# Artificial Neural Network (ANN)

Artificial Intelligent

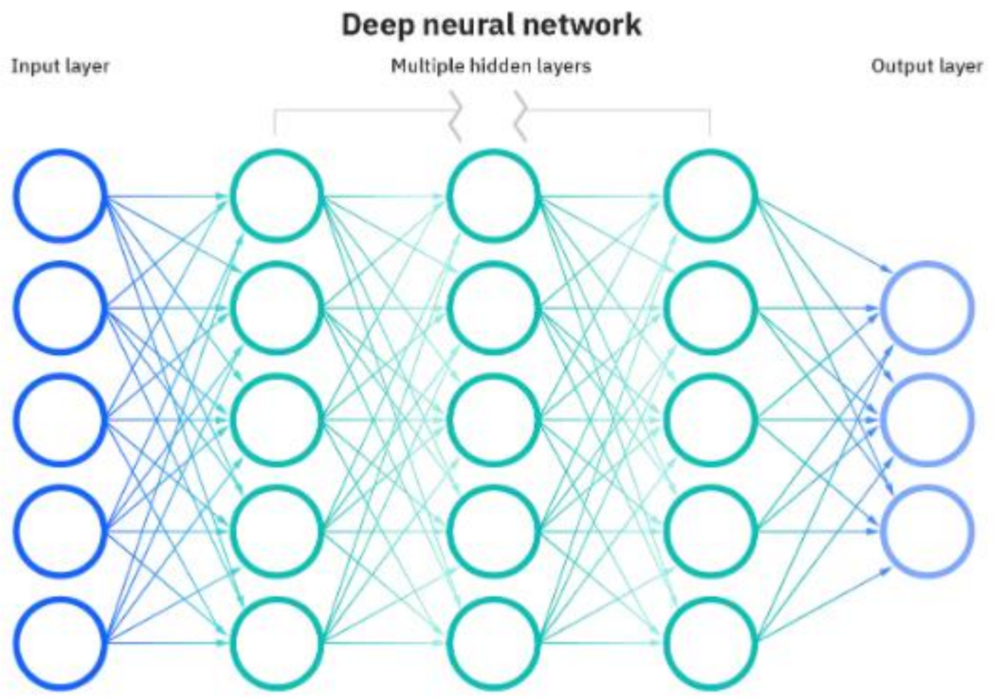
## 1. Introduction:

Neural networks reflect the behavior of the human brain, allowing computer programs to recognize patterns and solve common problems in the fields of AI, machine learning, and deep learning.

## 2. Simple Neural Network :

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

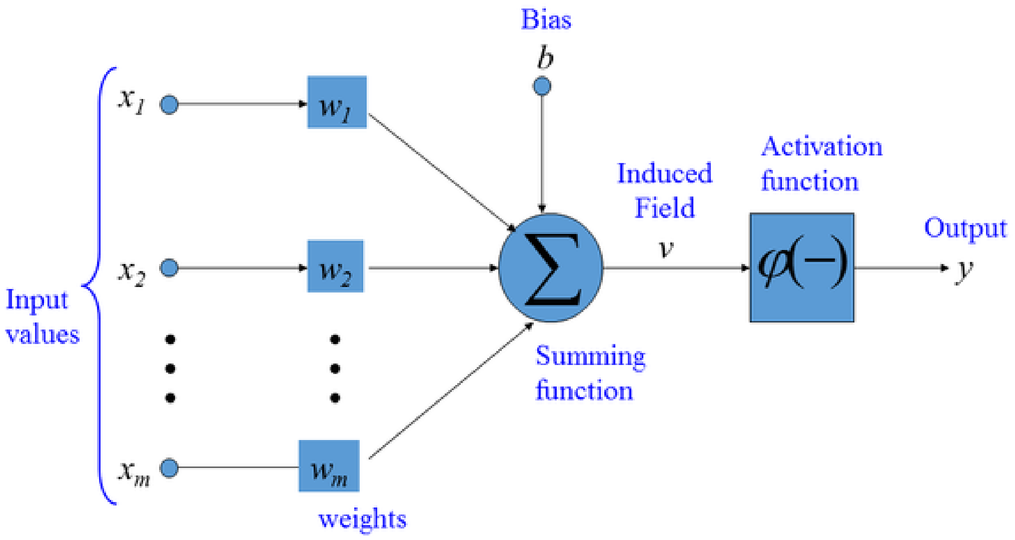


Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.

From IBM.com

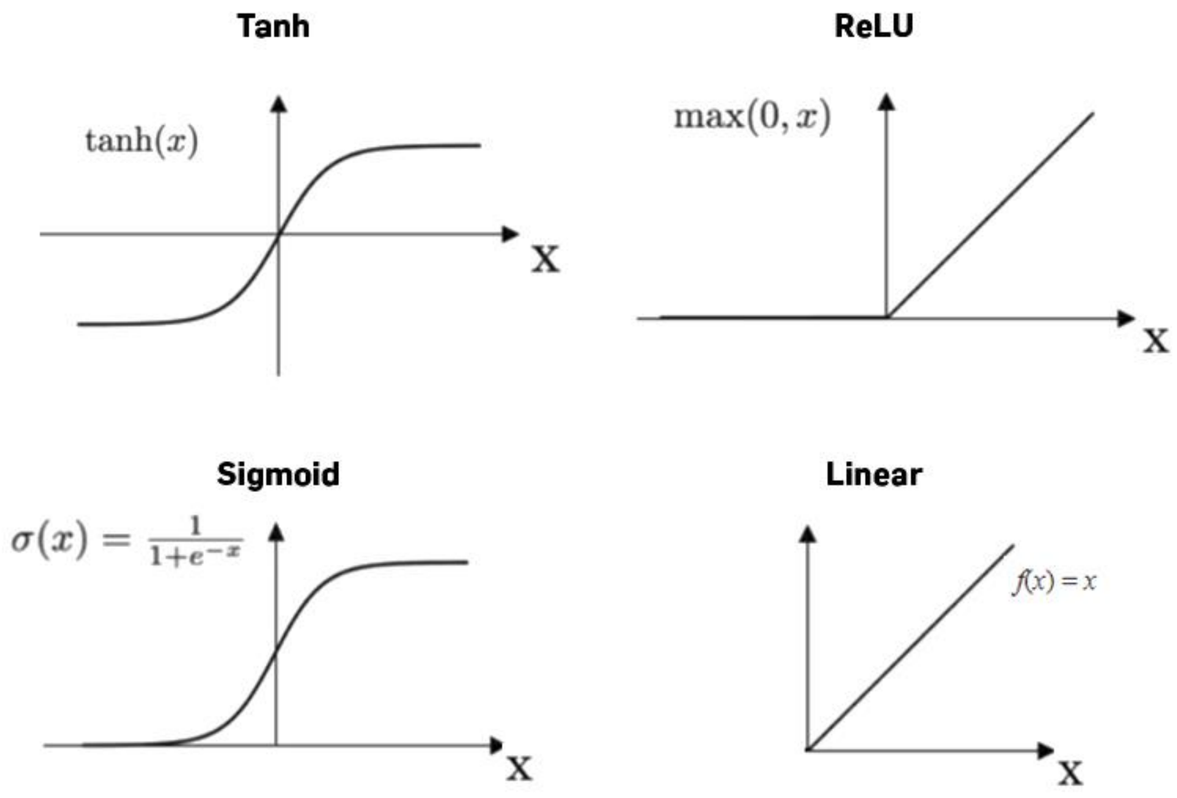
- **Neuron:**

Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs and sums them to produce an output (or activation). Usually each input is separately weighted, and the sum is passed through a non-linear function known as an activation function.



- **Activation functions:**

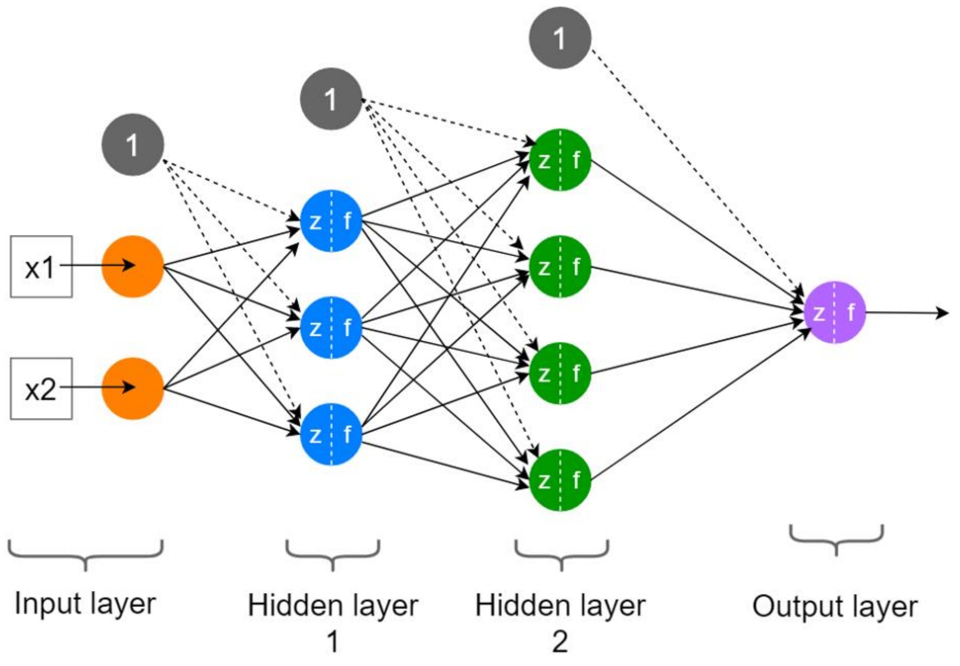
Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.



A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

- **Layers:**

There are three layers, an input layer, hidden layers, and an output layer. Inputs are inserted into the input layer, and each neuron provides an output value via an activation function. The outputs of the input layer are used as inputs to the next hidden layer.



Output layer can have one output or more than one.

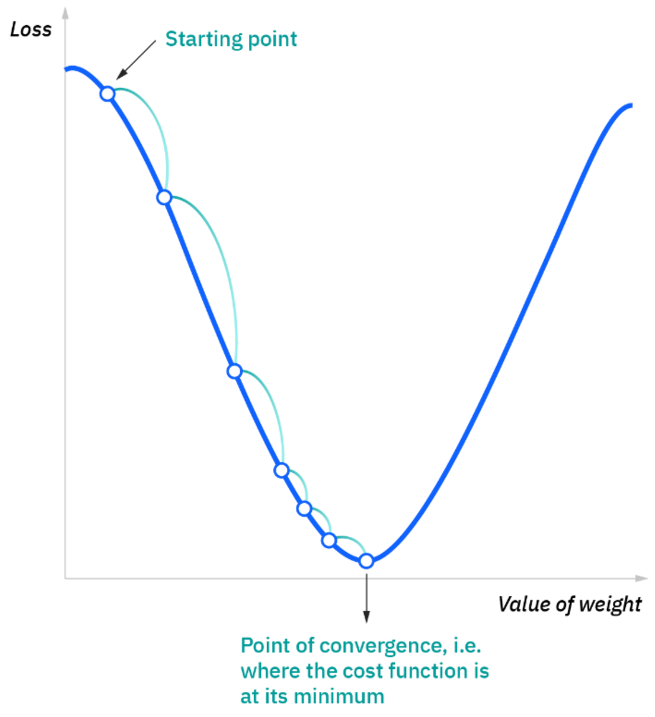
- **Gradient Descent:**

Gradient descent (GD) is the heart of Machine Learning algorithms. It's a mathematical iterative first-order optimization algorithm used to find a local minimum/maximum of a given function (training or Learning).

GD in Deep Learning or Machine Learning used to minimize the cost/loss function.

A configurable hyperparameter "learning rate" is used for optimization algorithms that determines the step size at each iteration while moving toward a minimum of a loss function.

There are other various methods to reduce cost error function such as Adam optimizer, Stochastic Gradient Descent (SGD), RMSProp and others…


Loss
Starting point
Value of weight
Point of convergence, i.e. where the cost function is at its minimum

α: Learning rate of the optimization algorithm.

θj: weights of the j corresponding neuron.

J(θ): the cost function.

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

**Gradient Descent** - Optimization algorithm

Other optimizers algorithms are little more complicated and have different advantages, you can check this link for more details " https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0 " .

- **Cost and Loss functions:**

The loss function is the error between the predictions made by our deep learning model and the corresponding true values with every training example (step). There are many types of losses functions such as Mean squared error (MSE(L2)), Mean absolute error, Log loss (cross entropy loss), Huber and others....

The cost function is the average value of the loss function over all training sample (episode). In Machine Learning, we usually try to optimize cost function rather than loss function.

n: number of steps in each episode.

Yi: the true values.

Ŷi: the predicted value.

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

**Loss function:** Mean squared error.

You can check this link " https://www.section.io/engineering-education/understanding-loss-functions-in-machine-learning/ " for more details.

- **Local and Global Optima:**

A local minimum of a function is a point where the function value is smaller than at nearby points, but possibly greater than at a distant point. A global minimum is a point where the function value is smaller than at all other feasible points. A function can have multiple local minima and only one global minima.

Gradient descent for Deep Learning is one of the techniques which can be used to find the local or global minima of the cost function.

- **How Neural Network work?**

A normal neural network consists of multiple layers called the input layer, output layer, and hidden layers. In each layer every node (neuron) is connected to all nodes in the next layer with parameters called weights.

Neural networks consist of nodes called perceptrons that do necessary calculations and detect features of neural networks. These perceptrons try to reduce the final cost error by adjusting the weights parameters. Moreover, a perceptron can be considered as a neural network with a single layer.

On the other hand, multilayer perceptrons are called deep neural networks. The perceptrons are activated when there is satisfiable input. Go through this wiki article "https://en.wikipedia.org/wiki/Perceptron" if you need to learn more about perceptrons.

Now let's move on to discuss the exact steps of a working neural network.

1. Initially, the dataset should be fed into the input layer which will then flow to the hidden layer.

2. The connections which exist between the two layers randomly assign weights to the input.

3. A bias is added to each input. Bias is a constant which is used in the model to fit best for the given data.

4. The weighted sum of all the inputs will be sent to a function that is used to decide the active status of a neuron by calculating the weighted sum and adding the bias. This function is called the activation function.

5. The nodes that are required to fire for feature extraction are decided based on the output value of the activation function.

6. The final output of the network is then compared to the required labeled data of our dataset to calculate the final cost error. The cost error is actually telling us how 'bad' our network is. Hence, we want the error to be as smallest as we can.

7. The weights are adjusted through backpropagation (by the gradient descent optimizer or other optimizers), which reduces the error. This backpropagation process can be considered as the central mechanism that neural networks learn. It basically fine-tunes the weights of the deep neural network in order to reduce the cost value.

In simple terms, what we do when training a neural network is usually calculating the loss (error value) of the model and checking if it is reduced or not. If the error is higher than the expected value, we have to update the model parameters (weights and bias), or do some changing on the hyperparameters (learning rate, number of neurons and layers and others...). once the loss is lower than the expected error margin we have to validate and test the model then use it.

- **Math behind Neural Network:**

**Step 1**: For each input, multiply the input value $x_i$ with weights $w_i$ and sum all the multiplied values. Weights represent the strength of the connection between neurons and decides how much influence the given input will have on the neuron's output. If the weight $w_1$ has a higher value than the weight $w_2$, then the input $x_1$ will have a higher influence on the output than $w_2$.

$$\sum = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n)$$

The row vectors of the inputs and weights are x = [$x_1$, $x_2$, … , xn] and w =[$w_1$, $w_2$, … , wn] respectively and their dot product is given by

$$x.w = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n)$$

Note that x as an input must be flattened for every type of data (Audio, image, texts…etc.)

**Step 2**: Add bias b to the summation of multiplied values and let's call this z. Bias — also known as the offset is necessary in most of the cases, to move the entire activation function to the left or right to generate the required output values.

$$z = x.w + b$$

**Step 3**: Pass the value of z to a non-linear activation function. Activation functions are used to introduce non-linearity into the output of the neurons, without which the neural network will just be a linear function.

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Note that these three steps represent the function of one neuron (perceptron), the weights propagate from a layer of neurons to the next layer, until the output layer. These steps called the neural network propagation.

**Step 4**: To know an estimation of how far are we from our desired solution a loss function is used. There are different loss functions, let's take the mean squared error loss function. which squares the difference between the true value ($y_i$) and predicted value ($\hat{y}_i$).

$$MSE_i = (y_i - \hat{y}_i)^2$$

Loss function is calculated for the entire training dataset batch and their average is the cost function:

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

**Step 5:** In order to find the best weights and bias for our Neural Network, we need to calculate the gradient of cost function C with respect to the weight $w_i$ using partial derivation. Since the cost function is not directly related to the weight $w_i$.

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$$

We get,

$$\frac{\partial C}{\partial w_i} = \frac{2}{n} \times sum(y - \hat{y}) \times \sigma(z) \times (1 - \sigma(z)) \times x_i$$

And for the bias b we get,

$$\frac{\partial C}{\partial b} = \frac{2}{n} \times sum(y - \hat{y}) \times \sigma(z) \times (1 - \sigma(z))$$

**The final step** is optimization, The weights and bias are updated as follows using gradient descent optimizer in this case (but there are also others optimizers), and the backpropagation and gradient descent is repeated until convergence.

$$w_i = w_i - \left(\alpha \times \frac{\partial C}{\partial w_i}\right)$$
$$b = b - \left(\alpha \times \frac{\partial C}{\partial b}\right)$$

- **Training, Validation and Testing datasets:**

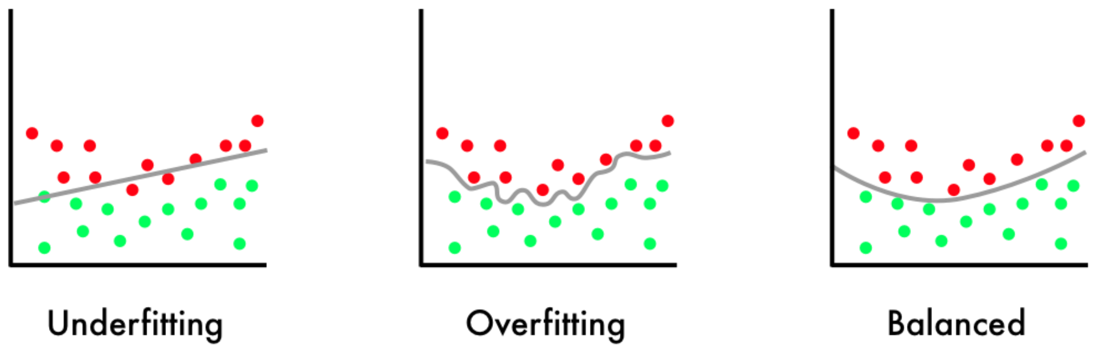
**Training Dataset**: The sample of data used to fit the model.

**Validation Dataset**:(unseen data) The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

**Test Dataset**:(unseen data) The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

- **Overfitting:**

In deep learning algorithms a model tries to fit the training data entirely and ends up memorizing the data patterns and the noise and random fluctuations. These models fail to generalize and perform well in the case of unseen data.



The difference between the training and validation accuracy is an indicator of an overfitting problem.

- **Techniques to avoid overfitting**

1.Train with more data.

2.Data augmentation: data look slightly different every time the model processes it.

3.Regularization: Regularization applies a penalty to the input parameters with the larger coefficients, which subsequently limits the model's variance.

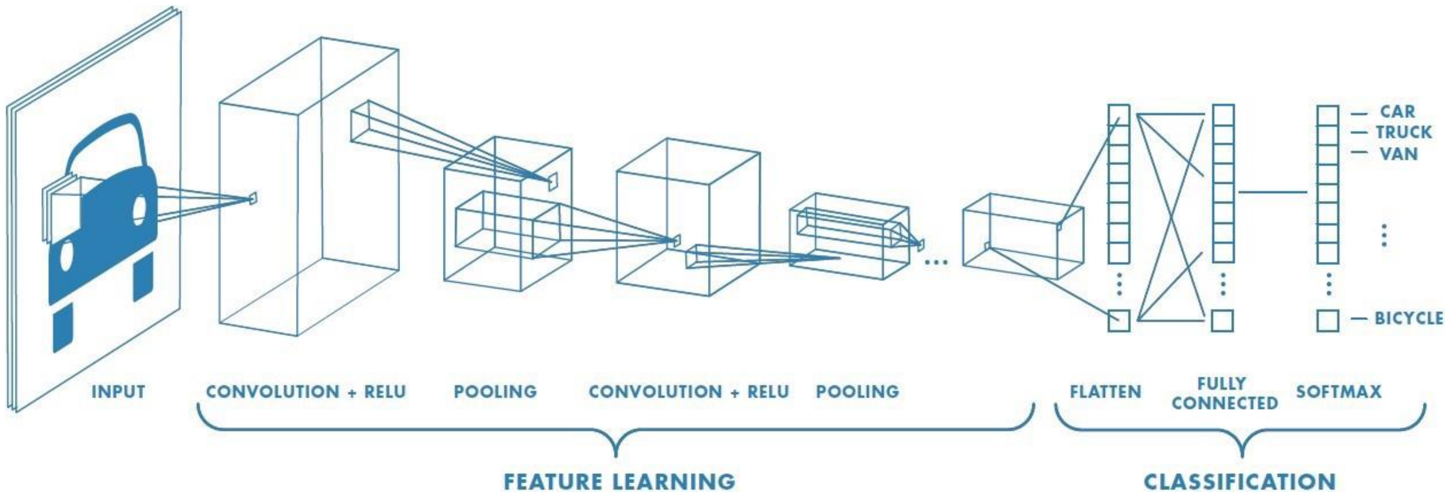
4.Dropout: dropping out nodes in the network is a simple and effective method to prevent overfitting. In regularization, some number of layer outputs are randomly ignored or "*dropped out*" to reduce the complexity of the model.

## 3. Convolutional Neural Network (CNN) :

Convolutional neural networks power image recognition and computer vision tasks. Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs, and based on those inputs, it can take action.

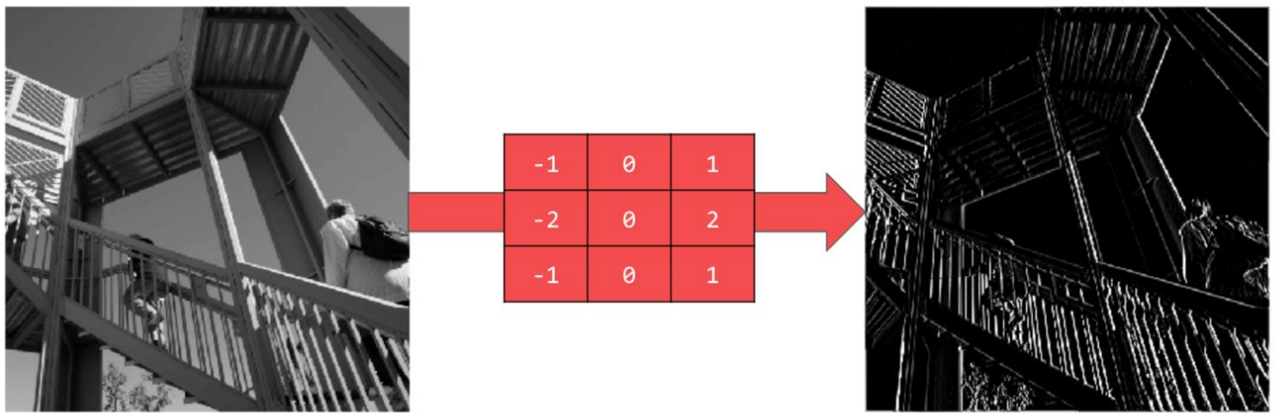
CNN is a type of deep learning model for processing data that has a grid pattern (matrix) such as images which is designed to automatically and adaptively learn spatial hierarchies of features from low to high-level patterns.

CNN is a mathematical construct that is typically composed of three types of layers (or building blocks) convolution, pooling, and fully connected layers. The first two, convolution and pooling layers perform feature extraction, whereas the third is a fully connected layer (Simple Neural Network), maps the extracted features into final output.



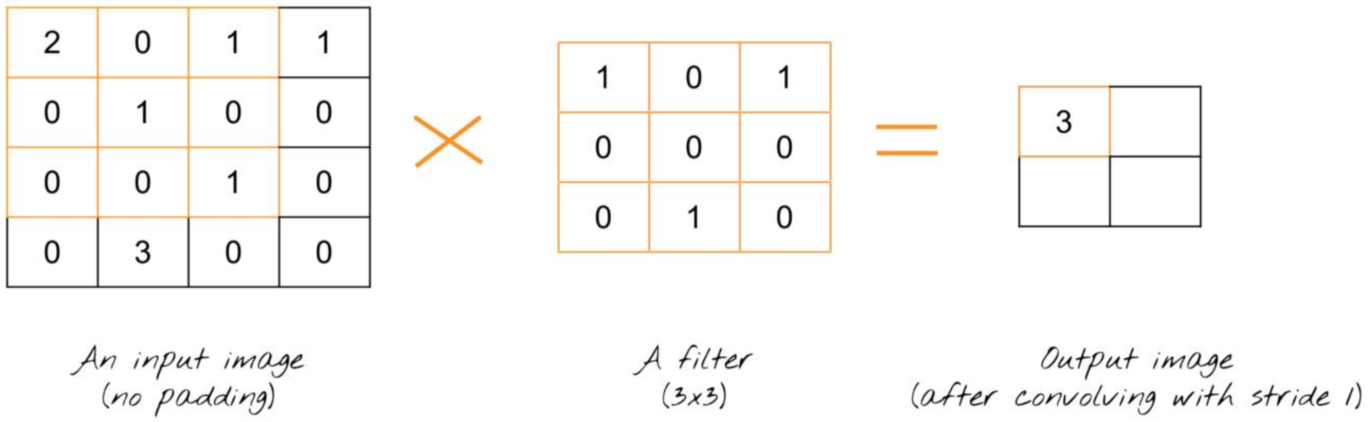
- **Convolution layers:**

A convolution is how the input is modified by a filter. In convolutional networks, multiple filters are taken to slice through the image and map them one by one and learn different portions of an input image. Imagine a small filter sliding left to right across the image from top to bottom and that moving filter is looking for, say, a dark edge. Each time a match is found, it is mapped out onto an output image.

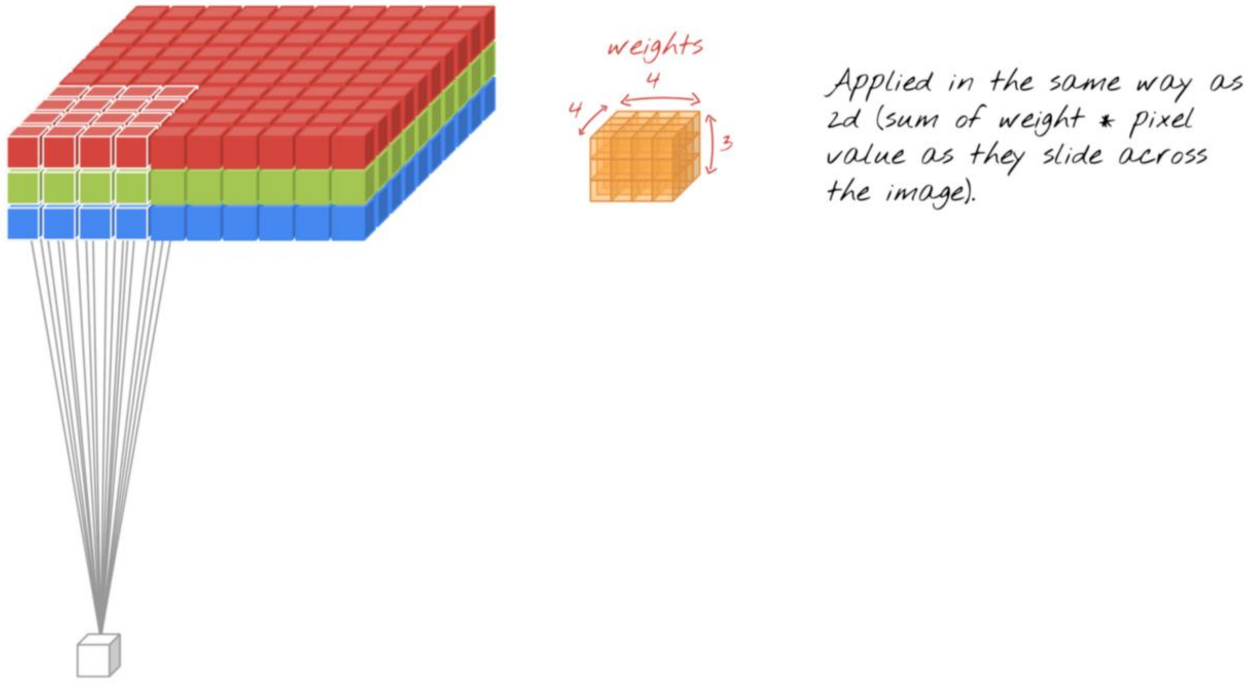


1. Filters:

Let’s take an example and start with a (4 x 4) input image with no padding and we use a (3 x 3) convolution filter to get an output image. The first step is to multiply the yellow region in the input image with a filter. Each element is multiplied with an element in the corresponding location. Then you sum all the results, which is one output value.


2 0 1 1
0 1 0 0
0 0 1 0
0 3 0 0
1 0 1
0 0 0
0 1 0
3
An input image (no padding)
A filter (3x3)
Output image (after convolving with stride 1)

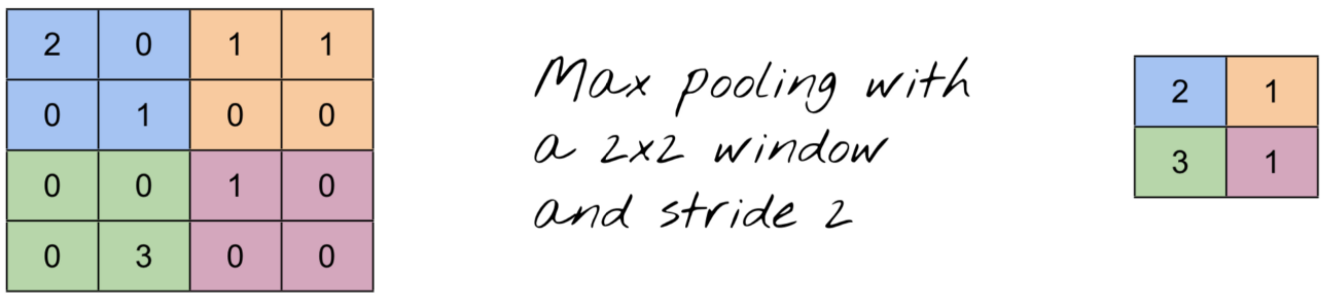
Convolution in 3D is just like 2D, except you are doing the 2d work 3 times, because there are 3 color channels.


weights
4
4
3
Applied in the same way as 2d (sum of weight * pixel value as they slide across the image).

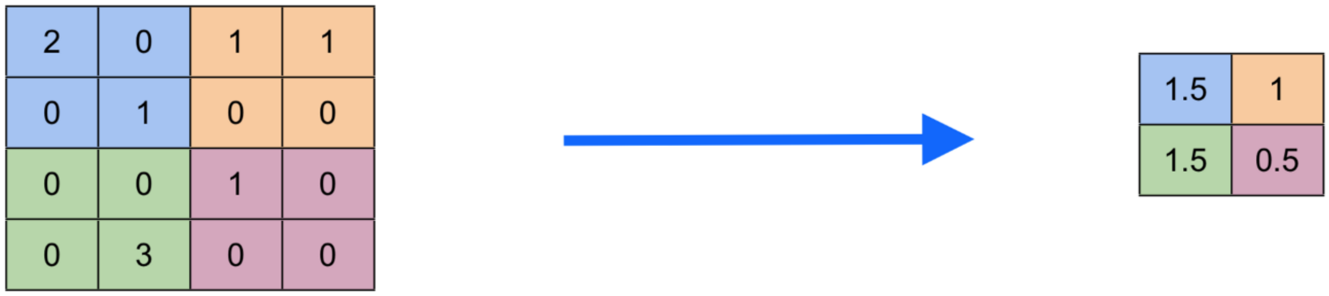
If you want to keep the output image at the same width and height without decreasing the filter size, you can add padding to the original image with zero’s and make a convolution slice through the image.

2. Pooling:

Note that pooling is a separate step from convolution. Pooling is used to reduce the image size of width and height. Note that the depth is determined by the number of channels. As the name suggests, all it does is it picks the maximum value in a certain size of the window. Although it’s usually applied spatially to reduce the x, y dimensions of an image. There are two types of pooling: Max-Pooling and Average-Pooling.



Max-Pooling

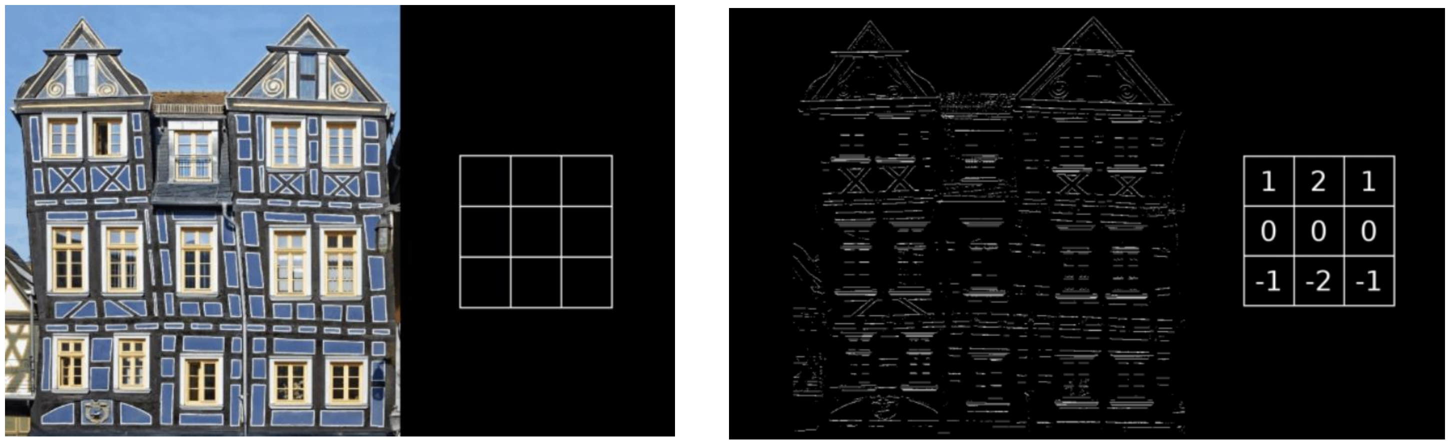


Average-Pooling

- **The math behind CNN**

**Step 1:** Filter (or Kernel) convolution is a process when we take a small matrix of numbers (in CNN these numbers are updatable) and pass it over our image and transform it based on the values from the filter.

First, we place our filter over a selected pixel, then we take each value from kernel and multiply them in pairs with corresponding values from the image. Finally, we sum up everything and put the result in the right place in the output feature map. Above we can see how such an operation looks like in micro scale, but what is even more interesting, is what we can achieve by performing it on a full image. For example:



feature map values using kernels are calculated according to the following formula:

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k]$$

the input image is denoted by f and kernel by h, The indexes of rows and columns of the result matrix are marked with m and n respectively.

If we look at how our kernel moves through the image, we see that the impact of the pixels located on the outskirts is much smaller than those in the center of image and the output filtered channels are smaller on shape. in this case we can pad our image with an additional border (usually in practice we put zeros on additional padding pixels). To guaranteed the same size of input and output channels, the padding width should meet the following equation:

$$p = \frac{f-1}{2}$$

where p is padding and f is the filter dimension (usually odd).

We have other option on the convolution filter called striding that can increase the passing steps of the kernel from one as default to another positive integer (step). (As default the stride step is one).

The dimensions of the output matrix taking into account padding and stride, can be calculated using the following formula.

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - f}{s} + 1 \right\rfloor$$

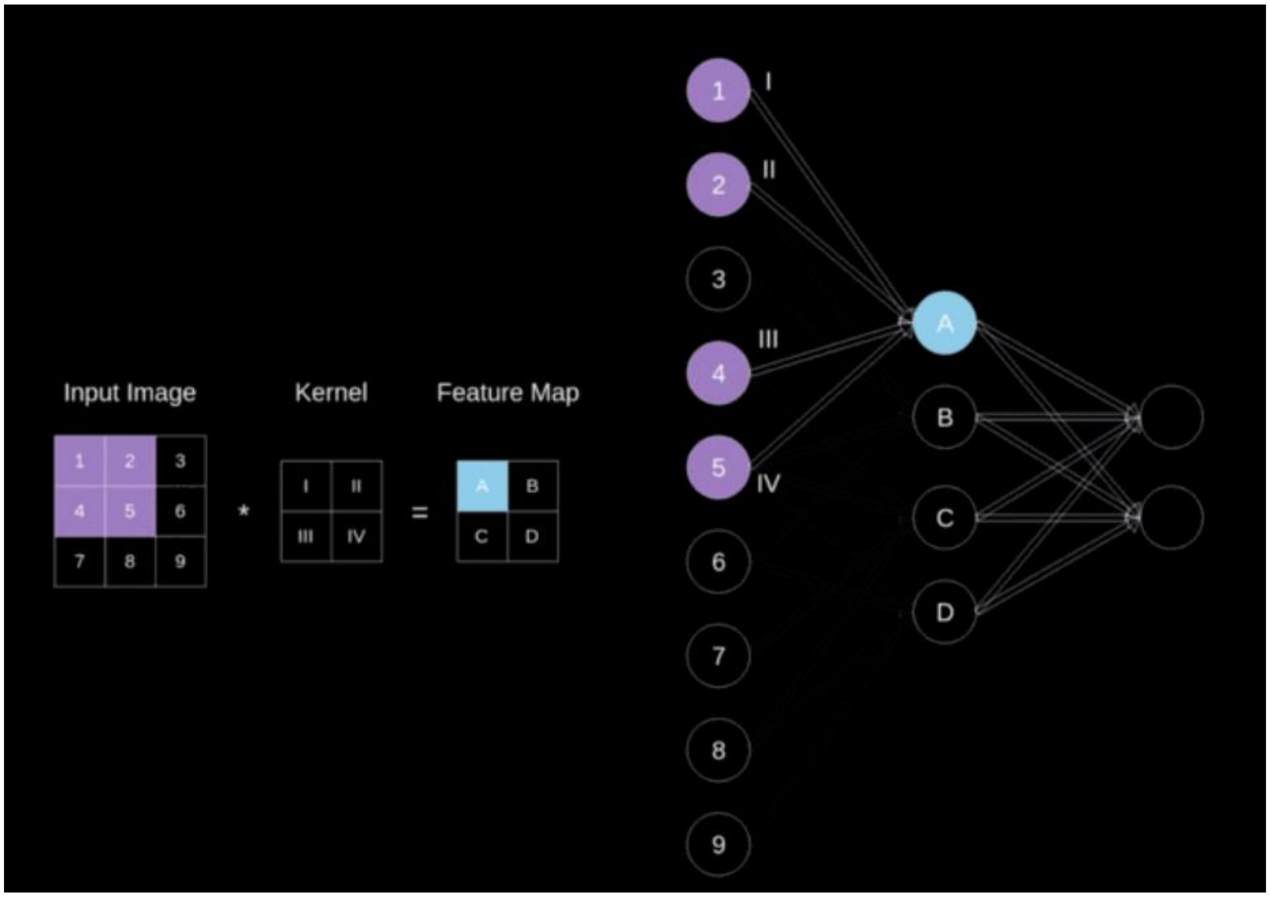
Max pooling and other pooling types are using to decrease the output size of channels.

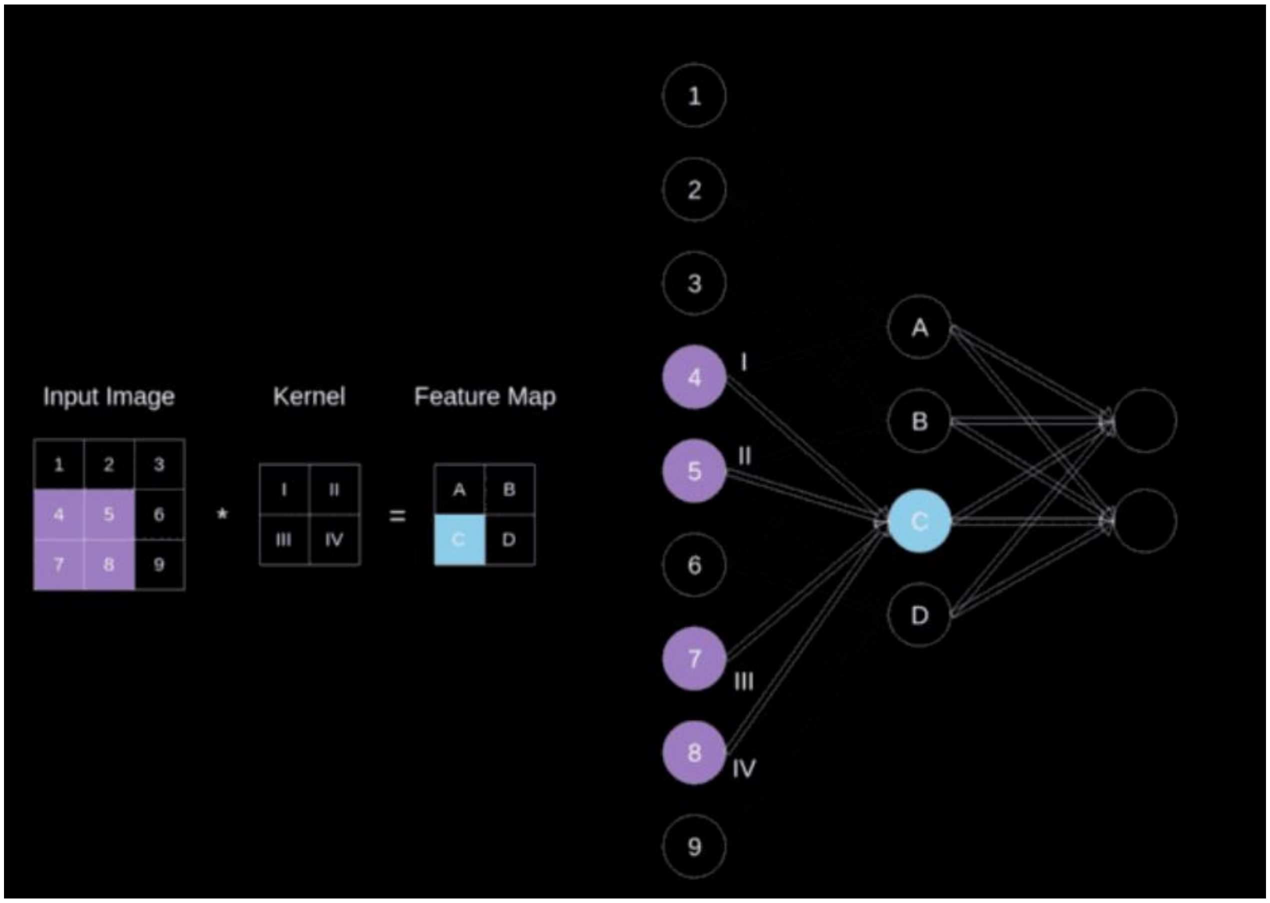
**Step 2:** The forward calculation, the methodology is almost identical to the one we used for densely connected neural networks (simple neural network), the only difference is that instead of using a simple matrix multiplication, we use convolution between image matrix and kernels. Kernels contains the weights that will updated with the backpropagation process.

Note, that CNN have the ability to construct filters (by updating weights) that can extract the most important features from image for our desired goal (decreasing the loss function), and this is the most powerful technique on computer vision.

The following picture explains how convolution matrix applied between filters and pixels on the neural network and you can see that not all neurons in the two consecutive layers are connected to each other, and this is why densely connected neural networks are poor at working with images because it needs to connect every pixel with every weight in the first layer and forward with this huge number of weights. So, it requires a huge number of parameters to be learned.

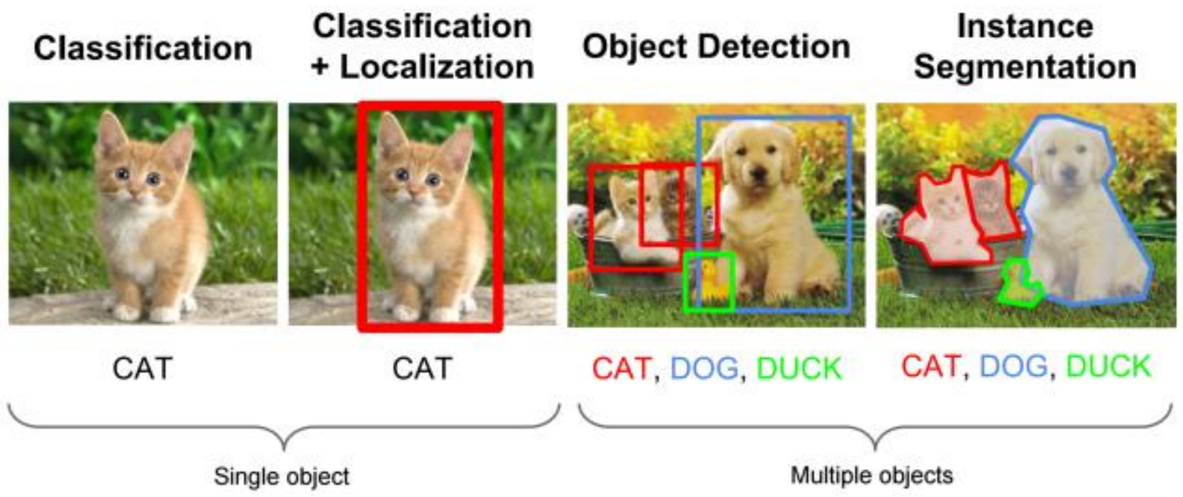
Note that the following images shown only two steps, typically there are four steps for this example.


Input Image
1
2
3
4
5
6
7
8
9
*
Kernel
I
II
III
IV
=
Feature Map
A
B
C
D
1
2
3
4
5
6
7
8
9
I
II
III
IV
A
B
C
D


Input Image
1
2
3
4
5
6
7
8
9
*
Kernel
I
II
III
IV
=
Feature Map
A
B
C
D
1
2
3
4
5
6
7
8
9
I
II
III
IV
A
B
C
D

**Computer vision:**

Object detection using deep learning provides a fast and accurate means to predict the location and segmentation of an object in an image. Deep learning is a powerful machine learning technique in which the object detector automatically learns image features required for detection tasks. Several techniques for object detection using deep learning are available such as Faster R-CNN, you only look once (YOLO) v2, YOLO v3, YOLO v4…v7, and single shot detection (SSD), GoogleNet, VGGNet and others….

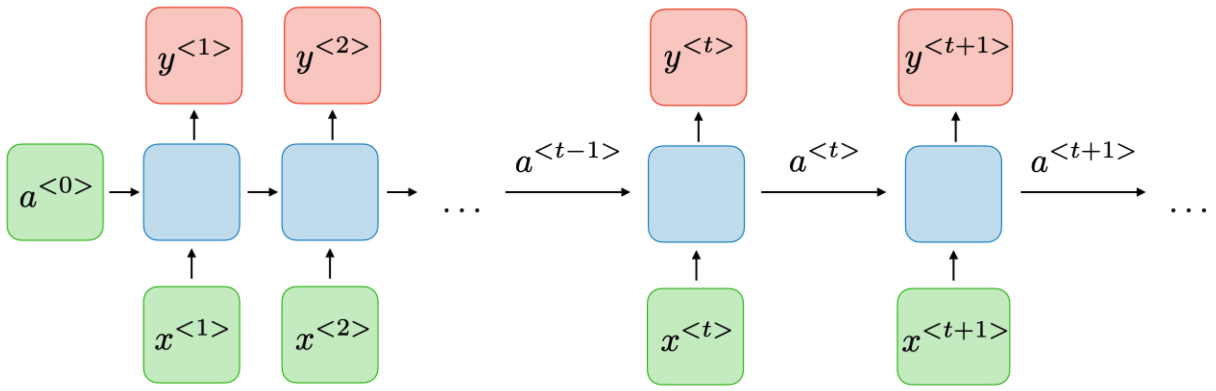


Applications for object detection include:

- Image classification
- Scene understanding
- Self-driving vehicles
- Surveillance

## 4. Recurrent Neural Network:

A recurrent neural network (RNN) is a special type of an artificial neural network adapted to work for time series data or data that involves sequences. Ordinary simple or convolutional neural networks are only meant for data points, which are independent of each other. However, if we have data in a sequence such that one data point depends upon the previous data point, we need to modify the neural network to incorporate the dependencies between these data points. RNNs have the concept of memory by allowing previous outputs to be used as inputs.



Architecture of traditional RNN

RNN models are mostly used in the fields of natural language processing and speech recognition and other applications such as music generation and sentiment classification and stock prediction and others…

- **RNN Advantages:**

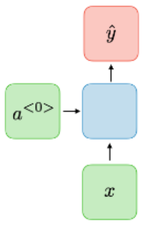
• Possibility of processing input of any length
• Model size not increasing with size of input
• Computation takes into account historical information
• Weights are shared across time

- **RNN Drawbacks:**

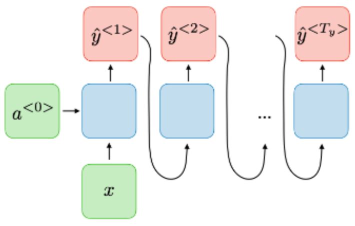
• Computation being slow
• Difficulty of accessing information from a long time ago
• Cannot consider any future input for the current state

- **Types of RNN:**

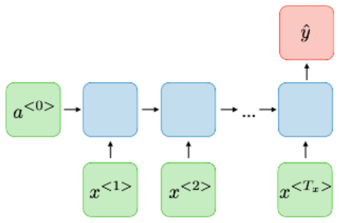
One To One



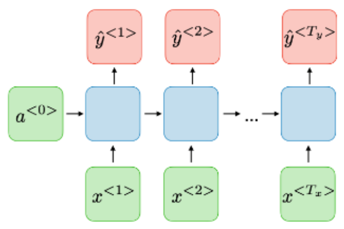
One to Many



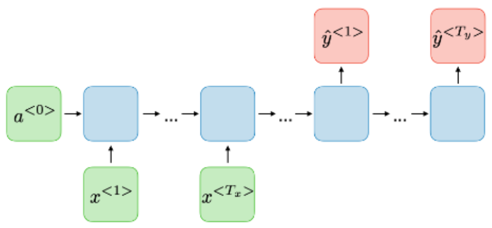
Many To One



Many To Many ($Tx = Ty$)

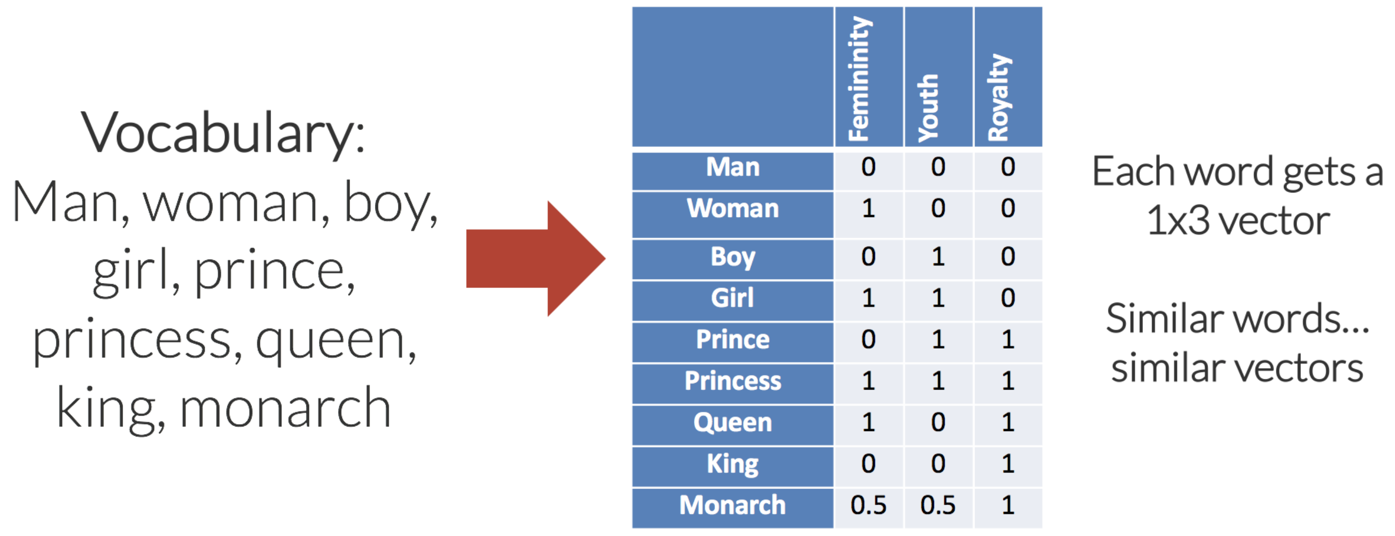


Many To Many ($Tx \neq Ty$)



- **Embedding words (word2vec)**

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Techniques for learning word embeddings can include Word2Vec, GloVe, and other neural network-based approaches that train on an NLP task such as language modeling or document classification.



Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

| | Femininity | Youth | Royalty |
|---|---|---|---|
| Man | 0 | 0 | 0 |
| Woman | 1 | 0 | 0 |
| Boy | 0 | 1 | 0 |
| Girl | 1 | 1 | 0 |
| Prince | 0 | 1 | 1 |
| Princess | 1 | 1 | 1 |
| Queen | 1 | 0 | 1 |
| King | 0 | 0 | 1 |
| Monarch | 0.5 | 0.5 | 1 |

Each word gets a 1x3 vector

Similar words... similar vectors

Technically, learning the embedding matrix can be done using target/context likelihood models.

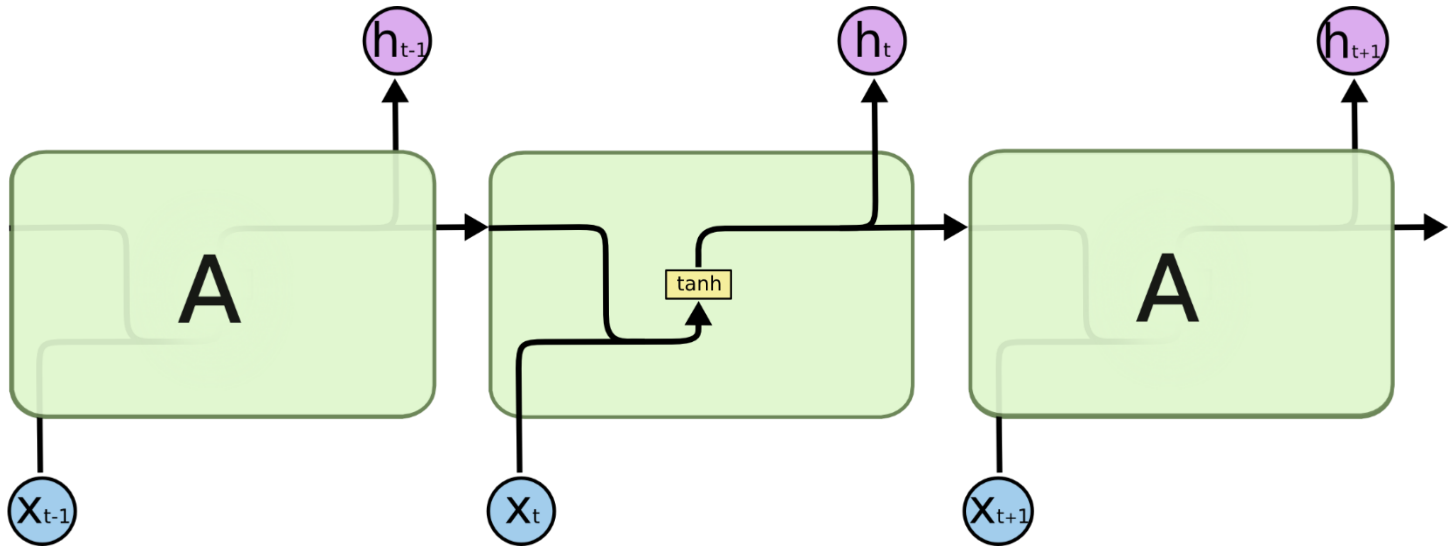
- **Vanishing gradient:** The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU

- **Commonly used RNN architecture:**

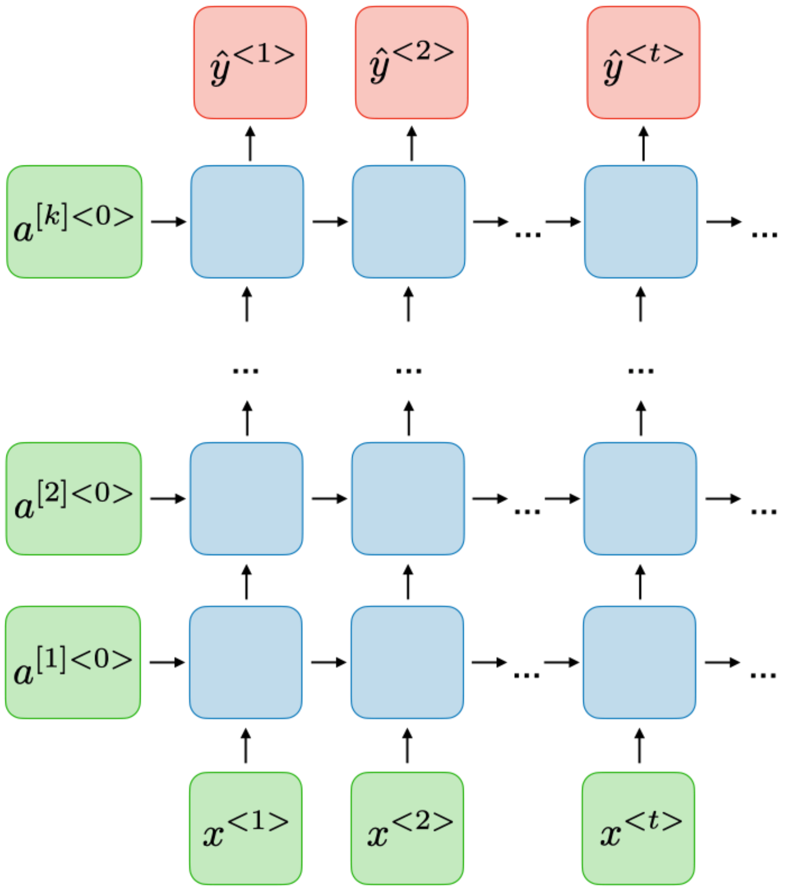
**1- Simple RNN**

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer



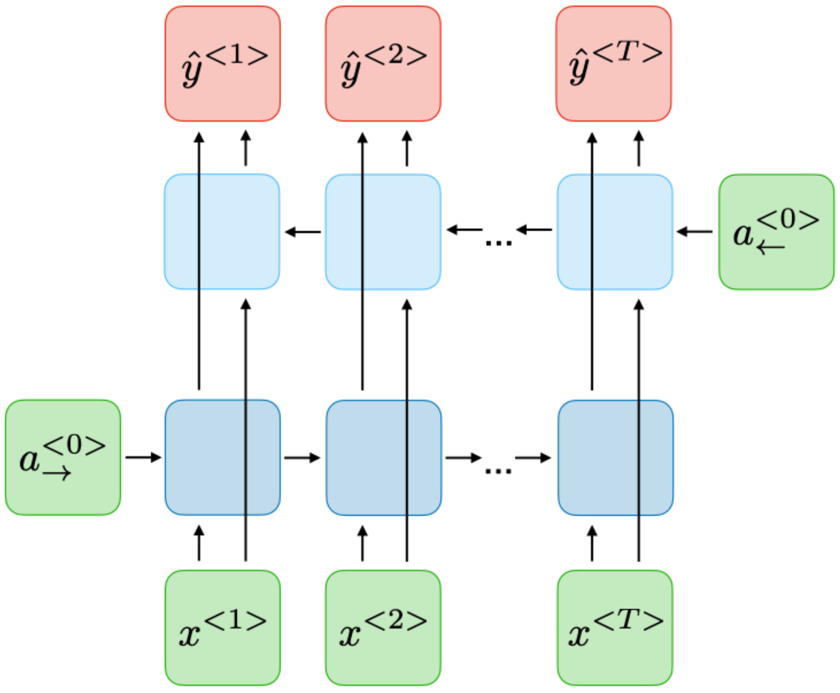
**2- Deep RNN**

it describes a RNN with multiple L-hidden layers.



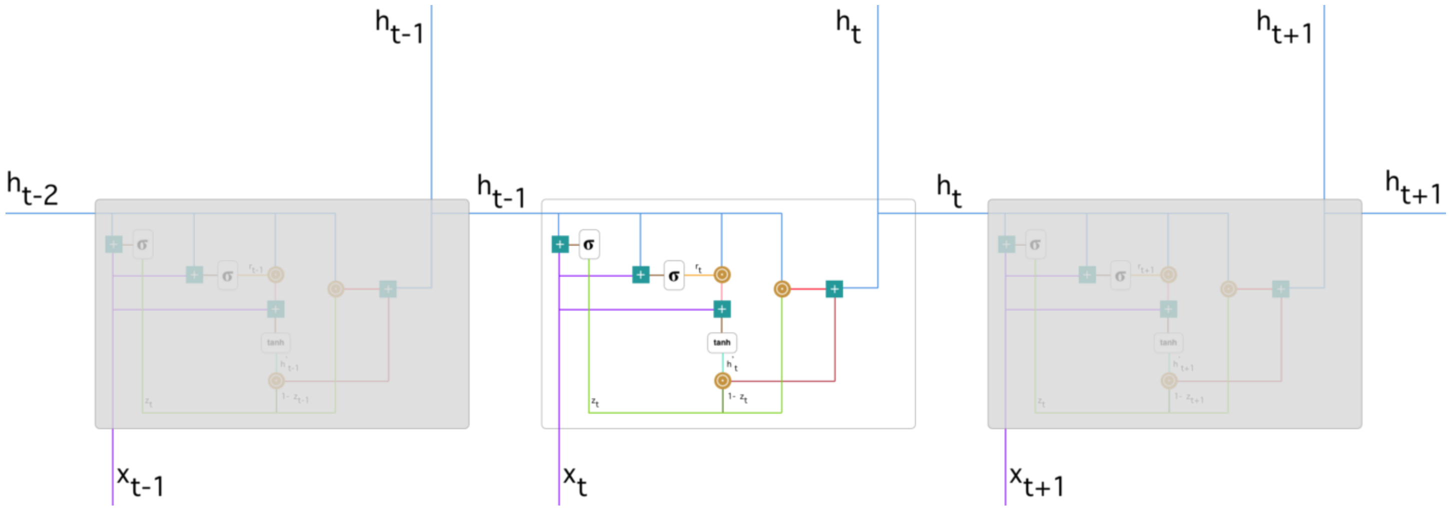
## 3- Bidirectional

Connect two hidden layers of opposite directions to the same output.

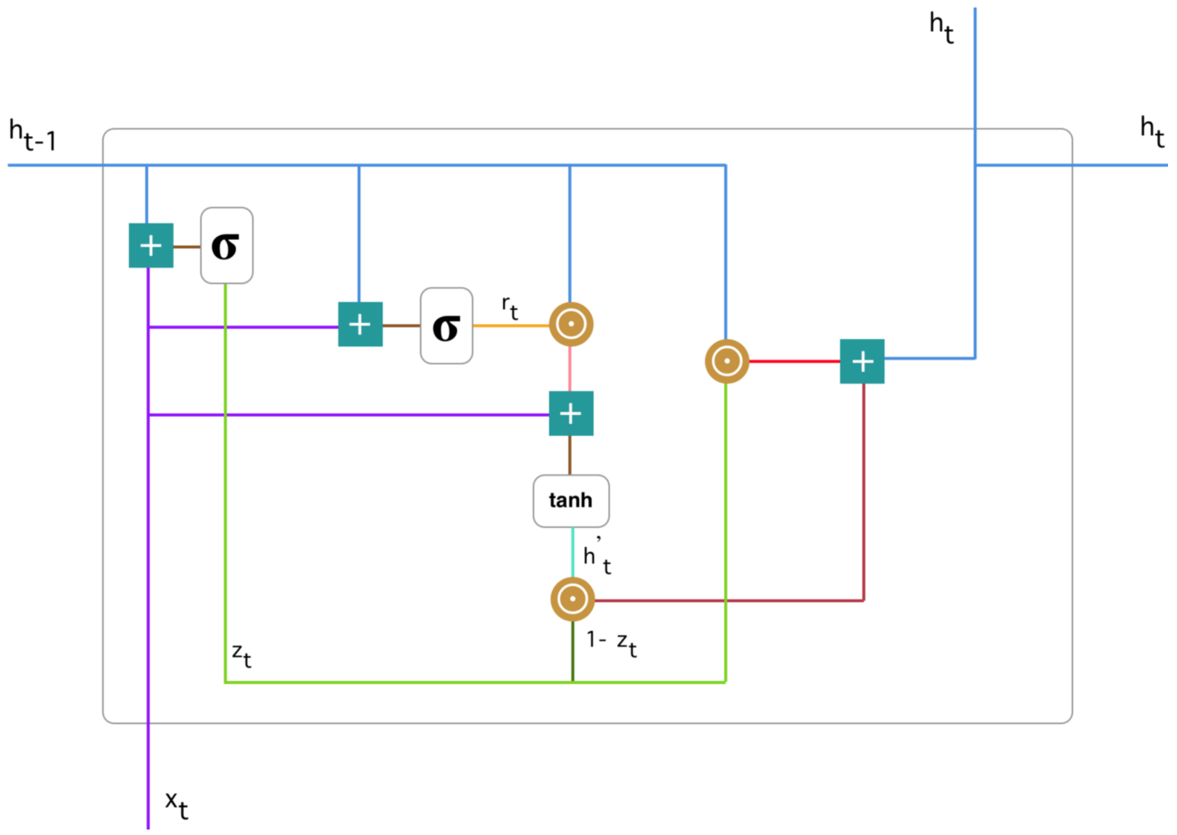


## 4- Gated Recurrent Unit (GRU)

GRU is to solve the vanishing gradient problem, it decides what information should be passed to the output. The special thing about it is that it can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.



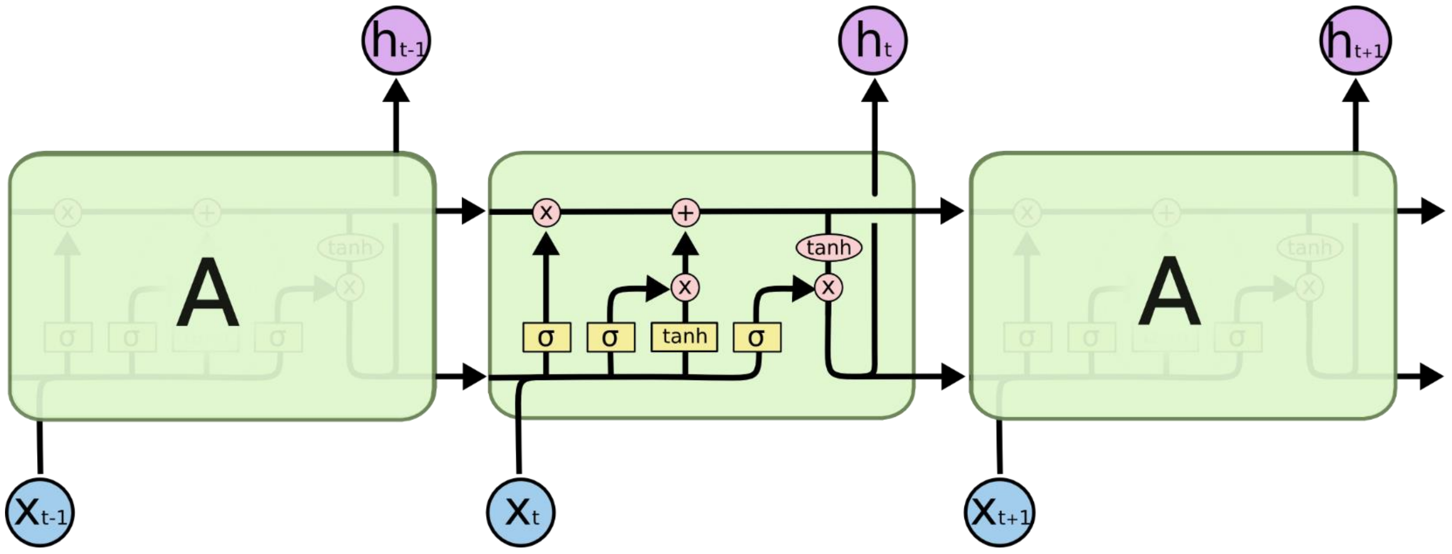
Recurrent Neural Network nodes with gated recurrent unit



Single gated recurrent unit

## 5- Long Short-Term Memory

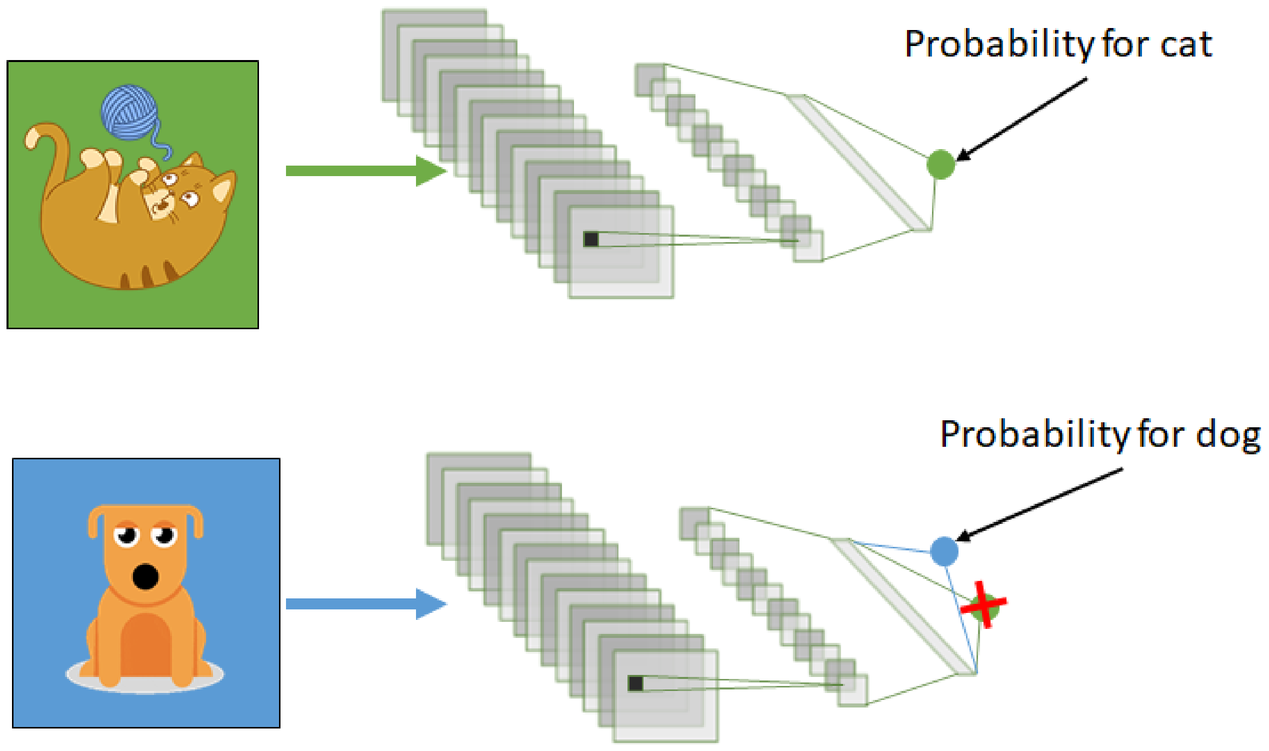
Long Short-Term Memory Network is an advanced RNN, that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN and also better then GRU.



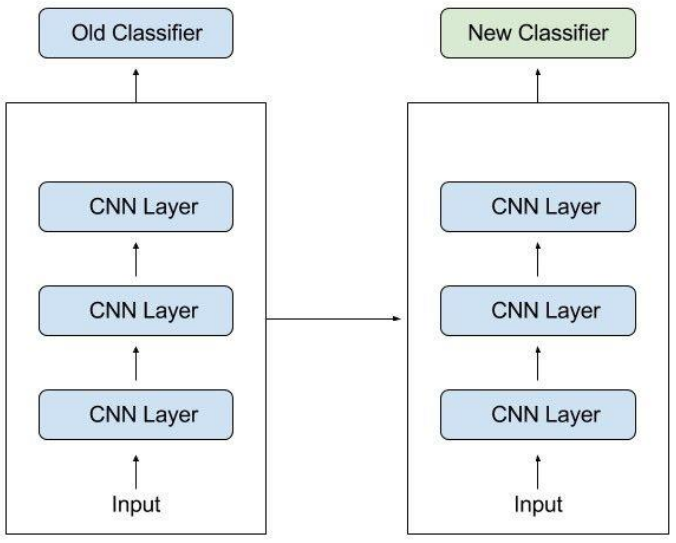
Long Short-Term Memory unit

## 5. Transfer Learning

The knowledge of an already trained machine learning model is transferred to a different but closely linked problem throughout transfer learning. For example, if you trained a simple classifier to predict whether an image contains a backpack, you could use the model's training knowledge to identify other objects such as sunglasses.



In computer vision, neural networks typically aim to detect edges in the first layer, forms in the middle layer, and task-specific features in the latter layers. The early and central layers are employed in transfer learning, and the latter layers are only retrained. It makes use of the labelled data from the task it was trained on.



If the model has trained to recognize objects in the earlier levels, we will simply retrain the subsequent layers to understand what distinguishes sunglasses from other objects.

Transfer learning offers a number of advantages, the most important of which are reduced training time, improved neural network performance, and the absence of a large amount of data. To train a neural model from scratch a lot of data is typically needed, but access to that data isn't always possible this is when transfer learning comes in handy.