1. Computer Vision and Visual Perception

Any AI system can perceive the environment and take actions based on its perceptions.

What is Computer vision?

Computer vision is a field of artificial intelligence (AI) and computer science that focuses on enabling computers to interpret and understand visual information from the world, much like humans do with their eyes and brain.

It involves in the *development of algorithms and techniques* that allow machines to extract meaningful insights and make decisions based on images or video data.

What is visual perception?

"**Visual perception** is the act of observing patterns and objects through sight or visual input. "

Visual perception is also defined as the ability to interpret the surrounding environment through vision (photopic vision (daytime vision), colour vision, scotopic vision (night vision), and mesopic vision (twilight vision)) using light in the visible spectrum reflected by objects in the environment.

For example, consider autonomous vehicle, visual perception means understanding the surrounding objects and their specific details
—such as pedestrians, or whether there is a particular lane the vehicle needs to be centred in.
—and detecting traffic signs and understanding what they mean.

Fig 1a. Visual perception

2. Types of Vision Systems:

Traditional image-processing techniques were considered as CV systems, but that is not totally accurate**.** Processing an image by a machine is far different from that machine understanding, what is happening within the image, which is not a trivial task. Image processing is now just a piece of a bigger, more complex system that aims to interpret image content.

a. Human Vision Systems:

The vision system for humans, animals and most living organisms consists of a sensor or an eye to capture the image and a brain to process & interpret the image. The system outputs a prediction of the image components based on the data extracted from the image.
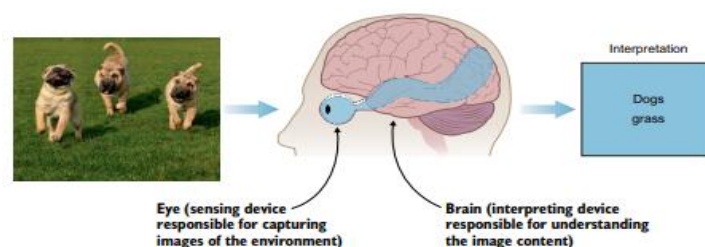

Figure2a: The human vision system uses the eye and brain to sense and interpret an image.

human vision system helps to interpret the image of dogs as shown in figure2a looks at it and directly understands that the image consists of a bunch of dogs (three, to be specific). It comes pretty natural to us to classify and detect objects in this image because we have been trained over the years to identify dogs.

b. AI Vision Systems:

To replicate human vision, two key components are required: a sensing device that mimics the eye's functionality, and a sophisticated algorithm that replicates the brain's ability to interpret and categorize visual content as shown in figure2b.
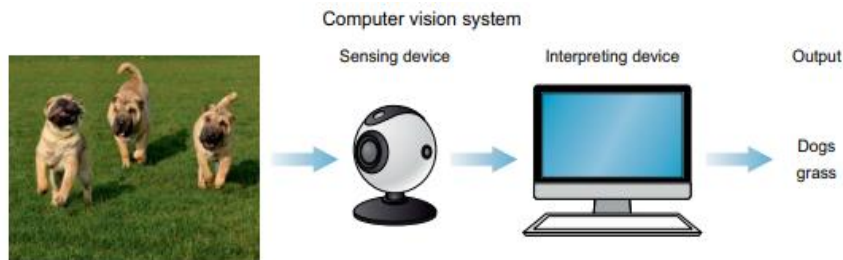

Figure2b : The components of the computer vision system are a sensing device and an interpreting device.

Example in Machine Learning Perspective:

- Learn to identify dogs by seeing labelled images.
  In supervised learning labelled data, data for which target is known.
  Eg: Shown a sample image of a dog and told that it was a dog. The human brain learned to associate the features and say the label: dog.
- Predict the classification for different object, a horse, shown and ask to identify. First, human brain thought it was a dog, because it hadn't seen horses before, and there will be a confusion of horse features with dog features. So, the prediction goes wrong.
- Now the brain starts adjusting the parameters to learn horse features. Now it will understand "Yes, both have four legs, but the horse's legs are longer. Longer legs indicate a horse." So, the brain starts learning until it doesn't make no mistakes. This is called training by trial and error.

Comparison of Image Processing Vs Computer Vision:

Image Processing	Computer Vision
Image processing is mainly focused on processing the raw input images to enhance them or preparing them to do other tasks	Computer vision is focused on extracting information from the input images or videos to have a proper understanding of them to predict the visual input like human brain.
Image processing uses methods like Anisotropic diffusion, Hidden Markov models, Independent component analysis, Different Filtering etc.	Image processing is one of the methods that is used for computer vision along with other Machine learning techniques, CNN etc.
Image Processing is a subset of Computer Vision.	Computer Vision is a superset of Image Processing.
Examples of some Image Processing applications are- Rescaling image (Digital Zoom), Correcting illumination, Changing tones etc.	Examples of some Computer Vision applications are- Object detection, Face detection, Hand writing recognition etc.

3. Basic components for Human and Machine Vision Systems:

Both human and machine vision systems have two components a *sensing device* and an *interpreting device*. Each is tailored to fulfil a specific task.

a. Sensing devices:

The important aspect in selecting a best sensing device to capture the surroundings of a specific environment, whether that is a camera, radar, X-ray, CT scan, Lidar, or a combination of devices to provide the full scene of an environment to fulfill the task at hand.

| Camera | X Ray | Lidar in Drones |

For example, the main goal of the *autonomous vehicle (AV)* vision system is to allow the car to understand the environment around it and move from point A to point B safely and in a timely manner. To fulfill this goal, vehicles are equipped vehicles the role of cameras and sensors are to detect 360 degrees of movement—pedestrians, cyclists, vehicles, roadwork, and other objects— from up to three football fields away.

Example2 for sensing devices used in *self-driving cars* is to perceive the surrounding area:

- LIDAR, a radar-like technique, uses invisible pulses of light to create a high resolution 3D map of the surrounding area.
- Cameras can see street signs and road markings but cannot measure distance.
- Radar can measure distance and velocity but cannot see in fine detail.

Medical diagnosis applications use X-rays or CT scans as sensing devices.

b. Interpreting devices

Interpreting devices in vision systems take the output image from the sensing device and learn features and patterns to identify objects. So, we need to build a brain. Thus, we have ANNs and CNNs

Artificial neural networks (ANNs) as shown in **Fig.3. a.** The analogy between the biological neurons and artificial systems has a main processing element, a neuron, with input signals (x1, x2, …, xn) and an output.
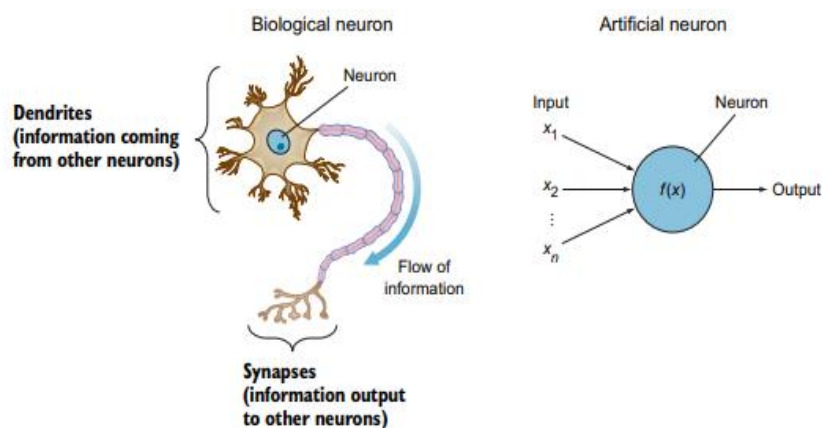

Figure.3. a: Similarities between biological neurons and artificial systems

Another system that connects millions of these neurons stacked in layers that is each neuron is connected to thousands of other neurons, yielding a learning behaviour.

Building a multilayer neural network is called Deep Learning as shown in **Fig.3. a.** The DL methods learn the representations through a sequence of transformations of data through layers of neurons. The different DL architectures are ANNs and Convolutional neural networks CNNs.
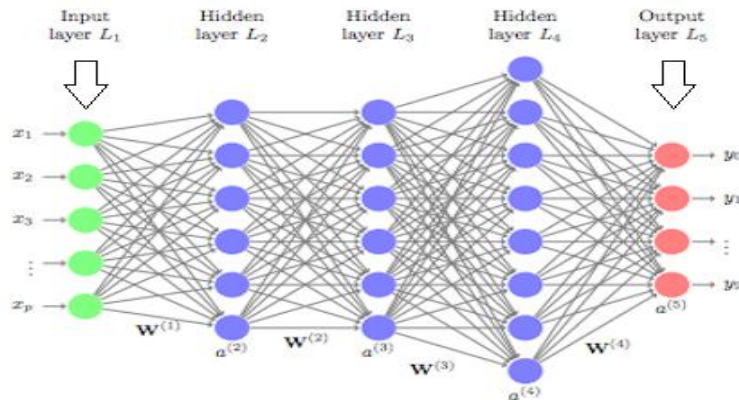

Fig. 3.b: Deep learning involves layers of neurons in a network.

4. Applications of Computer Vision:

Computers are able to recognize human faces in images decades ago, but now AI systems are able to classify objects in photos and videos because of the computational power and the amount of data availability.

The deep neural networks are successful for CV, natural language processing and voice user interface tasks.

AI and DL have managed to achieve best performance on many complex visual perception tasks like image or video search, captioning classification, and object detection.

a. Image classification is the task of assigning to an image a label from a predefined set of categories. A Convolutional neural network is a neural network type that truly shines in processing and classifying images in many different applications:

Lung cancer diagnosis—Lung cancer is a growing problem. The main reason lung

cancer is very dangerous is that when it is diagnosed, it is usually in the middle or of late stages. When diagnosing lung cancer, doctors typically use their eyes to examine CT scan images, looking for small nodules in the lungs. In the early stages, the nodules are usually very small and hard to spot. (figure 4a).
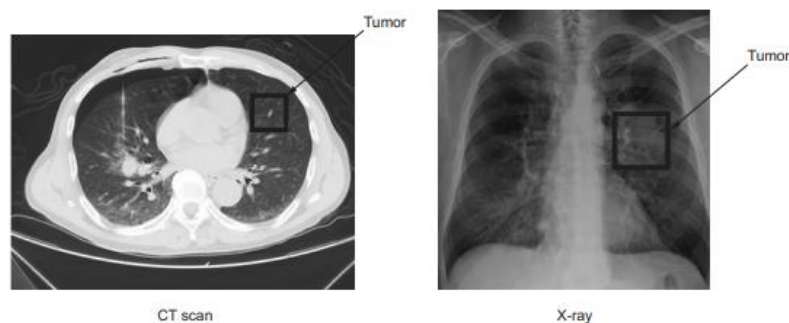

Figure 4a: X-ray images to identify tumors in earlier stages of development.

b. Traffic sign recognition—Traditionally, standard CV methods were employed to detect and classify traffic signs, but this approach required time-consuming manual work to handcraft important features in images. Instead, by applying DL to this problem, we can create a model that reliably classifies traffic signs, learning to identify the most appropriate features for this problem by itself (figure 4b).

Figure4b : Vision systems can detect traffic signs with very high performance.

c. Object detection and localization:

Localization involves determining the object's position and outlining it with a bounding box. On the other hand, object detection goes beyond by identifying and categorizing all objects within the image. Each object is assigned a class label and enclosed with a bounding box. To do that, we can build object detection systems like YOLO (you only look once), SSD (single-shot detector), and Faster R-CNN, which not only classify images but also can locate and detect each object in images that contain multiple objects as shown in (figure 4c).
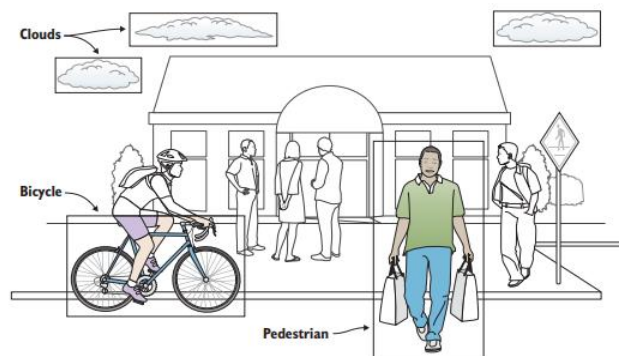

Figure.4c: Deep learning systems can segment objects in an image.

d. Generating art (style transfer)

Neural style transfer, one of the most interesting CV applications, is used to transfer the style from one image to another. Neural style transfer is a method that optimizes three images: a content image, a style reference image (like an artwork), and an input image you want to style. It combines these images to make the input image adopt the content of one image while being artistically transformed to match the style of another image, like a painting. (figure.4d).

Figure.4d: Style transfer onto the original image, producing a piece of art that feels

e. Creating images

A new DL model that can imagine new things called generative adversarial networks (GANs). GANs are sophisticated DL models that generate stunningly accurate synthesized images of objects, people, and places, among other things. If you give them a set of images, they can make entirely new, realistic-looking images. For example, StackGAN is one of the GAN architecture variations that can use a textual description of an object to generate a high-resolution image of the object matching that description. This is not just running an image search on a database. These "photos" have never been seen before and are totally imaginary (figure 4e)

Figure 4e: Generative adversarial networks (GANS)
can create new, "made-up" images from a set of existing images.

AI-generated artwork featuring a fictional person named Edmond de Belamy sold for $432,500. The artwork was created by a team of three 25-year-old French students using GANs. The network was trained on a dataset of 15,000 portraits painted between the fourteenth and twentieth century's, and then it created one of its own. The team printed the image, framed it, and signed it with part of a GAN algorithm.

f. Face recognition:

Face recognition (FR) allows us to exactly identify or tag an image of a person. Day-to day applications include searching for celebrities on the web and auto-tagging friends and family in images. Face recognition is a form of fine-grained classification.
Face Recognition system is categorized in two modes:
Face identification involves one-to-many matches that compare a query face image against all the template images in the database to determine the identity of the query face. Another face recognition scenario involves a watchlist check by city authorities, where a query face is matched to a list of suspects (one-to-few matches).

Face verification involves a one-to-one match that compares a query face image against a template face image whose identity is being claimed (figure below)
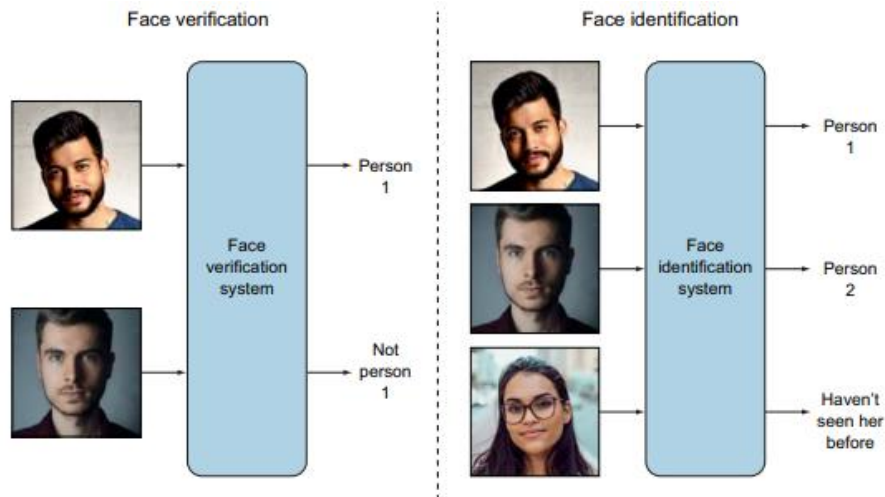

Figure: Example of face verification (left) and face recognition (right)

g. Image recommendation system:

In this task, a user seeks to find similar images with respect to a given query image. Shopping websites provide product suggestions (via images) based on the selection of a particular product, for example, showing a variety of shoes similar to those the user selected. An example of an apparel search is shown in figure below.
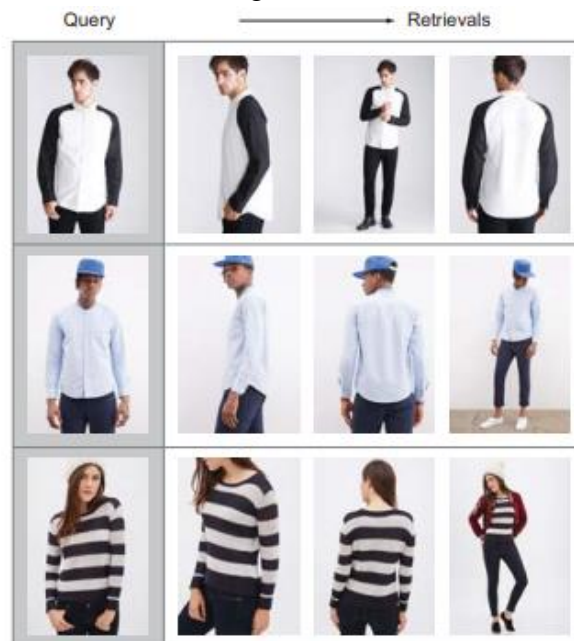

Figure: Apparel search.

5. Computer Vision Pipeline Process

The vision systems are composed of two main components,
* Sensing devices capture the image or signals through camera or sensors.
* The interpreting device is used to process and understand images.

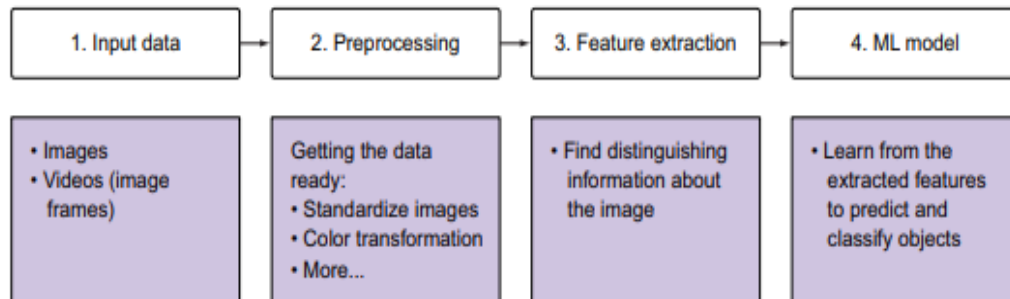The interpreting process uses "computer vision pipeline" that consists of four steps as shown in **Fig.5.a**

Fig.5.a; The computer vision pipeline

Computer Vision pipeline for image classification:

An image classifier is an algorithm that takes in an image as input and predicts a label or "class" that identifies that image. A class (also called a category) in machine learning is the output category of your data.

Example For a motorcycle image, if we want the model to predict the probability of the object from the following classes: motorcycle, car, and dog as shown in **Fig.5.b.**
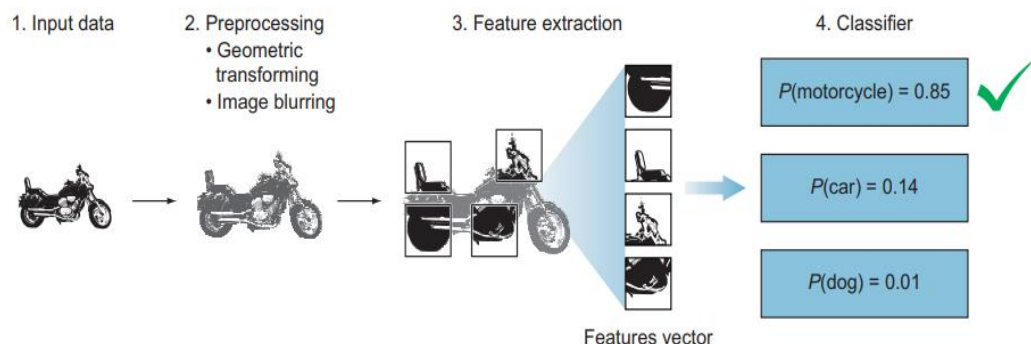

Fig.5.b. Predict the probability of the motorcycle object from the motorcycle, car, and dog classes

a. Input Data (Image):

For Computer Vision applications, the input data is either images or video data.
* The images are of two types, Grayscale (BW) and colour images (RGB) and videos are stacked sequential frames of images.
* An *image* can be represented as a function of two variables x and y, which define a two-dimensional area.
* A *digital image* is made of a grid of pixels.
* The pixel is the smallest unit of an image.
* Every image consists of a set of pixels in which their values represent the intensity of light that appears in a given place in the image.
* The image coordinate system is similar to the Cartesian coordinate system:

i) Grayscale Image:

In gray scale images are two-dimensional and lie on the x-y plane. The gray scale image has pixel values range from 0 to 255. Since the pixel value represents the intensity of light, the value 0 represents very dark pixels (black), 255 is very bright (white), and the values in between represent the intensity on the grayscale. **(Fig.5.c)**
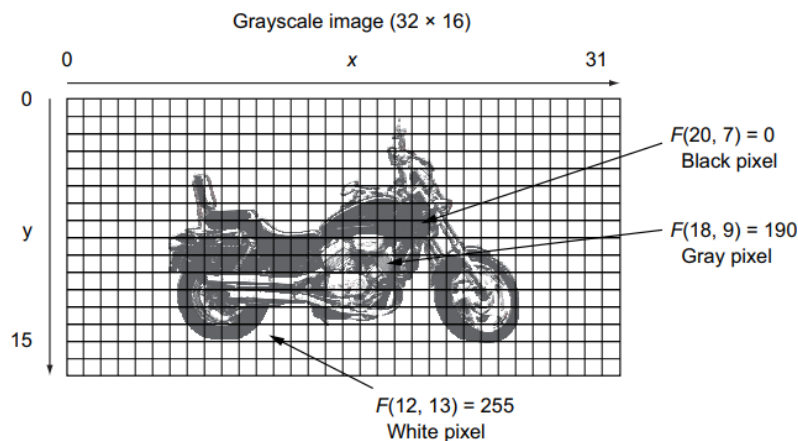
Example: motorcycle and the grid of pixel

Grayscale image (32 × 16)

F(20, 7) = 0
Black pixel

F(18, 9) = 190
Gray pixel

F(12, 13) = 255
White pixel

Fig 5.c Images consists of raw building blocks called pixels.

Grayscale => F(x, y) gives the intensity at position (x, y).
The pixel located at x = 12 and y = 13 is white; F(12, 13) = 255.

Mathematically an image is a function of F(x, y) and transforms it to a new image function G(x, y). The image transformation examples:

Application	Transformation
Darken the image.	G(x, y) = 0.5 * F(x, y)
Brighten the image.	G(x, y) = 2 * F(x, y)
Move an object down 150 pixels.	G(x, y) = F(x, y + 150)
Remove the gray in an image to transform the image into black and white.	G(x, y) = { 0 if F(x, y) < 130, 255 otherwise }

ii) Color images:

In colour images, the value of the pixel is represented by three numbers: the intensity of red, intensity of green, and intensity of blue In an RGB system, **Color image in RGB => F(x, y) = [red (x, y), green (x, y), blue (x, y)].** Color image is having three matrices stacked on top of each other; that's why it's a 3D matrix. The dimensionality of 700 × 700 color images is (700, 700, 3). The first matrix represents the red channel; then each element of that matrix represents an intensity of red color in that pixel, and likewise with green and blue.**(Fig 5.d)**
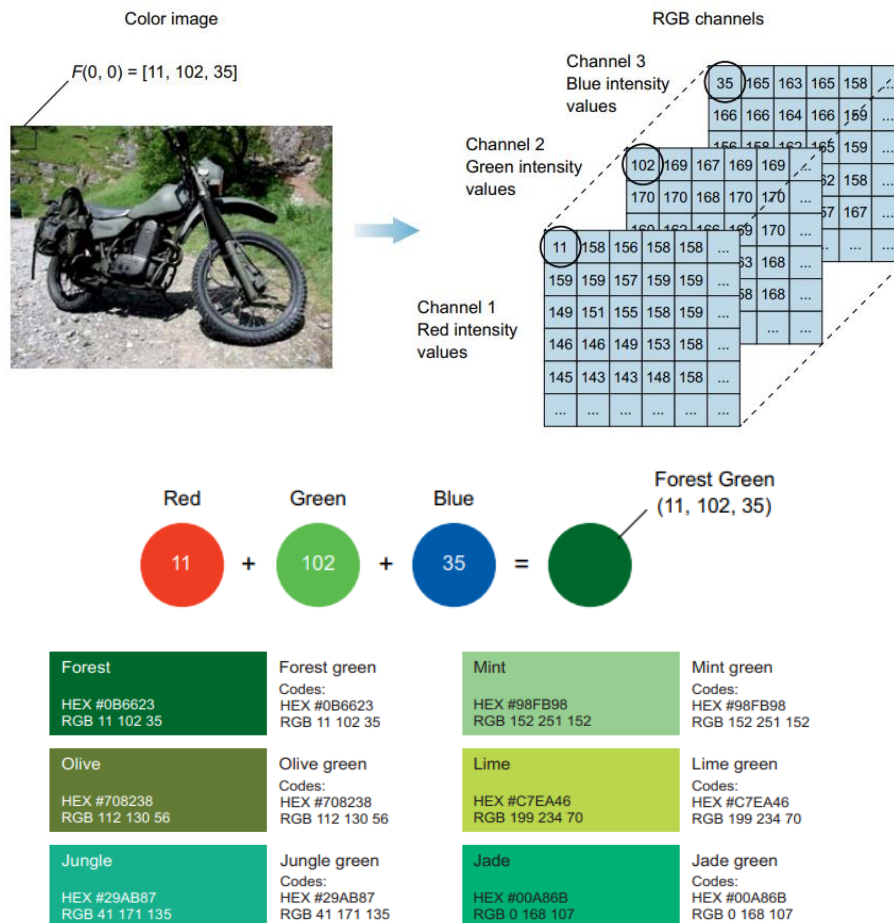
Fig .5.d. Color images and representation

b. Image Pre-processing:

Image data pre-processing converts image data into a form that allows machine learning algorithms to solve it. It is often used to increase a model's accuracy, as well as reduce its complexity. The techniques used to pre-process image data are image resizing, converting images to grayscale, and image augmentation etc., when the images exist in different formats, i.e., natural, fake, grayscale, etc., we need to standardize them before feeding them into a neural network.

The important steps in image pre-processing techniques are

- Grayscale conversion
- Normalization
- Data Augmentation
- Image standardization

i) Grayscale conversion simply converting images from colored to black and white as shown in **Fig.5.f.** It is normally used to reduced number of pixels that need to be processed and reduce computation complexity in machine learning algorithms.

This could be a not good approach for applications depend on color information because it losses information in conversion. Since most pictures don't need color to be recognized, it is wise to use grayscale, which reduces the number of pixels in an image, thus, reducing the computations required.

Fig.5.f. color image transformation to gray scale

Converting images to grayscale might not always be practical in solving some problems.

For examples where it would be impractical to use grayscale include: *trafficlights*, *healthcare diagnosis*, *autonomous vehicles*, *agriculture*, etc. The best way to know whether to use it or not depends on your human visual ability to identify an object without color.

ii) Normalization

Image normalization is a typical process in image processing that changes the range of pixel intensity values.
For example, when we perform a function that produces a normalization of an input image (grayscale or RGB). Then, we understand a representation of the range of values of the scale of the image represented between 0 and 255. i.e., very dark images become clearer.
The linear normalization of a digital image is performed according to the formula
$$\text{Output\_channel} = 255 * (\text{Input\_channel - min}) / (\text{max-min})$$
For grayscale image, normalize using one channel and color images normalize a RGB (3 channels)
Examples:
The left image depicts the original image is too dark and results clear after the normalization process.

Example 2: The left image depicts the right-side original image is very bright results with better contrast after the normalization process.

iii) Data augmentation:

Data augmentation helps in preventing a neural network from learning irrelevant features. This results in better model performance.

Data augmentation is the process of making minor alterations to existing data to increase its diversity without collecting new data. It is a technique used for enlarging a dataset.

There are two types of augmentation are *Offline augmentation* is used for small datasets. *Online augmentation* is used for large datasets. It is normally applied in real-time.

Standard data augmentation techniques include horizontal & vertical flipping, rotation, cropping, shearing, etc. as shown in **Fig.5.h**

Shifting is the process of shifting image pixels horizontally or vertically.

Flipping reverses, the rows or columns of pixels in either vertical or horizontal cases, respectively.

Rotation process involves rotating an image by a specified degree.

Changing brightness is the process of increasing or decreasing image contrast.

Cropping is the process of creating a random subset of an original image which is then resized to the size of the original image.

Scaling image can be scaled either inward or outward. When scaling an image outward, the image becomes more significant than the original and vise versa.
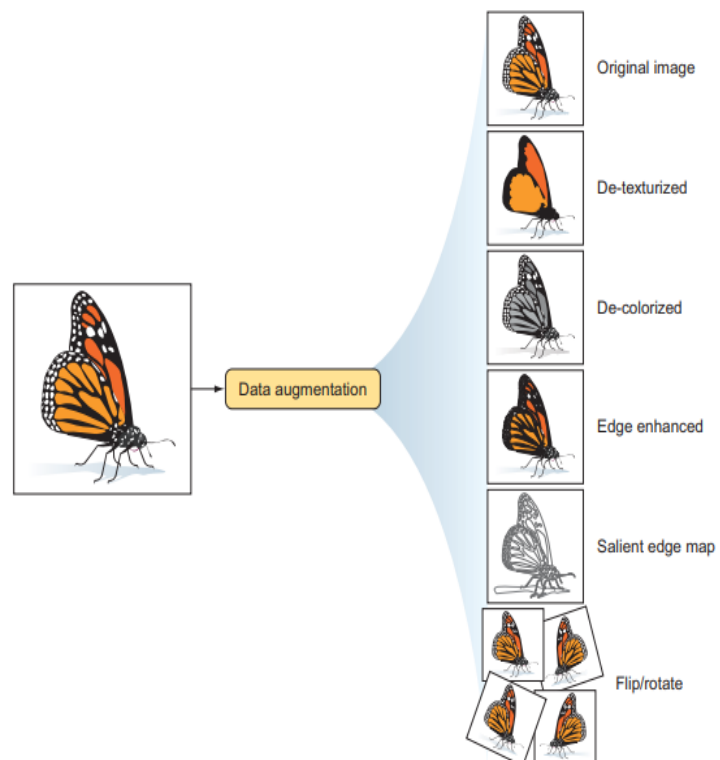
Fig.5. h. Data Augmentation

iv) Standardizing the images:

Standardization is a method that scales and pre-processes images to have similar heights and widths. It re-scales data to have a standard deviation of 1 (unit variance) and a mean of 0. Standardization helps to improve the quality and consistency of data. one important constraint that exists in some ML algorithms, such as CNNs, is the need to resize the images in your dataset to unified dimensions. This implies that your images must be pre-processed and scaled to have identical widths and heights before being fed to the learning algorithm

No Free Lunch Algorithm:

This is a phrase that was introduced by David Wolpert and William Macready in "No Free Lunch Theorems for Optimizations" (IEEE Transactions on Evolutionary Computation 1, 67).

- When working on ML projects, you will need to make many choices like building your neural network architecture, tuning hyperparameters, and applying the appropriate data preprocessing techniques. While there are some rule-of-thumb approaches to tackle certain problems, there is really no single recipe that is guaranteed to work well in all situations.
- You must make certain assumptions about the dataset and the problem you are trying to solve. For some datasets, it is best to convert the colored images to grayscale, while for other datasets, you might need to keep or adjust the color images.
- The good news is that, unlike traditional machine learning, DL algorithms require minimum data preprocessing because, as you will see soon, neural networks do most of the heavy lifting in processing an image and extracting features.

C. Feature extraction

Features are parts or patterns of an object in an image that help to identify image. The entire DL model works around the idea of extracting useful features that clearly define the objects in the image.

A raw data (image) is transformed into a feature vector using learning algorithm, which can learn the characteristics of the object.

For example — a square has 4 corners and 4 edges, they can be called features of the square, and they help us humans identify it's a square. Features include properties like corners, edges, regions of interest points, ridges, etc.

Example: when we feed the raw input image of a motorcycle into a feature extraction algorithm. the extraction algorithm produces a vector that contains a list of features as shown in figure below. This feature vector is a 1D array that makes a robust representation of the object.
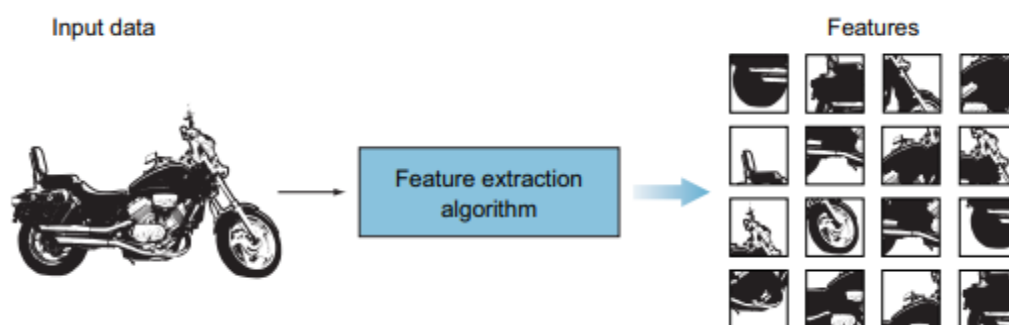

Fig.5. j. Feature Extraction

Feature extraction for Traditional Machine Learning:

The Process relies on domain knowledge (or partner with domain experts) to extract features that make ML algorithms work better. Feeding the produced features to a classifier like a support vector machine (SVM) or AdaBoost to predict the output (**Fig 5.l**).
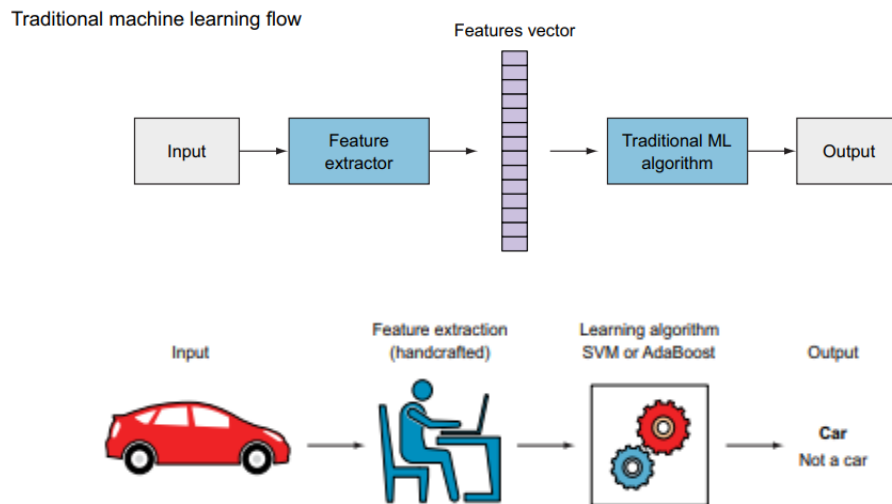
Fig 5.l: *Traditional machine learning algorithms require handcrafted feature extraction.*

For traditional ML problems, designers need spend a large amount of time in manual feature selection and engineering. Examples for CV uses Handcrafted feature Extractor Algorithms like:

Histogram of oriented gradients (HOG)- The technique counts occurrences of gradient orientation in localized portions of an image.
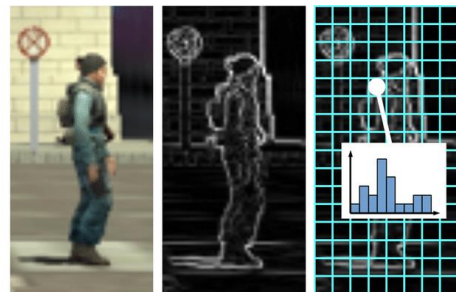
Haar Cascades extracts the features which are scalable, rectangular frames that are used to compare how pixels relate to each other; specifically, how dark one is to the other. There are three basic types of Haar features: Edge features, Line features, and Four-rectangle features.
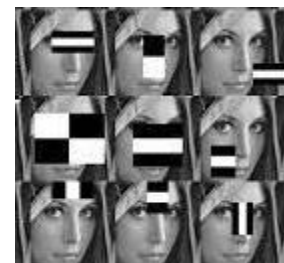
Harris Corner Detection Uses a Gaussian window function to detect corners.

Shi-Tomasi Corner Detector — The modified the scoring function used in Harris Corner Detection to achieve a better corner detection technique.
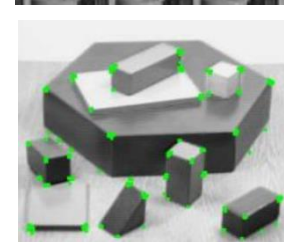
Scale-Invariant Feature Transform (SIFT) —is used to detect and describe local features in images.

Speeded-Up Robust Features (SURF) — This is a faster version of SIFT as the name says.
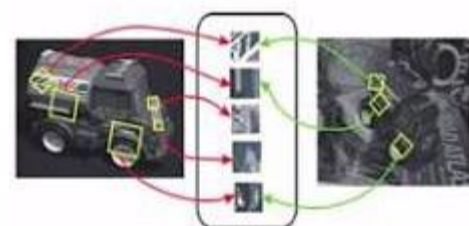
Feature extraction in Deep Learning:

In deep Learning, no need to manually extract features from the image. The network extract features automatically and learns their importance on the output by applying weights to its connections. When a raw image is fed to the network, while passing through the network layers, identifies features/ patterns within the image as shown in figure (**Fig 5.m**). Neural networks can be thought of as feature extractors plus classifiers that are end-to-end trainable, as opposed to traditional ML models that use handcrafted features.
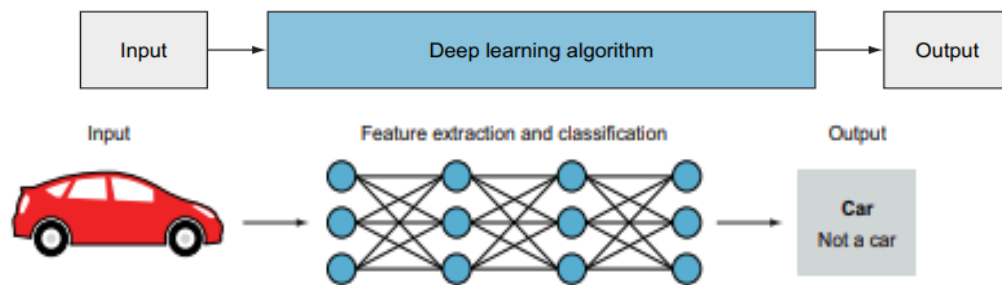
Deep learning flow

Fig.5.m : A DNN the input image through its layers to automatically extract features

D. Classification model:

The features vectors are fed into a classification model predicts the class of the image.

- First you see a wheel feature; could this be a car, a motorcycle, or a dog? Clearly it is not a dog, because dogs don't have wheels (at least, normal dogs, not robots). Then this could be an image of a car or a motorcycle.
- You move on to the next feature, the headlights. There is a higher probability that this is a motorcycle than a car.
- The next feature is rear mudguards—again, there is a higher probability that it is a motorcycle.
- The object has only two wheels; this is closer to a motorcycle.
- And you keep going through all the features like the body shape, pedal, and so on, until you arrive at a best guess of the object in the image.

The output of this process is the probability of each class., the dog has the lowest probability, 1%, whereas there is an 85% probability that this is a motorcycle. Although the model was able to predict the right class with the highest probability, it is still a little having confusion between cars and motorcycles because it was predicted as 14% chance of car. Since it is a motorcycle, we can say that our ML classification algorithm is 85% accurate.

The different approaches used to improve the accuracy of our model follow either step:
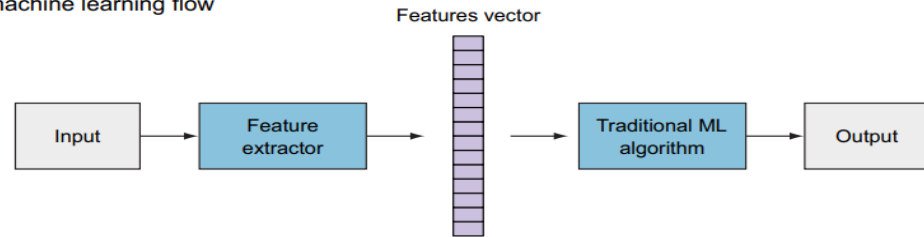- acquire more training images, or
- more processing to remove noise, or
- extract better features, or
- change the classifier algorithm and tune some hyperparameters, or
- even allow more training time.

6. Classifier learning algorithm:
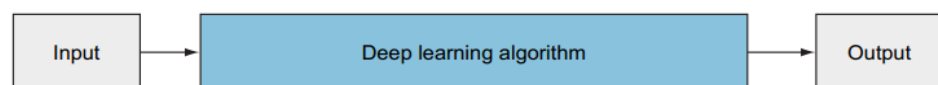
The classification task is done by two ways:

- By traditional ML algorithms like SVMs which might give result for some problems.

Fig : Traditional Classifier learning algorithm

- Deep neural network algorithms like CNNs give best results to classifying images for most complex problems.

Fig : Deep NN Classifier learning algorithm

Deep Learning Classifier:

The neural networks classifier automatically extracts useful features from your dataset(images), to predicts the class labels for new images.

The input images are passed through the hidden layers of the deep neural **network** to learn their features layer by layer. The deeper network with more hidden layers learns the features of the dataset will prone the model to overfitting and reduces the accuracy of the model. The last layer of the neural network is fully connected usually acts as the classifier that outputs the class label.

To improve the accuracy of the model, different approaches are used:
- acquire more training images, or
- more processing to remove noise, or
- extract better features, or
- change the classifier algorithm and tune some hyperparameters, or
- even allow more training time.

Deeplearning Flow:
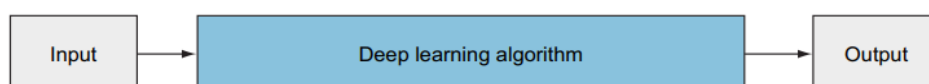
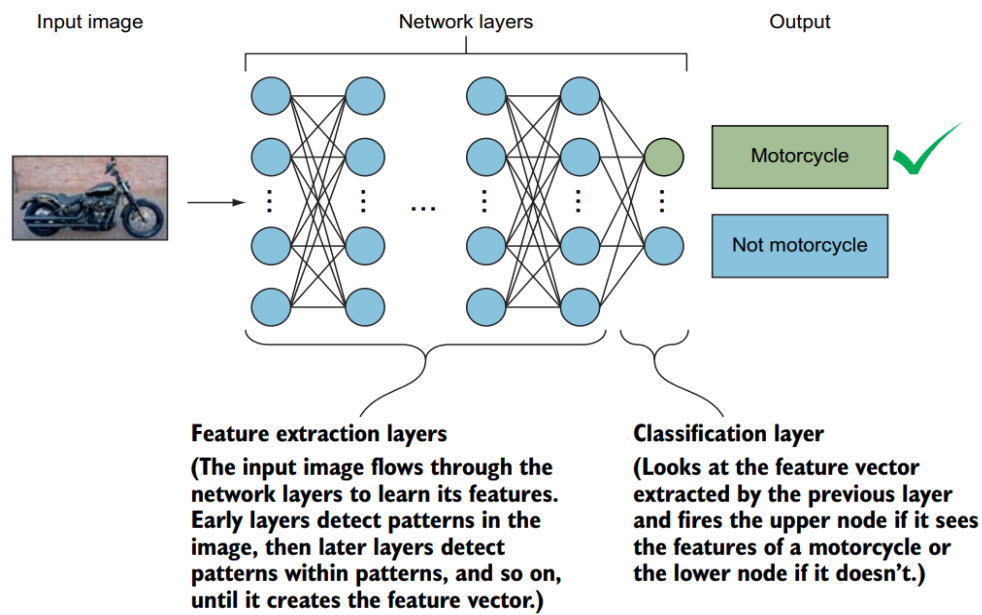
Fig : Deep NN Classifier learning algorithm

Fig : Deep NN Classifier learning algorithm

Types of Neural Networks:

7. Single layer Perceptron:

The deep Learning model uses neural network to extract features automatically by passing the input through hidden layers. Based on the depth, number of hidden layers and I/O capabilities neural network models are classified into different types. They are

1. Perceptron
2. Feed Forward Networks
3. Multi-Layer Perceptron
4. Convolutional Neural Networks
5. Advanced CNN- LNet, AlexNet, VGG.. Etc.,

The most basic component of the neural network is the perceptron, which consists of a single neuron. The perceptron functions are similar to a biological neuron.

Perceptron is a supervised deep learning model used for binary classification tasks. Performs two consecutive functions: it calculates the weighted sum of the inputs to represent the total strength of the input signals, and it applies a step function to the result to determine whether to fire the output 1 if the signal exceeds a certain threshold or 0 if the signal doesn't exceed the threshold.

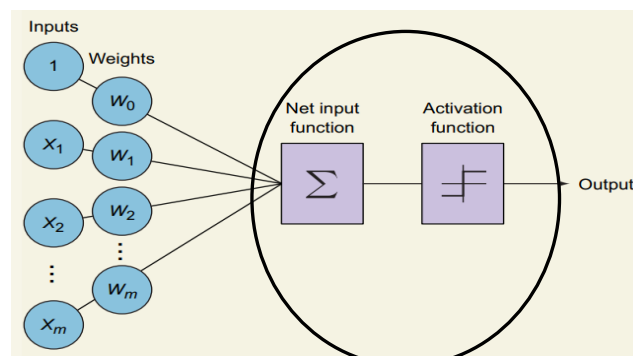**a. Perceptron architecture:** The perceptron is as shown in Fig. 6.a.

Fig. 6.a. Single Layer Perceptron

The Perceptron consists of

Input vector denoted by uppercase X (x1, x2, . . ., xn) fed to the neuron .

bias (b) is an extra weight used while learning and adjusting the neuron to minimize the cost function

Weights vector—Each x1 is assigned a weight value w1 that represents its importance to distinguish between different input data points.

Neuron functions—The calculations performed within the neuron to modulate the input signals: the weighted sum and step activation function.

Output—controlled by the type of activation function you choose for your network. There are different activation functions eg. a step function, the output is either 0 or 1. Other activation functions produce probability output or float numbers. The output node represents the perceptron prediction.

b. When to use Perceptron and How does they learn?

The single perceptron works with *linearly separable data*. This means the training data should be separated by a straight line as shown in **Fig 6.b**
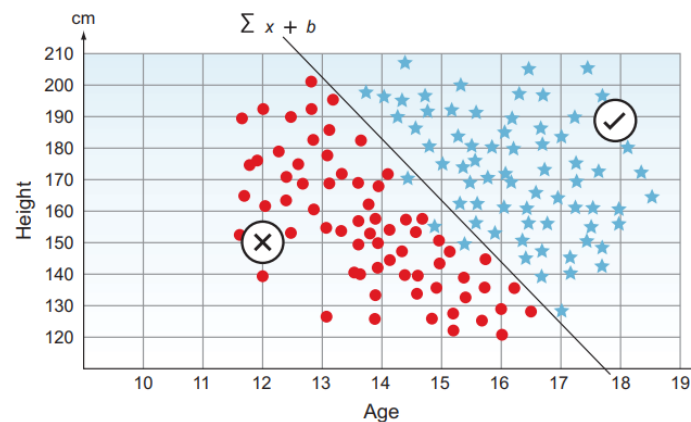

Fig 6.b. plot for best fit between Age and height

The perceptron uses trial and error to learn from its mistakes. It uses the weights as knobs by tuning their values up and down until the network is trained. That is the Weights are tuned up and down during the learning process to optimize the value of the loss function.

Steps of perceptron learning

1 The neuron calculates the weighted sum and applies the activation function to make a prediction yˆ . This is called the feed forward process.

$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + \cdots + x_n \cdot w_n + b$$

$$z = \sum x_i \cdot w_i + b \text{ (bias)}$$

$$\hat{y} = \text{activation}\left(\sum x_i \cdot w_i + b\right)$$

2 It compares the output prediction with the correct label to calculate the error

$$\text{error} = y - \hat{y}$$

3 It then updates the weight. If the prediction is too high, it adjusts the weight to make a lower prediction the next time, and vice versa.

4 Repeat!

This process is repeated many times, and the neuron continues to update the weights to improve its predictions until step 2 produces a very small error (close to zero),which means the neuron's prediction is very close to the correct value. At this point, we can stop the training and save the weight values that yielded the best results to apply to future cases where the outcome is unknown.

Problem:

A Neural network which takes binary valued inputs x1, x2 € {0,1} and the activation function is the threshold function h(x)=1 if x>0 else 0. Find which logical operation is done by neural network function f(x)=$8x_1+8x_2-4$? (hint: OR, AND, NAND, NOR, XOR)
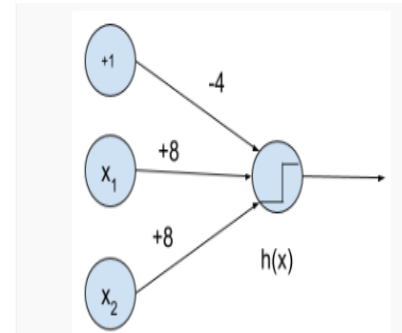
Here f(x) = $8x_1 + 8x_2$ -4

CASE I : x_1 = 0   and x_2 = 0
f(x) = 8 x 0 + 8 x 0 – 4
     = - 4
Since f(x) is negative, from the given threshold function h(x) :
    h(x) = 1 when f(x) > 0
    h(x) = 0 when f(x) < 0
    Hence output= 0

CASE II and III : x_1 = 0   and x_2 = 1
or x_1 = 1   and x_2 = 0
f(x) = 8 x 0 + 8 x 1 – 4
     = 4
Since f(x) is positive , from the given threshold function h(x) :
    h(x) = 1 when f(x) > 0
    Hence output= 1

Here f(x) = $8x_1+8x_2$ -4

CASE IV : x_1 = 1   and x_2 = 1
f(x) = 8 x 1 + 8 x 1 – 4
     = 12
Since f(x) is positive , from the given threshold function h(x) :
    h(x) = 1 when f(x) > 0
    Hence output= 1

	x2=0	x2=1
x1=0	0	1
x1=1	1	1

Hence its OR gate

C. How to Train a perceptron model to predict whether a player will be accepted into the college squad?

The first step is to collect all the data from previous years and train the perceptron to predict whether players will be accepted based on only two features (height and weight). The trained perceptron will find the best weights and bias values to produce the straight line that best separates the accepted from non-accepted (best fit).

The line has this equation: z = height · w1 + age · w2 + b. After the training is complete on the training data, we can start using the perceptron to predict with new players. When we get a player who is 150 cm in height and 12 years old, we compute the previous equation with the values (150, 12)as shown in **Fig .6.c**
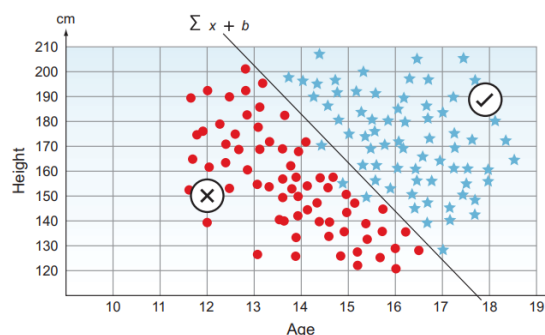

Fig 6.c plot for best fit between Age and height

8. Multi layer Perceptrons:

The deep Learning model uses neural network to extract features automatically by passing the input through hidden layers. Based on the depth, number of hidden layers and I/O capabilities neural network models are classified into different types. They are

1. Perceptron
2. Feed Forward Networks
3. Multi-Layer Perceptron
4. Convolutional Neural Networks
5. AdvancedCNN-
   LNet,AlexNet,VGG..Etc.,

The single perceptron works fine because our data was linearly separable. This means the training data can be separated by a straight line. But in real time isn't always that simple.

What happens when we have a more complex dataset that cannot be separated by a straight line (nonlinear dataset fig 7.a.))? We use Multi layer Perceptrons .
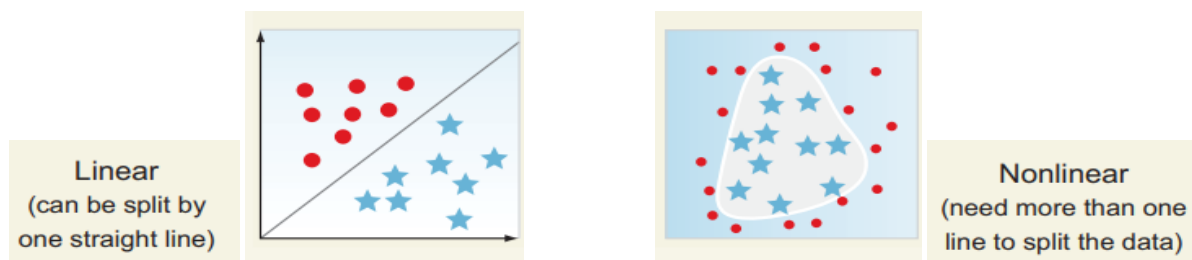

Fig .7.a Linear Data Vs Non Linear data

i) Multi layer Perceptron Architecture:

A very common neural network architecture is to stack the neurons in layers on top of each other, called hidden layers. Each layer has n number of neurons. Layers are connected to each other by weight connections. This leads to the multilayer perceptron (MLP) architecture in the **Fig 7.b.**
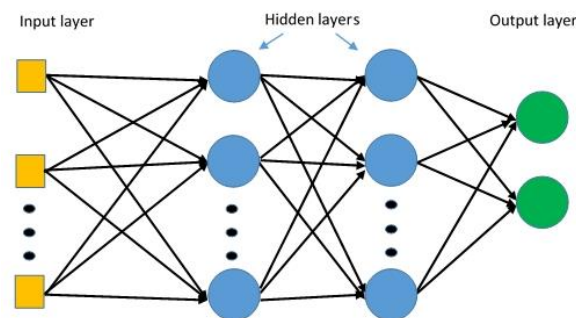

Fig 7.b. Multi Layer Perceptron(ANN)

The main components of the neural network architecture are

Input Layer

It is the initial or starting layer of the Multilayer perceptron. It takes input from the training data set and forwards it to the hidden layer. There are n input nodes in the input layer. The number of input nodes depends on the number of dataset features. Each input vector variable is distributed to each of the nodes of the hidden layer.

Hidden Layer

It is the heart of all Artificial neural networks. This layer comprises all computations of the neural network. The edges of the hidden layer have weights multiplied by the node values. This layer uses the activation function. There can be one or two hidden layers in the model. Several hidden layer nodes should be accurate as few nodes in the hidden layer make the model unable to work efficiently with complex data. More nodes will result in an overfitting problem.

Output Layer

This layer gives the estimated output of the Neural Network as shown in fig 7.d. The number of nodes in the output layer depends on the type of problem. For a single targeted variable, use one node. N classification problem, ANN uses N nodes in the output layer.
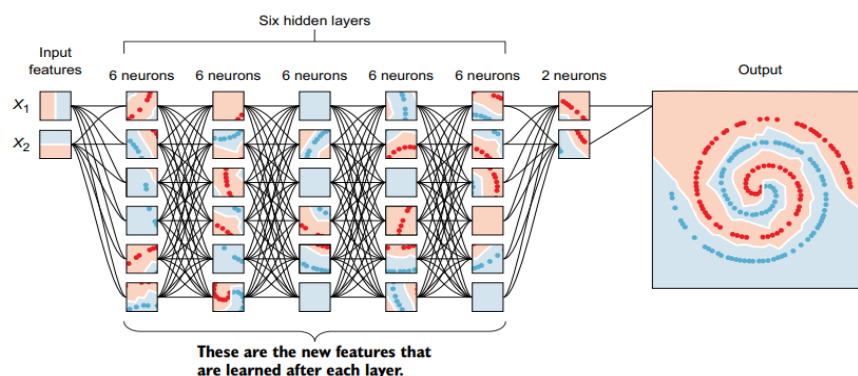

Fig.7.d. ANN with estimated output

How does a multilayer perceptron work?

1. The input node represents the feature of the dataset.
2. Each input node passes the vector input value to the hidden layer.
3. In the hidden layer, each edge has some weight multiplied by the input variable. All the production values from the hidden nodes are summed together. To generate the output
4. The activation function is used in the hidden layer to identify the active nodes.
5. The output is passed to the output layer.
6. Calculate the difference between predicted and actual output at the output layer.
7. The model uses backpropagation after calculating the predicted output.

Problem Design a MLP network :

Consider MLP network to be designed with 4 Inputs, bias , two hidden layers and each has five neurons : The Total neurons required are:

Weights_0_1: (4 nodes in the input layer) $\times$ (5 nodes in layer 1) + 5 biases [1 bias per neuron] = 25 edges

Weights_1_2: 5×5 nodes + 5 biases = 30 edges

Weights_2_output: 5×3 nodes + 3 bias = 18 edges

Total edges (weights) in this network = 73

Problem: For training a binary classification model with three independent variables, you choose to use neural networks. You apply one hidden layer with three neurons. What are the number of parameters to be estimated? Consider the bias term as a parameter)
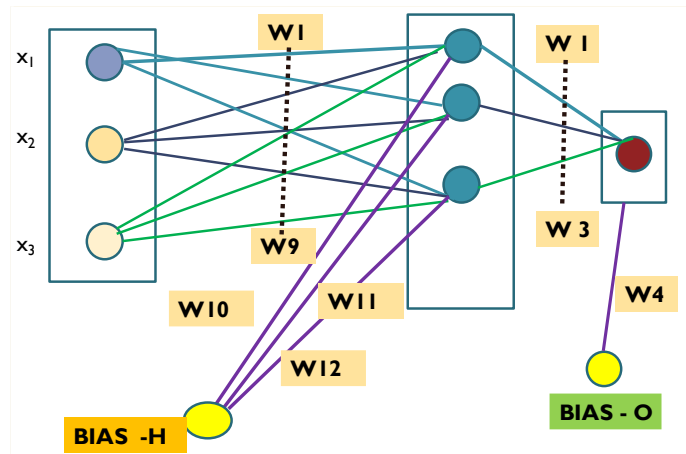
NUM. OF PARAMETERS = **[INPUT(3)** × **3**]+ **BIAS (3**)

+ **3** O/P +1 BIAS(1) = O/P

TO BE ESTIMATED   = **[3 * 3] + 3+3 +1 = 16**

Therefore 73 weights are required to design the simple MLP network. The values of these weights are randomly initialized, and then the network performs feed forward and Backpropagation to learn the best values of weights that most fit our model to the training data.

```
model = Sequential([
    Dense(5, input_dim=4),
    Dense(5),
    Dense(3)
])
```

```
model.summary()
```

```
_____
Layer (type)                    Output Shape                  Param #
=======================================================================
dense (Dense)                   (None, 5)                     25

_____
dense_1 (Dense)                 (None, 5)                     30

_____
dense_2 (Dense)                 (None, 3)                     18
=======================================================================
Total params: 73
Trainable params: 73
Non-trainable params: 0
```

ii) Some Hyper Parameters in a neural network are

Number of hidden layers—The general idea is that the more neurons you have, the better your network will learn the training data. But if you have too many neurons, this might lead to a phenomenon called overfitting: the network learned the training set so much that it memorized it instead of learning its features. Thus, it will fail to generalize. To get the appropriate number of layers, start with a small network, and observe the network performance. Then start adding layers until you get satisfying results.

Activation function—There are many types of activation functions, the most popular being ReLU and softmax. It is recommended that you use ReLU activation in the hidden layers and Softmax for the output layer.

Error function—Measures how far the network's prediction is from the true label. Mean square error is the metrics for regression problems, and cross-entropy is metric for classification problems.

Optimizer—Optimization algorithms are used to find the optimum weight values that minimize the error. There are several optimizers - batch gradient descent, stochastic gradient descent, and mini-batch gradient descent. Adam and RMSprop are two other popular optimizers.

Batch size—Mini-batch size is the number of sub-samples given to the network, after which parameter update happens. Bigger batch sizes learn faster but require more memory space. A good default for batch size might be 32. Also try 64, 128, 256, and so on.

Number of epochs—The number of times the entire training dataset is shown to the network while training. Increase the number of epochs until the validation accuracy starts decreasing even when training accuracy is increasing (overfitting).

Learning rate—One of the optimizer's input parameters that we tune is learning rate. If lr is too small is guaranteed to reach the minimum error (if you train for infinity time). A learning rate that is too big speeds up the learning but is not guaranteed to find the minimum error. The default lr value of the optimizer in most DL libraries is a reasonable start to get decent results.

9. Types of Activation functions:

A neuron in a neural network is a function that calculates the output of the neuron based on its inputs and the weights on individual inputs. Nontrivial problems can be solved only using a nonlinear activation function. Popular types of activation functions functionality is as shown in Fig 8.a

   1. Binary Step Function

   2. Linear Function   6. ReLU
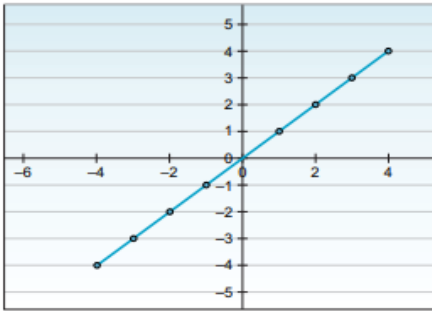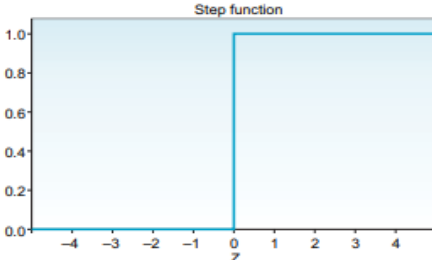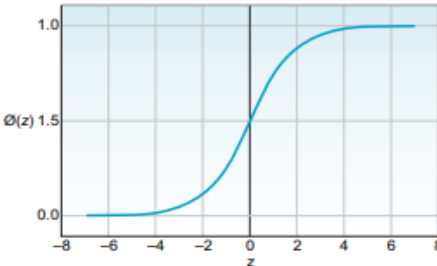
   3. Sigmoid   7. Leaky ReLU..... Etc.,

   4. Softmax

   5. Tanh
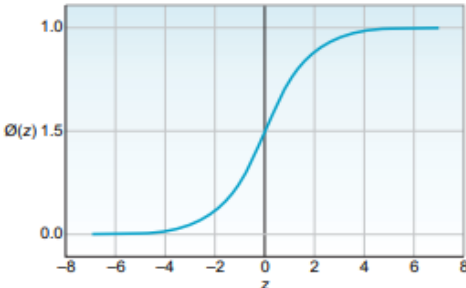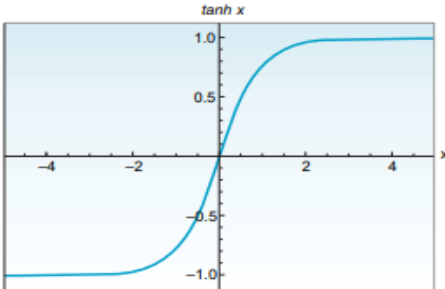
Activation function	Description	Plot	Equation
Linear transfer function (identity function)	The signal passes through it unchanged. It remains a linear function. Almost never used.		$f(x) = x$
Heaviside step function (binary classifier)	Produces a binary output of 0 or 1. Mainly used in binary classification to give a discrete value.		$\text{output} = \begin{cases} 0 \text{ if } w \cdot x + b \leq 0 \\ 1 \text{ if } w \cdot x + b > 0 \end{cases}$
Sigmoid/ logistic function	Squishes all the values to a probability between 0 and 1, which reduces extreme values or outliers in the data. Usually used to classify two classes.		$\sigma(z) = \dfrac{1}{1 + e^{-z}}$
Softmax function	A generalization of the sigmoid function. Used to obtain classification probabilities when we have more than two classes.		$\sigma(x_j) = \dfrac{e^{x_j}}{\sum_i e^{x_i}}$

Fig 8.a   Popular types of activation Functions

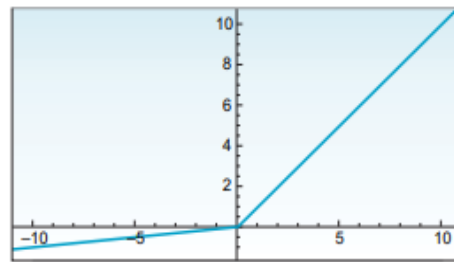|			
Hyperbolic tangent function (tanh)	Squishes all values to the range of −1 to 1. Tanh almost always works better than the sigmoid function in hidden layers.		$\tanh(x) = \dfrac{\sinh(x)}{\cosh(x)}$ $= \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

| Rectified linear unit (ReLU) | Activates a node only if the input is above zero. Always recommended for hidden layers. Better than tanh. | | $f(x) = \max(0, x)$ |
| Leaky ReLU | Instead of having the function be zero when $x < 0$, leaky ReLU introduces a small negative slope (around 0.01) when (x) is negative. | | $f(x) = \max(0.01x, x)$ |
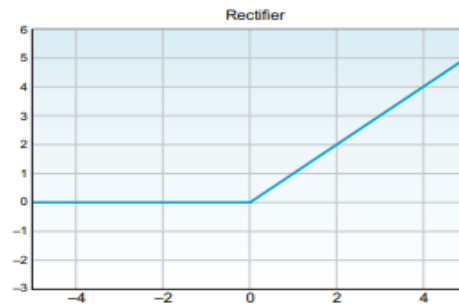
ii) Advantages and Disadvantages of Activation Functions are shown in Fig 8.b

Activation Functions		
Sigmoid	**Advantages** • Sigmoid functions has a well-defined nonzero derivative everywhere unlike step function, which allows the optimizer to make some progress at every step during training.	**Disadvantages** • suffers from vanishing gradient problem. • Slow due to its exponential operation and also, not zero centric.
Tanh	**Advantages** • Derivatives are steeper compared to Sigmoid. • Output is zero centric.	**Disadvantages** • Suffers from vanishing gradient problem.
ReLU	**Advantages** • Rectifies the vanishing gradient problem. • ReLU is less computationally expensive compared to tanh and sigmoid due to its simpler mathematical operations.	**Disadvantages** • Suffers from dying ReLU problem due to negative input. Weights updation stops and no information is passed to next layers.
Leaky ReLU	**Advantages** • Leaky ReLU fixed dying ReLU problem by introducing small negative slope.	**Disadvantages** • As it processes linearity, it can't be used for complex Classification. • Suffers from vanishing gradient problem.
ELU	**Advantages** • Unlike ReLU,ELU can produce negative outputs. • Fixes both dying ReLU and Vanishing gradient problem.	**Disadvantages** • For x>0, it can blow up the activation with the output range of $[0,\infty]$. • Computationally intensive.
Softmax	**Advantages** • Helps for Multiclass prediction.	**Disadvantages** • Usually used in last layer.

Fig 8.b   advantages and Disadvantages activation Functions

10. Training a MLP model

Training a neural network involves showing the network many examples (a training dataset); the network makes predictions through feedforward calculations and compares them with the correct labels to calculate the error. Finally, the neural network needs to adjust the weights (on all edges) until it gets the minimum error value, which means maximum accuracy. To get Max accuracy we need to do is build algorithms that find the optimum weights.

The Training neural network is done in three main steps:

- Feed forward Process
- Calculate Error,
- Optimize Weights

a. Feed forward Process:

The neural network learns by implementing the complete forward-pass calculations to produce a prediction output. The process of computing the linear combination and applying the activation function is called feed forward. The term feed forward is used to imply the forward direction in which the information flows from the input layer through the hidden layers, all the way to the output layer.

This process happens through the implementation of two consecutive functions:

- the weighted sum and
- the activation function.

In short, the forward pass is the calculations through the layers to make a prediction.

Let's us consider a simple three-layer neural network as shown in figure Fig.9.a ,its components:
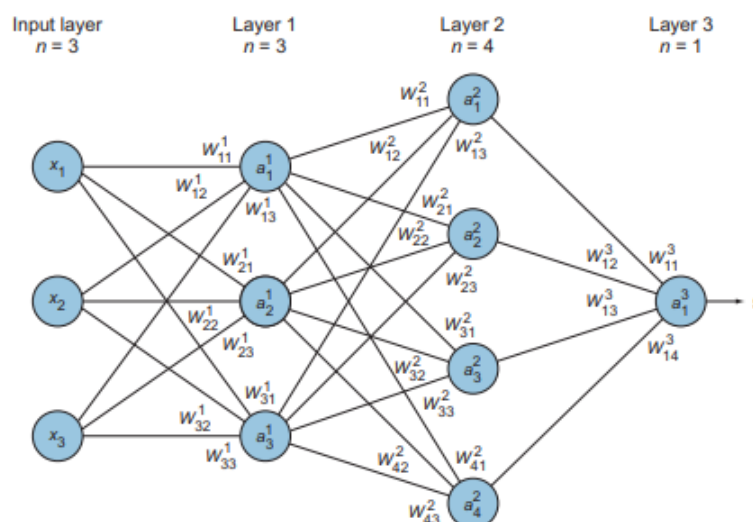

Fig9.A. Three-Layer Neural Network

Layers—This network consists of an input layer with three input features, and three hidden layers with 3, 4, 1 neurons in each layer

Weights and biases (w, b)—The edges between nodes are assigned random weights denoted as $W_{ab}^{(n)}$, where (n) indicates the layer number and (ab) indicates the weighted edge connecting the ath neuron in layer (n) to the bth neuron in the previous layer (n – 1). For example, $W_{23}^{(2)}$ is the weight that connects the second node in layer 2 to the third node in layer 1 (a_2^2 to a_1^3). (Note that you can see different denotations of $W_{ab}^{(n)}$ in other DL literature, which is fine as long as you follow one convention for your entire network.) The biases are treated similarly to weights because they are randomly initialized, and their values are learned during the training process. So, for convenience, from this point forward we are going to represent the basis with the same notation that we gave for the weights (w). In DL literature, you will mostly find all weights and biases represented as (w) for simplicity.

Activation functions σ(x)—In this example, we are using the sigmoid function σ(x) as an activation function.

Node values (a)—We will calculate the weighted sum, apply the activation function, and assign this value to the node a_m^n, where n is the layer number and m is the node index in the layer. For example, a_2^3 means node number 2 in layer 3.

Feed forward calculations:

Step1: start the feed forward calculations

$$a_1^{(1)} = \sigma(w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3)$$

$$a_2^{(1)} = \sigma(w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{32}^{(1)} x_3)$$

$$a_3^{(1)} = \sigma(w_{13}^{(1)} x_1 + w_{23}^{(1)} x_2 + w_{33}^{(1)} x_3)$$

Step2: do the same calculations for layer 2

$$a_1^{(2)}, a_2^{(2)}, a_3^{(2)}, \text{ and } a_4^{(2)}$$

Step3: all the way to the output prediction in layer 3

$$\hat{y} = a_1^{(2)} = \sigma\ (w_{11}^{(3)}\ a_1^{(2)}\ +\ w_{12}^{(3)}\ a_2^{(2)}\ +\ w_{13}^{(3)}\ a_3^{(2)}\ +\ w_{14}^{(3)}\ a_4^{(2)}\ )$$

$$\hat{y} = \sigma \cdot W^{(3)} \cdot \sigma \cdot W^{(2)} \cdot \sigma \cdot W^{(1)} \cdot (x)$$

$$\hat{y} = \sigma \begin{bmatrix} w_{11}^3 & w_{12}^3 & w_{13}^3 & w_{14}^3 \end{bmatrix} \cdot \sigma \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \\ w_{41}^2 & w_{42}^2 & w_{43}^2 \end{bmatrix} \cdot \sigma \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$W^{(3)}$ — Layer 3
$W^{(2)}$ — Layer 2
$W^{(1)}$ — Layer 1
Input vector

b. Error Function:

The error function is a measure of how "wrong" the neural network prediction is with respect to the expected output (the label). It quantifies how far we are from the correct solution. example, if we have a high loss, then our model is not doing a good job. The smaller the loss, the better the job the model is doing. The larger the loss, the more our model needs to be trained to increase its accuracy.

The error in simplest form, is calculated by comparing the prediction yˆ and the actual label y.

$$\text{error} = \left| \hat{y} - y \right|$$

Need of Error function

Optimization problems focus on defining an error function and trying to optimize its parameters to get the minimum error . The ultimate goal is to find the optimum variables (weights) that would minimize the error function as much as we can. If we don't know how far from the target we are, how will we know what to change in the next iteration? The process of minimizing this error is called error function optimization. Using several optimization methods all we need to know from the error function is how far we are from the correct prediction, or how much we missed the desired degree of performance.

A visualization of loss functions of two separate models plotted over time is shown in figure 9.b. You can see that model #1 is doing a better job of minimizing error, whereas model #2 starts off better until epoch 6 and then plateaus
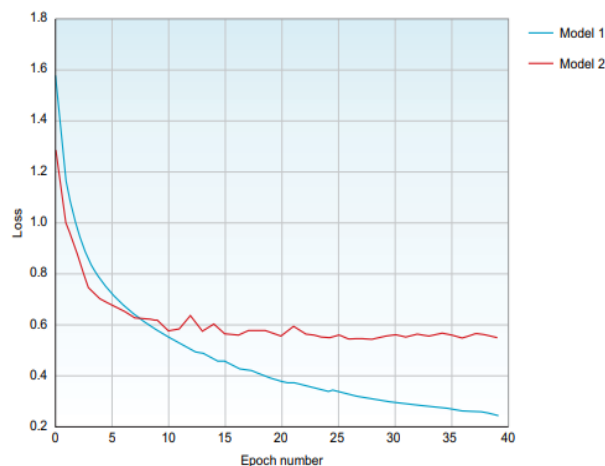

Fig 9.b loss functions of two separate models

Types of error Functions:

Different loss functions will give different errors for the same prediction, and thus have a considerable effect on the performance of the model. two most commonly used loss functions: mean squared error (and its variations), usually used for regression problems, and cross-entropy, used for classification problems.

a. Mean squared error (MSE) is commonly used in regression problems that require the output to be a real value (like house pricing). Instead of just comparing the prediction output with the label ($\hat{y}_i - y_i$), the error is squared and averaged over the number of data points, as you see in this equation

$$E(W, b) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

MSE is a good choice for a few reasons. The square ensures the error is always positive, and larger errors are penalized more than smaller errors. MSE is quite sensitive to outliers, since it squares the error value. This might not be an issue for the specific problem that you are solving. In fact, this sensitivity to outliers might be beneficial in some cases. For example, if you are predicting a stock price, you would want to take outliers into account, and sensitivity to outliers would be a good thing. In other scenarios, you wouldn't want to build a model that is skewed by outliers, such as predicting a house price in a city. In that case, you are more interested in the median and less in the mean. mean absolute error (MAE) was developed just for this purpose. It averages the absolute error over the entire dataset without taking the square of the error.

$$E(W, b) = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

Problem: Calculate the feed forward Error between actual and predicted value
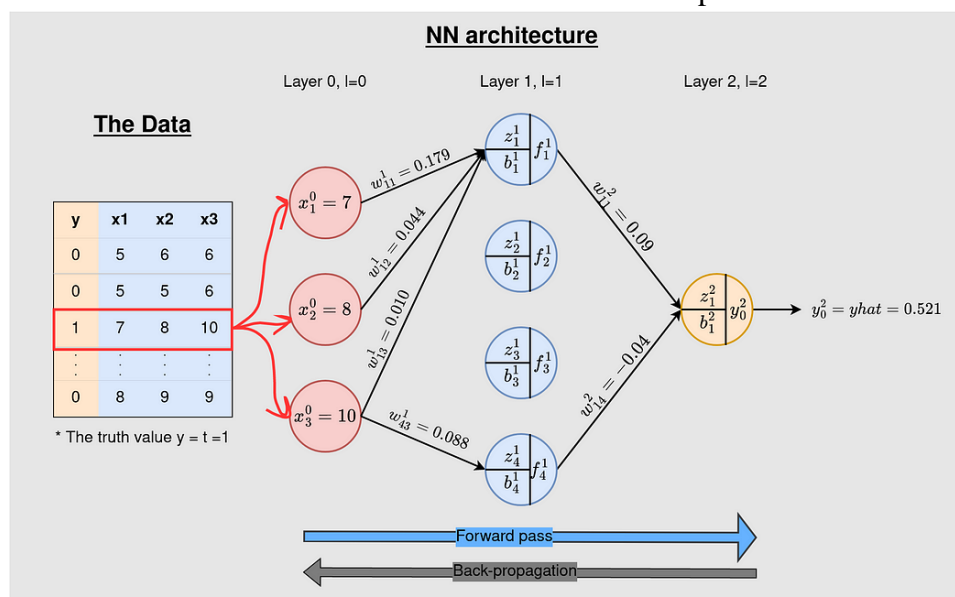
b. Cross-entropy is commonly used in classification problems because it quantifies the difference between two probability distributions.

For example, suppose that for a specific training instance, we are trying to classify a dog image out of three possible classes (dogs, cats, fish). The true distribution for this training instance is as follows:

Probability(cat)	P(dog)	P(fish)
0.0	1.0	0.0

We can interpret this "true" distribution to mean that the training instance has 0% probability of being class A, 100% probability of being class B, and 0% probability of being class C. Now, suppose our machine learning algorithm predicts the following probability distribution:

Probability(cat)	P(dog)	P(fish)
0.2	0.3	0.5

How close is the predicted distribution to the true distribution? That is what the crossentropy loss function determines. We can use this formula:

$$E(W, b) = -\sum_{i=1}^{m} \hat{y}_i \log(p_i)$$

where (y) is the target probability, (p) is the predicted probability, and (m) is the number of classes. The sum is over the three classes: cat, dog. the loss is 1.2

```
E = - (0.0 * log(0.2) + 1.0 * log(0.3) + 0.0 * log(0.5)) = 1.2
```

To calculate the cross-entropy error across all the training examples (n), we use this general formula:

$$E(W, b) = -\sum_{i=1}^{n}\sum_{i=1}^{m} \hat{y}_{ij} \log(p_{ij})$$

C. Optimization:

In neural networks, optimizing the error function means updating the weights and biases until we find the optimal weights, or the best values for the weights to produce the minimum error.

Optimization algorithms are responsible for reducing losses and provide most accurate results possible. The weight is initialized using some initialization strategies and is updated with each epoch according to the equation. The best results are achieved using some optimization strategies or algorithms called Optimizer.

Different types of optimizers a used to work to minimize the loss function are

1. Gradient Descent
2. Stochastic Gradient Descent (SGD)
3. Mini Batch Stochastic Gradient Descent (MB-SGD)
4. SGD with momentum
5. Nesterov Accelerated Gradient (NAG)
6. Adaptive Gradient (AdaGrad)
7. AdaDelta
8. RMSprop
9. Adam

Gradient descent simply means updating the weights iteratively to descend the slope of the error curve until we get to the point with minimum error as shown in 2D visualization below.
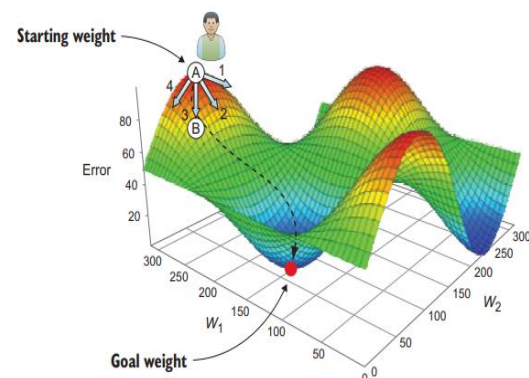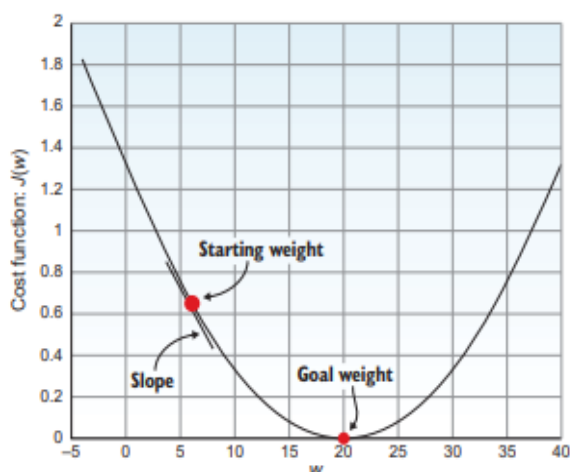
Fig9.e Gradient descent

Fig9.e Gradient descent

The GD hyperparameters are learning rate and batch_size . To visualize how gradient descent works, the plot the error function in a 3D graph (figure below) and go through the process step by step. The random initial weight (starting weight) is at point A, and our goal is to descend this error mountain to the goal w1 and w2 weight values, which produce the minimum error value. The way we do that is by taking a series of steps down the curve until we get the minimum error. In order to descend the error mountain, we need to determine two things for each step: The step direction (gradient), The step size (learning rate).(**Fig9.e**)

Three types of Gradient Decent are *Batch, Stochastic, And Mini-Batch.*
 All follow the same concept:
 – Find the direction of the steepest slope: the derivative of the error with respect to the weight
– Setting the learning rate (or step size). The algorithm will compute the slope, but you will set the learning rate as a hyperparameter that you will tune by trial and error.
– Start the learning rate at 0.01, and then go down to 0.001, 0.0001, 0.00001. The lower you set your learning rate, the more guaranteed you are to descend to the minimum error (if you train for an infinite time). Since we don't have infinite time, 0.01 is a reasonable start, and then we go down from there.

i) Batch GD updates the weights after computing the gradient of all the training data. This can be computationally very expensive when the data is huge. It doesn't scale well. Batch Gradient descent is a very powerful algorithm to get to the minimum error. But it has two major pitfalls. First, not all cost functions look like the simple bowls we saw earlier. There may be holes, ridges, and all sorts of irregular terrain that make reaching the minimum error very difficult. Consider figure below, where the error function is a little more complex and has ups and downs. Complex error functions are represented by more complex curves with many local minima values. Our goal is to reach the global minimum value. (**Fig9.f**)
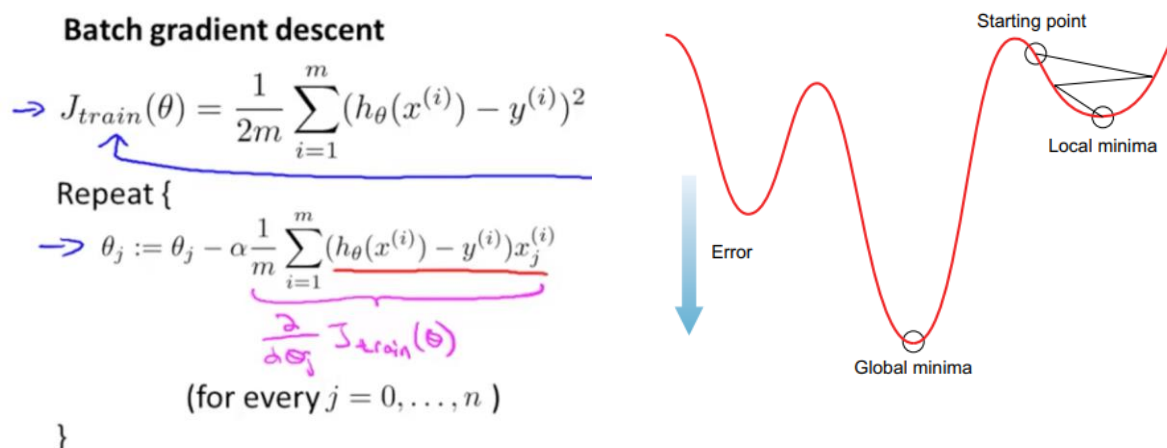

Batch gradient descent

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

$$\frac{\partial}{\partial \theta_j} J_{train}(\theta)$$

(for every $j = 0, \ldots, n$)

}

Fig 9.f. **Batch GD**

ii) The stochastic gradient descent algorithm randomly selects data points across the curve and descends all of them to find the local minima. it updates the weights after computing the gradient of a single instance of the training data. SGD is faster than BGD and usually reaches very close to the global minimum. (**Fig9.g**)

Stochastic gradient descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$
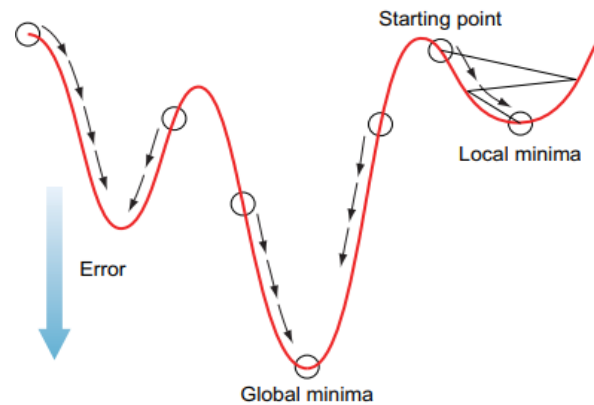
1. Randomly shuffle dataset.
2. Repeat {
   for $i := 1, \ldots, m$ {
   $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$
   (for $j = 0, \ldots, n$)
   }
   }

Fig9.g: **The stochastic gradient descent**

iii) Mini-batch GD is a compromise between batch and stochastic, using neither all the data nor a single instance. Instead, it takes a group of training instances (called a mini-batch), computes the gradient on them and updates the weights, and then repeats until it processes all the training data. In most cases, MB-GD is a good starting point.
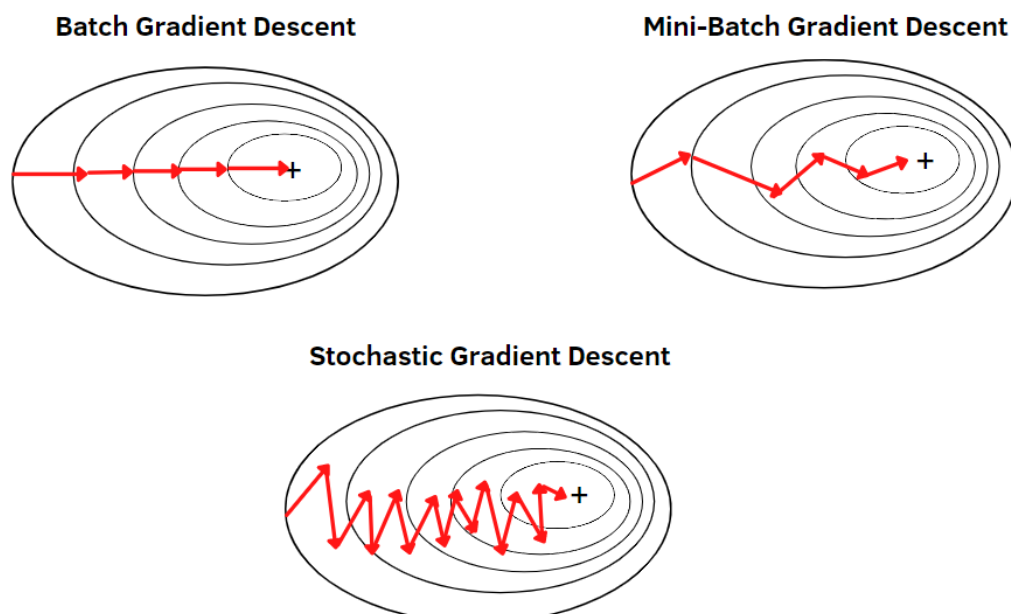
Mini-batch gradient descent

Say $b = 10, m = 1000$.
Repeat {
    for $i = 1, 11, 21, 31, \ldots, 991$ {
$$\theta_j := \theta_j - \alpha\frac{1}{10}\sum_{k=i}^{i+9}(h_\theta(x^{(k)}) - y^{(k)})x_j^{(k)}$$
(for every $j = 0, \ldots, n$)
    }
}

Fig10.d: **Mini Batch gradient descent**

Batch Gradient Descent   **Mini-Batch Gradient Descent**

Stochastic Gradient Descent

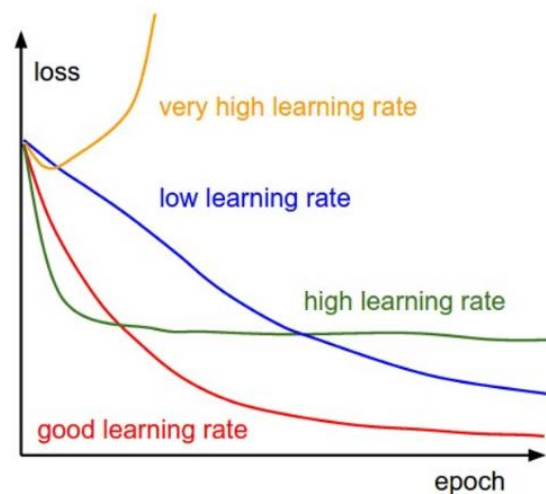**Fig10.e**: **Finding Optimal Minima using different   gradient descents**

Fig10.f: **Epoch Vs Loss at different learning Rates**

Comparison of Gradient descent and stochastic Gradient Descent:

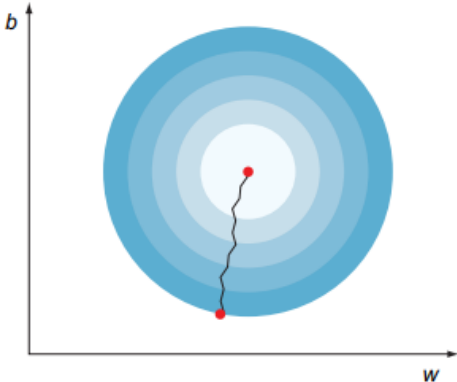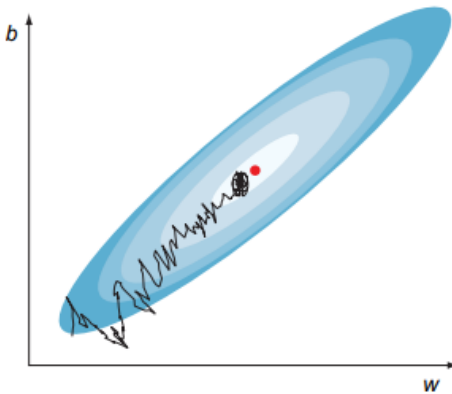| GD	Stochastic GD
1 Take all the data. 2 Compute the gradient. 3 Update the weights and take a step down.	1 Randomly shuffle samples in the training set. 2 Pick one data instance. 3 Compute the gradient. 4 Update the weights and take a step down. 5 Pick another one data instance. 6 Repeat for n number of epochs (training iterations).

4 Repeat for n number of epochs (iterations).

A smooth path for the GD down the error curve

An oscillated path for SGD down the error curve

Fig10.g Comparison of Gradient descent and stochastic GD

D. What is Backpropagation?

Backpropagation is the core of how neural networks learn. Up until this point, you learned that training a neural network typically happens by the repetition of the following three steps:

- **Feedforward:** get the linear combination (weighted sum), and apply the activation function to get the output prediction ($\hat{y}$):

$$\hat{y} = \sigma \cdot W^{(3)} \cdot \sigma \cdot W^{(2)} \cdot \sigma \cdot W^{(1)} \cdot (x)$$

- Compare the prediction with the label to **calculate the error** or loss function:

$$E(W, b) = |\hat{y}i - yi\,|$$

- Use a **gradient descent optimization** algorithm to compute the Δw that optimizes the error function:

$$\Delta wi = -\alpha \; dE/dwi$$

- **Backpropagate** the Δw through the network to update the weights:

Backpropagation, or backward pass, means propagating derivatives of the error with respect to each specific weight dE/dW_i .

from the last layer (output) back to the first layer (inputs) to adjust weights. By propagating the change in weights Δw backward from the prediction node ($\hat{y}$) all the way through the hidden layers and back to the input layer, the weights get updated:

$$(w_{next-step} = w_{current} + \Delta w)$$

This will take the error one step down the error mountain. Then the cycle starts again (steps 1 to 3) to update the weights and take the error another step down, until we get to the minimum error.

Backpropagation might sound clearer when we have only one weight. We simply adjust the weight by adding the Δw to the old weight $w_{new} = w - \alpha \; dE/dw_i$.

But it gets complicated when we have a multilayer perceptron (MLP) network with many weight variables. To make this clearer, consider the scenario in figure 2.35.
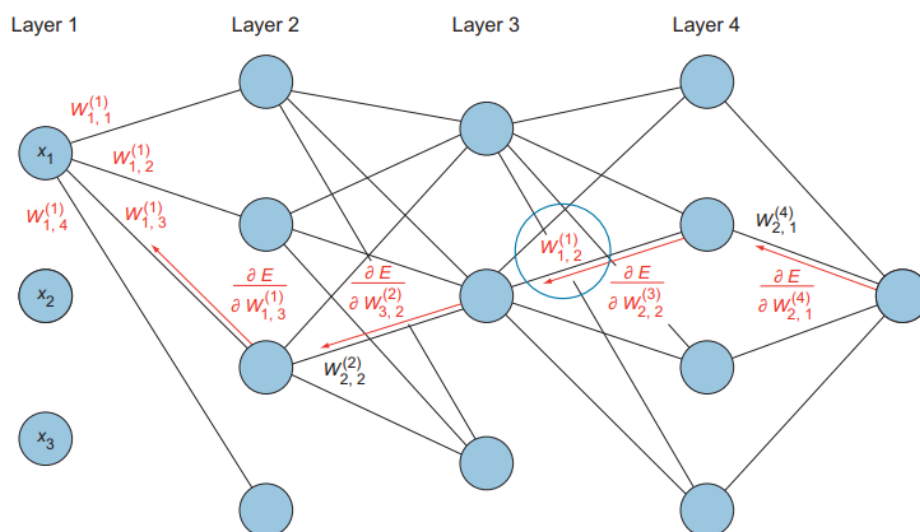

Figure 10.h. Backpropagation in MLP network with many weights' variables

How do we compute the change of the total error with respect to dE/dw$_{13}$?

How much will the total error change when we change the parameter w$_{13}$?

how to compute by applying the derivative rules on the error function.

That is straightforward because w21 is directly connected to the error function. But to compute the derivatives of the total error with respect to the weights all the way back to the input, we need a calculus rule called the **chain rule.**

Figure 10.i shows how backpropagation uses the chain rule to flow the gradients in the backward direction through the network. Let's apply the chain rule to calculate the derivative of the error with respect to the third weight on the first input w$_{1,3}$ (1) , where the (1) means layer 1, and w$_{1,3}$ means node number 1 and weight number 3:

$$\frac{dE}{dw_{1,3}^{(1)}} = \frac{dE}{dw_{2,1}^{(4)}} \times \frac{dw_{2,1}^{(4)}}{dw_{2,2}^{(3)}} \times \frac{dw_{2,2}^{(3)}}{dw_{3,2}^{(2)}} \times \frac{dw_{3,2}^{(2)}}{dw_{1,3}^{(1)}}$$

---$\rightarrow$Chain rule
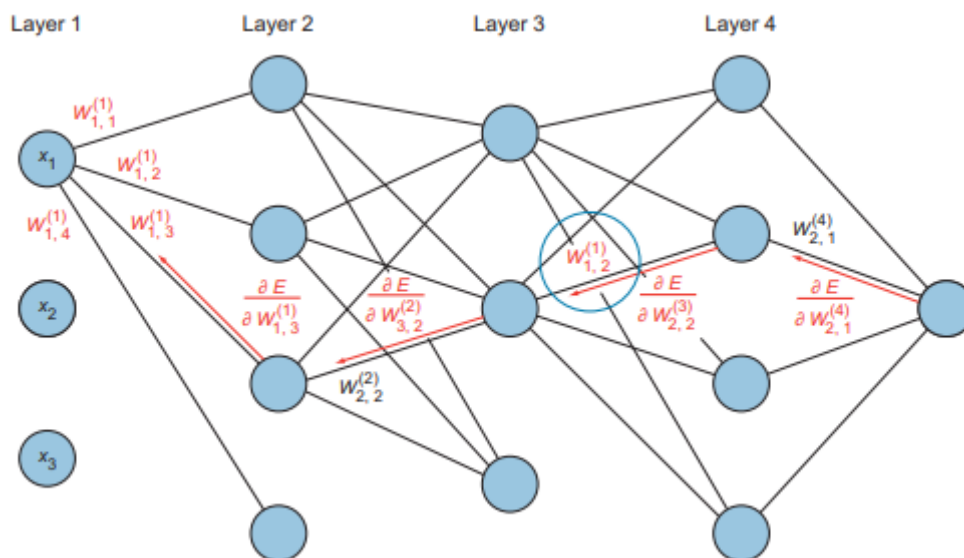
The error back propagated to the edge w$_{1,3}$ (1) = effect of error on edge 4 · effect on edge 3 · effect on edge 2 · effect on target edge.

Thus, the backpropagation technique is used by neural networks to update the weights to solve the best fit problem.