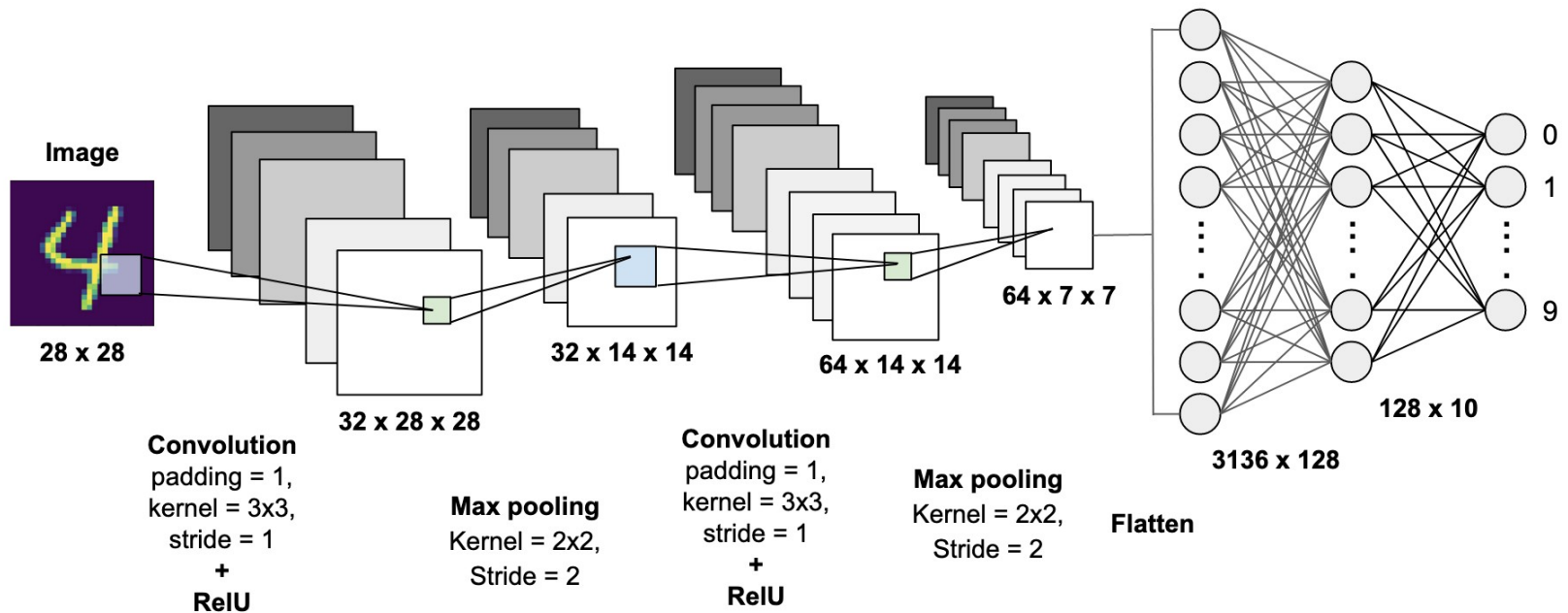
# Example: MNIST Handwritten Chars



# Building CNN

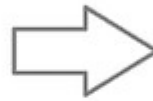


# CNN Architecture



# Why ConvNet of FF Nets?

1	1	0
4	2	1
0	2	1

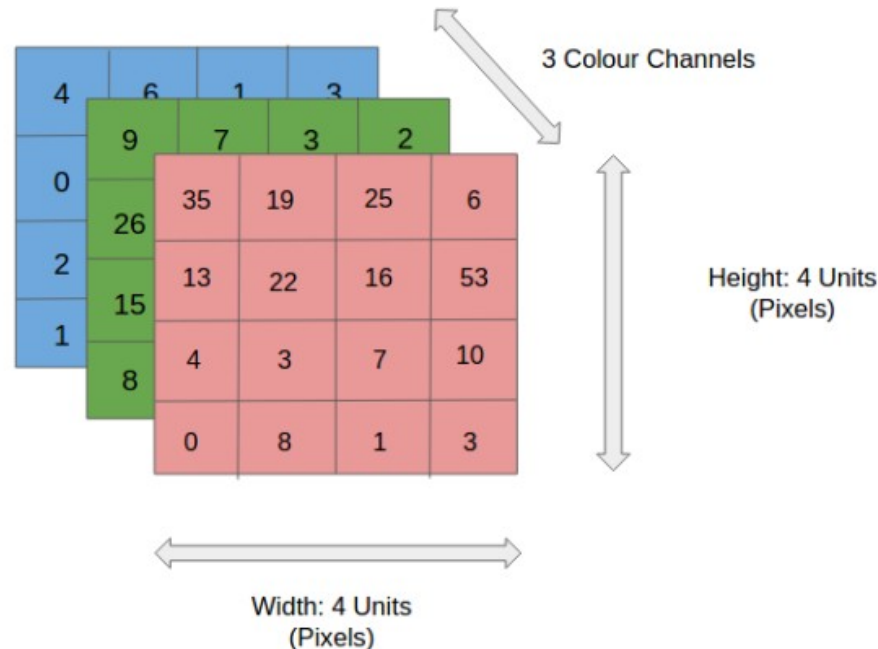


1
1
0
4
2
1
0
2
1

# Why ConvNet over FF NN ?

- A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters.
- The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights.
- In other words, the network can be trained to understand the sophistication of the image better.

# Input image formatting



The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.



# Convolution Layer

- Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)
- The green section resembles our 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix.
- Kernel/Filter, K =

1 0 1

0 1 0

1 0 1

# Convolution Layer

1	1	1	0	0
0	1	1	1	0
0	0 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	1
0	0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	0
0	1 <sub>x1</sub>	1 <sub>x0</sub>	0 <sub>x1</sub>	0

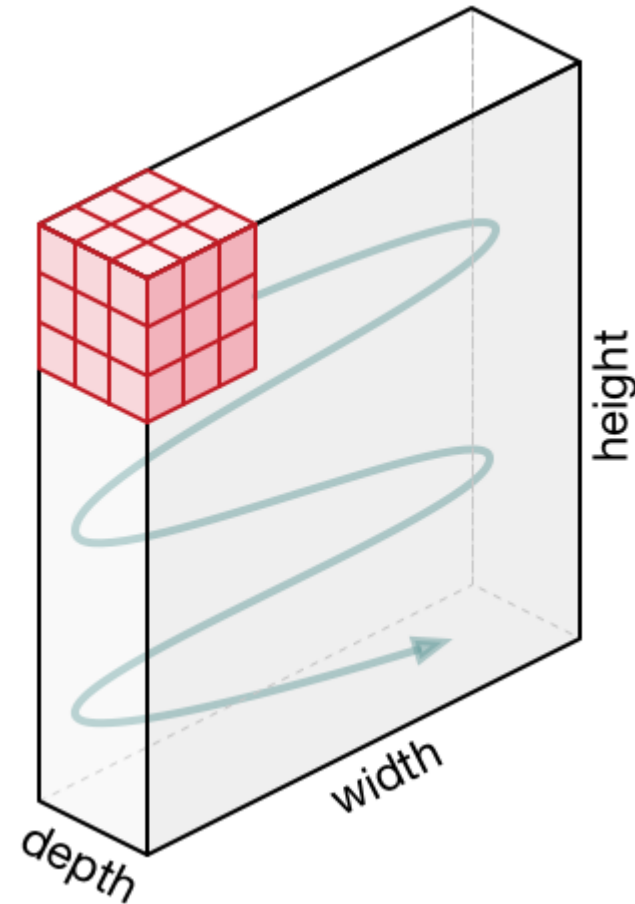
Image

4	3	4
2	4	3
2	3	

Convolved  
Feature

# Movement of the Kernel

- The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.
- The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.



# Movement of the Kernel

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



161

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-9

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



659

+

+

+ 1 = 812

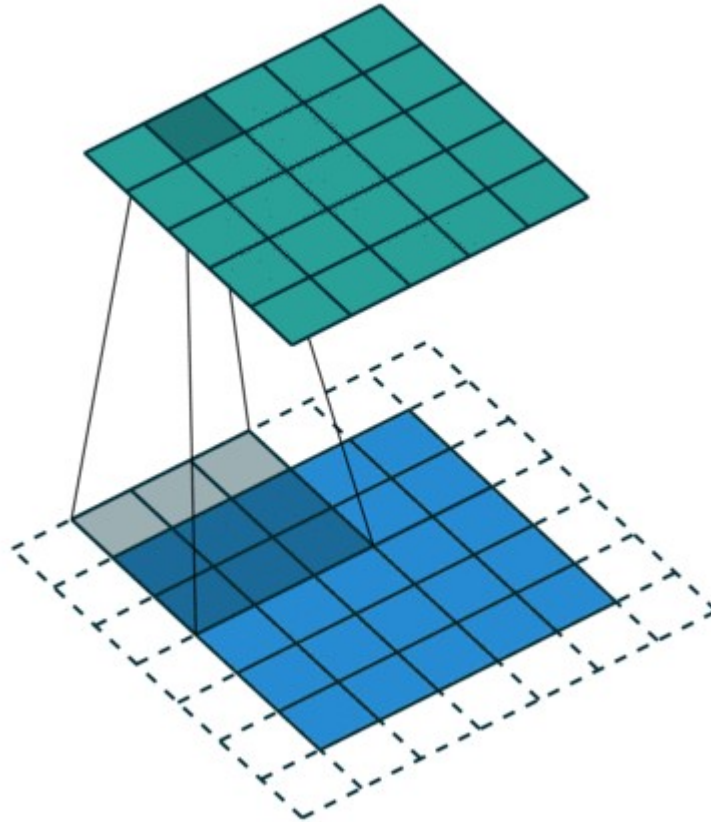


Bias = 1

Output

-25	466	466	475	...
295	787	798	812	...
				...
				...
...	...	...	...	...

# Same Padding



# Pooling Layer

- The Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction.
- It is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.
- There are two types of Pooling: Max Pooling and Average Pooling.
  - Max Pooling returns the maximum value from the portion of the image covered by the Kernel.
  - Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

# Pooling Layer

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

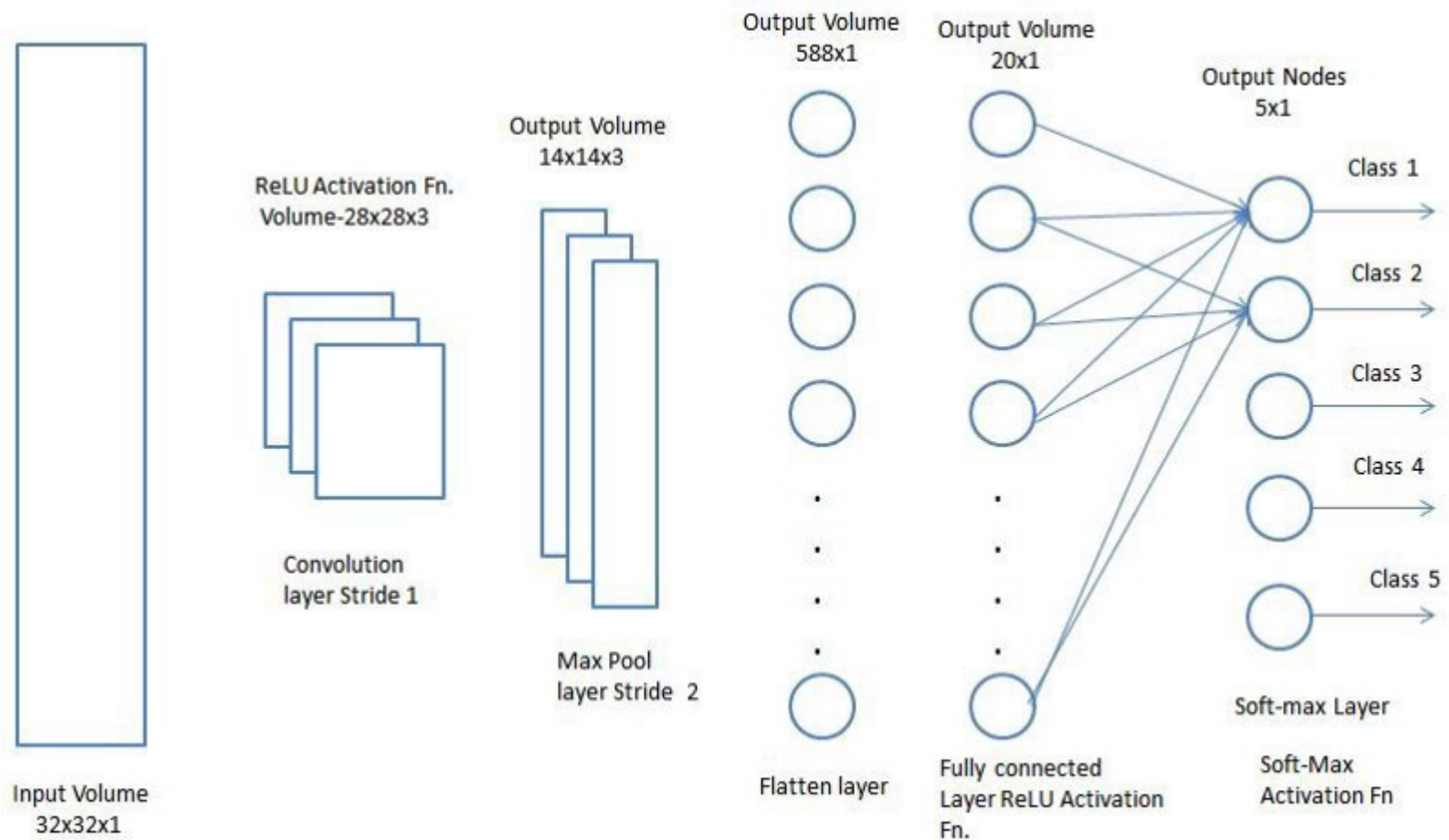
max pooling

20	30
112	37

average pooling

13	8
79	20

# Classification

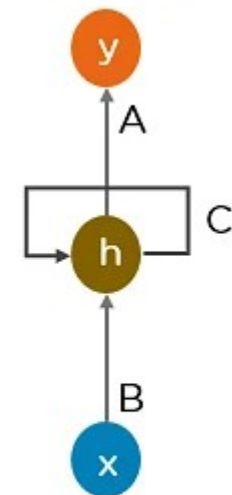
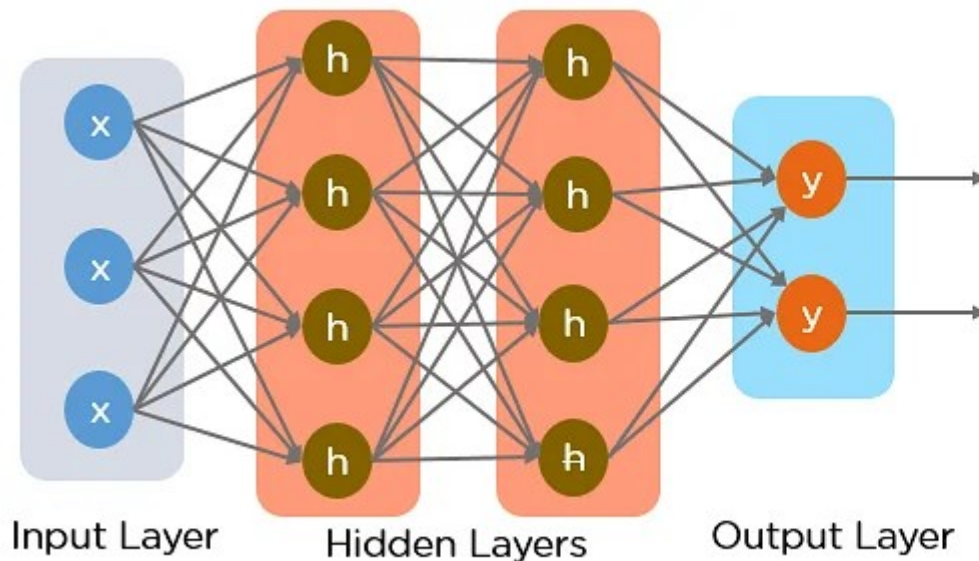




# Recurrent Neural Network

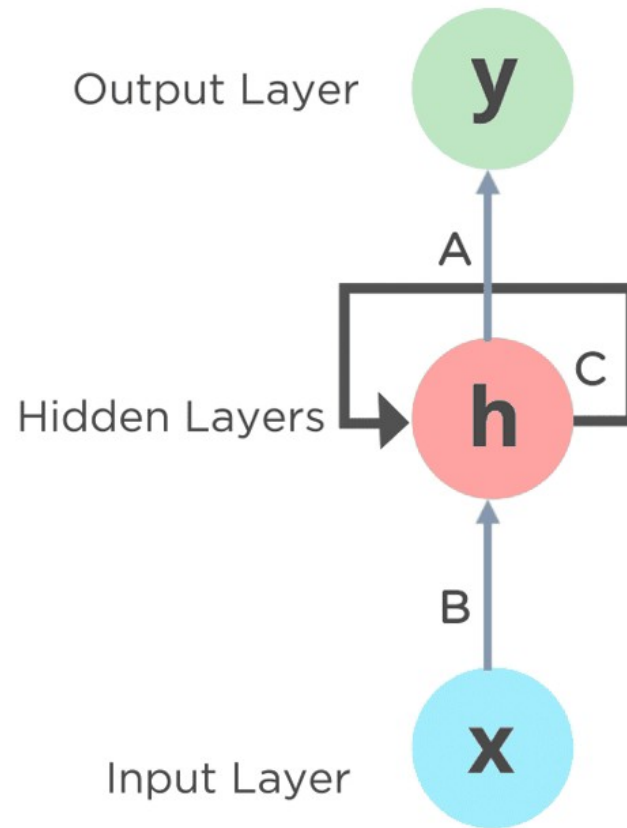
- RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.
- Below is how you can convert a Feed-Forward Neural Network into a Recurrent Neural Network:

# Recurrent Neural Network



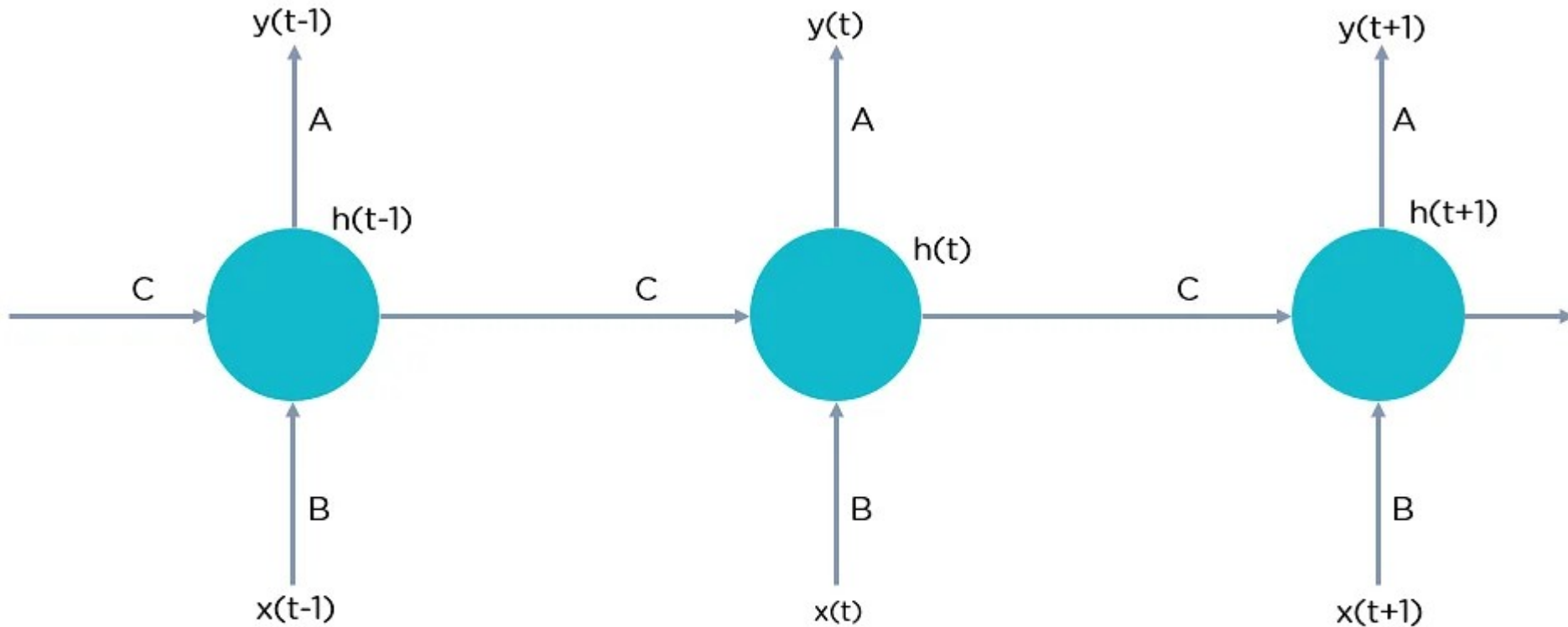
Recurrent Neural Network

# Recurrent Neural Network



A, B and C are the parameters

# Fully Connected RNN



$$h(t) = f_c(h(t-1), x(t))$$

$h(t)$  = new state

$f_c$  = function with parameter  $c$

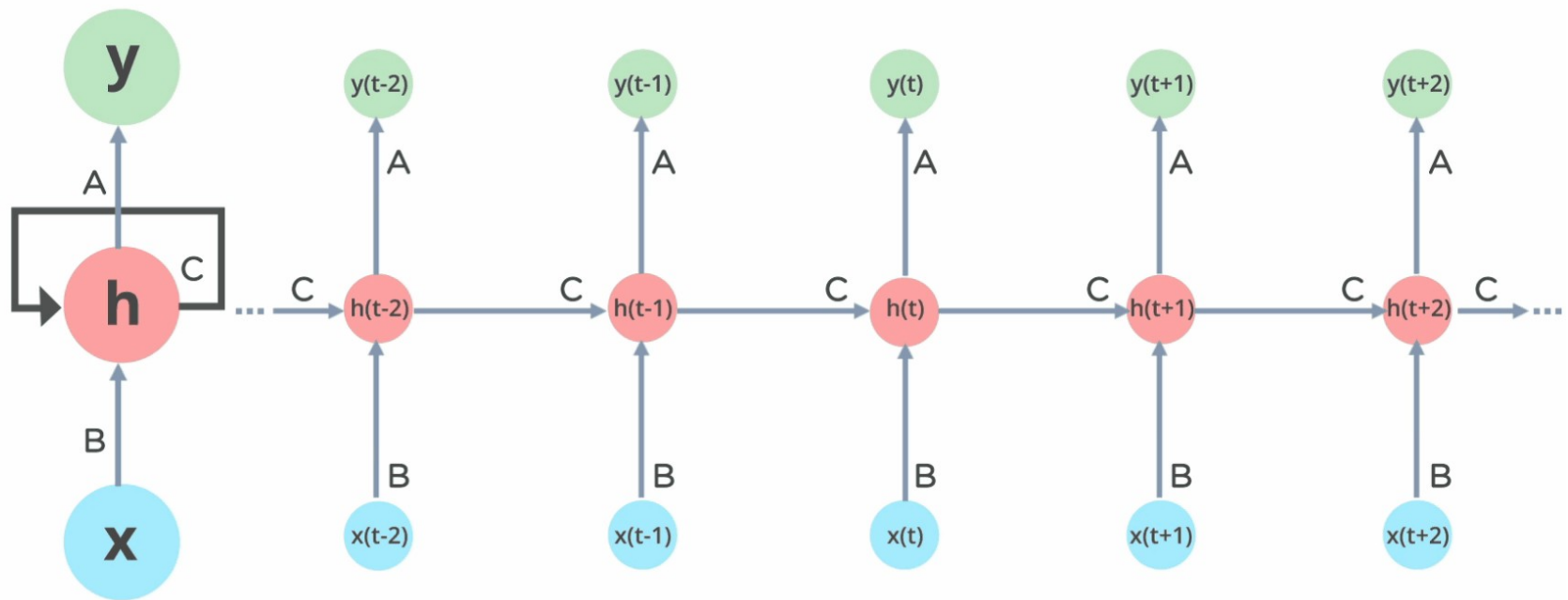
$h(t-1)$  = old state

$x(t)$  = input vector at time step  $t$

# Why RNN?

- RNN were created because there were a few issues in the feed-forward neural network:
  - Cannot handle sequential data
  - Considers only the current input
  - Cannot memorize previous inputs
- The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously received inputs.
- RNNs can memorize previous inputs due to their internal memory.

# Working of RNN

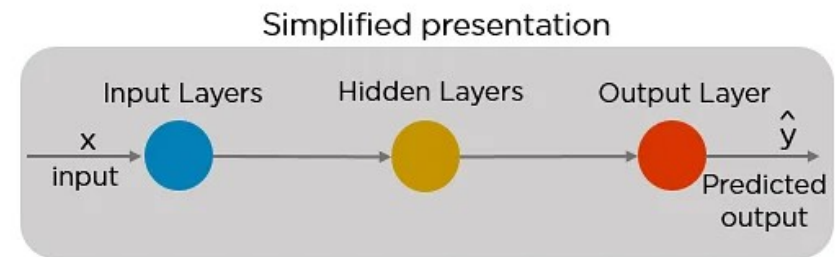
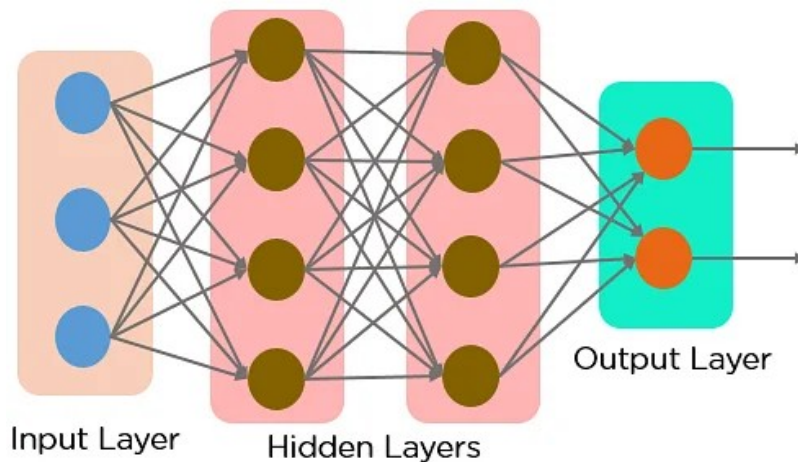


# Working of RNN

- The input layer 'x' takes in the input to the neural network and processes it and passes it onto the middle layer.
- The middle layer 'h' can consist of multiple hidden layers, each with its own activation functions and weights and biases. If you have a neural network where the various parameters of different hidden layers are not affected by the previous layer, ie: the neural network does not have memory, then you can use a recurrent neural network.
- The Recurrent Neural Network will standardize the different activation functions and weights and biases so that each hidden layer has the same parameters. Then, instead of creating multiple hidden layers, it will create one and loop over it as many times as required.

# FFNN vs. RNN

- A feed-forward neural network allows information to flow only in the forward direction, from the input nodes, through the hidden layers, and to the output nodes. There are no cycles or loops in the network.
- Below is how a simplified presentation of a feed-forward neural network looks like:



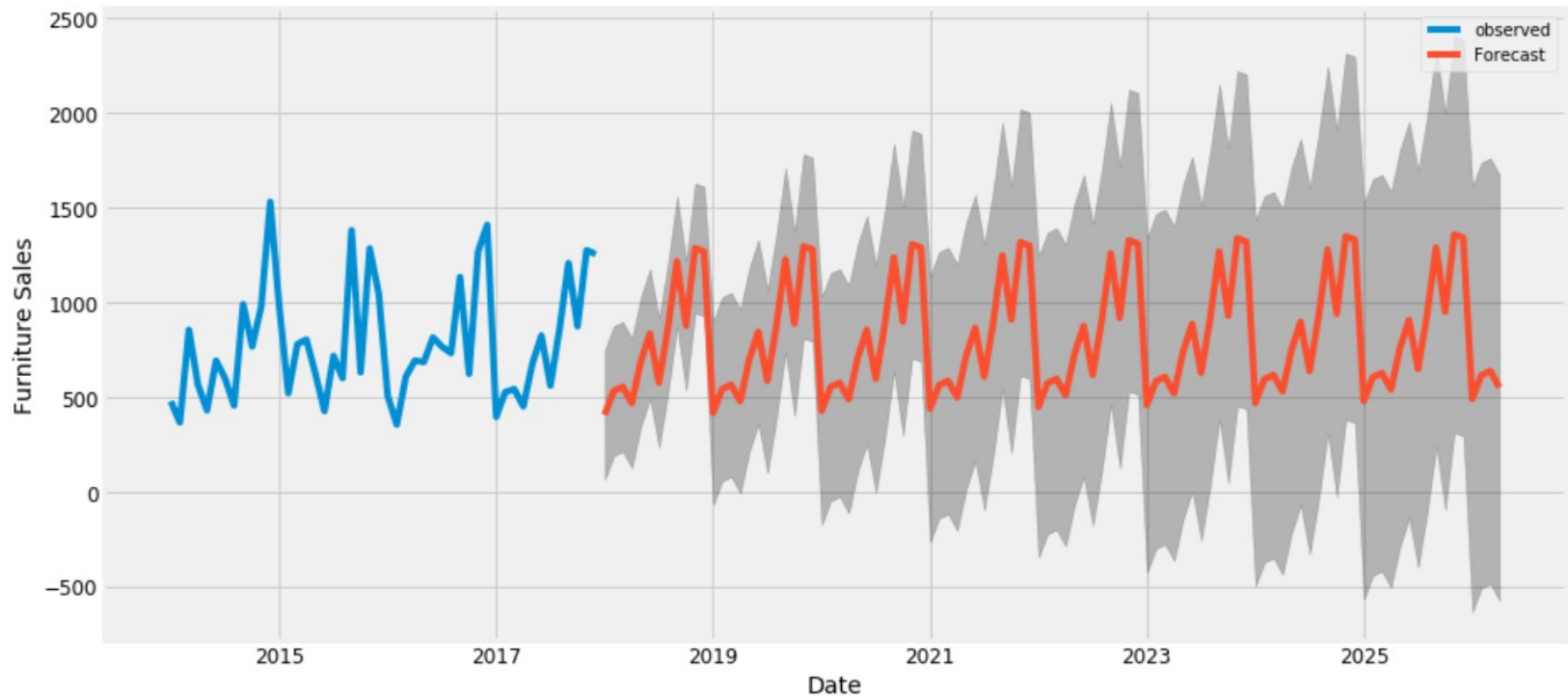


# Application of RNN



"A Dog catching a ball in mid air"

# Application of RNN



# Application of RNN



Natural Language Processing

When it rains, look for rainbows.  
When it's dark, look for stars.

Positive Sentiment

# Application of RNN



Machine Translation

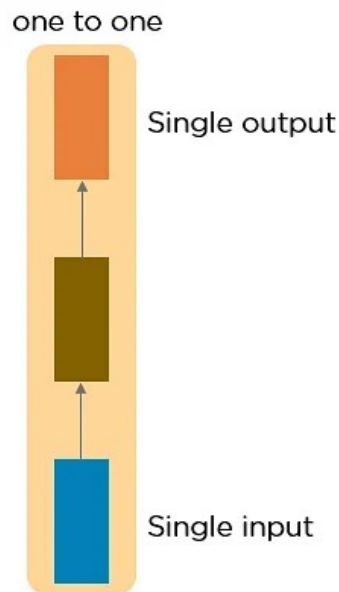
Here the person is speaking in English and it is getting translated into Chinese, Italian, French, German and Spanish languages

# Types of RNN

- There are four types of Recurrent Neural Networks:
  - One to One
  - One to Many
  - Many to One
  - Many to Many

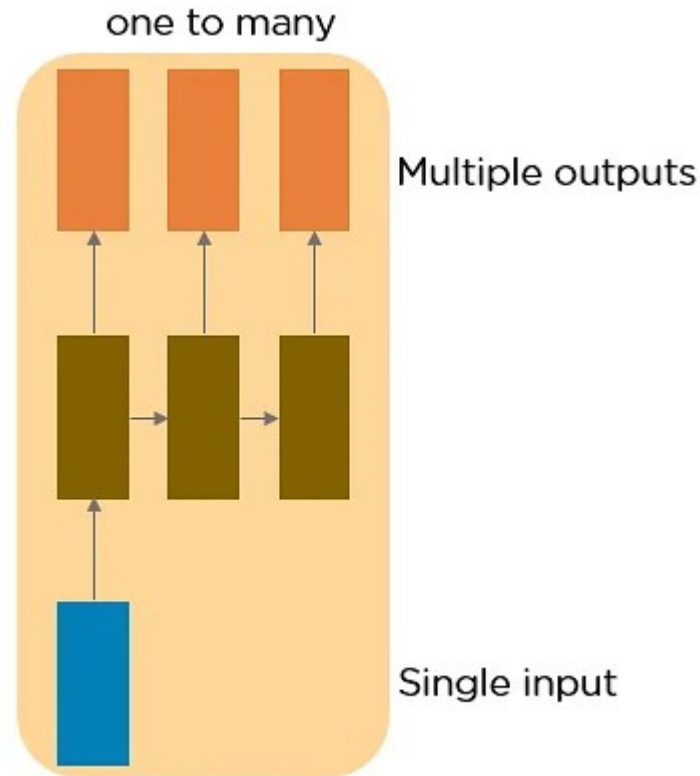
# Types of RNN

- One to One RNN
  - This type of neural network is known as the Vanilla Neural Network. It's used for general machine learning problems, which has a single input and a single output.



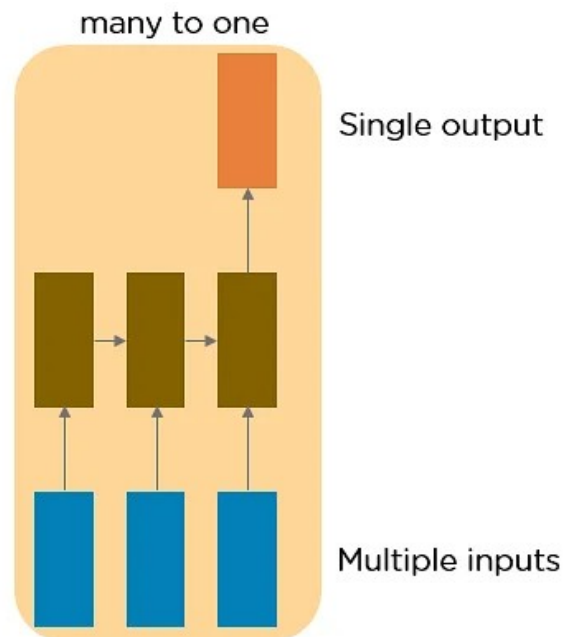
# Types of RNN

- This type of neural network has a single input and multiple outputs. An example of this is the image caption.



# Types of RNN

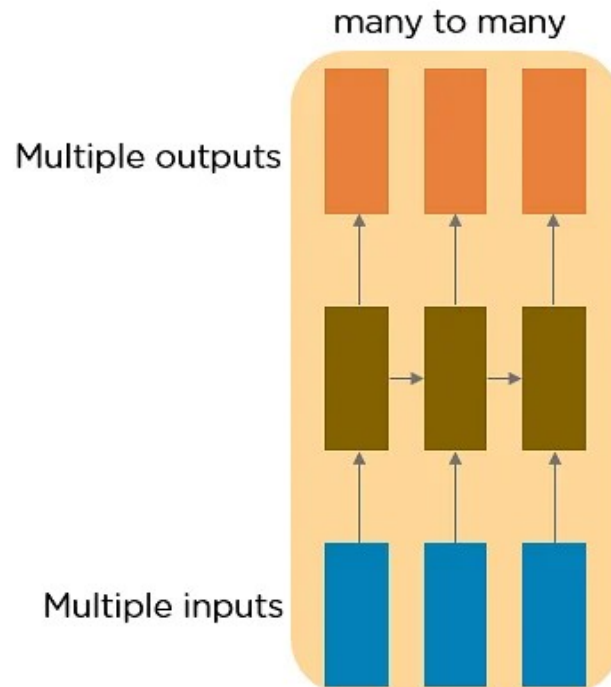
- Many to One RNN
  - This RNN takes a sequence of inputs and generates a single output. Sentiment analysis is a good example of this kind of network where a given sentence can be classified as expressing positive or negative sentiments.





# Types of RNN

- Many to Many RNN
  - This RNN takes a sequence of inputs and generates a sequence of outputs. Machine translation is one of the examples.



# Two Issues of Standard RNNs

- The word you predict will depend on the previous few words in context. Here, you need the context of Spain to predict the last word in the text, and the most suitable answer to this sentence is “Spanish.”
- The gap between the relevant information and the point where it's needed may have become very large. LSTMs help you solve this problem.

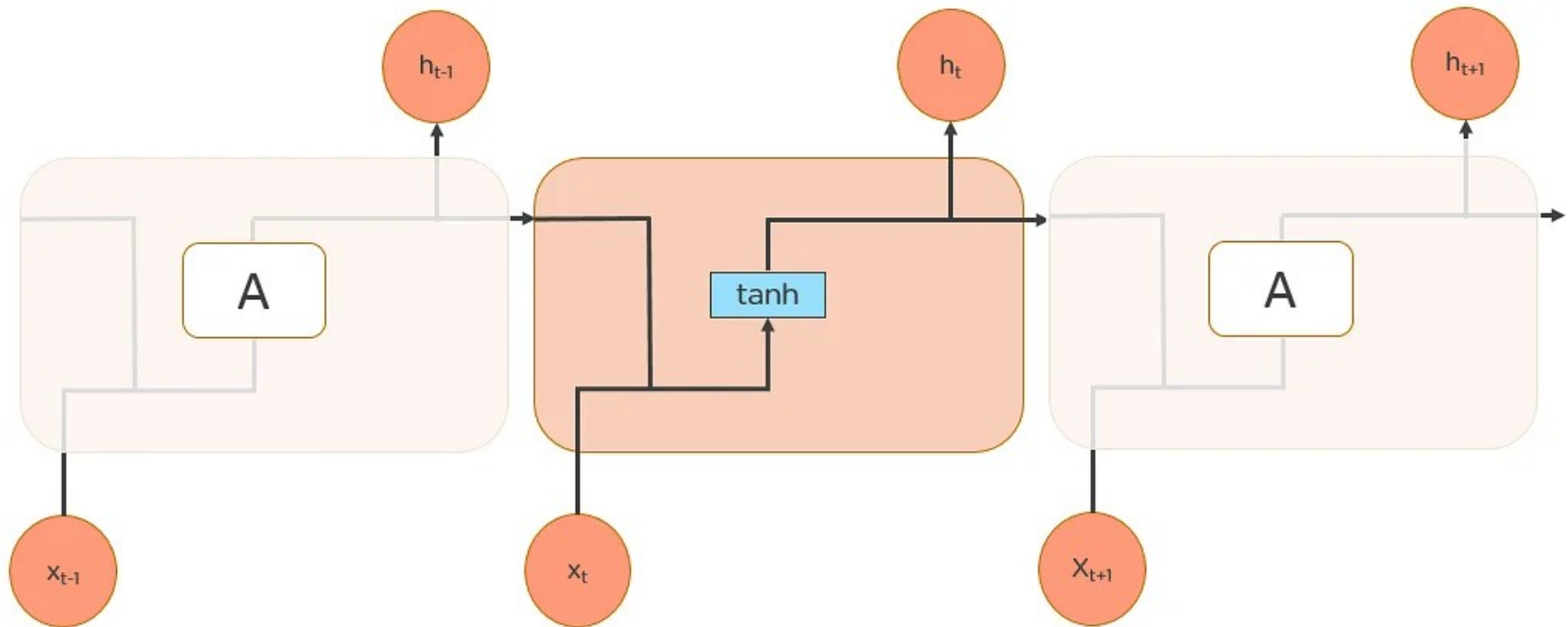
# Backpropagation Through Time

- Backpropagation through time is when we apply a Backpropagation algorithm to a Recurrent Neural network that has time series data as its input.
- In a typical RNN, one input is fed into the network at a time, and a single output is obtained. But in backpropagation, you use the current as well as the previous inputs as input. This is called a timestep and one timestep will consist of many time series data points entering the RNN simultaneously.
- Once the neural network has trained on a timeset and given you an output, that output is used to calculate and accumulate the errors. After this, the network is rolled back up and weights are recalculated and updated keeping the errors in mind.

# Long Short-Term Memory Networks

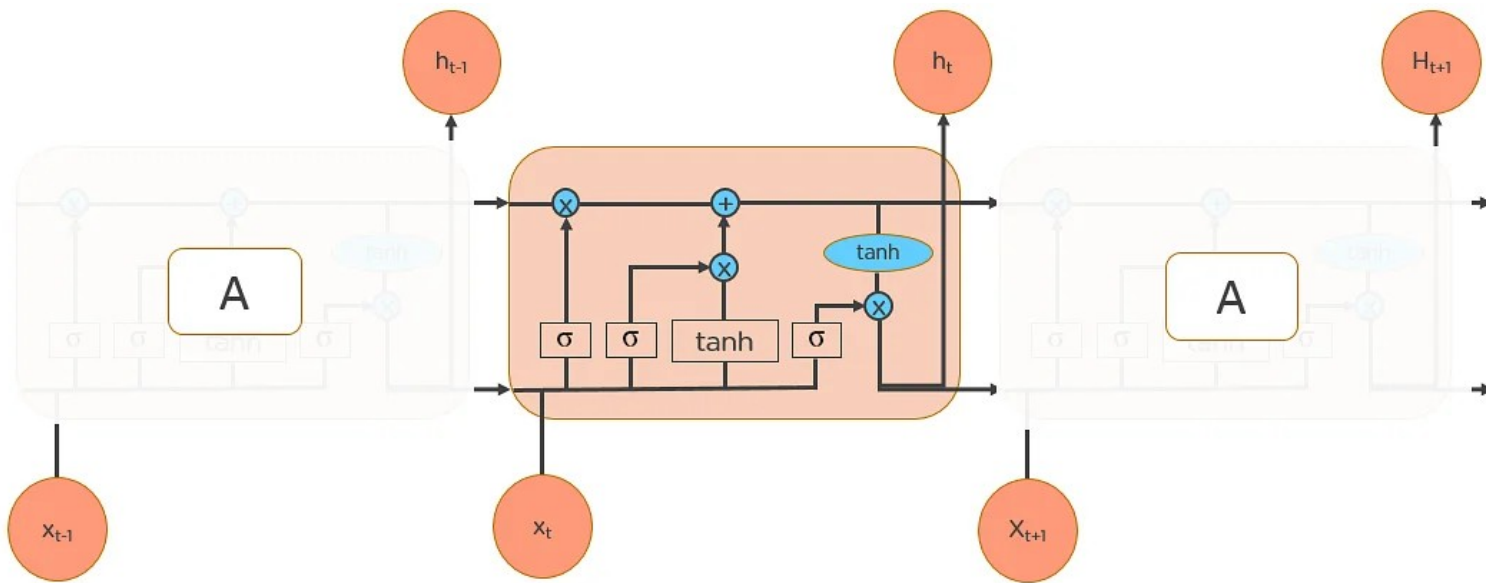
- LSTMs are a special kind of RNN — capable of learning long-term dependencies by remembering information for long periods is the default behavior.
- All RNN are in the form of a chain of repeating modules of a neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

# Long Short-Term Memory Networks

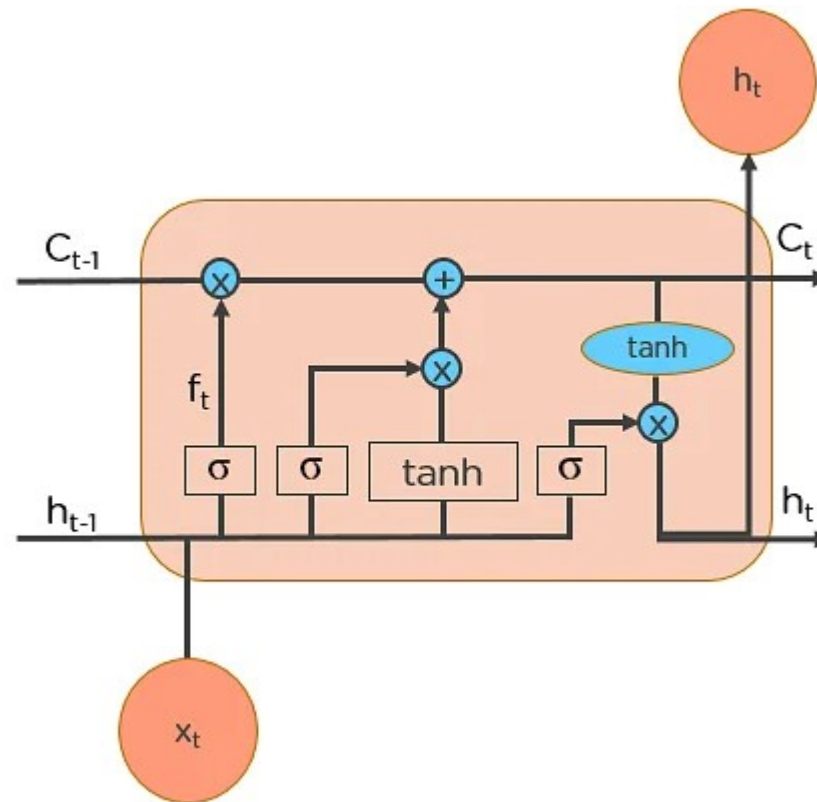


# Long Short-Term Memory Networks

LSTMs also have a chain-like structure, but the repeating module is a bit different structure. Instead of having a single neural network layer, four interacting layers are communicating extraordinarily.



# Long Short-Term Memory Networks



# Long Short-Term Memory Networks

- Step 1: Decide How Much Past Data It Should Remember
  - The first step in the LSTM is to decide which information should be omitted from the cell in that particular time step. The sigmoid function determines this. It looks at the previous state ( $h_{t-1}$ ) along with the current input  $x_t$  and computes the function.

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

$f_t$  = forget gate  
Decides which information to  
delete that is not important  
from previous time step



# Long Short-Term Memory Networks

- Consider the following two sentences:
- Let the output of  $h(t-1)$  be “Alice is good in Physics. John, on the other hand, is good at Chemistry.”
- Let the current input at  $x(t)$  be “John plays football well. He told me yesterday over the phone that he had served as the captain of his college football team.”
- The forget gate realizes there might be a change in context after encountering the first full stop. It compares with the current input sentence at  $x(t)$ .
- The next sentence talks about John, so the information on Alice is deleted. The position of the subject is vacated and assigned to John.

# Long Short-Term Memory Networks

- Step 2: Decide How Much This Unit Adds to the Current State
- In the second layer, there are two parts. One is the sigmoid function, and the other is the tanh function. In the sigmoid function, it decides which values to let through (0 or 1). tanh function gives weightage to the values which are passed, deciding their level of importance (-1 to 1).

$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$i_t$  = input gate  
Determines which information to let  
through based on its significance in  
the current time step

# Long Short-Term Memory Networks

- With the current input at  $x(t)$ , the input gate analyzes the important information — John plays football, and the fact that he was the captain of his college team is important.
- “He told me yesterday over the phone” is less important; hence it's forgotten. This process of adding some new information can be done via the input gate.

# Long Short-Term Memory Networks

- Step 3: Decide What Part of the Current Cell State Makes It to the Output
- The third step is to decide what the output will be. First, we run a sigmoid layer, which decides what parts of the cell state make it to the output. Then, we put the cell state through tanh to push the values to be between -1 and 1 and multiply it by the output of the sigmoid gate.

$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

$o_t$  = output gate  
Allows the passed in information to  
impact the output in the current  
time step

# Long Short-Term Memory Networks

- Let's consider this example to predict the next word in the sentence: "John played tremendously well against the opponent and won for his team. For his contributions, brave \_\_\_\_ was awarded player of the match."
- There could be many choices for the empty space. The current input brave is an adjective, and adjectives describe a noun. So, "John" could be the best output after brave.

# Useful resources

- [www.mitu.co.in](http://www.mitu.co.in)
- [www.pythonprogramminglanguage.com](http://www.pythonprogramminglanguage.com)
- [www.scikit-learn.org](http://www.scikit-learn.org)
- [www.towardsdatascience.com](http://www.towardsdatascience.com)
- [www.medium.com](http://www.medium.com)
- [www.analyticsvidhya.com](http://www.analyticsvidhya.com)
- [www.kaggle.com](http://www.kaggle.com)
- [www.stephacking.com](http://www.stephacking.com)
- [www.github.com](http://www.github.com)

# Thank you

*This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License*



@mitu\_skillologies



/mITuSkillologies



@mitu\_group



/company/mitu-  
skillologies



c/MITUSkillologies

## Web Resources

<https://mitu.co.in>  
<http://tusharkute.com>

[contact@mitu.co.in](mailto:contact@mitu.co.in)  
[tushar@tusharkute.com](mailto:tushar@tusharkute.com)