DEEP LEARNING

SWAPNIL SAURAV

eka PUBLISHING

CHAPTER ONE: INTRODUCTION

Motivation of Deep Learning, and Its History and Inspiration

On a conceptual level, deep learning is inspired by the brain but not all of the brain's details are relevant. For a comparison, aeroplanes were inspired by birds. The principle of flying is the same but the details are extremely different.

The history of deep learning goes back to a field which changed its name now to cybernetics. It started in the 1940s with McCulloch and Pitts. They came up with the idea that neurons are threshold units with on and off states. You could build a Boolean circuit by connecting neurons with each other and conduct logical inference with neurons. The brain is basically a logical inference machine because neurons are binary. Neurons compute a weighted sum of inputs and compare that sum to its threshold. It turns on if it's above the threshold and turns off if it's below, which is a simplified view of how neural networks work.

In 1947, Donald Hebb had the idea that neurons in the brain learn by modifying the strength of the connections between neurons. This is called hyper learning, where if two neurons are fired together, then the connection linked between them increases; if they don't fire together, then the connection decreases.

Later in 1948, cybernetics were proposed by Norbert Wiener, which is the idea that by having systems with sensors and actuators, you have a feedback loop and a self-regulatory system. The rules of the feedback mechanism of a car all come from this work.

In 1957, Frank Rosenblatt proposed the Perceptron, which is a learning algorithm that modifies the weights of very simple neural nets.

Overall, this idea of trying to build intellectual machines by simulating lots of neurons was born in 1940s, took off in 1950s, and completely died in late 1960s. The main reasons for the field dying off in 1960 are:

The researchers used neurons that were binary. However, the way to get backpropagation to work is to use activation functions that are continuous. At that time, researchers didn't have the idea of using continuous neurons and they didn't think they can train with gradients because binary neurons are not differential.

With continuous neurons, one would have to multiply the activation of a neuron by a weight to get a contribution to the weighted sum. However, before 1980, the multiplication of two numbers, especially floating-point numbers, were extremely slow. This resulted in another incentive to avoid using continuous neurons.

Deep Learning took off again in 1985 with the emergence of backpropagation. In 1995, the field died again and the machine learning community abandoned the idea of neural nets. In early 2010, people start using neuron nets in speech recognition with huge performance improvement and later it became widely deployed in the commercial field. In 2013, computer vision started to switch to neuron nets. In 2016, the same transition occurred in natural language processing. Soon, similar revolutions will occur in robotics, control, and many other fields.

Supervised Learning

Deep learning applications use supervised learning. Supervised learning is a process by which, you collect a bunch of pairs of inputs and outputs, and the inputs are feed into a machine to learn the correct output. When the output is correct, you don't do anything. If the output is wrong, you tweak the parameter of the machine and correct the output toward the one you want. The trick here is how you figure out which direction and how much you tweak the parameter and this goes back to gradient calculation and backpropagation.

Supervised learning stems from Perceptron and Adaline. The Adaline is based on the same architecture with weighted inputs; when it is above the threshold, it turns on and below the threshold, it turns off. The Perceptron is a 2-layer neuron net where the second layer is trainable and the first layer is fixed. Most of the time, the first layer is determined randomly and that's what they call associative layers.

History of Pattern Recognition and introduction to Gradient Descent

The foregoing is the conceptual basis of pattern recognition before deep learning developed. The standard model of pattern recognition consists of feature extractor and trainable classifier. Input goes into the feature extractor, extracting relevant useful characteristics of inputs such as detecting an eye when the purpose is recognizing the face. Then, the vector of features is fed to the trainable classifier for computing weighted sum and comparing it with the threshold. Here, a trainable classifier could be a perceptron or single neural network. The problem is feature extractor should be engineered by hand. Which means, pattern recognition/computer vision focus on feature extractor considering how to design it for a particular problem, not much devoted to a trainable classifier.

After the emergence and development of deep learning, the 2-stage process changed to the sequences of modules. Each module has tunable parameters and nonlinearity. Then, stack them making multiple layers. This is why it is called "deep learning". The reason why using nonlinearity rather than linearity is that two linear layers could be one linear layer since the composition of two linear is linear.

The simplest multi-layer architecture with tunable parameters and nonlinearity could be: the input is represented as a vector such as an image or audio. This input is multiplied by the weight matrix whose coefficient is a tunable parameter. Then, every component of the result vector is passed through a nonlinear function such as ReLU. Repeating this process, it becomes a basic neural network. The reason why it is called a neural network is that this architecture calculates the weighted sum of components of input by corresponding rows of a matrix.

Back to the point of supervised learning, we are comparing the resulting output with target output then optimize the objective function which is the loss, computing a distance/penalty/divergence between the result and target. Then, average this cost function over the training set. This is the goal we want to minimize. In other words, we want to find the value of the parameters that minimize this average.

The method of how to find it is computing gradient. For example, if we are lost in a smooth mountain at foggy night and want to go to the village in the valley. One way could be turning around and seeing which way the steepest way is to go down then take a small step down. The direction is (negative) gradient. With the assumption that the valley is convex, we could reach the valley.

The more efficient way is called Stochastic Gradient Descent (SGD). Since we want to minimize average loss over the training set, we take one sample or small group of samples and calculate the error, then use gradient descent. Then, we take a new sample and get a new value for the error, then get the gradient which is a different direction normally. Two of the main reasons for using SGD are that it helps a model to converge fast empirically if the training set is very large and it enables better generalization, which means getting similar performance on various sets of data.

Evolution of CNNs

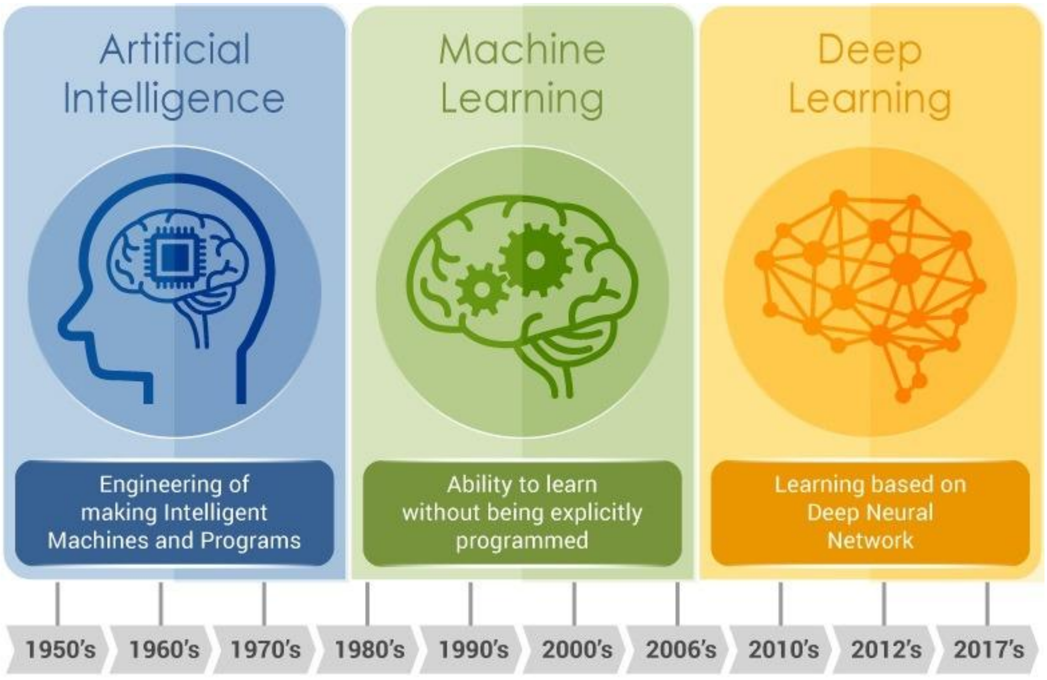
In animal brains, neurons react to edges that are at particular orientations. Groups of neurons that react to the same orientations are replicated over all of the visual field. Fukushima (1982) built a neural net (NN) that worked the same way as the brain, based on two concepts. First, neurons are replicated across the visual field. Second, there are complex cells that pool the information from simple cells (orientation-selective units). As a result, the shift of the picture will change the activation of simple cells, but will not influence the integrated activation of the complex cell (convolutional pooling). LeCun (1990) used backprop to train a CNN to recognize handwritten digits. There is a demo from 1992 where the algorithm recognizes the digits of any style. Doing character/pattern recognition using a model that is trained end-to-end was new at that time. Previously, people had used feature extractors with a supervised model on top. These new CNN systems could recognize multiple characters in the image at the same time. To do it, people used a small input window for a CNN and swiped it over the whole image. If it activated, it meant there was a particular character present.

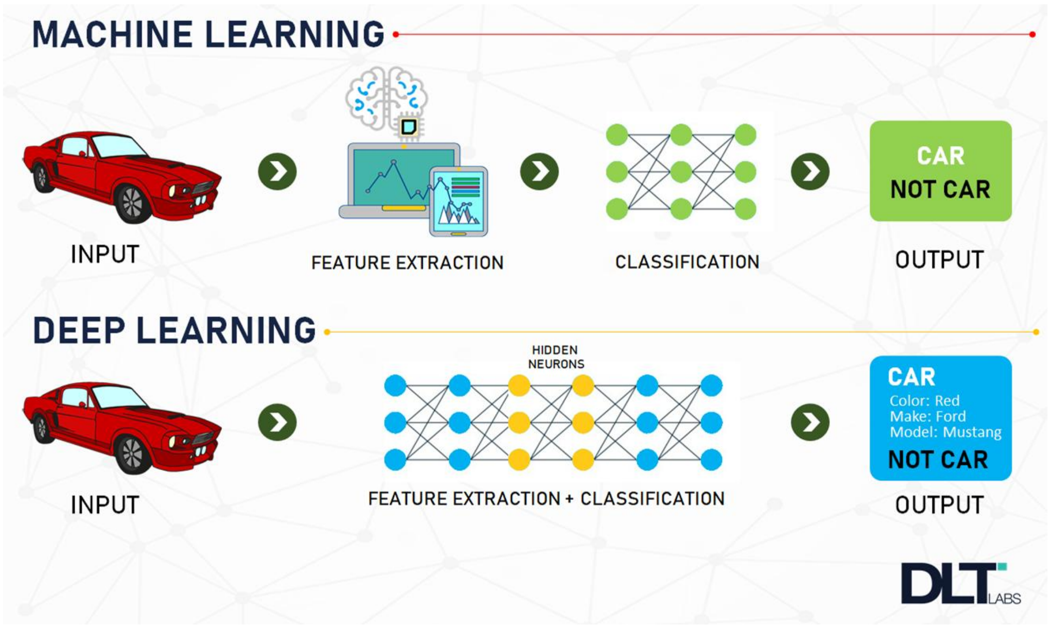
Later, this idea was applied to faces/people detection and semantic segmentation (pixel-wise classification). Examples include Hadsell (2009) and Farabet (2012). This eventually became popular in industry, used in autonomous driving applications such as lane tracking. Special types of hardware to train CNN were a hot topic in the 1980s, then the interest dropped, and now it has become popular again. The deep learning (though the term was not used at that time) revolution started in 2010-2013. Researchers focused on inventing algorithms that could help train large CNNs faster. Krizhevsky (2012) came up with AlexNet, which was a much larger CNN than those used before, and trained it on ImageNet (1.3 million samples) using GPUs. After running for a couple of weeks AlexNet beat the performance of the best competing systems by a large margin – a 25.8% vs. 16.4% top-5 error rate.

After seeing AlexNet's success, the computer vision (CV) community was convinced that CNNs work. While all papers from 2011-2012 that mentioned CNNs had been rejected, since 2016 most accepted CV papers use CNNs.

Over the years, the number of layers used has been increasing: LeNet – 7, AlexNet – 12, VGG – 19, ResNet – 50. However, there is a trade-off between the number of operations needed to compute the output, the size of the model, and its accuracy. Thus, a popular topic now is how to compress the networks to make the computations faster.

Deep Learning and Feature Extraction


Artificial Intelligence
Engineering of making Intelligent Machines and Programs
Machine Learning
Ability to learn without being explicitly programmed
Deep Learning
Learning based on Deep Neural Network
1950's
1960's
1970's
1980's
1990's
2000's
2006's
2010's
2012's
2017's



Multilayer networks are successful because they exploit the compositional structure of natural data. In compositional hierarchy, combinations of objects at one layer in the hierarchy form the objects at the next layer. If we mimic this hierarchy as multiple layers and let the network learn the appropriate combination of features, we get what is called Deep Learning architecture. Thus, Deep Learning networks are hierarchical in nature. Deep learning architectures have led to an incredible progress in computer vision tasks ranging from identifying and generating accurate masks around the objects to identifying spatial properties of an object. Mask-RCNN and RetinaNet architectures mainly led to this improvement.

Mask RCNNs have found their use in segmenting individual objects, i.e. creating masks for each object in an image. The input and output are both images. The architecture can also be used to do instance segmentation, i.e. identifying different objects of the same type in an image. Detectron, a Facebook AI Research (FAIR) software system, implements all these state-of-the-art object detection algorithms and is open source.

Some of the practical applications of CNNs are powering autonomous driving and analysing medical images.

Although the science and mathematics behind deep learning is fairly understood, there are still some interesting questions that require more research. These questions include: Why do architectures with multiple layers perform better, given that we can approximate any function with two layers? Why do CNNs work well with natural data such as speech, images, and text? How are we able to optimize non-convex functions so well? Why do over-parametrised architectures work?

Feature extraction consists of expanding the representational dimension such that the expanded features are more likely to be linearly separable; data points in higher dimensional space are more likely to be linearly separable due to the increase in the number of possible separating planes.

Earlier machine learning practitioners relied on high quality, hand crafted, and task specific features to build artificial intelligence models, but with the advent of Deep Learning, the models are able to extract the generic features automatically. Some common approaches used in feature extraction algorithms are highlighted below:

- Space tiling
- Random Projections
- Polynomial Classifier (feature cross-products)
- Radial basis functions
- Kernel Machines

Because of the compositional nature of data, learned features have a hierarchy of representations with increasing level of abstractions. For example:

Images - At the most granular level, images can be thought of as pixels. Combination of pixels constitute edges which when combined forms textons (multi-edge shapes). Textons form motifs and motifs form parts of the image. By combining these parts together we get the final image.

Text - Similarly, there is an inherent hierarchy in textual data. Characters form words, when we combine words together we get word-groups, then clauses, then by combining clauses we get sentences. Sentences finally tell us what story is being conveyed.

Speech - In speech, samples compose bands, which compose sounds, which compose phones, then phonemes, then whole words, then sentences, thus showing a clear hierarchy in representation.

There are those who dismiss Deep Learning: if we can approximate any function with 2 layers, why have more?

For example: SVMs find a separating hyperplane "in the span of the data", meaning predictions are based on comparisons to training examples. SVMs are essentially a very simplistic 2 layer neural net, where the first layer defines "templates" and the second layer is a linear classifier. The problem with 2 layer fallacy is that the complexity and size of the middle layer is exponential in NN (to do well with a difficult task, need LOTS of templates).

An analogy is designing a circuit to compute a boolean function with no more than two layers of gates – we can compute any boolean function this way! But, the complexity and resources of the first layer (number of gates) quickly becomes infeasible for complex functions.

What is “deep”?

- An SVM isn’t deep because it only has two layers
- A classification tree isn’t deep because every layer analyses the same (raw) features

A deep network has several layers and uses them to build a hierarchy of features of increasing complexity

TensorFlow & Keras

Frameworks are a collection of packages and libraries which help in simplifying the overall programming experience for building a specific kind of application. Keras and TensorFlow are among the most popular frameworks when it comes to Deep Learning. TensorFlow is an end-to-end open-source platform for machine learning. It’s a comprehensive and flexible ecosystem of tools, libraries and other resources that provide workflows with high-level APIs. The framework offers various levels of concepts for you to choose the one you need to build and deploy machine learning models. For instance, if you need to do some large machine learning tasks, you can use the Distribution Strategy API in order to perform distributed hardware configurations and if you need a full production machine learning pipeline, you can simply use TensorFlow Extended (TFX). Some of the salient features are described below:

- Easy Model Building: TensorFlow offers multiple levels of abstraction to build and train models.
- Robust ML Production Anywhere: TensorFlow lets you train and deploy your model easily, no matter what language or platform you use.
- Powerful Experimentation For Research: TensorFlow gives you flexibility and control with features like the Keras Functional API and Model Subclassing API for the creation of complex topologies.

Keras, on the other hand, is a high-level neural networks library that is running on the top of TensorFlow, CNTK, and Theano. Using Keras in deep learning allows for easy and fast prototyping as well as running seamlessly on CPU and GPU. This framework is written in Python code which is easy to debug and allows ease for extensibility. The main advantages of Keras are described below

- User-Friendly: Keras has a simple, consistent interface optimized for common use cases which provides clear and actionable feedback for user errors.
- Modular and Composable: Keras models are made by connecting configurable building blocks together, with few restrictions.
- Easy To Extend: With the help of Keras, you can easily write custom building blocks for new ideas and researches.
- Easy To Use: Keras offers consistent & simple APIs which helps in minimizing the number of user actions required for common use cases, also it provides clear and actionable feedback upon user error.

So Should You Go For Keras Or TensorFlow?

There are several differences between these two frameworks. Keras is a neural network library while TensorFlow is the open-source library for a number of various tasks in machine learning. TensorFlow provides both high-level and low-level APIs while Keras provides only high-level APIs. In terms of flexibility, Tensorflow’s eager execution allows for immediate iteration along with intuitive debugging. Keras offers simple and consistent high-level APIs and follows best practices to reduce the cognitive load for the users. Both frameworks thus provide high-level APIs for building and training models with ease. Keras is built in Python which makes it way more user-friendly than TensorFlow.

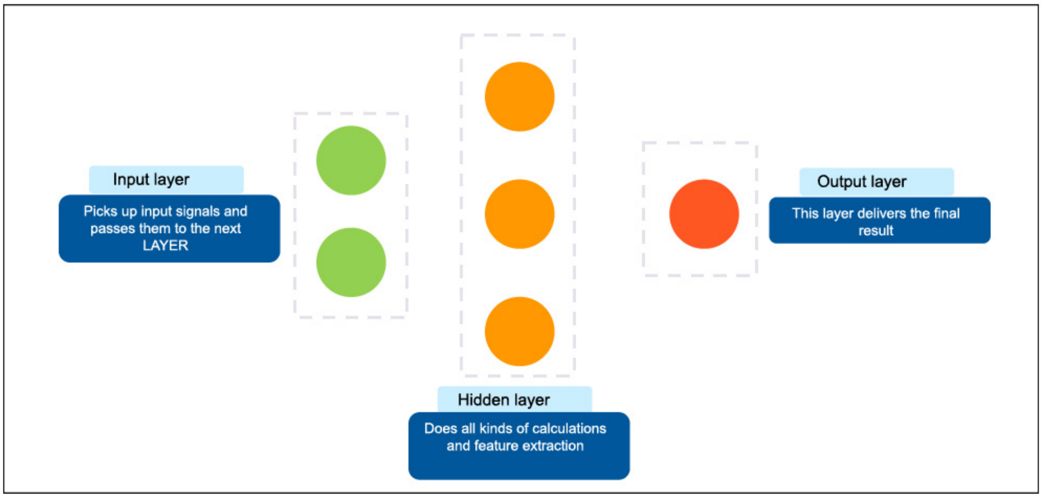
CHAPTER TWO: ARTIFICIAL NUERAL NETWORK

Neural networks can perform the following tasks:

- Translate text
- Identify faces
- Recognize speech
- Read handwritten text
- Control robots
- And a lot more

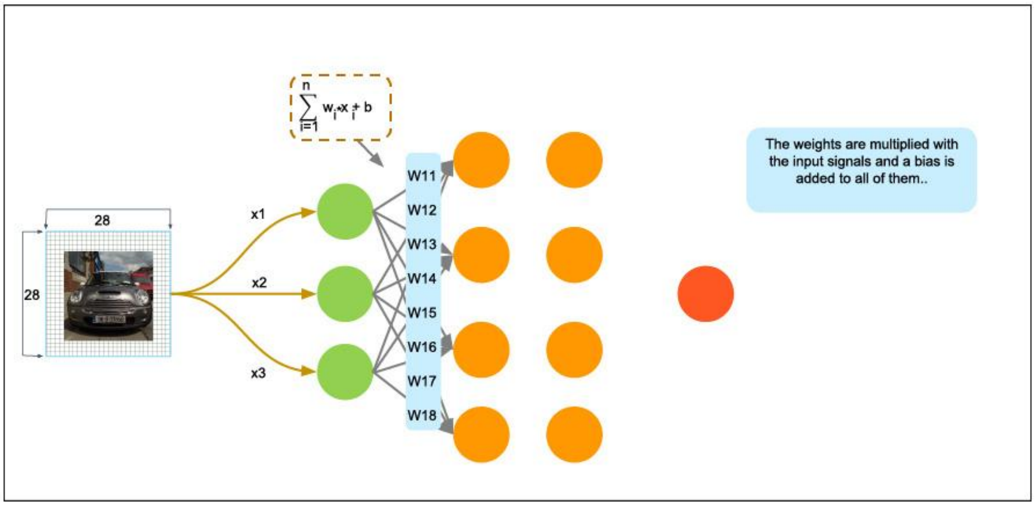
Working of Neural Network

A neural network is usually described as having different layers. The first layer is the input layer, it picks up the input signals and passes them to the next layer. The next layer does all kinds of calculations and feature extractions—it's called the hidden layer. Often, there will be more than one hidden layer. And finally, there's an output layer, which delivers the final result.


Input layer
Picks up input signals and passes them to the next LAYER
Hidden layer
Does all kinds of calculations and feature extraction
Output layer
This layer delivers the final result

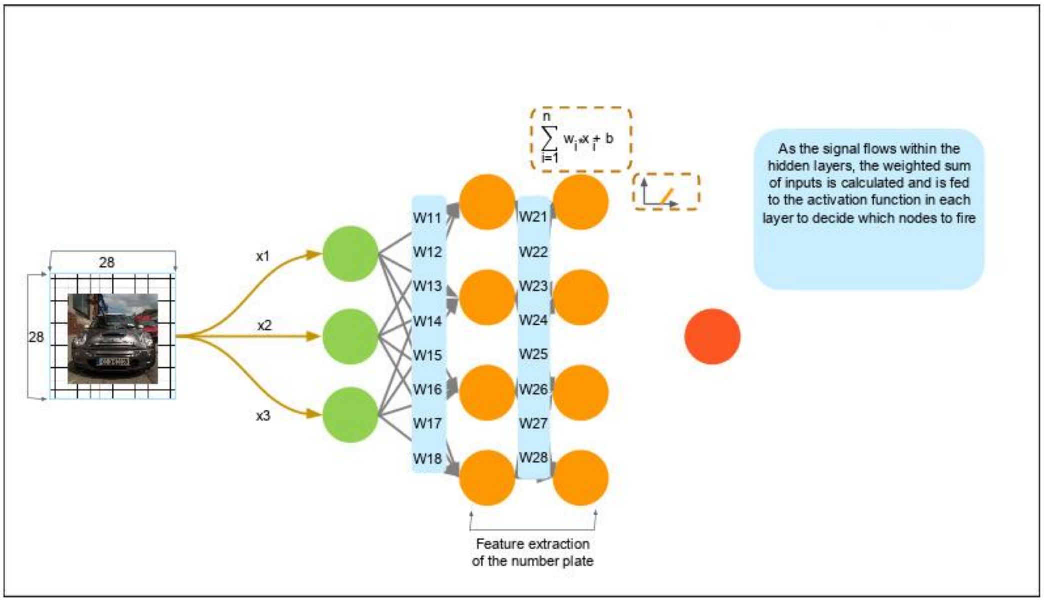
Let's take the real-life example of how traffic cameras identify license plates and speeding vehicles on the road. The picture itself is 28 by 28 pixels, and the image is fed as an input to identify the license plate. Each neuron has a number, called activation, which represents the grayscale value of the corresponding pixel, ranging from 0 to 1—it's 1 for a white pixel and 0 for a black pixel. Each neuron is lit up when its activation is close to 1.

Pixels in the form of arrays are fed into the input layer. If your image is bigger than 28 by 28 pixels, you must shrink it down, because you can't change the size of the input layer. In our example, we'll name the inputs as X1, X2, and X3. Each of those represents one of the pixels coming in. The input layer then passes the input to the hidden layer. The interconnections are assigned weights at random. The weights are multiplied with the input signal, and a bias is added to all of them.



$\sum_{i=1}^{n} w_i * x_i + b$
The weights are multiplied with the input signals and a bias is added to all of them..
28
28
x1
x2
x3
W11
W12
W13
W14
W15
W16
W17
W18

The weighted sum of the inputs is fed as input to the activation function, to decide which nodes to fire for feature extraction. As the signal flows within the hidden layers, the weighted sum of inputs is calculated and is fed to the activation function in each layer to decide which nodes to fire.

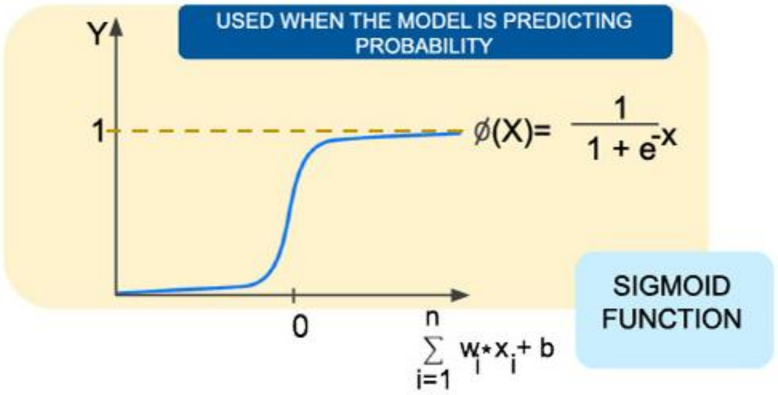


$\sum_{i=1}^{n} w_i * x_i + b$
As the signal flows within the hidden layers, the weighted sum of inputs is calculated and is fed to the activation function in each layer to decide which nodes to fire
28
28
x1
x2
x3
W11
W12
W13
W14
W15
W16
W17
W18
W21
W22
W23
W24
W25
W26
W27
W28
Feature extraction of the number plate

There are different types of activation functions.

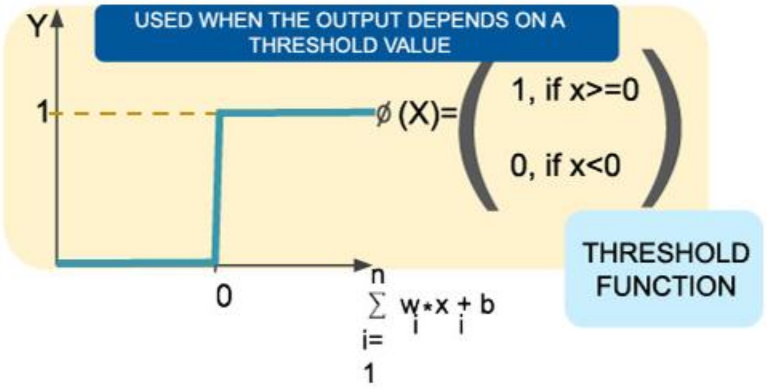
Sigmoid Function

The sigmoid function is used when the model is predicting probability.



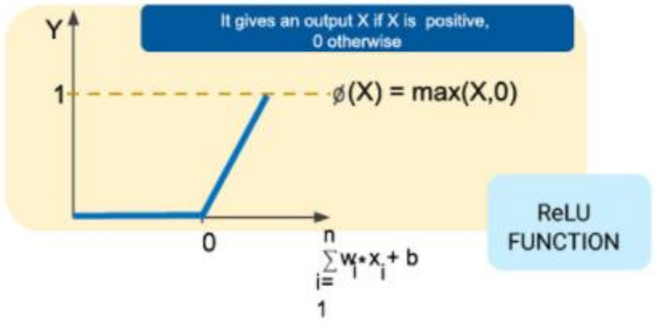
Threshold Function

The threshold function is used when you don’t want to worry about the uncertainty in the middle.



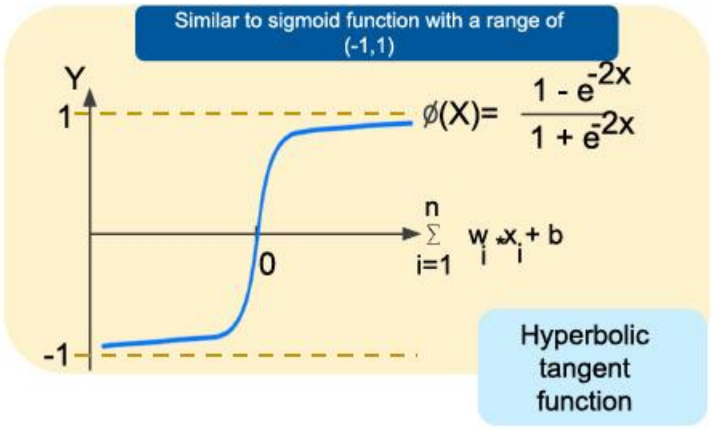
ReLU (rectified linear unit) Function

The ReLU (rectified linear unit) function gives the value but says if it’s over 1, then it will just be 1, and if it’s less than 0, it will just be 0. The ReLU function is most commonly used these days.



Hyperbolic Tangent Function

The hyperbolic tangent function is similar to the sigmoid function but has a range of -1 to 1.



Types of Neural Networks

The different types of neural networks are discussed below:

Feed-forward Neural Network

This is the simplest form of ANN (artificial neural network); data travels only in one direction (input to output). This is the example we just looked at. When you actually use it, it’s fast; when you’re training it, it takes a while. Almost all vision and speech recognition applications use some form of this type of neural network.

Radial Basis Functions Neural Network

This model classifies the data point based on its distance from a center point. If you don’t have training data, for example, you’ll want to group things and create a center point. The network looks for data points that are similar to each other and groups them. One of the applications for this is power restoration systems.

Kohonen Self-organizing Neural Network

Vectors of random input are input to a discrete map comprised of neurons. Vectors are also called dimensions or planes. Applications include using it to recognize patterns in data like a medical analysis.

Recurrent Neural Network

In this type, the hidden layer saves its output to be used for future prediction. The output becomes part of its new input. Applications include text-to-speech conversion.

Convolution Neural Network

In this type, the input features are taken in batches—as if they pass through a filter. This allows the network to remember an image in parts. Applications include signal and image processing, such as facial recognition.

Modular Neural Network

This is composed of a collection of different neural networks working together to get the output. This is cutting-edge and is still in the research phase.