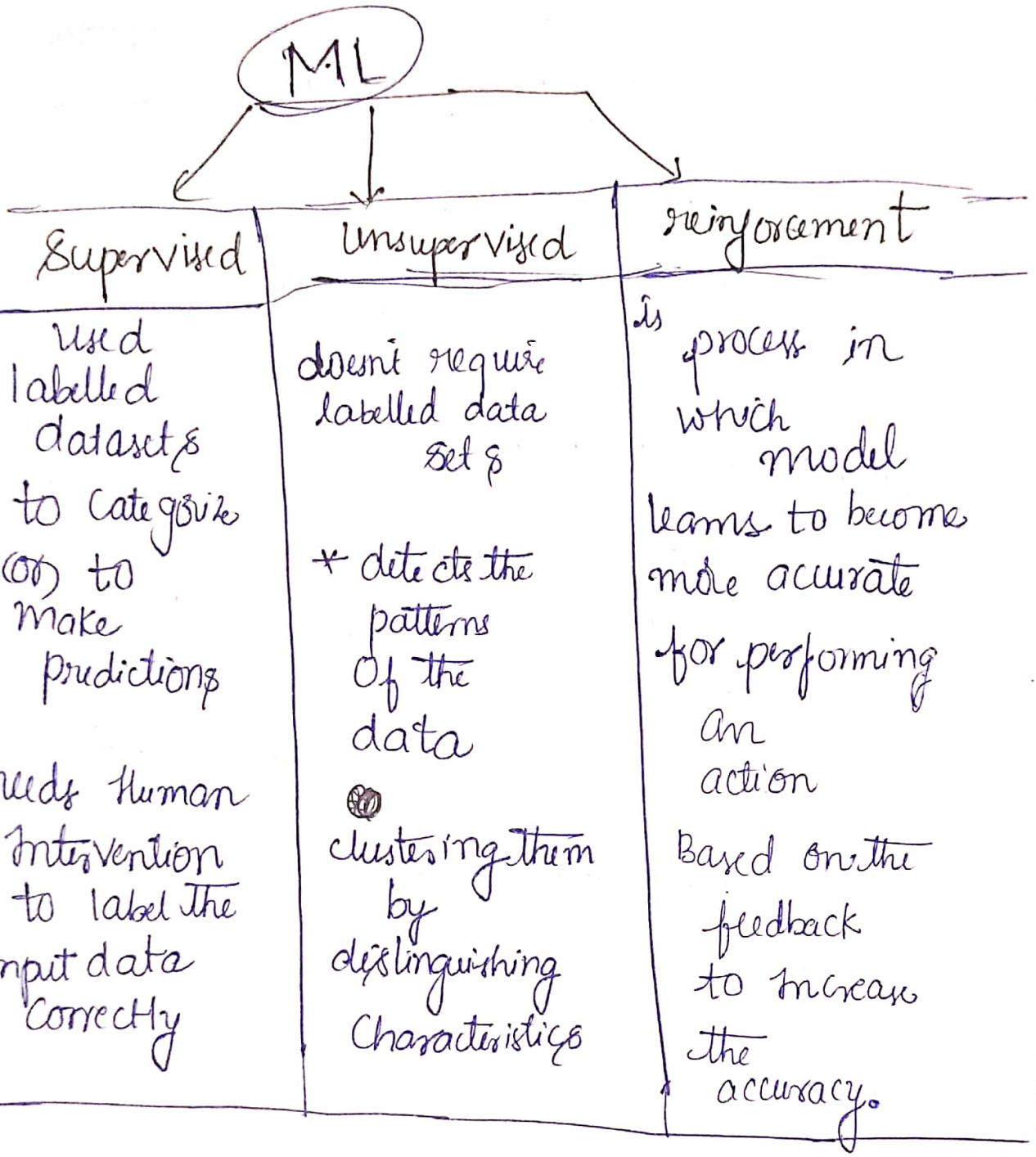


ML —

Takes Input data set & features

Classification
regression

Class label as output ← produces

deep learning :- * subset of Machine learning
* essentially a neural network with three or more layers.

ex :- technique that teaches computers to do what comes naturally to humans
1) driverless cars

These neural networks attempt to simulate Human Brain
* learning from large amounts of data
* additional Hidden layers helps to optimize and refine for accuracy

deep learning attempts to mimic the human brain

* deep learning drives many AI applications & services
↳ Improve Automation, performing analytical tasks without Human intervention

ex :-
1) digital Assistants
2) Voice enabled Tv remotes
3) Credit card fraud detection
4) self driving cars.

deep learning → * multilayered neural networks.
each layer has neurons

ML : ML V/S DL

algorithms leverage structured, labelled data to make predictions

↳ Specific features defined from input data & organized into tables

* goes through some preprocessing to organize into structure format

DL

elimates data pre-processing (which typically involved in ML)

* Algos can process unstructured data like Text & Image and automates feature extraction.

removes dependency on Human experts (some)

---

ex:- distinguishing among animals cat, dog etc

ML → needs Hierarchy of features establish by Human experts

DL → can determine which features are most important

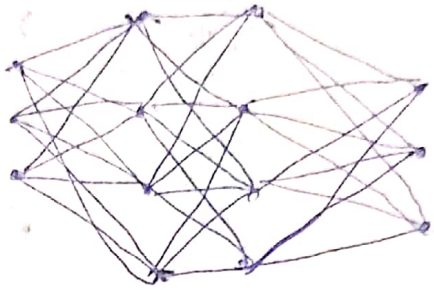
DL Input layer takes input dataset

Output layer produces Class labels.

ML := To achieve ai from algorithms that are trained with data.

DL = type of ML.
Inspired structure of Human Brain

DL ⇐ Artificial neural networks



ML ex:-

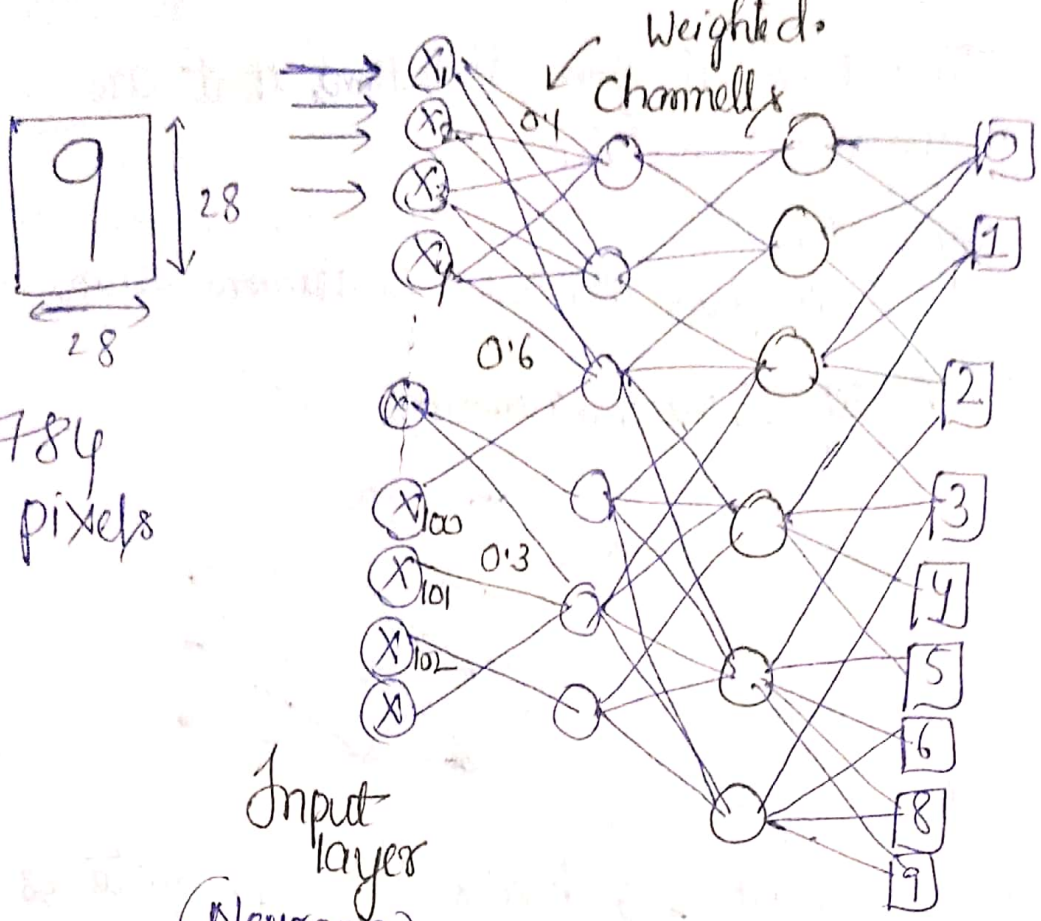
Seperating tomatoes & cherries by the features size & type of stem
(features Hierarchy given by Human experts)

DL Seperating the tomatoes & cherries by selecting the features..
Here features are extracted/picked by the neural networks are done without Human Intervention

Working of neural networks :-

distinguishing the Number written by humans

9 9 9 9 ⟶ detect as 9



Channels/connections are weighted

All the neurons are associated with unique numbers ⟶ Bias

* this bias is added to the weighted sum of inputs reaching the neurons

$$B_1 + (X_1 * 0.1 + X_2 * 0.2)$$

= function node.

Activation function

↓ determines if the neuron gets activated/not.

every activated neuron passes on the information to the following layer

Same repeats until all the hidden layers upto last Output layer

that one neuron activates in the output layer is the output

Where is deep learning applied?

* customer support agents (bot)
* Medical care → neural networks detects cancer cells (MRI) images
* Self driving cars

Apple
Tesla
NISSAN

limitations

① data → needs massive volume of data

(Takes unstructured data)

② Computational power

neural networks needs (GPU) more cores

- GPU > CPU

③ Training time

Hours/months of Training

Time ∝ amount of data.

Training

Working of neural networks

1) weighted sum of inputs calculated
2) Bias is added
3) result is fed to activation function
4) specific neuron is activated.

---

neural network ?

Abstract:-

Predicting traffic based on the
1) Historical traffic flow data
2) Contextual factors

Traffic flow → Start time, end time period → Timestamp

Contextual factors
- day
- Week
- Weather
- Season
- Holiday
- Big event
- working day / weekends

multilayer-supervised learning Algorithm to generate/mine relationship b/w the Traffic flow data & contextual factors

---

Introduction

1) government Agencies
2) Travellers

Appropriate Traffic Management

~~Esc~~ forecasting Trafficflow for whole day

ANN > (Better) Classical/Statistical Method & Averaging Smoothing

Literature Review :

forecasting Based on:-

day divided into time intervals
averaging values across each interval
→ forecasting result of each interval.

daily patterns

Morning & evening → peaks
midnight ⟷ low

Steps :- 2016

1) Historical data into several groups
Based on similar traffic Flow

2) select Appropriate group of Historical data for target day

3) finish forecasting by taking Appropriate data

2014 → ① collects historical data from all work days of prev 6 months (128) days

② used to predict the Traffic flow on target days.

2016

* divide Historical data into two groups
  acc to degree of similarity
①
② then use wavelet analysis
③ Split into basic series & deviation series
④ Historical data is should first be selected based on
  Whether the target day is
  is working day ?
  or NOT
⑤ Average of Basic series of whole group
  ↳ = Basic forecasting result.

then
deviation of target day → Based on forecasting result

[Common Time Series Forecasting Methods.]

2002

2016

ARIMA

Autoregressive Integrated moving Average Algorithm

GRNN

generalized regression neural network

(Working days & Weekends)

1) Week day / Saturday / Sunday
2) Summer day / non Summer day
3) rainy day / not
3) Holiday / not

Euclidean distance
Spectral distance

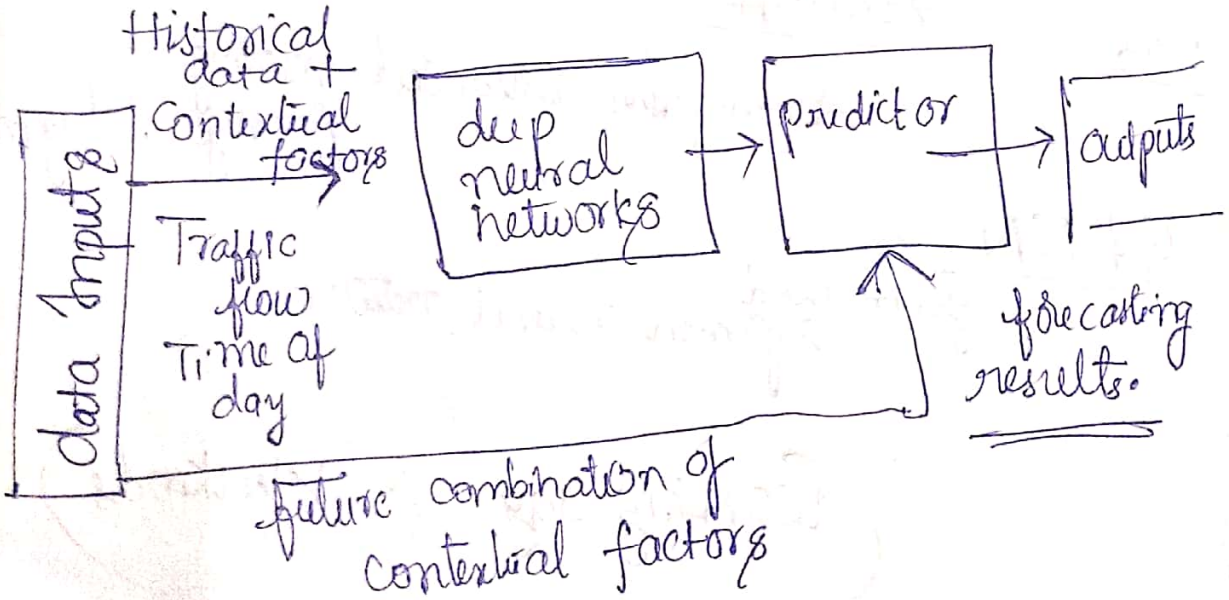
literature survey

- k-means
- wavelet analysis
- Support Vector Machines
- k-nearest neighbours
- neural networks
- Statistical

* Combination of Contextual factors → and, starting & ending time point

---

Methodology :-



Batch Training & predictor generation

dataset — Over 1 lakh records → each CSV

2015, 2016

stamp → 5 min
10 min
15 min
:
60 min

00:00:00 to 23:59:00 → whole year → 12 months one year

→ 6 lakh records → 6 CSV files

2015, 2016

attributes

- Year
- month
- day
- hour
- minute
- Weekday
- holiday
- rain
- Volume

→ features — (9)

2015

5 min — 1 CSV — 1 lakh
10 min
:
60 min — 1 CSV — 1 lakh.

6 Lakh

01| 2016 → 5 min, 10 min, ... 60 min — 6 × 1500

02| 2016

03| 2016

3 × (6 CSV files)* 1500 records

⇒ 18 CSV files

18 + 6

↓ ↓

(18 × 1500) (6 × 100,000)

⇒ 27000 records

6 Lakh records

24 CSV files

= 6L+

3000 epochs

epoch: complete pass over the entire training data set during the training phase of a neural network

ex

epoch: 10 ⟶ 10 complete passes

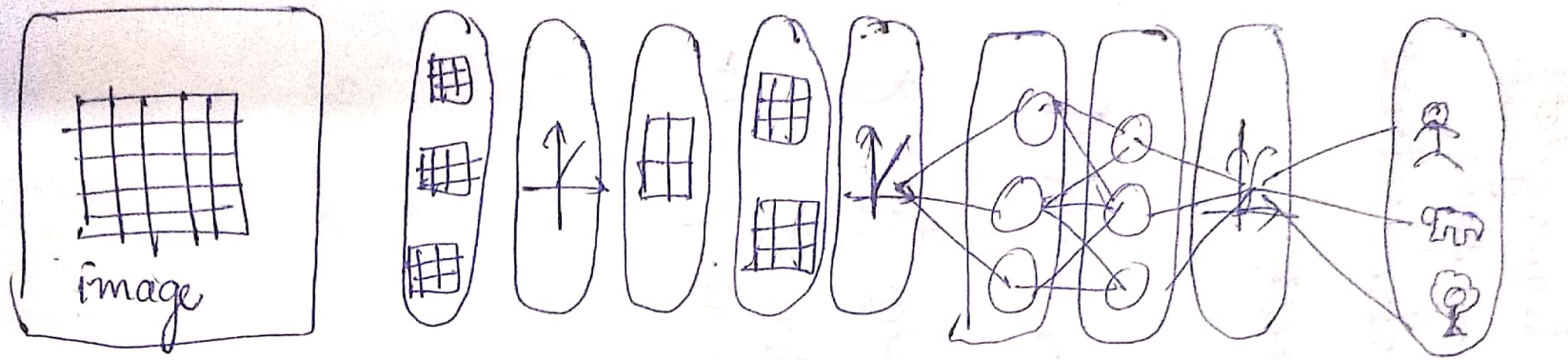
epochs ↑ ⟶ patterns ↑ good

efficient training for model ↑

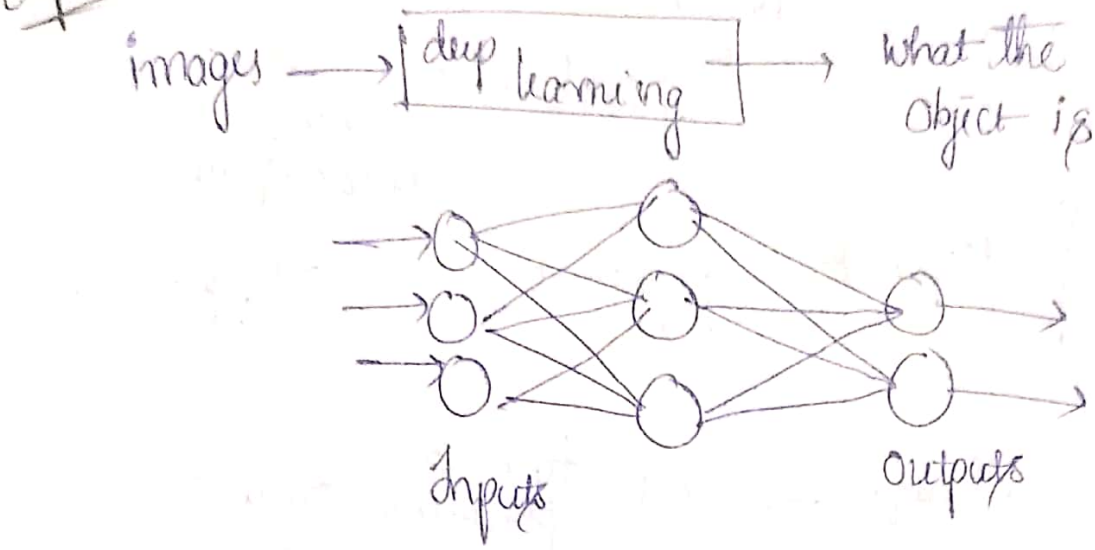
more epochs ⟶ model start Memorizing training data instead of generalizable patterns. (Overfitting)

less epochs ⟶ (Underfitting)

Optimal No. of epochs



dep learning



dep learning — to recognize images
letters
objects

Image recognition

diagnosis
Tesla. auto-pilot mode for car driving

Task 1

1) Import images

I = imread ("filename.png")

*

img1 = imread ("file01.jpg")

GIF : png
png : JPEG

2) To display an image stored in Matlab Variable.

imshow (I)

*

imshow (img1)

Task 2

img2 = imread ("file02.jpg")

imshow (img2)

To Create copy of predfined deep network "GoogLeNet" in matlab work space

* `deepnet = googlenet` (Task 3)

Classify ⇒ used to make prediction on the image

Addn

`pred = classify (net, img)`

*

`pred1 = Classify (deepnet, img 1)`

* `pred2 = Classify (deepnet2 img2)`
* `pred3 = classify (deepnet, img3)`

(Task 1)

deepnet = deep convolutionary network

Variable property

Inspect the layers in deep convolutionary network.

`Var = deepnet.Layers`

*

`dy = deepnet.Layers`

extracts layers of deepnet

ly = array of network layers.

Input

Task 2

Index individual layers by indexing into ly using regular matlab indexing

ex:

layer3 = ly(3)

* `inlayer = ly(1)` extract first input layer of the network into "inlayer"

---

each layer of the network → has properties relevant to type of layer

↓

properties relevant to layer

ex: Input Size ⇒ Size (dimensions) of image the network expects as input

Task 3

extract InputSize of first layer of network

* `insz = inlayer.InputSize`

Task 4 To extract last layer `ly(end)`

* `outlayer = ly(end)`

Tasks — Classes property ⇒ extract the classes of the last layer

* `categorynames = outlayer.Classes`

---

classify func ⇒ gives class to which network assigns highest score

Obtain predicted scores for all classes requiring second output from classify

classify — function
net — pretrained GoogLeNet network
img — image stored in variable
pred = Network prediction
scores = all the prediction scores

* `[pred, scores] = classify(net, img)`

---

bar(scores) = Network

Vector of prediction scores to investigate network classification

* `bar(scores)`

* `highscores = scores > 0.01`

logical indexing → `bar(scores(highscores))`

bar chart of prediction values that above threshold value.

`xticklabels(categorynames(highscores))`

`xticklabels(categorynames(highscores))` *

* label the Bar Chart with appropriate predicted class names.

If labels overlap at bar chart

* `xtickangle(60)` →

Image Input

`imread`

each image is read into memory

`classify` (pre-trained network to classify)

`dog` (Output class label)

Thousands of Images

folder

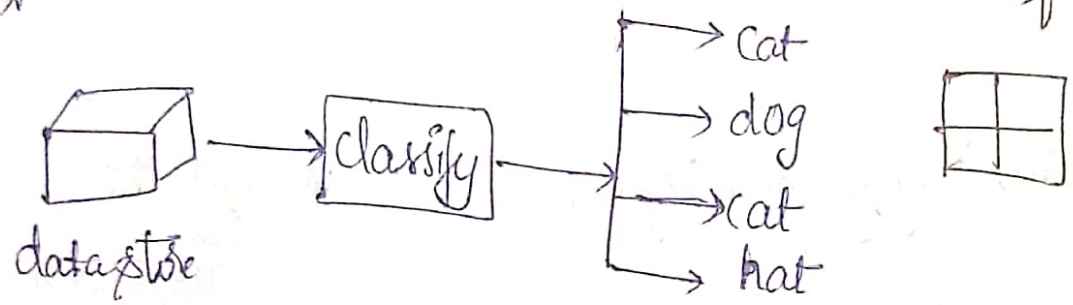
read

datastore
Matlab Variable

files

folder of Image file

CNN: In deep learning works with datastores



Create datastore.

function :- imageDatastore

argument :- "file*.jpg" (foldername)

* → Specify multiple files

```
imds = imageDatastore("file*.jpg")
```

datastore contains meta information about data files.

Files = property ⇒ extract file names of images.

```
fname = imds.Files
```

manually import data from data store

read — Imports image one at a time order

readimage — Imports single specific image

readall — Imports all the images into a single cell array with each image in a separate cell.

$$I = \text{readimage}(ds, n)$$

nth image of datastore ds imported into array I.

* $$img = \text{readimage}(imds, 7)$$

$$preds = \text{classify}(net, ds)$$

array of predicted classes → one for each image

To classify image data store to classify.

Instead of using individual image in CNN

$$preds = \text{classify}(net, imds)$$

- preds → array predicted classes for all the images respectively.
- net → pretrained network
- imds → data store.

Classify function running 12 images then GoogLeNet time see to execute

* `[preds, scores] = classify(net, imds)`
  `max(scores, [], 2)`

  → prediction classes along with the scores for predictions.

  ↓ maximum element in each row to see how confident the classification was for each image.

```
[preds, scores] = classify(net, imds)
max(scores, [], 2)

bar(max(scores, [], 2))
xticklabels(preds)
xtickangle(60)
ylabel("Score of Prediction")
```

Adjust Input Images

* `Sz = size(img)`

Size = function
img = input

`ExpectedInputSize = inputLayer.InputSize`

InputSize → property

inputLayer

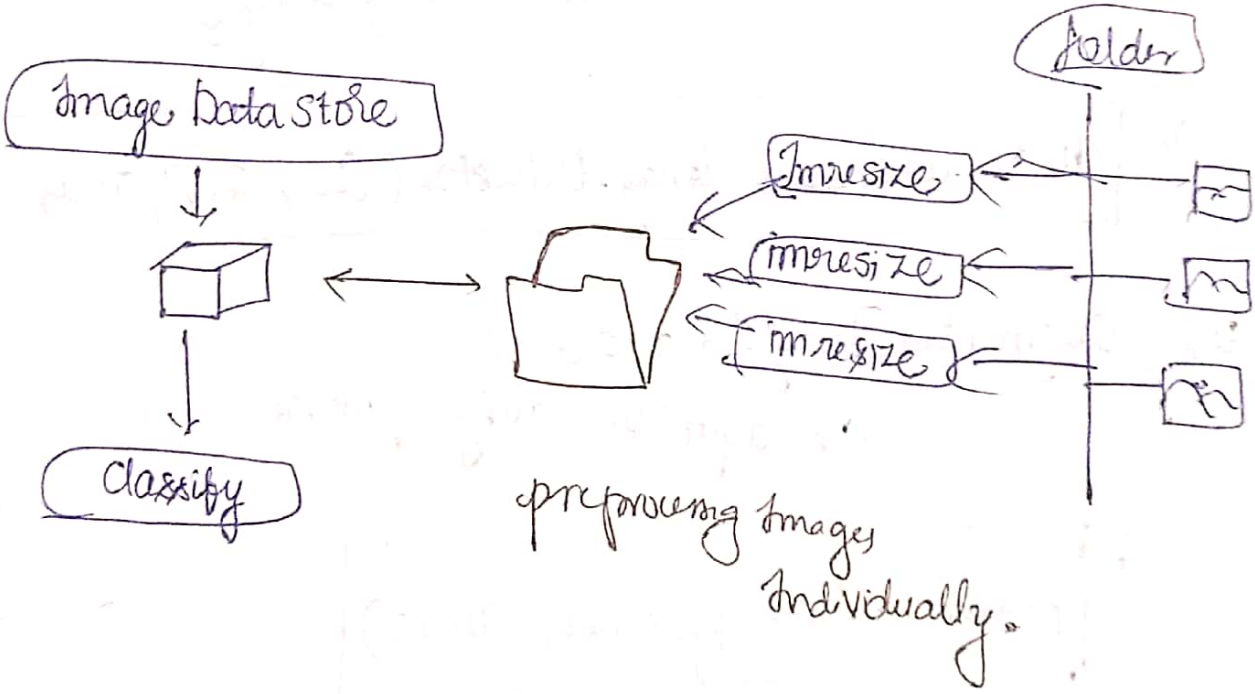
To get InputSize of Inputlayer 1

```
net = googlenet
inlayer = net.Layers(1)
insz = inlayer.InputSize
```

To resize the image ⇒

```
imresz = imresize(img,[numrows, numcols]);
imshow(imresz)
```

```
img = imresize(img,[224,224])
imshow(img)
```



Resize Images in datastore :-

Create image datastore that refers to image files with extension .jpg in the current folder

```
ds = imageDatastore("*.jpg")
```

* `imds = imageDataStore("*.jpg")`

`auds = augmentedImageDatastore([r,c], imds)`

augmented image datastore can perform simple preprocessing on entire collection of images

* `auds = augmentedImageDatastore([224 224], imds)`

use augmentedImageDatastore as input to classify function

* `preds = classify(net, auds)`

classifying images in auds using classify function using network stored in net

preds = predictions all images

resizing is one of the most common preprocessing steps when classifying images.

grayscale → RGB

Preprocess Color Using a datastore.

montage(imds) ⟹ display all images in the datastore

* `montage(imds)`

*

```
auds = augmentedImageDatastore([224,224], imds,
                               "ColorPreprocessing", "gray2rgb");
```

Create an Augmented datastore from imds preprocess

* `[pred,s] = classify(net, auds)`

classify images in auds using classify.

GoogleNet is stored in net

* `flwrds = imageDatastore("Flowers", "IncludeSubfolders", true)`

*

```
resizeds = augmentedImageDatastore([224,224],flowers);
```

*

```
preds = classify(net, resizeds)
```

CNN

Classify the type of flower with input image

pretrained network can classify an image into 100's of predetermined categories.

Take a network and modify it to our required problem

Transfer learning:

1) Take a pretrained network
2) modify it
3) re-training it on new data

Very effective to attack on many deep-learning problems.

pretrained network like 'GoogLeNet'

We have no flexibility in the way the network operates & network probably wont solve the exact problem you are trying to solve.

Transfer layers Checklist

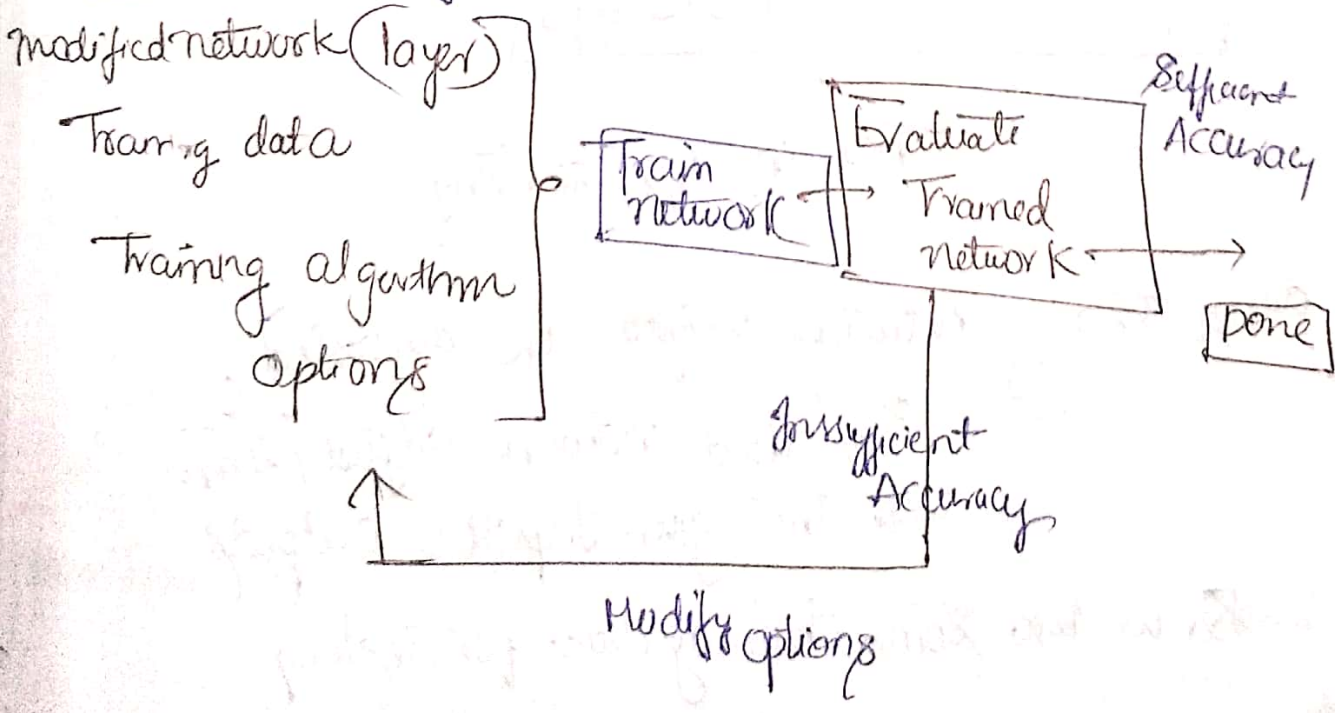
1) Network layer
2) Training data → network is trained with the known correct answer
3) Algorithm options (Supervised learning in ML Terminology)

↳ parameters for Algorithms

1) Batch Size
   (How many training image we use at each step
2) Max no of iterations take
3) Learning rate
   (How aggressively new iteration should change network parameter.

```
newnet =
trainNetwork( data, layer, options)
```

Transfer learning :



for training network

provide the known labels for training images.

```
flwrds = imageDataStore(pathToImages,
                        "IncludeSubfolders",
                        true,
                        "LabelSource",
                        "foldernames");
```

flwrds = Creates datastore

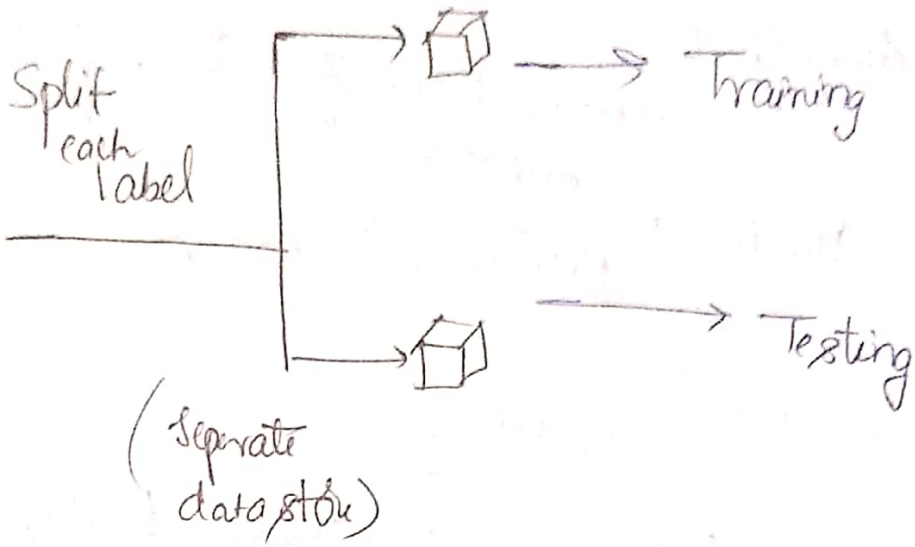
Creates, & gets all the images in the subfolders of the folders path. Stored in the variable pathToImages

* `flowernames = flwrds.Labels`

Training V/s testing

during training network weights are adjusted so that network adjust/learns with the given inputs & outputs

So we use some training data for testing




```
[ds1, ds2] = splitEachLabel(imds, p)
```

p = 0 to 1

```
* [[flwrTrain, flwrTest] = splitEachLabel(flwrds, 0.6)
```

- flwrTrain → training
- flwrTest → testing test
- flwrds → data store as a whole
- p → proportion

proportion of images with each label from flwrds

flwrTrain

flwrTest

keeps the files in order.

```
[ds1, ds2] = splitEachLabel(imds, p, "randomised")
```

randomly shuffle the files.

```
* [flwrTrain, flwrTest] = splitEachLabel(flwrds, 0.8, "randomized")
```

such that random selection of 80% of files are in "fleur train" dataset.

Unbalanced Training data

In some Applications, it is common to have many more images of one class than another.

for ex: when trying to detect defects, usually easy to obtain many images → No defect

Hard to obtain images with defects.

In this case

during data proportionally by class will result in the network being trained primarily by the images with no defects.

"Bias Training"

resulting in a network that plays the percentages." rather than truly learning to identify the features that show the presence of defects.

To avoid this, Training data : Split images have equal numbers on each class.

flwrds => Split the datastore into two datastores.
such that 500 files / n files in each category are in flwrTrain

[ds1, ds2] = splitEachLabel(imds, n)

[flwrTrain flwrTest] = splitEachLabel(flwrds, 500)

flwrTrain → set for training with each of the 5 categories having number of (n) files

flwrTest → testing

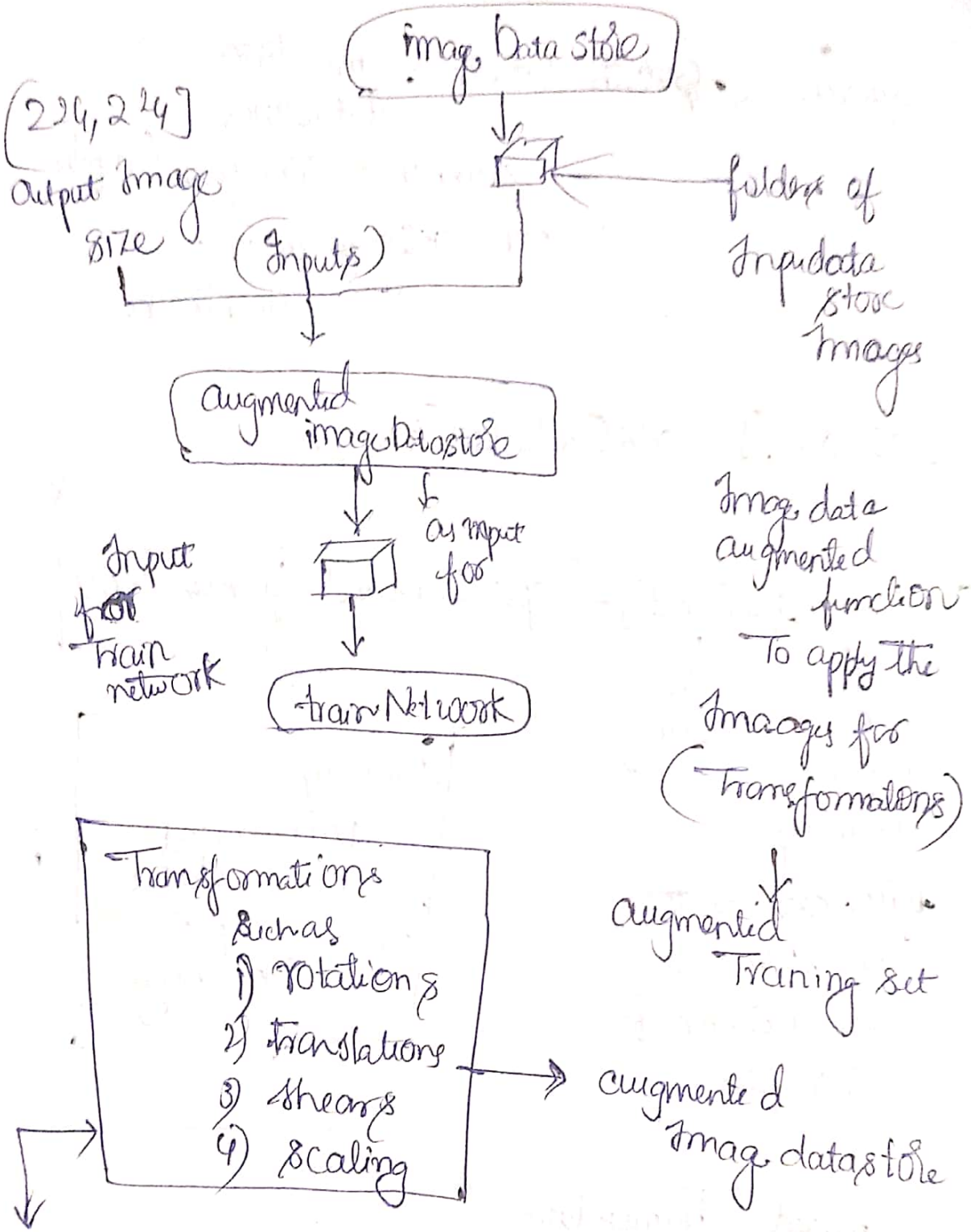
avoiding unbalanced data for training

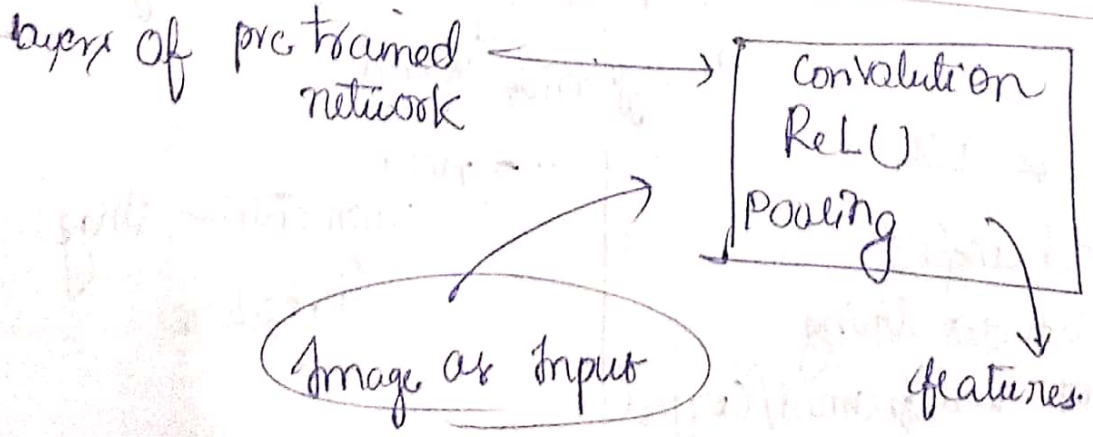
Augmented Training data

When training the network over training images may not be sufficiently varied

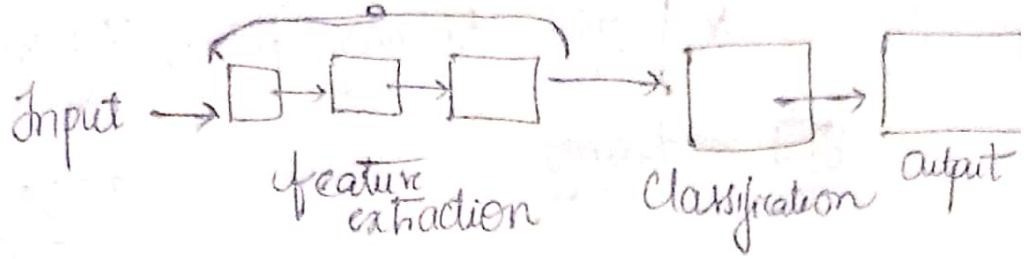
flowers with
- Odd angles
- Substandard lining
- Not well framed/cropped

→ need augmented image datastore.

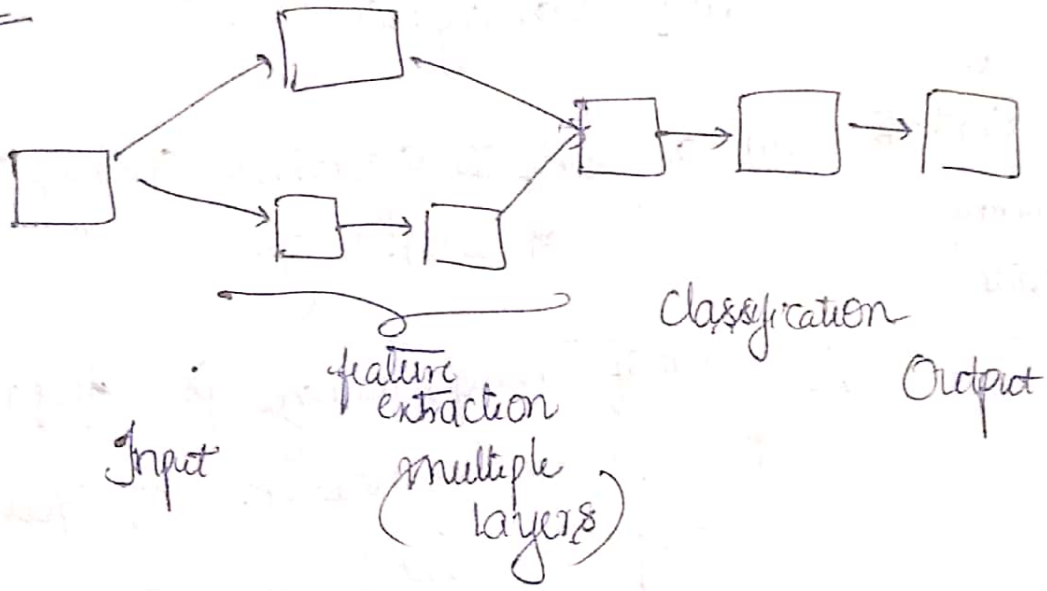




Series network

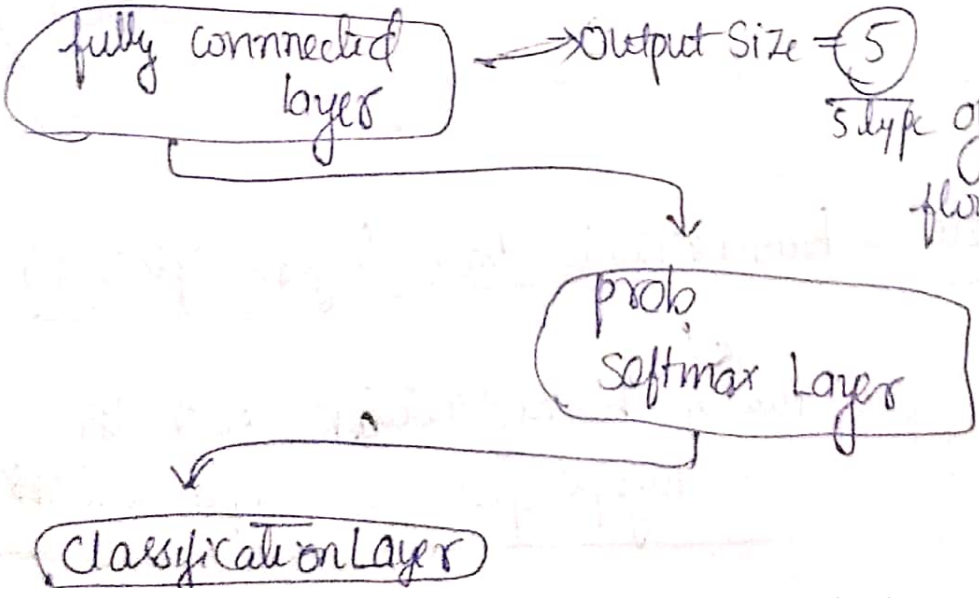


DAG network



Series network
└ layers

DAG network
→ layers
→ connections



* opts = trainingOptions('sgdm')

~~opts = trainng~~

SGDM optimizer.

Changing learning rate

Intial learning Rate = 0.01

↓

Controls how aggressively the algorithm changes the network weight.

learning rate

the goal of Transfer learning is to fine tune the existing network. (So you typically want to change the weights less Aggressively than when Traing from Scratch.

*

opts = trainingOptions ("sgdm", "InitialLearnRate", 0.001)

newnet = trainNetwork(data, layer, options)

↓

newly Trained network with the layers just like Input network.

Text display = progress of Training

Mini batch Accuracy :- % percentage of Training images that the network classifies correctly.

during Training Accuracy increases.

Network How confident about prediction.

Loss

measure of how far from a perfect prediction network works.

Totalled over the set of Training images

↓ decreases during the training.

Accuracy ↑ Loss ↓ } network getting better

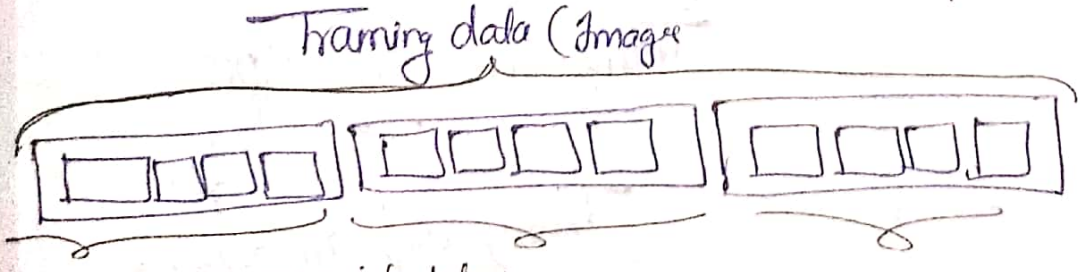
---

mini-batch

at each iteration (subset of Training images — known as mini-batch) used to update the weights

each iteration uses a diff mini batch

once whole Training set used. known as epoch.



max no of epochs (MaxEpochs)

Size of the mini-batches (MiniBatchSize)

* MaxEpochs
* MiniBatchSize

} parameters to set to the Training algorithm

* Mini batch loss
* Mini batch Accuracy

} for the reported during training are for the mini-batch being used in the current iteration.

* By default, the images are shuffled once prior to being divided into mini batches.

Shuffle

Using GPU

Speeds up many computations. Required for deep learning

(Graphics processing unit)

Can take long time

Training on Single GPU

Preferable for

Training deep learning networks

Evaluating network after Training

flowernet ⇒ network obtained by performing transfer learning on GoogLeNet. With flower species data

Trained with 30-epochs

enough time To learn the data

info = Information about Training

TrainingLoss
Training Accuracy record of performance of the network of the Training data at each iteration.

Info
- TrainingLoss
- TrainingAccuracy
- Base learnrate

* plot (info. TrainingLoss)

flwrpreds = Classify (flowernet, resizeTestImgs)

flwrActual = testImgs.Labels ⇒ How many of testImgs correctly classified by comparing Predicted Classification

labels prpty from the test Img

No of elements of two Arrays Match

numequal = nnz (a == b)

↓

To compare How many predicted classifications match the Coorect classification.

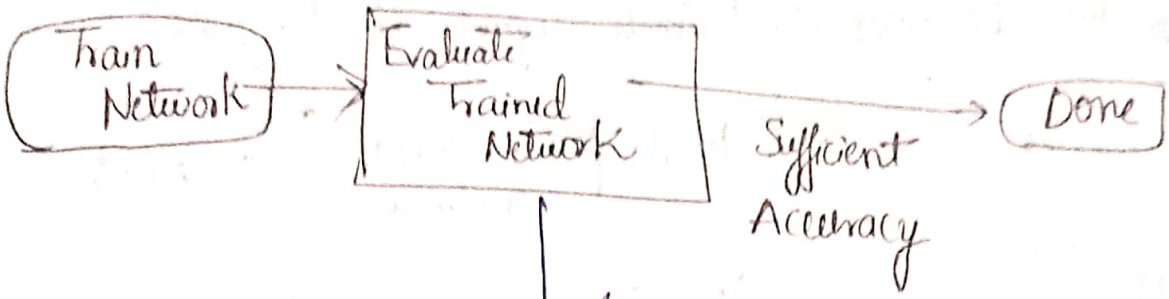
* numCorrect = nnz (flwrActual == flwrPreds)

* fracCorrect = numCorrect / numel (flwrPreds)

Loss, Accuracy = give overall measure of the network performance.

*

Confusionchart (flwrActual, flwrPreds)



Trained Network performs well on the training data

worst performance on Testing data

So

To avoid word performance.

Training data → Training data set

→ Validation dataset

CNN = convolutional neural network.

Object detection

↳ automatic car driving System.

Sequence classification.

Transfer learning Summary :-

Creat network

deepNetworkDesigner ⇒ launch deepnetwork designer

googlenet ⇒ load pretrained network "GoogleNet"

supported networks ⇒ list of available, pretrained network.

fully connected layer = Create new fully connected Layer

Classification Layer = Create new Output layer

Get Traing Images

Image Datastore = Create data store reference to Image files

augmented ImageData Store ⇒ pre process a collection of Image files.

SplitEachLabel = divide datastore into multiple datastores.

Set Traing Algorithm options

trainingOptions = (perform Training)
Create Variable containg training algo options

perform Training

Train network = perform Training

Use Trained Network to perform Classifications

Classify ⇒ obtain Trained network Classification of Input Images

evaluate Trained Network

nnz = Count non zero elements in an array

Confusion chart = Calculate Confusion matrix

Matplotlib

%matplotlib inline

```
import matplotlib.pyplot as plt

x= [1, 2, 3, 4, 5, 6, 7]
y= [50, 51, 52, 48, 47, 49, 46]

plt.plot(x, y)
plt.show()
```

```
plt.plot(x, y)
plt.plot(x, y, color="green", linewidth=5)
plt.show()
```

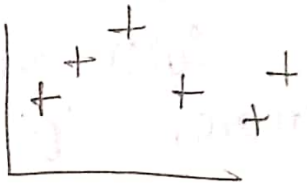
```
plt.plot(x, y, color='green')
plt.show()
```

```
plt.plot(x, y, color='green'
                linewidth = 5,
                linestyle = 'dotted')
plt.show()
```

```
plt.title('Weather')
plt.xlabel('Day')
plt.ylabel('Temperature')
plt.plot(x, y, color='green', linewidth=5)
plt.show()
```

```
plt.plot(x, y, 'g+')
plt.show()
```

green '+'



```
plt.plot(x, y, '+')
plt.show()
```

```
plt.plot(x, y, '--')
plt.show()
```

dashed plot

red dashed

```
plt.plot(x, y, 'r--')
plt.show()
```

red-diamonds

```
plt.plot(x, y, 'rD')
plt.show()
```

red stars

```
plt.plot(x, y, 'r*')
plt.show()
```

red stars joined with dashed line

```
plt.plot(x, y, 'r*--')
plt.show()
```

---

color = red
marker = 'D' diamonds
line style = ''

```
plt.plot(x, y, color='red', marker='D',
                linestyle='')
plt.show()
```

```
plt.plot(x, y, color='red', marker='*',
                linestyle='Dashed')
plt.show()
```

```
plt.plot(x, y, color='red', marker='+',
                markersize=20)
```

```
plt.plot(x, y, color='red', marker='+',
                markersize=15)
```

alpha ⇒ opacity factor

```
plt.plot(x, y, color='red', alpha=0.5)
plt.show()
```

```
plt.plot(x, y, color='red', alpha=1)
plt.show()
```

Bar chart

```
days = [1, 2, 3, 4, 5, 6, 7]
max_t = [50, 51, 52, 48, 47, 49, 46]
min_t = [43, 42, 40, 44, 43, 35, 37]
avg_t = [45, 48, 48, 46, 40, 42, 41]
```

```
plt.plot(days, max_t)
plt.plot(days, min_t)
plt.plot(days, avg_t)
```

```
plt.title("Weather")
plt.xlabel("Days")
plt.ylabel("Temperature")
plt.plot(days, max_t, label='max')
plt.plot(days, min_t, label='min')
plt.plot(days, avg_t, label='avg')
```

plt.legend(loc='best')
plt.show()

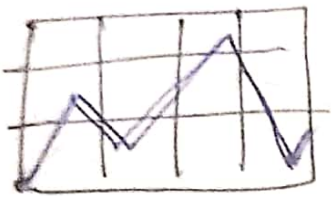
plt.legend(loc='best', shadow=True)

plt.legend(loc="best", shadow=True,
fontsize=large)

plt.legend(loc='best', shadow=True,
fontsize=small)

plt.grid()

plt.show()



Histograms

```
import matplotlib.pyplot as plt
import numpy as np

company = ['ABC', 'DEF', 'AMZN', 'FB']
revenue = [90, 136, 89, 27]
```

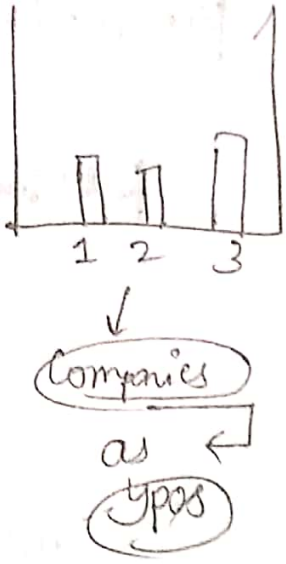
```
ypos = np.arange (len(company))
ypos

plt.bar(ypos, revenue)
plt.show()

plt.bar(company, revenue)
plt.show()
```

```
plt.bar (ypos, revenue)

plt.xticks(ypos, company)
plt.ylabel ("revenue(bln)")
plt.title("US Tech stocks")
plt.bar (ypos, revenue)
plt.legend(loc="best")
plt.show()
```



```
profits = [1, 2, 3, 4].
plt.bar(ypos - 0.2, revenue, width=0.4,
                    label='Revenue')
plt.legend()
plt.bar(ypos + 0.2, profits, width=0.4,
                    label="profit")
plt.show()
```

Histograms

```
blood_sugar = [113, 85, 90, ... 127]
plt.title("Hospital")
plt.hist(blood_sugar, bins=3, rwidth=0.75)
plt.show()
```

```
plt.hist(blood_sugar, bins=[80, 100, 120], color="green")
plt.show()
```

Pie Chart

```
exp_labels = ["Homerent", 'phone', 'car', 'Food', 'Others']
exp_vals = [1000, 500, 300, 200, 700]
plt.title("expenditure")
plt.axis("equal")
plt.pie(exp_vals, labels = exp_labels)
plt.show()
```

```
plt.pie(exp_vals, labels = exp_labels,
        radius = 0.75,
        autopct = '%0.2f %%')
plt.show()
```

```
plt.axis('equal')
plt.pie(exp_vals, labels = exp_labels, radius = 1.5,
        autopct = '%0.1f %%',
        shadow = True)
plt.show()
```

Save Matplotlib file

```
plt.savefig("piechart.jpg")
plt.savefig("piechart.png")
plt.savefig("piechart.png", bbox_inches = "tight",
            pad_inches = 2,
            transparent = True)
```

```
plt.savefig("C:/code/piechart.pdf");
```

pandas :- data Manipulation & Analysis

Tabular data

2D memory — dataframe

Series, dataframe

Series → 1D

Data frame → 2D

panel → 3D

```
import pandas as pd
df = pd.read_csv("G:\\project\\set.csv")
Print(df)
```

CSV

```
d = pd.read_csv("G:\\project\\....csv")
df = pd.DataFrame(d)
Print(df)
```

dictionary

```
di = { "Name": ["abc", "def", "ghi"],
       "Roll": [1, 2, 3],
       "perc": [90, 80, 70] }
```

```
df = pd.DataFrame(di)
print(df)
```

list of Tuples

```
l = [("Name", "Roll", "Perc"),
     ("abc", 1, 90),
     ("def", 2, 100),
     ("ghi", 3, 50)]

df = pd.DataFrame(l)
```

```
print(df.head()) ⟶ (Top 5)
print(df.head(2))
print(df.head(10))
```

```
print(df.tail())
print(df.tail(2))
print(df.tail(10))
```

```
print(df.describe())
print(df.shape)

print(df.shape)
```

```
print(df[0:10:1])
print(df["year"])
print(df["month"])          //retrieve by column
print(df[["year", "month"]]  //retrieve multiple columns
print(df[["year", "month"]][1:20:2])
```

get details of individual records for 3 records

```
for rec in df.head(3).iterrows():
    Print(rec)

for rec in df.iterrows():
    print(rec)
```

---

loc[] (Stop index included)

Column names are specified as names

iloc[] Stop index excluded

Column names are specified as indexes.

df.loc [row_number]

df.loc [5] get 5th row of dataset.

df.loc [10]

19th row, "year"

df.loc [19, ["year"]]

df.loc [19, ["year", "weekday"]]

* df.loc [row_number]

df.loc[1]

* df.loc [row_number, [column_name1, ...]]

df.loc [1, ["weekday", "year"]]

* df.loc[start : stop]

df.Loc [0:5]

df.Loc [start: stop, "column_name"]

df.loc[start:stop, ["column1", "column2" ...]] is

```
df.loc[0:5, "year"]
df.loc[0:5, "month"]
df.loc[0:5, ["year", "month"]]
```

df.loc[start:stop, "column1":"columnn"]

```
print(df.loc[0:5, "month":"rain"])
```

for ~~first~~ 5 rows from month to rain.

```
df.iloc[0:5]
```

→ Stop index excluded.

```
df.iLOC[0:5, 1:6]
```

1st → 5th column

for 0 → 4 rows

Stop not included

df.iLoc[1] only row 1

df.iLoc[0:3] get only top 3 rows

df.iloc[0:3, 1] Top 3 rows only column index-1

df.iloc[0:3, 1:6] Top 3 rows

[1 → 6)

(1 .. 5) 6 excluded

6th column excluded.

df.iloc[0:3, [1, 2, 3]]

df.iloc[0:3, [1, 2, 3, 4, 5]]

df.iloc[0:3, 1:6]

range for both rows & columns

df.iloc[1, 1]

Specified row with specified column

df.iloc[row number, column_numbr]

df.iloc[1, 1]

df.iloc[row_start : row_stop, column_num]

df.iloc[0:3, 1]

df.iloc[row_start : row_stop, [column1, column2]]

df.iloc[0:3, [1, 2, 3]]

df.iloc[row_num, [column1, column2 ...]]

df.iloc[1, [1, 2, 3, 4, 5]]

df.iloc[[row1, row2 ...]]

df.iloc[[1, 2, 3]]

df.iloc[[row1, row2, row3], [col1, col3, col5]]

df.iloc[[1, 2, 3], [1, 2, 3, 4, 5]]

df.iloc[: , [col1, col2 ...]]

all rows & specific cols

all rows all columns

print(df.loc[:])

slicing all the rows.

---

* df.sort_values("volume")

df.sort_values("Column_name")

* df.sort_values("day")

descending order

df.sort_values("columnname", ascending=False)

df.sort_values("Columnname", ascending=False)

df.sort_values("volume", ascending=False)

Sort dataset by multiple columns

```
df.sort_values(["Volume", "day"])
```

Manipulating dataframe

Add column

```
df["new column"] = default value
```

```
df["Total"] = 0
df["pass|fail"] = "fail"
df["grade"] = "pass"
df["holiday"] = 0.0
df["new_col"] = df["col 1"] + df["col 2"]
df["value"] = df["day"] + df["Month"]
print(df["value"])
print(df["value"][1:5])
```

df["new_col"] = Expression / Condition.

removing column

* df.drop(Columns = "Column_name")

→ get the dataset without the Spefied Column name

To delete the Column permanently from dataframe

* df.drop(Columns = "Value", inplace = True)

df.drop(Columns = 'Value')

df.drop(Columns = "Value", inplace = True)

→ removes the column from the dataframe permanently.

* remove duplicates

To identify whether data set is duplicated

* df.duplicated() boolean result: True, False

if any row duplicated ⇒ True

False
False
False
True

4th row duplicated

*

df.drop_duplicates()

↳ print the df without duplicates.

*

df = df.drop_duplicates(inplace = True)

print(df)

remove duplicates from dataframe permanently.

Handling Missing data

* remove missing data

dataframe.dropna()

dataframe.dropna(inplace = True)

df.dropna() ← prints dataframe after removing rows with missing data

df.dropna(inplace = True) → removes the rows permanently from the dataframe.

fill with default values

* dataframe.fillna(default value)

df.fillna('80') ← fill missing with 80

df["social"].fillna('80') → data frame with "social" column

df["rain"].fillna(1) → missing data = 80

"1"

---

* data filtering & conditional changes

① df.loc[Simple_condition]

ex: get data for the persons who got 'Maths' > 85

* df.loc[df["Maths"] > 85]
* df.loc[df["Maths"] < 85]

② df.loc[Compound_condition]

* df.loc[(df["Maths"] > 60) & (df["Maths"] < 85)]

* df.loc[(df["Maths"] > 60) & (df["Physics"] > 60)]

* get student details whose name contains "n"

df.loc[df["Name_student"].str.contains("n")]

df.loc[df["Name"].str.contains("n")]

③ df.loc[Simplecondn.str.Contains(str)]

④ df.loc[Simple condn.str.startswith(str)]

* df["percentage"] = (df["Total"]/500) * 100

print(df)

---

df["Grade"] = "pass/Fail"

print(df)

Set grade by percentage by Conditions

*

df.loc[df["percentage"] < 40, ["Grade"]] = "Fail"

*

df.loc[(df["perc"] ≥ 40) & (df["perc"] < 60), ["Grade"]] = "pass"

*

df.loc[(df["perc"] ≥ 60) & (df["perc"] < 70), ["Grade"]] = "First-Class"

*

df.loc[df["perc"] ≥ 70, ["Grade"]] = "distinction"

Export dataframe to
Excel
CSV &
Text File

```
df.to_excel(PATH)
df.to_excel(PATH, index=False)
df.to_csv(PATH)
df.to_csv(PATH, index=False)
```

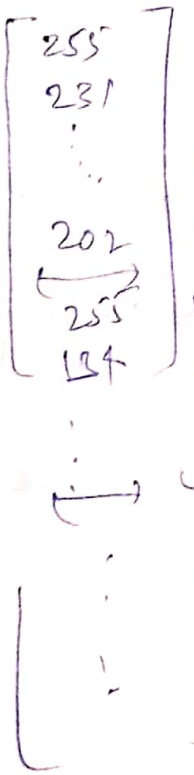
→ Basics of Neural Network:-

1) Binary classification:-

→ Logistic Regression Is thus Used, for the Binary classification.

→ InCase, Consider a Image of 64x64, and Inorder to Compute, Computer might go for '3' Matrices of Each "64x64", to store Individual Pixel. slot.

→ So, Those '3' Matrices which contains the Details of Pixels → we this UnRoll them Into the "Vector" say X



(64x64) + (64x64) + (64x64)

= 3x64x64 → 12288 → Dime

$n = n_x = 12288$

Which Is often size of My Matrix unrolled to Vector.

X → Y

(Input) (0 or 1)

→ How it looks?

$X \longrightarrow Y$

$X \in \mathbb{R}^{n_x}$ & $y \in \{0, 1\}$.

How this It all flows,

→ we have m training set } $M_{train} = m$

$M_{test} = \#test$

→ What train, usually contains } $\{x^{(0)}, y^{(0)}, x^{(1)}, y^{(1)} \ldots x^{(m)}, y^{(m)}\}$.

$X = \begin{bmatrix} | & | & | & & | \\ x_1 & x^{(2)} & x^{(3)} & \cdots & x^{m} \\ | & | & | & & | \end{bmatrix}$

↓ Capital

→ m training along

→ The Command for finding the Shape of my Matrix → X.shape()

→ Which Is Size of My Vector say n_x.

No of Rows

$X \in \mathbb{R}^{n_x \times m}$

$X.shape = (n_x, m)$.

↓ Input Matrix of My Model

$Y = [y_1\ y_2 \ \ldots \ y_m]$

as 0 & 1 as output wega-

$Y \in \mathbb{R}^{1 \times m}$

$Y.shape = (1, m)$

Given X, } → where it, each scenario that Is Image, which to be predicted, so, ($\hat{y}$)

Is the Estimated output, for X as Input

$$\hat{y} = P(y=1 \mid X)$$

↳ It is the Probabilistic Estimation of $y=1$, for the Particular X,

* Assume, two Parameters that these are two things $w \in \mathbb{R}^{n_x}$ & $b \in \mathbb{R}$

$\hat{y} = w^T X + b$ } but $0 \le \hat{y} \le 1$

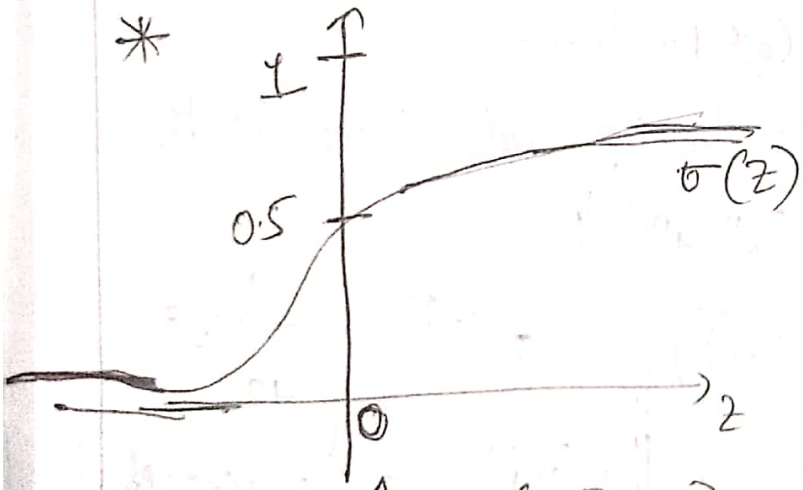
$\hat{y} = \sigma(w^T X + b)$

→ So, Thus In order to Adjust stigmoid Func is Used

This Normal linear Regression, gives Just Bigger >1 & -ve too!

→ which Is often Invalid, to our Result

* → Graphical Representation of Stigmoid Funct



$\hat{y} = \sigma(w^T x + b)$ $\hat{y} = \sigma(z)$

↳ z

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

Results From it,

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

If 'z' is too large, then,

$\frac{1}{1+\frac{1}{\text{large}}} = \frac{1}{1+0} = 1$ //

* If z is very small -ve, Negative Number, then

$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\frac{1}{e^{z}(\text{large})}} \approx \frac{1}{\text{large}} \to 0$

→ Other Notation Way:-

* Assume $x_0 = 1$, $x \in \mathbb{R}^{n+1}$ → which is one Column itself
↓ the Size of Vector.

$\vec{y} = \sigma(w^T x)$

$\boxed{\vec{y} = \sigma(\theta^T x)}$

$\theta^T = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ ($\theta_0 \to b$, $\theta_1 \ldots \theta_n \to w$)

other way, $\begin{bmatrix} \\ \\ \\ \end{bmatrix}$ → nx values In Vector.

(3.) Logistic Regression Cost Function:-

So, we given For the training Set, many inputs,

$\{(x^1, y^1), (x^2, y^2), \ldots (x^m, y^m)\}$.

$\hat{y}^{(i)} = \sigma(w^T(x^i) + b)$ $\quad \sigma(z^i) = \frac{1}{1+e^{-z^i}}$

$\sigma(z^i) = \frac{1}{1+e^{-(w^T(x^i)+b)}}$

Loss (ERROR) function:-

→ Why this? Inorder to Identify How well our Algorithm Is Doing these.

→ $L(\hat{y}, y) = \frac{1}{2}(\hat{y}-y)^2 = \frac{1}{2}(y - f(z))^2$

General Compare them. → one of the ways Inorder to Reach Optimization.

→ So, In Logistic Regression, the loss function, might be,

$\mathcal{L}(\hat{y}, y) = -\left(y \log \hat{y} + (1-y) \log (1-\hat{y})\right)$

→ has to be small.

→ If $y=1$

$L(\hat{y}, 1) = -y \log \hat{y}$

$= -\log \hat{y}$ = $\hat{y}$ → large to be so that $\log \hat{y}$ Too large.

→ If $y=0$,

$L(\hat{y}, 0) = -\log(1-\hat{y})$

→ $\log(1-\hat{y})$ to be large, so that the overall value Is Negative In Nature, which tells less loss (or) Comparison [$\hat{y}$ → too small]

→ Limitations of loss function:-
→ This is often Restricted to one Model | Neuron.

→ Cost Function:- ✓
↳ which is Applied to Many layers (or) Neurons in my Model

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

( ) , ( ) , ( ) ( )
1st Neuron, 2nd Neuron, 3rd Neu
m Neurons

$(\hat{y}^{(i)}, y^{(i)})$ $\hat{y}( )$ $(\hat{y}^{(m)}, y^{(m)})$

So, Summation of all the Neurons Loss Function.

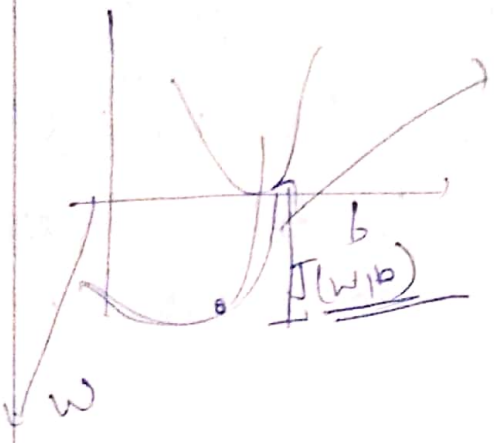
$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)}) \right]$$

OTHER Parameter (Gradient Descent):-

So, Indirectly, $J(w,b) = -ve$ } → we thus, Always Needed to be a Negative loss

Indirectly, we thus Have to Select w & b Such that, It Minimises $J(w,b)$

→ [$J(w,b)$] } → Representation In Graph.



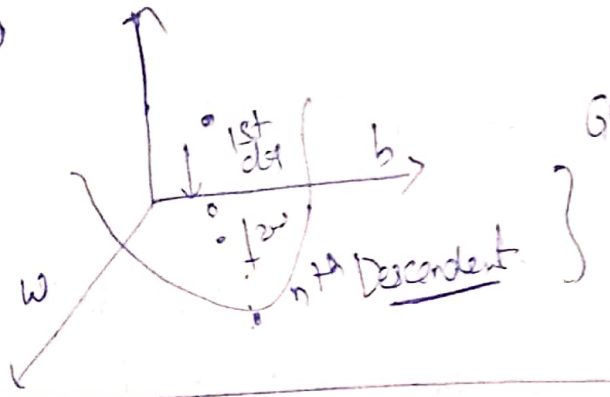
So, it Is the Convex Bowl type Repo

① to be on w/ b

b=0 Inorder to be Minimum In NATURE

→ The Elevatory steepDownward, of Gradient Descent to Reach the final optimal Point of Cost Function.

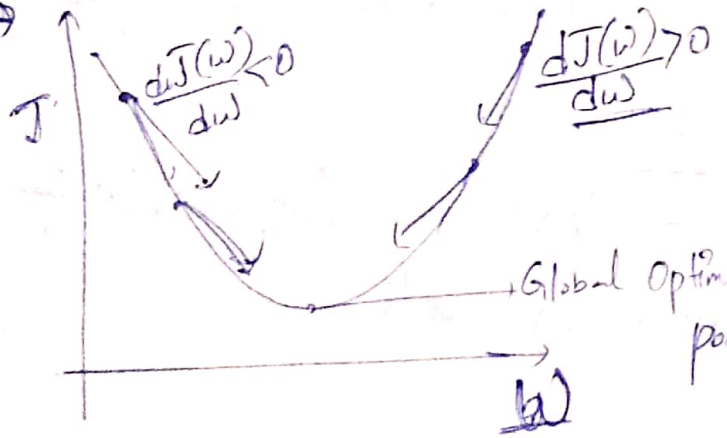
→



Simultaneous Updation of Values:-

Gradient Descent:-

→



Repeat {

$$w_j := w - \alpha \frac{dJ(w)}{dw}$$

$$w' := w - \alpha w$$

}

$\frac{dJ(w)}{dw}$ which Is the slope of curve

↳ Based on which we do process

$J(w,b) =$ $\begin{cases} w := w - \alpha \frac{dJ(w,b)}{dw} \\ b := b - \alpha \frac{dJ(w,b)}{db} \end{cases}$ $\frac{\partial}{\partial w} \frac{dJ(w,b)}{db}$ } q. for partial Derivative

* So, the New values which we obtained Replace it In Equation, so, Inorder to Derive, New Vs

---

→ An Explore to the Derivative:-

* Usually, the Slope of Straight line Remains Same → So $\frac{d}{dx} f(x)$ } → Remains Constant Itself.

→ INTRO To the Computational GRaPH:—

$J(a,b,c) = 3(a+bc)$ } → Consider a Example Of Computing this kind of fun'

$bc \to u$
$v \to a+u$
$3v \to J$

→ Backward Propagation } → Thus Uses finding, the Derivative

→ Forward } → Used for Computation of the Functions As well

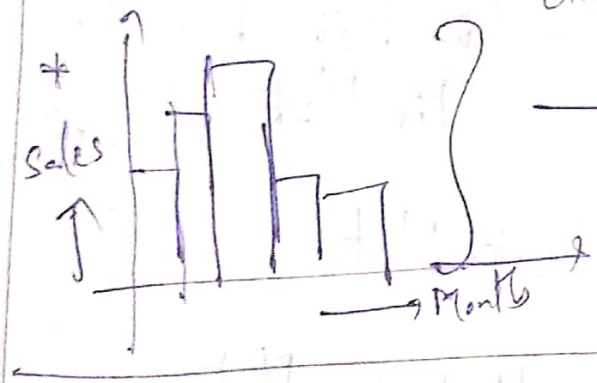
→ Consider,

A Example to get Info

→ The Matplotlib Library :-

* In the Python
* Mostly Used library, for the Data Visualisation, things!!

* Data } output (Mostly) → Represent through Graph, Chart, Bar Graph & Also the other formation!.



→ Many things like, Max, Min, Same, Mode, } all other Info Can be gathered/ known From the charted plot.

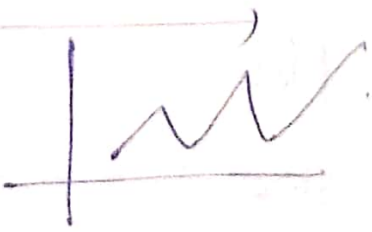
* One Can be Easily Understand those kinds of Visuals too!!

* Preferred in the Business Models, too!!
→ Like the, 1) Decision Taking Process (1).
2) Visual. (2)
3) large Data could be Interpreted (3)

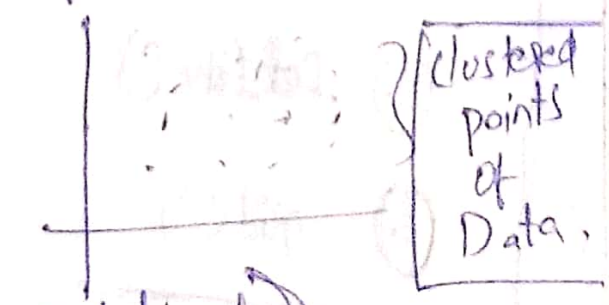
PYTHON →

* Various Types of DataVisualisation:-

→ (1) Line plotting.



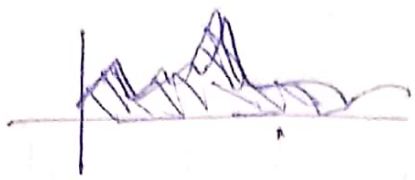
→ (2) Scatter Plot



→ (3) Histogram

→ (4) Bar Graph (Vertical Mirrored Histogram Evolution)).

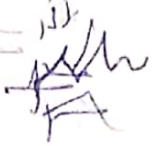
* Pie-chart Way.
* Areal / Integral Mode of Representation



* Which to Use when?

→ When the Outputs of the one Entities are supposed to be compared with others.

⟶ Comparison Mode :- Line Plot, Bar Plot, Pie Plot.



⟶ Distribution :- Scatter Plot.

⟹ For Continuous Data Comp. : Histogram Plot,

(*) Matplotlib Concepts :- ⟶ this library

* pip install matplotlib. ✓

imp matplotlib.pyplot as plt

* The Most Fundamental Functions In the Plot:

1. Plot() ⟶ to plot some graph.
2. Show() ⟶ shown Screen
3. ~~Set~~title() ⟶ For visual that we draw we set title.
4. grid() ⟶ Sets True (or) sets false for it.

* Even try to Able → Change color of plotted Axis, Which we got Here (r, b, g),

* linestyles for line plot } single dotted, Space Dashed Dashed, & others forms

*Markers } → Data Points are Highlighted (or) marked Here. P, S, V, 'O' }

→ LINE PLOTTING :-

→ For the Comparison of the Data, We Use it.

→ Steps

First → + Has to be Made this Is called As Figure( )

Plot → Finally, plotting

Show → Displaying the Output Here!

→ fig = plt.figure (figsize = (10,5));
plt.plot ('X_data');
plt.show( );

→ So, the Raw Plot of the Graph we Got,
→ Then Comes the Seting Name / Titles to the Both the X-axis & Y-axis too!!

* plt.xlabel ('X_data');
  plt.ylabel ('X_data');

* InCase, if we wish to provide or give title to it then,
  plt.title ("Student form Registry");

→ InCase, if we wish to have two plots Inside the Graph, then, we go for the Concept of the Subplots
→ for the Main Plot Itself try to Create the Subplot.
→ Based on which plot As Many things As If you want. (a greater scope for Visualisation)
*

→ Recurrent Neural Network (RNN) :—

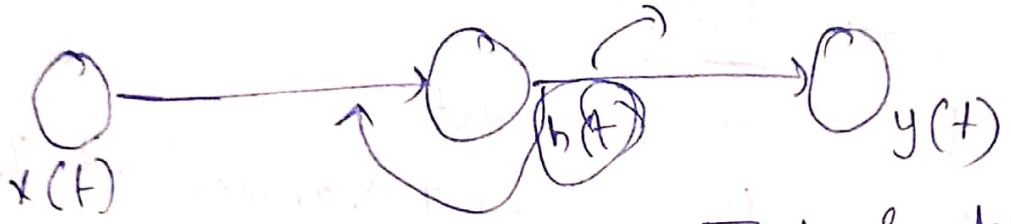
* Why RNN Why Not ANN

No loop Back link Connected to the Previous Node

→ So, we make the hidden feature ("hidden state") to Depend on the Previous Hidden State.

Hidden Weightage

→



→ So, Hidden Part Depends on the Input & also on the Previous Hidden State too!!

→ So, loop Back Makes the Time Delay of 1 step Backwards.

→ and It is the Non-linear function As Well Too!!

$$h_t = \sigma\left(W_{xh}^T x_t + W_{hh}^T h_{t-1} + b_h\right)$$

$$\hat{y}_t = \sigma\left(W_o^T h_t + b_o\right)$$

b → Constant

→ Sigmoid function

→ Transpose of the Weighted Graph / Matrix Feature

$h_t = \sigma(W_{xh}^T x_t + W_{hh}^T h_{t-1} + b_h)$

$\hat{y}_t = \sigma(W_o^T h_t + b_o)$.

$x \rightarrow$ Input

$h \rightarrow$ hidden

$o \rightarrow$ output

$Y_h \rightarrow$ Input to hidden

h_n = hidden to hidden

---

* How to calculate the Output?

$x_1, x_2 \ldots x_t$ } $\rightarrow$ x-Variables and those Corresponding values

Shape(x_i) = [ ].

First step: $h_1 = \sigma(W_{xh}^T x_1 + W_{hh}^T h_0 + b_h)$.

h_0? $\rightarrow$ which is the Initial hidden state

$\rightarrow$ It is often Referred to As learned Parameter, (OR) the Value Set as zero.

Now. $\hat{y}_1 = \sigma(W_o^T h_1 + b_o)$

Similarly, we can find h_2

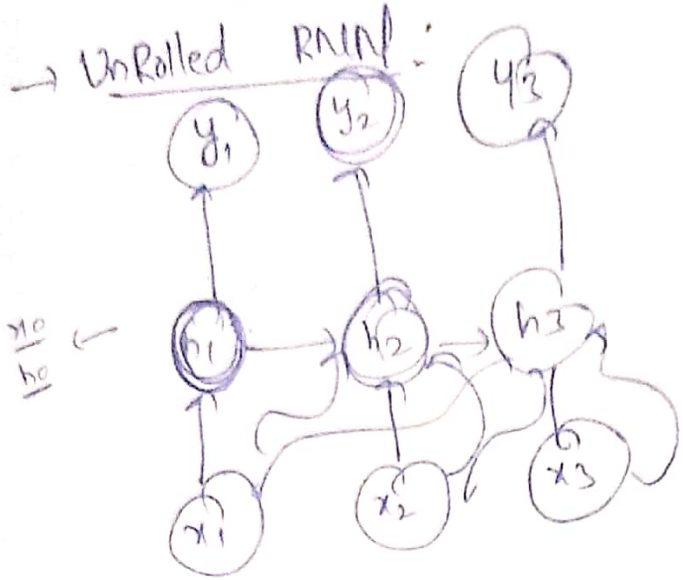
$h_2 = \sigma(W_{xh}^T x_2 + W_{hh}^T h_1 + b_h)$

$\hat{y}_2 = \sigma(W_o^T h_2 + b_o)$

$\rightarrow$ keep on going to find all

→ $P(y=k|x)$ $P(y_t=k$

→ UnRolled RNN:

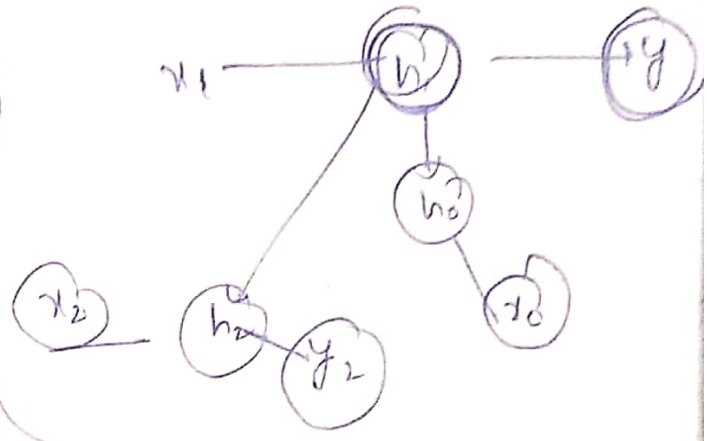


→ classification Probability :-

$P(y_1=k|x_1)$

$P(y_2=k|x_1,x_2)$

$P(y_3=k|x_1,x_2,x_3)$



Pseudo Code for RNN

Given:-

W_{xh} → Input to hidden

W_{hh} → hidden to hidden

b_h → hidden bias

W_o → hidden to output weight

b_o → output bias

x → T x D Input Matrix

tanh hidden Activation

Softmax output Activation.

→ Simple RNN Code Preparation :-

Steps: ① load the Data
→ Although the shape, that we use from the DataSet Is Different (N x T x D).

② We Build the Model.
→ Train
→ Evaluate Model
→ Make Predictions from Model.

---

③ RNN Expects 3-D Input
→ [N x T x 1] as Input

→ From (tensorflow.keras.layers) import Input, SimpleRNN, Dense,

In this, Corresponding Module we get

- i = Input (shape = (T, 1)), Mentioning Hidden Layers
- x = SimpleRnn (5, activation = 'relu') (i)

We make it Dense x = Dense (1) (x)

model Model (i, x)

→ Thus Creating model using Input & Dense Cx

Training the Model :-

```
r = model.fit( X[:-N//2], Y[:-N//2],
               epochs = 80,
               validation_data = (X[-N//2:], Y[-N//2:]))
```

Input In Shape :- N x T x D

- N → No. of Sample
- T → No. of Sequences
- D → No. of Features

→ Output of RNN } N x K → No. of Output Nodes.

→ If Talone given Then,
Reshape as (1, T, 1) [0.0] → Make as form

Linear In Nature

* Set the Activation In Python Code As → None.

*Modern RNN Units :-

- LSTM (long-Short Term Mem)
- GRU Gated Recurrent Unit)

→ Base Paper:- Long-Term Traffic Prediction

* For Mining the Relationship B/w the traffic Flow Data and the Combination of Key Contextual Factors. } → A Multi-layered Supervised Algorithm Is Used.

* Attributes like:
1) → Day of Week
2) → Weather
3) → Season.
4) Big Events

} Using Batch-Training Method. (we try to Reduce the Training Times)

* Time Stamp used for → Starting & Ending Points.

→ UnSupervised Algo
Clustering, Association
Comes Under UnSuper

Regression
Classifier
} Supervised

* Introduction:-
→ More Useful for Local Government Agencies to Predict potential congestion, Among Roads & to find Any Alternate certain Roads
→ Make the Tourists & Travelless, to Travel / Make their trip Based on Knowning the Traffic In More Advance.

* Two types } long term & short term } We Mostly Focuses on the Predicting the Traffic on the which Day, By fixing Target Day

→ Short-term

Statistical Method

ML Methods

ANN & other Hybrid Methods

→ Long-Term

Completely Made Using the DNN (Deep Neural Networks)

→ Literature Review

① Using Historical Data

Days are classified Into Equal Time Intervals

& Each Time Interval Is Due/ Averaged

& Finally that Average Value Is, Taken as o/p

This's called Prediction. After classfiee

Done with Smoothing & Historical Averaging things

* Patterns are Not gets changed By Weekends, But also due to Contextual Factors

Paper focuses on Predicting Traffic Flow on the Whole Day Next to FutuRE.

→ For long-term}
Similar & Significant Difference B/w Daily Patterns also Exists. & will affect Forecast.

→ Liter (2016)
→ Historical Data Is Splitted Into gRoups, & Transformed Into Some Wavelet Analysis, things.

1) JVS
2) Cisco
3) Fidelity
4) Wells
5) Optum
6) Walmart
7) Paypal
8) Lyft
9) Wells Fargo
10) Amazon

→ (2014)

→ Directly Taken & is Used to predict the Traffic Flow on the Particular Day Based on the Historical Data

→ (2016)

→ Split Into two Groups that Is, Basic Series and the Deviation Series As well!!

→ Incase If their Is Any Deviation, that Is Done | Calculated Using the Common Time Series

→ → Historical Data Is first Selected, Based on the Target day Is Workday/Not.

→ Using the ARIMA and the Regression Neural Network (GRNN) we Done it.

Both Comes Under Common Time Series Forecasting Things

AutoRegression Integrated Moving Algorithm (ARIMA)

(24) categories/groups like:-
(1) Weekend, Sunday/Saturday
(2) Summer/Non-Summer
(3) Rainy/Not Rainy
Etc

→ K-Means Clustering
→ Wavelet Analysis.
→ SVM (Support Vector Machines)
→ Euclidean Distance

No clear Method for the Selection of (Data Exists)

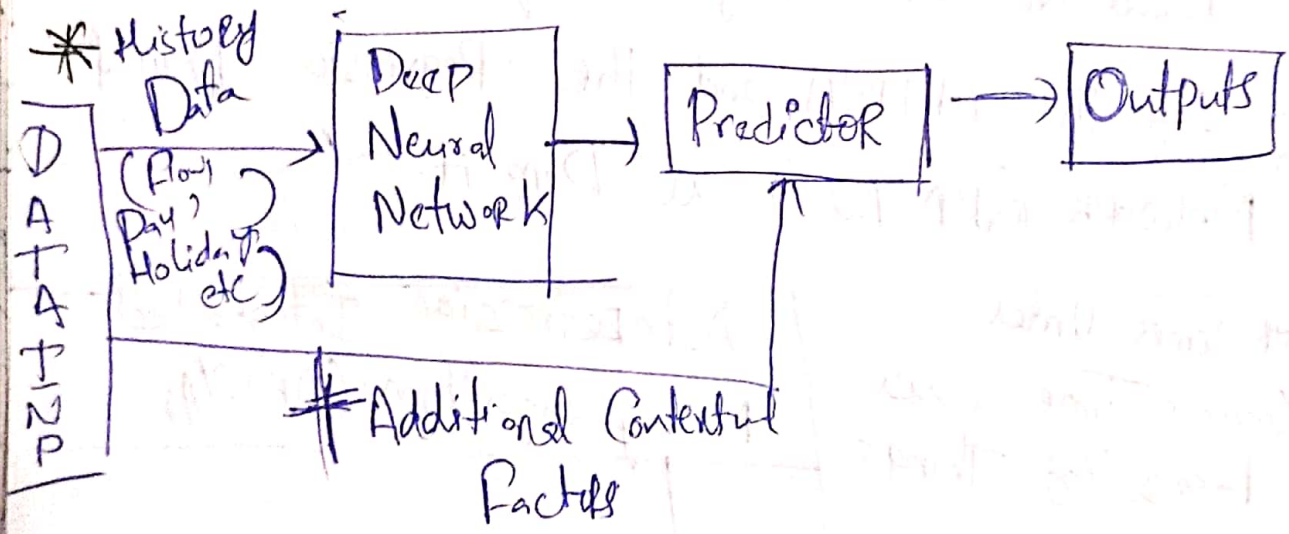
---

* Machine learning Algorithm } for a particular Predictor

→ 6:00 - 6:10 considering as Starting & Ending Point which are again Additional Contextual Factor

---

Methodology:-



---

(1) Methods Used ARe:-

(a) Prediction logic ——— DNN

(b) Batch Training. ——→ To generating Predicator

→ 23 csv files → 1 lakh lines in each csv files

Stamp	YEAR	Month	Day	Hour	Min	Week day	Holi day	Day light	Pain	V

→ Every "5 minutes"
Every '10' min
Every '15' "
Every "30" "
Every "45" "
Every "60" "

2015 , 2016

Stamp
- 5 lakhs

00:00:00
↓
24hr

Whole Year

2015 (Whole Yes) → 2016 (3 months only) } Trained

2016-01 (30 minutes) A copy

5
10
15
20 25
~~25~~ 50
30 55
35 60
40

5
10
15
20
30
60

6 lakhs Records
&
6 CSV files

1 lakh In Each file

2016 Each Months → 1500
3 × 1500 → 4500

* Epochs Used For Training } → 30,000

* Epoch :- "Complete Pass over the Entire, training Data Set During the Training phase, of the Neural Network"

---

* If

Epochs ↑ → Patterns good ↑

Efficient Model Training ↓ ↑

→ more Epochs → Model Starts Memorizing training Data Instead of generalisable pattern (Overfitting)

→ less Epochs → Underfitting

→ Features → 9.

Numpy :- Scientific computations

Lists = slow

Numpy = faster

5 → 00000101 → 00000000 00000000 00000000 00000101

Numpy → int 32

numpy ⇒ fixed type

faster — faster to read less bytes of Memory

Why? — No type checking when iterating through Objects

contigous memory

Numpy

usys for — Scientific computations. (Matlab)

Machine learning

plotting (Matplotlib)

Backend.

---

import numpy as np

→ a = np.array([1,2,3])

print(a) → [1,2,3]

→ b = np.array([[1,2,3],[4,5,6]])

print(b) → [[1,2,3],
[4,5,6]]

→ print(a.ndim) 1

print(b.ndim) 2

→ print(a.dtype) — int32 (3,)

print(b.dtype) — int32 (2,3)

print(a.shape) (3,)
print(b.shape) (2, 3)

---

print(a.itemsize) 4 bytes each item
print(b.itemsize) 4 bytes each item

---

print(a.size) 3 elements
print(b.size) 6 elements

---

print(a.nbytes) (3*4) ⇒ 12
print(b.nbytes) (6*4) ⇒ 24 (bytes total size)

---

a = np.array([[1,2,3,4,5,6,7],
[8,9,10,11,12,13,14]])

[[1,2,3,4,5,6,7]
[8,9,10,12,13,14]]

→ print(a[1,5]) ← (13)

value at 2nd row 6th Column

→ print(a[0:])

~~1st Column~~

1st row

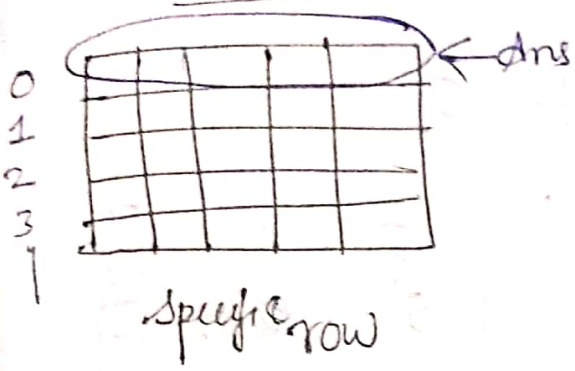
start → end

[1,2,3,4,5,6,7]

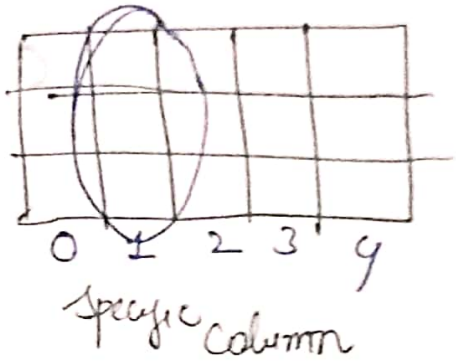
Specific column

```
print(a[:,1])
print(a[:,2])
```

a[0, :]



a[:, 1]



Specific row with

Start index : end index : Step size :-

Includes | not Includes | >0 Integer

print(a[0, 1:6]) == print(a[0, 1:6:1])

Column [1-6)

row 0

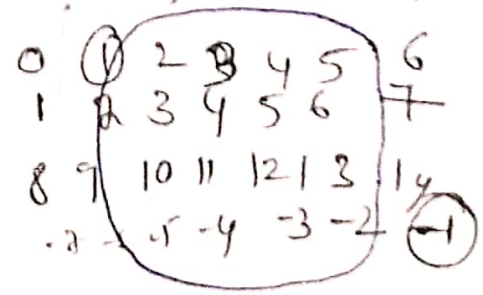
step : 1 default ⟶ [2, 3, 4, 5, 6]

print(a[0, 1:6:2]) ⟶ [2, 4, 6]

```
[ 0 1 2 3 4 5 ]
[-6-5-4-3-2-1 ]
```

print(a[0, 1:-1:2]

[2, 4, 6] ⟵



Change Values

```
a[1,5] = 100
print(a[1,5])
```

Set Column with all 5's

```
a[:, 2] = 5
```

```
[[1  2  5  4  5  6  7]
 [8  9  5  11 12 100 14]]
```

```
b = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
print(b)
```

→ `print(b[0,1,1])`

```
[[[1,2]
  [3,4]]
 [[5,6]
  [7,8]]]
```

```
[[[1,2]      [[5,6]
  [3,4]]      [7,8]]]
```

b[0,1,1] ⇒ 4

→ `print(b[:,1,:])`

(3, 4, 7, 8)

```
array([[3,4],[7,8]])
```

print(b[:,0,0]) ⇒ array([1,5])

#replace b[:,1,:] = [[9,9],[8,8]]

print(b) ⇒ [[[1,2]
[9,9]]
[[5,6]
[8,8]]]

---

different types of Arrays

np.zeros(5) ⟶ array([0.,0.,0.,0.,0.])

np.zeros((2,3)) ⟶ array([[0.,0.,0.],
[0.,0.,0.]])

np.zeros((2,3,3)) ⟶ array([[[0.,0.,0.],
[0.,0.,0.],
[0.,0.,0.]],
[[0.,0.,0.],
[0.,0.,0.],
[0.,0.,0.]]])

np.Ones((2,2)) ⟶ [[1,1]
[1,1]]

np.ones((1,2,3)) ⟶ [[[1.,1.,1.]
[1.,1.,1.]]]

np.ones((4,2,2), dtype='int32') [[1,1]
[1,1]]

np.zeros((2,2), dtype='int32') [[0,0]
[0,0]]

np.zeros((3,3), dtype='int32') [[0,0,0]
[0,0,0]
[0,0,0]]

Any other number

→ np.full((2,2), 100, dtype='int32')

⇒ array([[100, 100],
[100, 100]])

→ np.full((2,2), 99, dtype='int32')

⇒ array([[99, 99],
[99, 99]])

---

a = [[1, 2, 3, 4, 5, 6, 7]
[8, 9, 10, 11, 12, 13, 14]]

to get an array (same shape) with value as (4)

print(np.full_like(a, 4))

[[4, 4, 4, 4, 4, 4, 4],
[4, 4, 4, 4, 4, 4, 4]]

---

random values specified size

np.random.rand(4, 2)

print(np.random.rand(4, 2)) ⇒ array([[0.8321, 0.0431],
[0.1326, 0.1...],
[0.4, 0.6],
[0.4, 0.9]])

print(np.random.random_sample(a.shape)

random Integers

~~np.random.randint(7, shape=(3,3))~~

np.random.randint(7, size=(3,3))

- 7 → upto 7
- size=(3,3) → size/shape of array

```
[[5, 3, 3],
 [4, 4, 5],
 [0, 4, 0]]
```

np.random.randint(4, 7, size=(3,3))

```
[[6, 5, 4],
 [4, 4, 5],
 [6, 6, 5]]
```

np.random.randint(-4, 8, size=(3,3))

```
array([[-2, -4, -4],
       [ 6,  6,  3],
       [ 3,  2,  2]])
```

Identity Matrix

np.identity(3)

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
arr = np.array([1,2,3])
r1 = np.repeat(arr, 3, axis=0)
print(r1)
```

[1, 1, 1, 2, 2, 2, 3, 3, 3]

```
arr = np.array([[1,2,3]])
r1 = np.repeat(arr, 3, axis=0)
print(r1)
```

[[1,2,3]
 [1,2,3]
 [1,2,3]]

```
Output = np.ones((5,5))
print(Output)
```

[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]

```
Z = np.zeros((3,3))
```

replace

```
Z[1,1] = 9
print(Z)
```

⟹

[[0. 0. 0.]
 [0. 9. 0.]
 [0. 0. 0.]]

Z =

[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

replace set of values

```
Output[1:4, 1:4] = Z
print(Output)
```

[row1 → row4) [Column1 → Column4)

[[1, 1, 1, 1, 1]
 [1, 0, 0, 0, 1]
 [1, 0, 9, 0, 1]
 [1, 0, 0, 0, 1]
 [1, 1, 1, 1, 1]]

```
a = np.array([1,2,3])
b = a
b[0] = 100
```

Both a & b Changes

print(a) = [100 2 3]
print(b) = [100 2 3]

Copy

```
a = [1,2,3]
b = a.copy()
b[0] = 100
```

a = [1,2,3]
b = [100 2,3]

Arithmetic operations

```
a = np.array([1,2,3,4,5])
```

a+2 ⟹ [3 4 5 6 7]
a-2 = [-1 0 1 2 3]
a*2 = [2, 4, 6, 8, 10]
a/2 = [0.5 1 1.5 2 2.5]
a**2 = [1, 4, 9, 16, 25]

```
b = [1,2,3,4,5]
print(a+b)

print(np.sin(a))
print(np.cos(a))
```

```
a = np.ones((2,3), dtype='int32')
b = np.full((3,2), 2)
```

⟶ [[1,1,1], [1,1,1]]
[[2,2], [2,2],

```
np.matmul(a,b)
matmul(a,b)
```

array([[6, 6], [6, 6]])

```
C = np.identity(3)
np.linalg.det(C)
```

Statistics

```
Stats = np.array([[1,2,3],[4,5,6]])
print(Stats)
[[1, 2, 3],
 [4, 5, 6]]
```

```
np.min(Stats)   → 1
np.max(Stats)   → 6
```

```
np.min(stats, axis=0)  → [1, 2, 3]
```
Column wise

```
np.max(stats, axis=0)  → [4, 5, 6]
np.min(stats, axis=1)  → [1, 4]
np.max(stats, axis=1)  → [3, 6]
np.sum(stats, axis=0)  → [5 7 9]
```
Sum of all values Column wise

```
np.sum(stats, axis=1)  → [6, 15]
```

~~np.avg(stats, axis=0)~~ ~~avg column wise~~

~~np.avg(stats, axis=1)~~ ~~avg rows wise~~

reshape

```
a = np.array([[1,2,3,4],
              [5,6,7,8]])
```

```
[[1, 2, 3, 4],
 [5, 6, 7, 8]]
```

```
b = a.reshape((8,1))
```

```
[[1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]]
```

```
b = a.reshape((2,4))
```

```
[[1, 2, 3, 4],
 [5, 6, 7, 8]]
```

```
b = a.reshape((4,2))
```

```
[[1, 2],
 [3, 4],
 [5, 6],
 [7, 8]]
```

```
v1 = np.array([1,2,3,4])
v2 = np.array([5,6,7,8])
np.vstack([v1, v2])
```

```
[[1, 2, 3, 4],
 [5, 6, 7, 8]]
```

```
v3 = np.vstack([v1, v2])
```

```
[[1 2 3 4],
 [5, 6, 7 8],
 [1, 2, 3, 4],
 [5, 6, 7, 8]]
```

```
v4 = np.vstack([v1, v2, v3])
```

```
v = np.vstack([v1, v4, v2, v1])
```

```
[[1, 2, 3, 4],
 [5, 6, 7, 8],
 [5, 6, 7, 8],
 [1, 2, 3, 4]]
```

```
h1 = np.ones((2,4))
h2 = np.zeros((2,2))
```

```
[[1, 1, 1, 1],
 [1, 1, 1, 1]]
[[0, 0]
 [0, 0]]
```

```
h = np.hstack((h1, h2))
```

```
[[1, 1, 1, 1, 0, 0]
 [1, 1, 1, 1, 0, 0]]
```

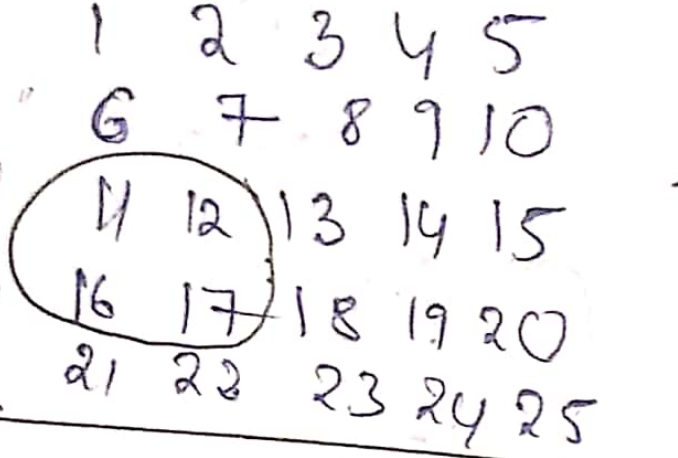
```
h = np.hstack((h2, h1))
```

```
[[0 0 1 1 1 1]
 [0 0 1 1 1 1]]
```

Miscellaneous

load data from file :-

que :-



a[2:4, 0:2]

[[11, 12]
[16, 17]]