Machine Learning 2

NATURAL LANGUAGE PROCESSING

Natural Language Processing (or NLP) is applying Machine Learning models to text and language. Teaching machines to understand what is said in spoken and written word is the focus of Natural Language Processing. Whenever you dictate something into your iPhone / Android device that is then converted to text, that's an NLP algorithm in action.

You can also use NLP on a text review to predict if the review is a good one or a bad one. You can use NLP on an article to predict some categories of the articles you are trying to segment. You can use NLP on a book to predict the genre of the book. And it can go further, you can use NLP to build a machine translator or a speech recognition system, and in that last example you use classification algorithms to classify language. Speaking of classification algorithms, most of NLP algorithms are classification models, and they include Logistic Regression, Naive Bayes, CART which is a model based on decision trees, Maximum Entropy again related to Decision Trees, Hidden Markov Models which are models based on Markov processes.

A very well-known model in NLP is the Bag of Words model. It is a model used to preprocess the texts to classify before fitting the classification algorithms on the observations containing the texts.

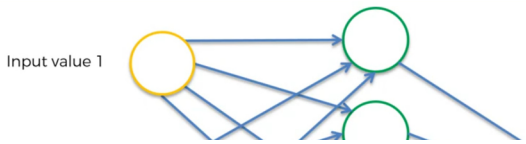
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


# Import the dataset
dataset = pd.read_csv('Reataurant_review.tsv', delimiter = '\t', qouting =3)


# Cleaning the texts
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
review = re.sub('[^a-zA-Z]', ' ', dataset['review'][0])
review = review.lower()
review = review.split()
review = [word for word in review if not word in set(stopwords.words('english'))]
```

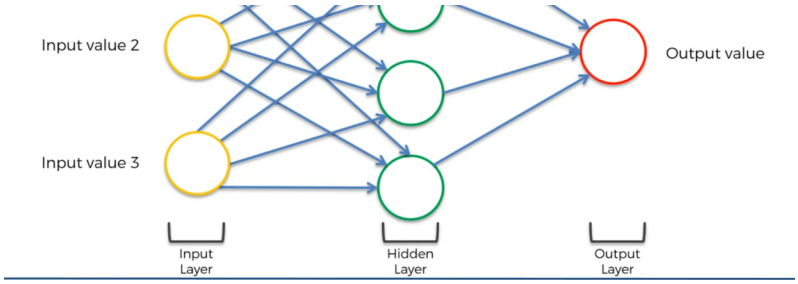
######incomplete

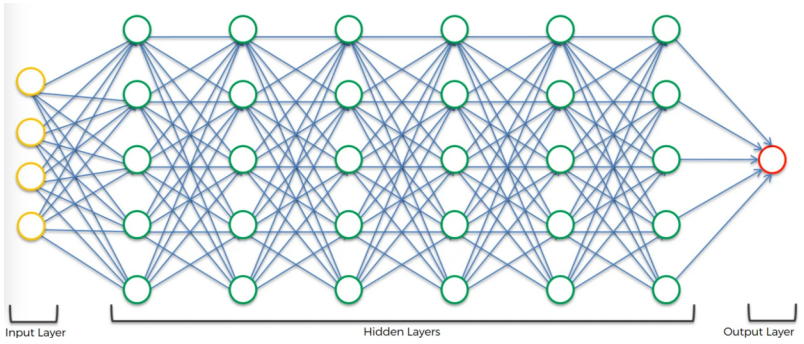
DEEP LEARNING

Simply means applying ML on large amount of data using neural network

Neural Network look something like this



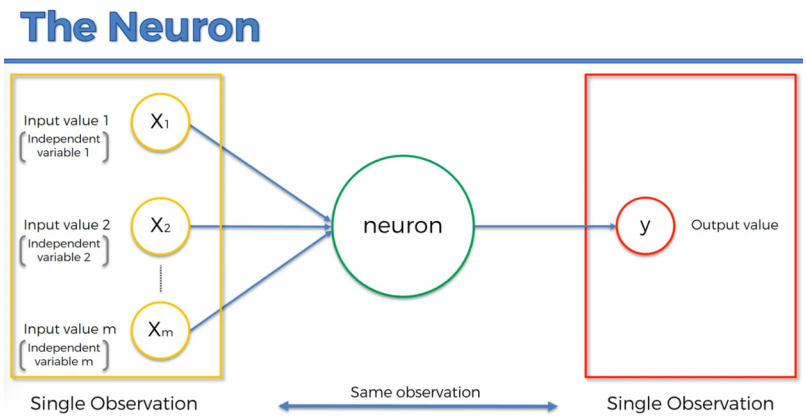


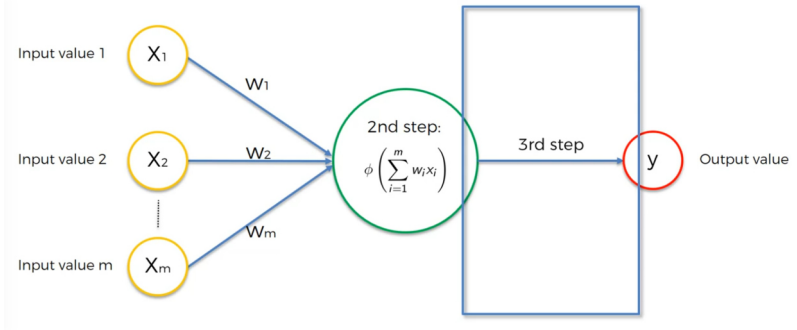


here we have an input layer then there are hidden layers then output layers and the computer make these connection between each node by learning from the data

ARTIFICIAL NEURAL NETWORK

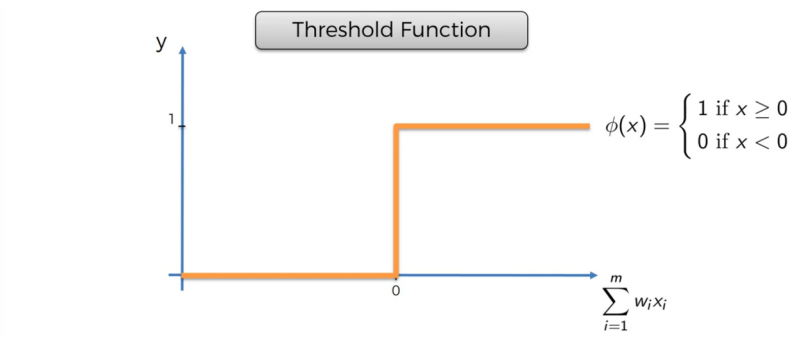
Theory



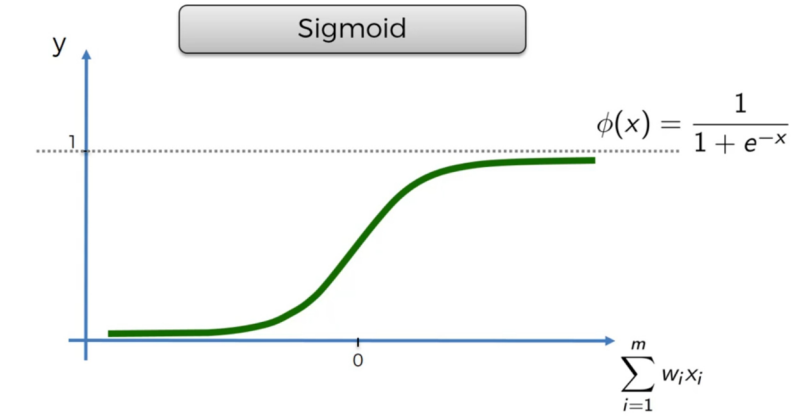


Different activation functions for output are -:

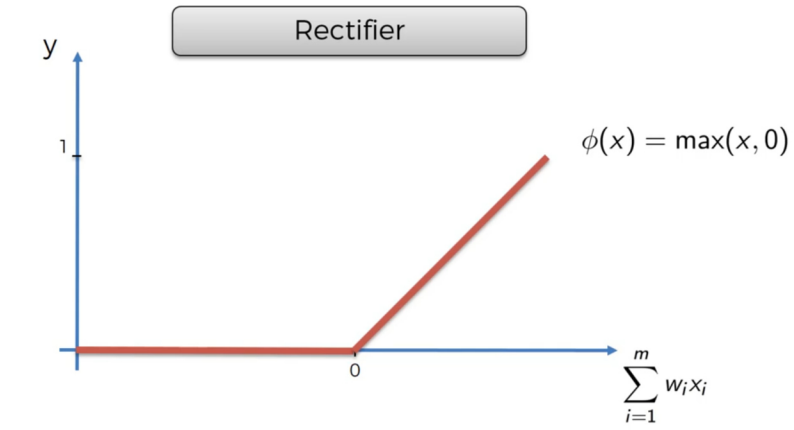
1 - threshold function- yes/no type



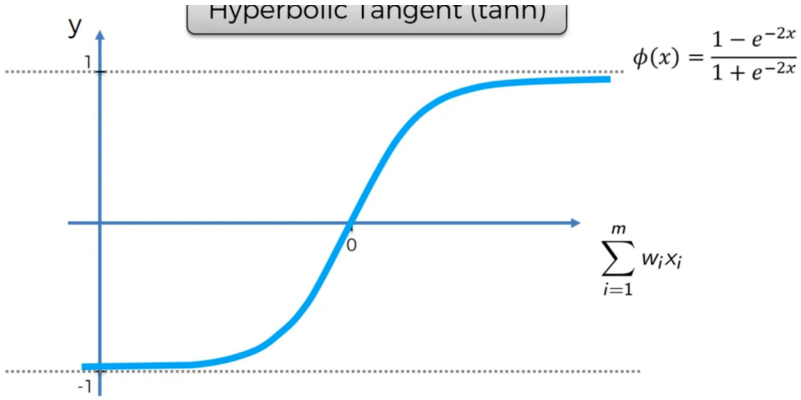
2 - sigmoid function - give the probability of happening of yes or 1



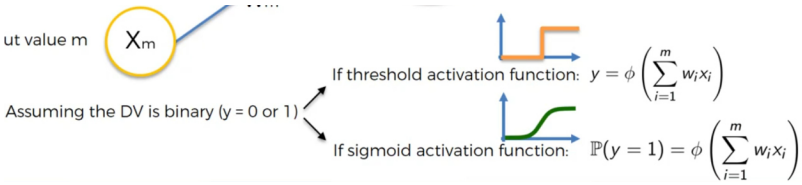
3 - rectifier function - it is one of the most of popular function



4 - hyperbolic function



http://jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf



A single layer feed forward network is called a perceptron output value is y hat as y is refer to actual value and output value has some deviation from the real value so how this work is that perceptron calculate the output value and then compare it to actual value and calculate the cost function given by c=1/2(yˆ - y)^2 then we update the weight to lower the cost function

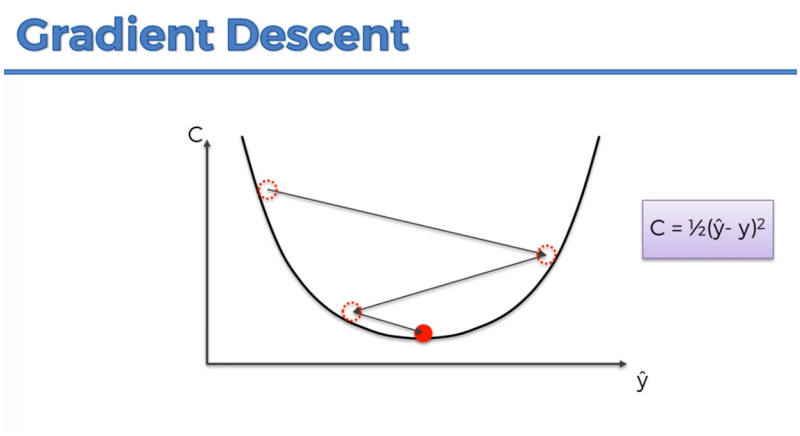
as computer don't have any relation for weight so it adjust them randomly.

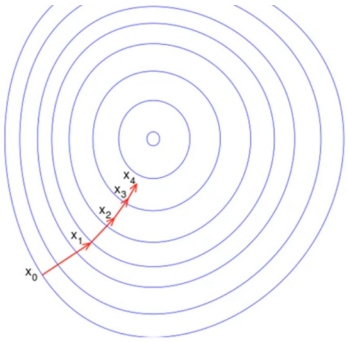
we are doing this on single row or single data so multiple data we calculate all the yˆ for each set and take the summation of their cost function.

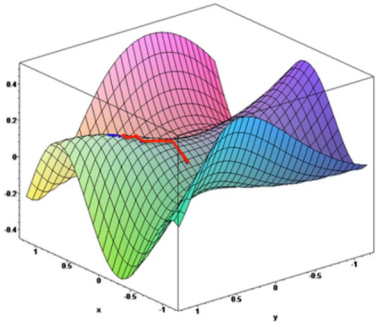
we can use brute force but when the number of synopsis and input dimension increases we have face the curse of dimensionality.

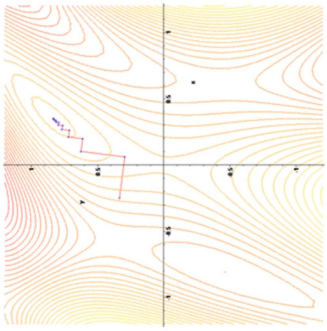
The problem with neural network is that there are lots of possibilities so lots of computation is require ,if we brute force a even smaller neural network like with 5 node in one hidden layer it require trillions of trillion fo years on world fastest computer so how we solve this situation

we use gradient descent technique in this we plot a graph against the cost function and yˆ and found the minima using the gradient descend





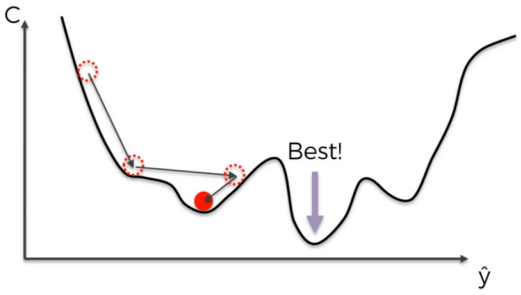




above image shows gradient decent in 1D ,2D and 3D

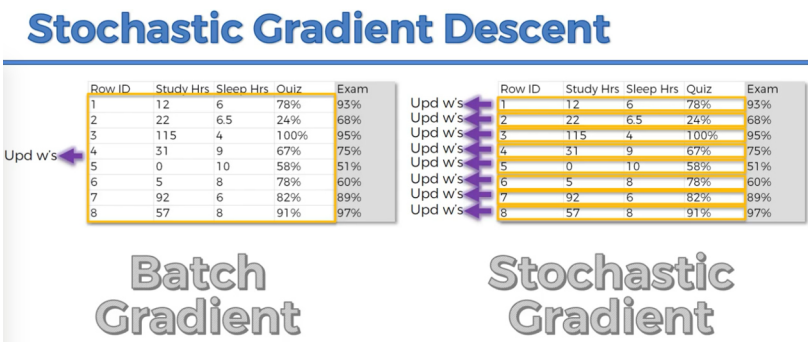
for gradient decent we need convex function

batch gradient decent - In this we take the whole dataset and adjust the weight at the end and it create a problem of local minimum like in the below fig we can get stuck at local minima at the final result



Stochastic Gradient decent - using this we reach the global minimum, in this method instead of taking all the data once we take one dataset at a time and adjust the weight at each iteration

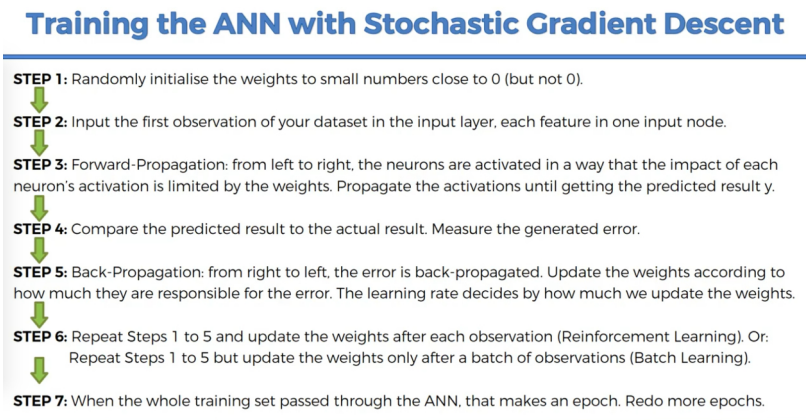
diff btw batch and stochastic



Descent Descent

read more - https://iamtrask.github.io/2015/07/27/python-network-part2/

must read - http://neuralnetworksanddeeplearning.com/chap2.html



install the following library for artificial neural network

Theano ,Tensorflow ,Keras

then do the basic importing the libraries , dataset and splitting the dataset into train and test set and bla bla bla

for ann we are only using keras library but keras need tensorflow and theano so we have to install them too. while building ann only install keras

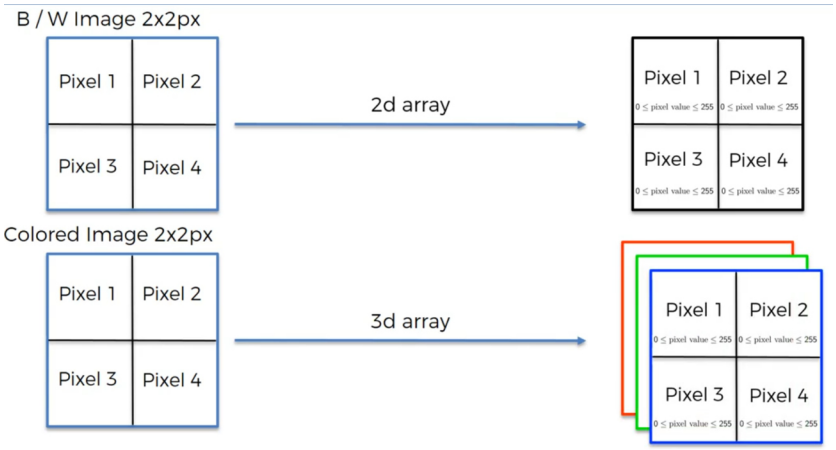
```
import keras
from keras.models import Sequential
from keras.layers import Dense


# Iniatialing the ANN
classifier = Sequential()


#initialing the classier as seqential since we are using sequential ann - stochastic gradient and then we are going to use rectifier function or hidden layer and sigmoid function for output layer


# Adding the input layer and the first hidden layer and activation function as rectifier
classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim=11))


# Adding the second hidden layer
classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))


# Adding the output layer and activation function as sigmoid
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))


# Compiling the ANN and using stochastic gradient(adam)
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
# Fitting the ANN
classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)


# Predicting the result
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)


# Making the Confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Convolutional Network

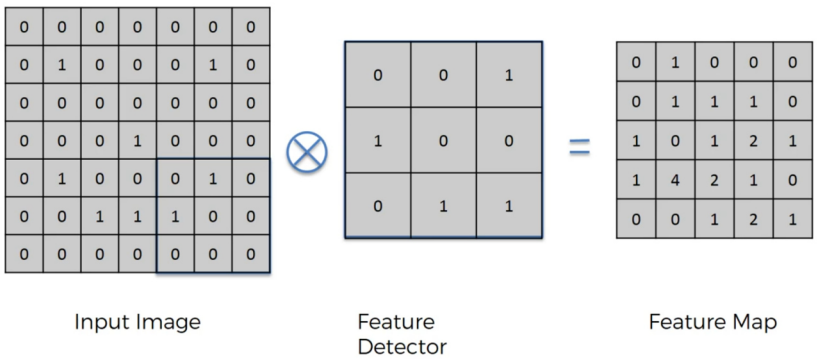
Convolutional network is used for deep learning on images

an black and white image is made up of 2d array while colored images are made up of 3d array

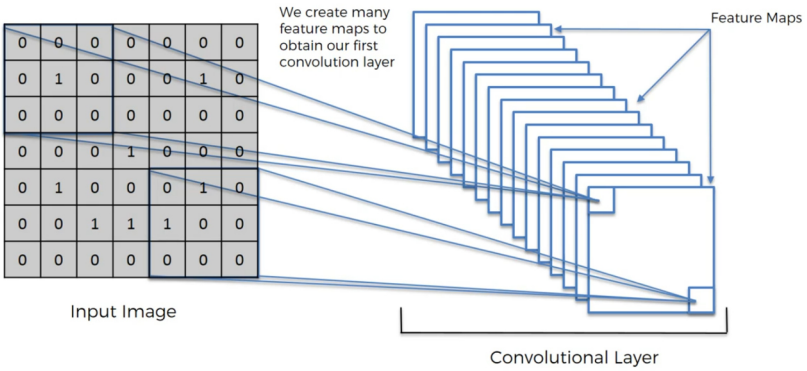


$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

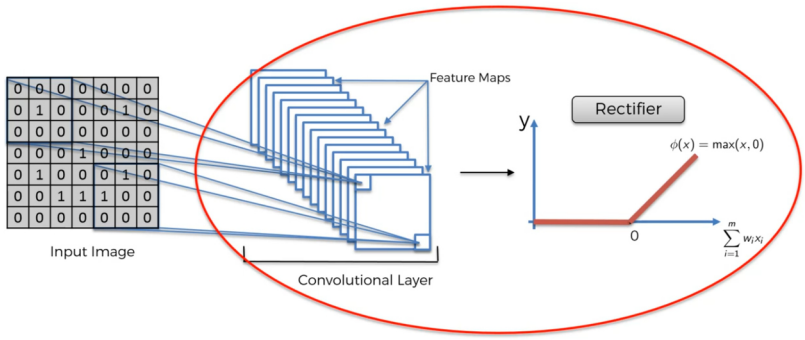
above fig is convolutional function



in convolution we use a feature detector matrix(it normally of 3x3 for basic but can be of any size) and we take this matrix and multiply it with our image matrix piece by piece and obtain a feature map it reduce the size the image but in creating feature map we lots some information which may or may not be important

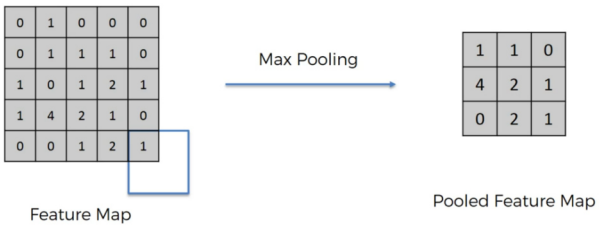


we create multiple feature map from a single image because we look for many features like in distinguishing dog from cat we are comparing multiple features like ear, mouth, eyes etc

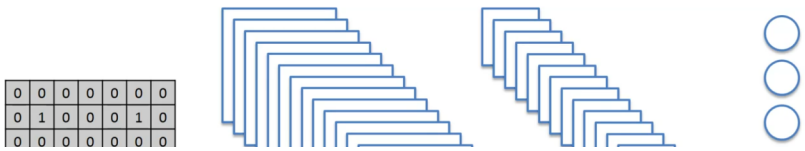


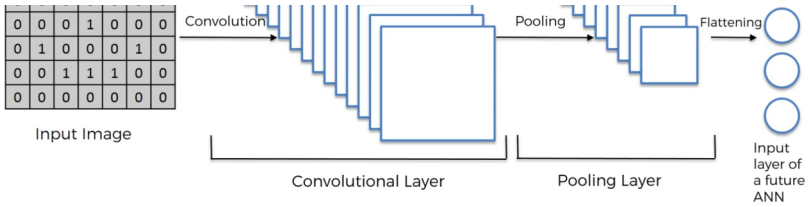
after generating feature map we apply rectifier function to increase the non linearity because images are non linear as there are different object, border, color etc and while applying feature detector we fear that we may make it linear so we apply rectifier

Then we apply pooling

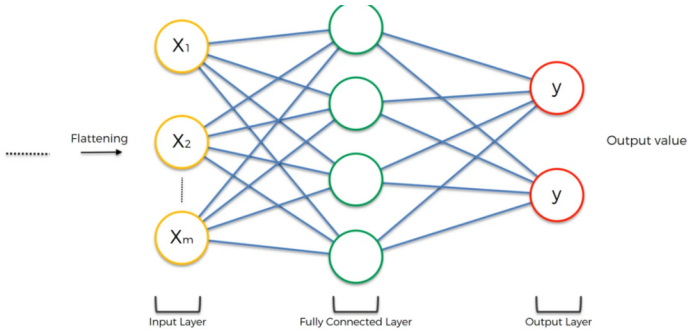


In max pooling we select a stride(sub matrix) and we record the max value of the stride in new matrix and this disregard the 70% of the unnecessary information from the image and we get only the feature this is principle behind the pooling this also reduce the size and no of parameter thus avoiding over fitting and reducing computing time

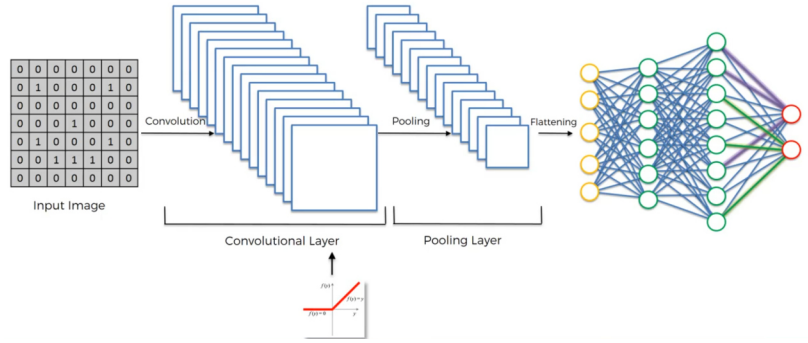


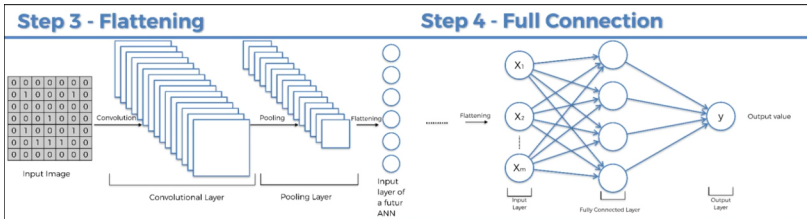


then we flatten are image means convert the matrix into vector so neural network can be applied basically ann

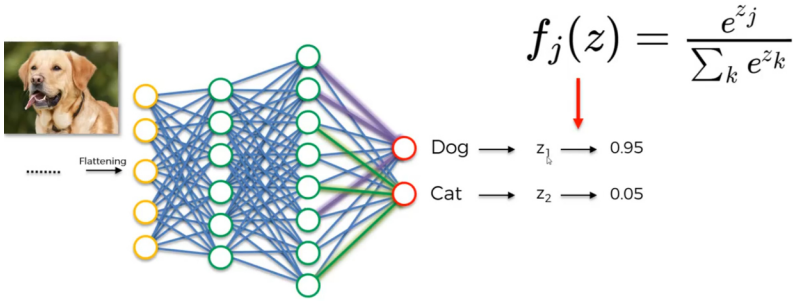


in this we call hidden layer as fully connected layer as they are fully connected





Softmax and Cross Entropy



$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

When we get the output from the neuron they return the probability now how do we know that they add up to one since it a

probability so the output from each neuron should add up to one right but in classic definition they don't necessarily until we introduce the softmax function it squishes a k dimensional vector and normalise the function and force the output in 0 to 1 and also add up to 1

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

$$H(p,q) = -\sum_x p(x)\,\log q(x)$$

cross Entropy function work hand in hand with softmax function

In this section there is no datapreprocessing as we are dealing with the images so this step is done manually by creating folder and stuff but there is feature scaling step we have to do but in different way


```
# Part 1 - Importing libraries
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense


#Intialising the CNN
classifier = Sequential()


#step 1 - convolution
classifier.add(Convolution2D(32, 3, 3, input_shape = (64, 64, 3), activation = 'relu'))


#step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))


#step 3 - Flattening
classifier.add(Flatten())


#step 4 - Full connection
```

```
classifier.add(Dense(output_dim = 128, activation = 'relu'))
classifier.add(Dense(output_dim = 1, activation = 'sigmoid'))


# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])


# Part 2 - Fitting the CNN to the images
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
     rescale = 1./255,
     shear_range = 0.2,
     zoom_range = 0.2,
     horizontal_flip = true)
test_datagen = ImageDataGenerator(rescale = 1./255)
training_set = train_datagen.flow_from_directory(
     'data/training_set',
     target_size = (64, 64),
     batch_size=32,
     class_mode = 'binary')
test_set = test_datagen.flow_from_directory(
     'dataset/test_set',
```