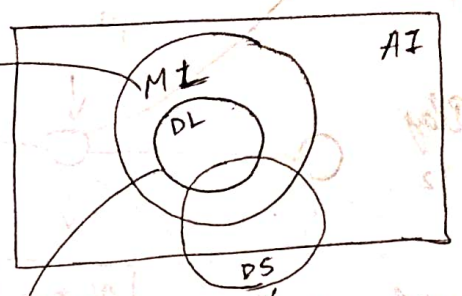
AI :- Application which can do its own task without human intervention.

1. Netflix app (provides automatic recommendation).
2. Self driving cars — Amazon app

AI which integrated s/w that is already existing to make the experience better.



Stats tool to analyse the data, visualize data, predictions, forecasting, clustering

Goal is to create AI application.

Researchers (1958)

Main aim mimic the human brain

=> Why Deep Learning is becoming popular?

① 2005 -> FB, Insta, WApp, LinkedIn, Twitter (We will be interacting with data)

DATA -> Exponentially

2008 -> {Big Data} -> Efficiently

2013 -> Company had huge amount of Data {AI -> popular}

⇓

Seamless Products

Panasonic :- AC's, TV's, Refrigerator {Data}

⇓

Model -> Reduce the electricity Bills ↓↓

⇓ sold in subscription basis => {Company generate revenue, Better Decisions}

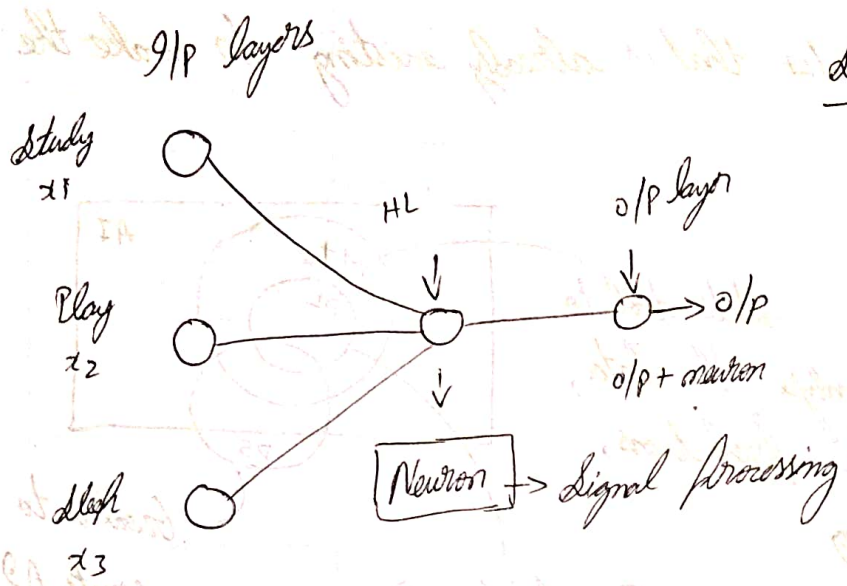
② Hardware Advancement (NVIDIA) → GPU's → Training the model

* Perceptron & Single layered Neural N/W's

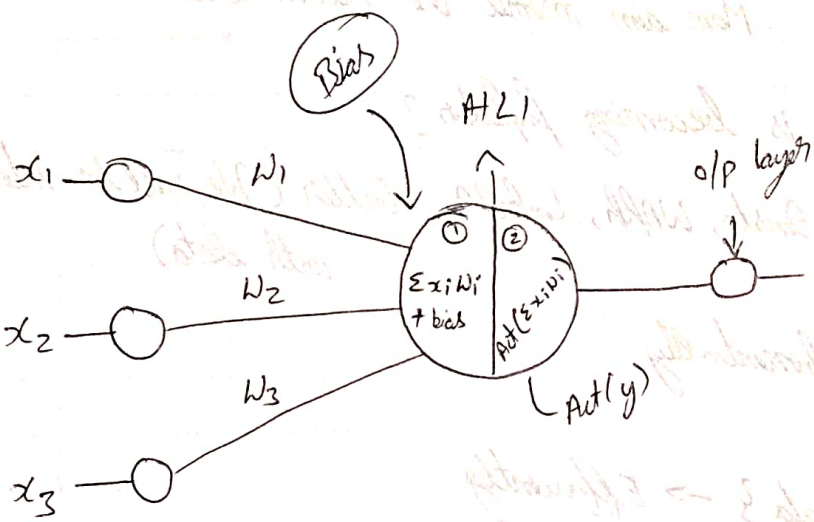
Binary classification

Dataset

Study	Play	Sleep	P/F
7	3	7	1
2	5	8	0
4	3	7	1



$y = mx + c$
↳ constant



$$\Sigma x_i W_i = x_1 W_1 + x_2 W_2 + x_3 W_3$$
$$= W^T x$$

⇒ Putting a hot obj in right hand, suddenly I'll move my right hand away. Bcoz the neurons are passing from here these are getting activated. Neurons passes the signals to the brain.

⇒ During the training of neural n/w, we will be taking care that which weight should have value so that neuron should activate on till that level.

When calculating the o/p of node, the i/p are multiplied by weights, & a bias value is added to the result.

The Bias allows the activation function to be shifted to left or right, to better fit the data.

⇒ Putting bot obj & moving the bond; this basically shows that wether the neurons should activate or deactivated.

≡⇒ We took i/p, added a bias & activate.

——————→ Forward Propagation.

→ 7 \
3 – × × × = o/p = 0 → $\hat{y}$ $y = 1$
7 – × ×

$(y - \hat{y}) = 1$ ⇒ Loss function (Diff b/w predicted & real value)
$= \downarrow$
0

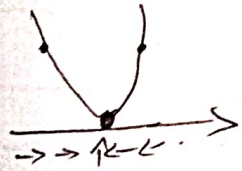
To Reduce the errorness

←——————— Back Propagation & update weights &

Gradient Descent ⇒ kind of optimizer

Optimizers are algo used to adjust the parameters (weights & bias) of neural n/w to minimize the loss function.

Ex:- Gradient descent (It is changing the coefficient).



$y - \hat{y}$ ⇒ Minimal

Conclusion

① I/P layers
② Weights
③ Bias
④ Activation

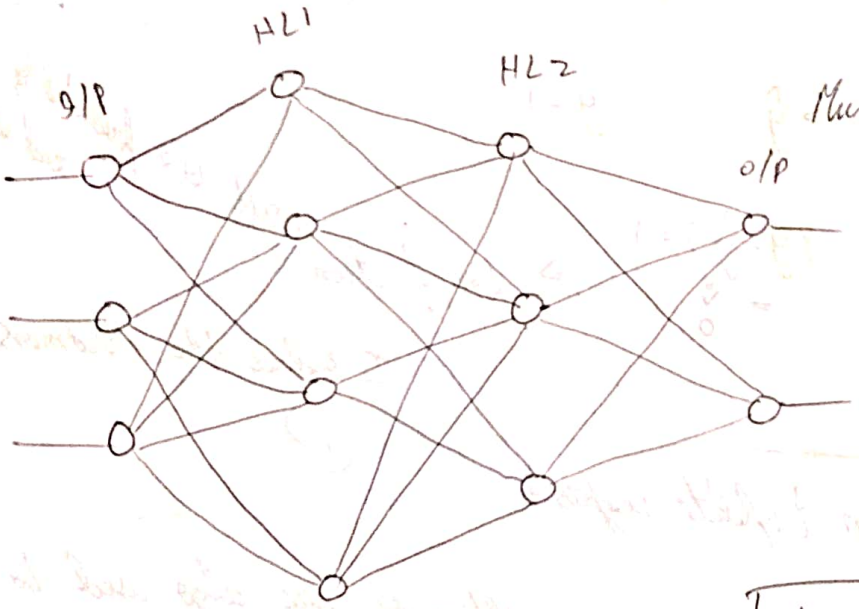
} Forward propagation

Every things works on

⑤ Loss function & y-ŷ ↓↓↓
⑥ Optimizers
⑦ Update the weight

} Backward propagation

① ANN
② CNN
③ RNN
④ Object Detection



Multi Layered Neural n/w

$$y = w^T x + b$$

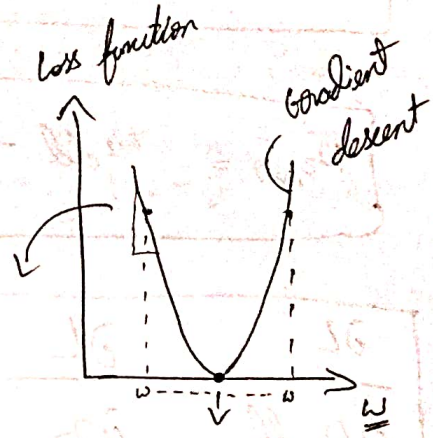
$$L \cdot (y - \hat{y}) \downarrow\downarrow$$

① Weight updation formula

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$

$\eta \rightarrow$ Learning rate

$\frac{\partial L}{\partial W_{old}} \rightarrow$ Slope



-ve slope

$$W_{new} = W_{old} - \eta(-ve)$$

$$W_{new} >> W_{old}$$

$\eta \approx$ small number

$$W_{new} = W_{old} - \eta(+ve)$$

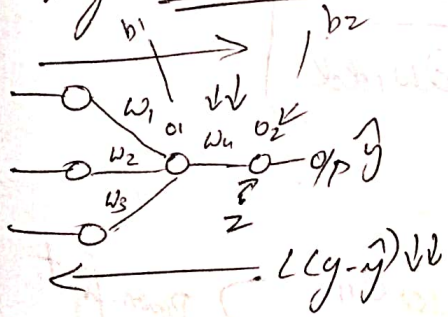
$$W_{new} << W_{old}$$

$\eta = 0.01$

If it may large num the point can jump here & there.

② Chain Rule of Differentiation

→ To updates weights during training, we need compute gradient of loss with respect to each parameter, across many layers; can be done using Chain Rule.



$$W_{4new} = W_{4old} - \eta \frac{\partial L}{\partial W_{4old}}$$ ← & Chain Rule of Differentiation &

$$\frac{\partial L}{\partial W_{4old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial W_{4old}}$$

$z = \sigma(O_1 W_4 + b_{2old})$

→ loss is dependent on O_2.

→ O_2 is dependent on O_1.

→ O_1 is dependent on W_1.

$$W_{1new} = W_{1old} - \eta \frac{\partial L}{\partial W_{1old}}$$

If wanted to update bias,

$$b_{2new} = b_{2old} - \frac{\partial L}{\partial b_{2old}}$$

$$\frac{\partial L}{\partial w_{1\,old}} = \frac{\partial L}{\partial o_2} * \frac{\partial o_2}{\partial o_1} * \frac{\partial o_1}{\partial w_{1\,old}}$$

⇒ to update w_1, this eqn become chain rule of differentiation.

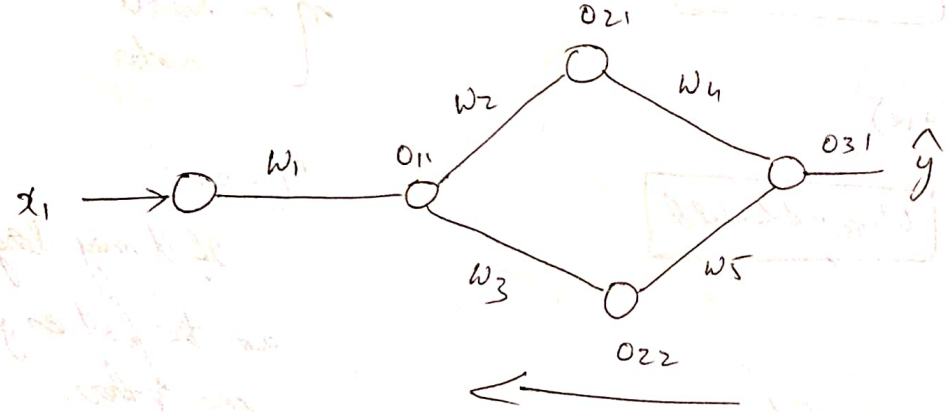
$$w_{2\,new} = w_{2\,old} - \eta \frac{\partial L}{\partial w_{2\,old}}$$

$$\frac{\partial o_2}{\partial w_4} * \frac{\partial w_4}{\partial o_4}$$

$$\Leftarrow$$

$$\frac{\partial L}{\partial w_{2\,old}} = \frac{\partial L}{\partial o_2} * \frac{\partial o_2}{\partial o_1} * \frac{\partial o_1}{\partial w_{2\,old}}$$

Optimizer



$$w_{1\,new} = w_{1\,old} - \eta \frac{\partial L}{\partial w_{1\,old}}$$

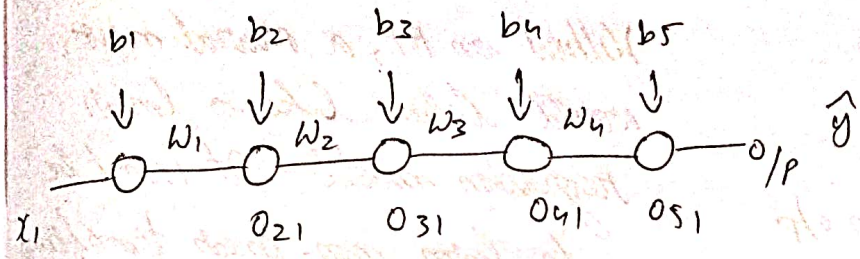
⇒ Chain Rule of Differentiation

$$\frac{\partial L}{\partial w_{1\,old}} = \left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{21}} * \frac{\partial o_{21}}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{1\,old}} \right] \text{upper part}$$

$$+$$

$$\left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{22}} * \frac{\partial o_{22}}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{1\,old}} \right] \text{Down part}$$

③ Vanishing Gradient Problem

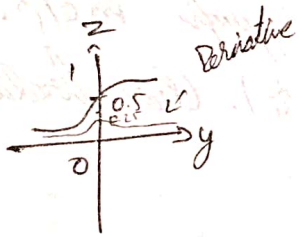


$$loss = \frac{1}{2}(y-\hat{y})^2$$ MSE

$$W_{1new} = W_{1old} - \eta \frac{\partial L}{\partial W_{1old}}$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial O_{51}} * \frac{\partial O_{51}}{\partial O_{41}} * \frac{\partial O_{41}}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial W_1}$$

$$= 0.25 * 0.15 * 0.10 * 0.05 * 0.02$$ ⇒ Assumption



Sigmoid AF

$$y = \frac{1}{1+e^{-x}} \rightarrow O_{51}$$

$$0 \le \sigma(y) \le 0.25$$

Derivative condition

$$O_{51} = \sigma\left[(O_{41} * W_4) + b\right]$$

↓ Sigmoid AF

$$W_{new} = W_{old} - \eta \text{ (small not)}$$

small

$W_{new} \approx W_{old}$ ⇒ Vanishing gradient problem

⇓

No change in weights

sol is to use another AF.

Activation Function, is a mathematical function that determines whether a neuron should be "activated" or not.

Without an AF, a Neural n/w would behave like a linear regression model - It won't be to learn non-linear transformations.

1. Sigmoid function

Used for binary classification o/p layer (eg probability b/w 0 & 1).

→ The function o/p is not centered on 0, which will reduce the efficiency of weight.

→ Sigmoid function performs exponential operations. Which is slower for computers.

Advantages

* Smooth gradient, prevents "jumps" in o/p.
* o/p values bound b/w 0 & 1. Clear predictions, i.e very close to 1 or 0.

Disadvantages

* Prone to gradient vanishing.
* Func o/p is not 0 centered.
* Power operations were really time consuming.

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

Used in o/p layer (binary classification)

2. tanh function (hyperbolic tangent function)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

* Value ranges -1 to 1.
* The whole func is 0-centric, which is better than Sigmoid.
* But still Vanishing Gradient.

=> For binary classification problems, the tanh func is used for hidden layer & Sigmoid function for o/p layer.

---

3. ReLU function (Rectified Linear Unit)

$$f(x) = \max(0, x)$$

Ranges $[0, \infty]$

* Positive values stay the same, -ve values become 0.
* But may cause dead neurons
-> When i/p is +ve, there is no gradient saturation problem.

Used for most hidden layer.

Disadvantages
* When i/p is -ve, ReLU is completely inactive.
One -ve no/- entered ReLU will die.

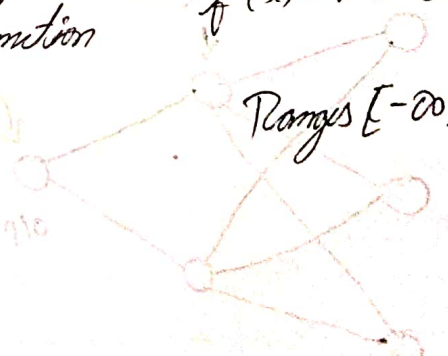
---

4. Leaky ReLU function

$$f(x) = \max(0.01x, x)$$

Ranges $[-\infty, \infty]$

* The i/p of -ve value, cannot 0 in this function
* It solves dead neuron problem, by allowing small gradient for -ve i/p.



5. ELU (Exponential Linear Units) function.

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$

* Proposed to solve ReLU problems.
→ No Dead ReLU issues.
→ The mean of o/p is close to zero. 0-centered.
* Computationally expensive, bcz we are using exponential value.

---

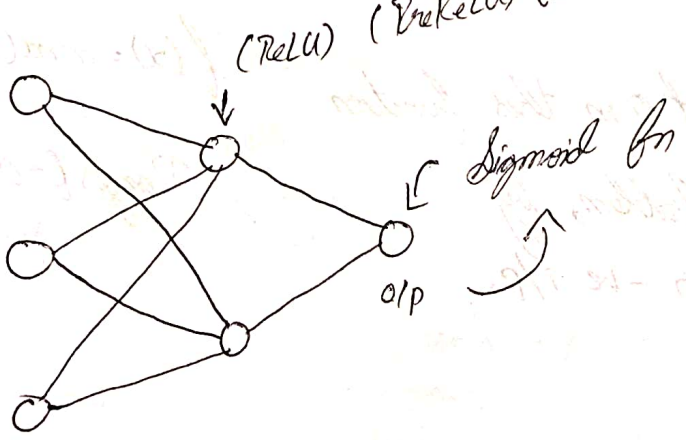
6. Softmax function

Used for o/p layer of multiclass classification

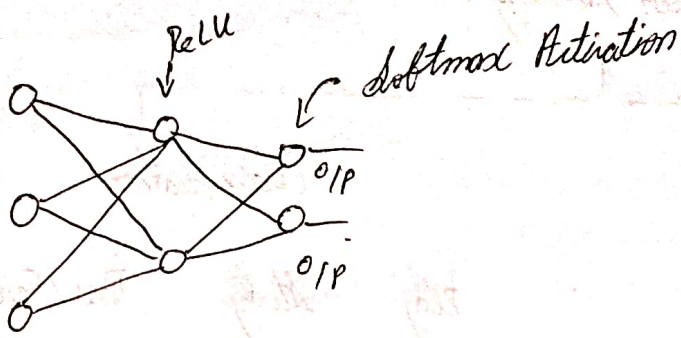
$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Technique which activation fn we should use;

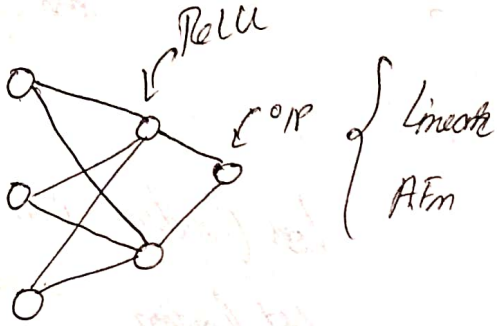
Binary Classification



Multiclass



Regression (continuous o/p)



Deep Learning (ANN)

Regression

Exp	Degree	Salary
10	PHD	—
—	—	—
—	—	—

Classification

Play	Study	Pass / Fail
10	2	Fail
4	3	Fail
5	5	Maybe
2	7	Pass

Regression

① MSE

② MAE

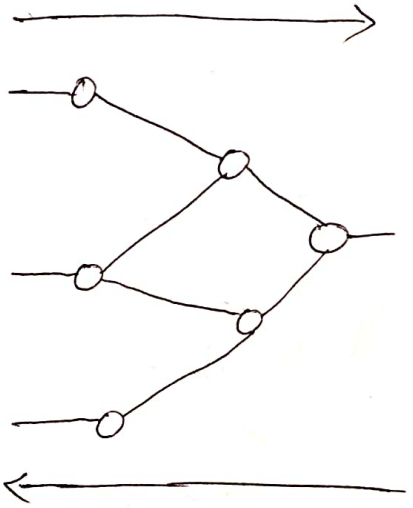
③ Huber loss

Loss function AND cost function

10 rec

⇓

Dataset = 100 records



$$loss = \frac{1}{2m}(y - \hat{y})^2$$

⇓

$$cost = \frac{1}{2}\sum_{i=1}^{m}(y - \hat{y})^2$$

* In loss fn, we will be providing one data point.

* In cost fn, I specifically provide batch of data points.

① Mean Squared Error (MSE) → is a loss fn used to measure avg squared diff b/w predicted value & actual value.

$$\text{loss fn} = \frac{1}{2n}(y-\hat{y})^2 \qquad \text{cost fn} = \frac{1}{2}\sum_{i=1}^{n}(y-\hat{y})^2$$

⇓

Quadratic eqn.

Advantages
* Differentiable
* It has only 1 local or global minima.
* It converges faster

Disadvantages
* Not robust to outlier.

---

② Mean Absolute Error (MAE) → is regression loss fn that calculates the avg of absolute diff b/w predicted & actual values.

$$\text{loss fn} = \frac{1}{2n}(y-\hat{y})$$

$$\text{cost fn} = \frac{1}{2}\sum_{i=1}^{n}(y-\hat{y})$$

* Robust to outlier.

---

③ Huber loss → combination of MSE & MAE.

$$\text{loss fn} = \begin{cases} \frac{1}{2n}(y-\hat{y})^2 & \text{if } |y-\hat{y}| \le \delta \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

(when outlier not present)

(δ ← Hyper parameter) value set before training that controls how ML model learns. It is not learned from data.

Classification

Cross Entropy
- Binary cross entropy (Binary classification)
- Categorical cross entropy (Multiclass classification)

① Binary Cross entropy

$$loss = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y}) \Rightarrow \text{logistic Regression}$$

$$\Downarrow$$

$$loss = \begin{cases} -\log(1-\hat{y}) & \text{if } y=0 \\ -\log(\hat{y}) & \text{if } y=1 \end{cases} \Big\} \text{ Binary classification}$$

$$\hat{y} = \frac{1}{1+e^{-x}}$$

---

② Categorical Cross entropy

one hot encoding

f1	f2	f3	o/p		Good	Bad	Neutral
2	3	4	Good	[	1	0	0]
5	6	7	Bad		0	1	0
8	9	10	Neutral		0	0	1

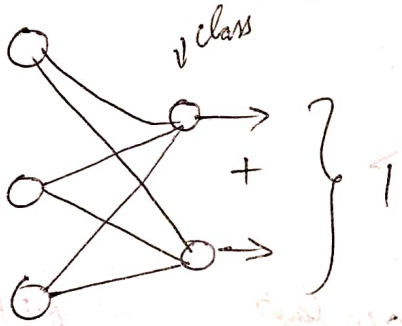
$$L(x_i, y_i) = -\sum_{j=1}^{C} y_{ij} * \ln(\hat{y}_{ij})$$

i = row
j = col

$$y_i = [y_{i1}, y_{i2}, y_{i3} \cdots y_{ic}]$$

$$y_{ij} = \begin{cases} 1 & \text{if the element is in class.} \\ 0 & \text{otherwise} \end{cases}$$

$\hat{y}_{ij}$ = Softmax Activation ← o/p layer



for multiclass classification

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Conclusion :-

ReLU, Softmax ⇒ Multiclass classification → categorical cross entropy (loss)

ReLU, Sigmoid ⇒ Binary → Binary cross entropy (loss)

ReLU, Linear Act^n → loss: MAE, MBE, Huber loss ⇒ Linear Regression

Optimizers

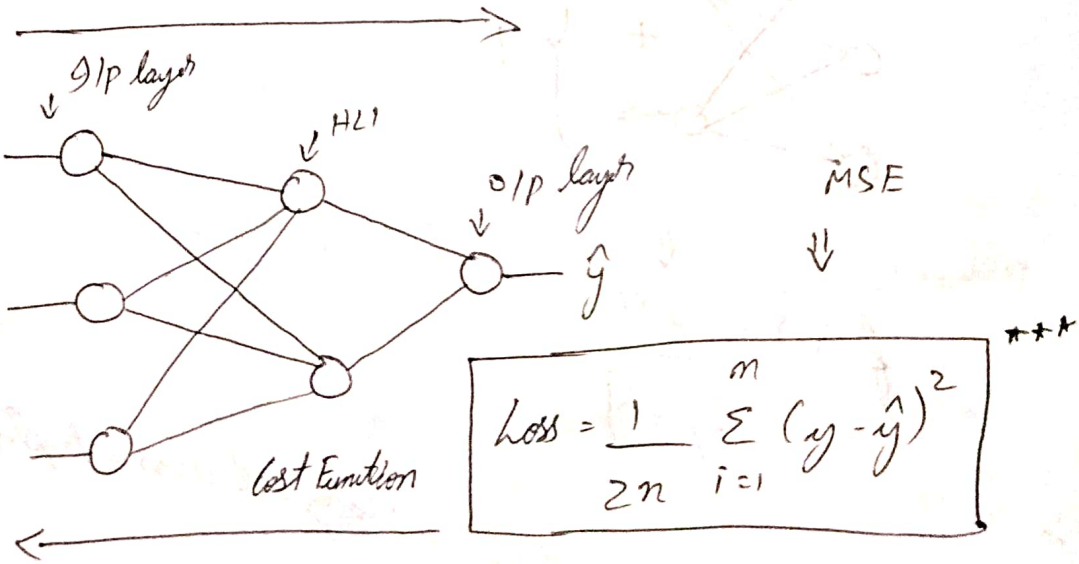
-> Gradient Descent.
-> SGD (Stochastic GD)
-> Mini Batch SGD
-> SGD with Momentum
-> Adagrad
-> RMS PROP
-> Adam optimizers

Batch, Epochs, Iterations

① Gradient Descent

Weight Updation formula

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$



$$Loss = \frac{1}{2n} \sum_{i=1}^{n} (y - \hat{y})^2 \quad ***$$

1000000 -> 1 Epoch

Epoch <-

{1000000}

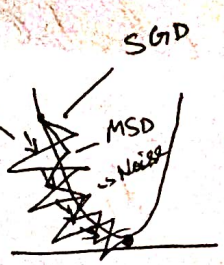
Disadv

-> Resource extensive & Huge RAM

② Stochastic GD

→ RAM ↓↓

1 rec → Epoch 1 → $\hat{y}$ ← loss → Update weight } → Iteration 1



Disadv

+ Convergence will be very slow
+ TC will be high

2 rec → ← } → Iteration 2

---

③ Mini batch SGD

1000000 batch size = 1000

→ Resource Intensive.
→ Convergence will be better.
→ TC will be improved.

E1 1000 → ← } → 91

E2 1000 → ← } → 92

⋮

} → 91000

---

④ SGD with Momentum

{ Exponential Weighted Avg }

⇓

Time Series

⇓

ARIMA, ARMA

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$

$$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$$

$$W_t = W_{t-1} - \eta \frac{\partial L}{\partial W_{t-1}}$$

Exponential Weighted Avg.

t_1 t_2 t_3 t_4 .... t_n

a_1 a_2 a_3 a_4 .... a_n

$\beta \Rightarrow$ Hyperparameter

$V_{t_1} = a_1$

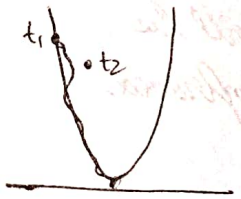
$\beta = 0$ to 1

$\Downarrow$

0.95

$V_{t_2} = \beta * V_{t_1} + (1-\beta) * a_2$

$= (0.95) * V_{t_1} + (0.05) * a_2$



$V_{t_3} = \beta * V_{t_2} + (1-\beta) * a_3$

Exponential Weighted Average

$W_t = W_{t-1} - \eta V_{dw}$

$$V_{dw_t} = \beta * V_{dw_{t-1}} + (1-\beta) * \frac{\partial L}{\partial W_{t-1}}$$

-> Noise will reduced.

-> Quick convergence.

⑤ Adagrad (Adaptive Gradient Descent)

η = fixed ⇒ adaptive ⇒ Learning rate ⇒ decreasing ⇒ Global minima

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

$$w_t = w_{t-1} - \eta' \frac{\partial L}{\partial w_{t-1}}$$

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_t}\right)^2$$

Huge not

t=1 t=2 t=3

η=0.01 η=0.05 η=0.002

⑥ Adadelta And RMSPROP

$$\eta' = \frac{\eta}{\sqrt{Sdw + \epsilon}}$$

$$Sdw = 0$$

$$Sdw_t = \beta Sdw_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

$\beta = 0.95$

$$Sdw_t = (0.95) Sdw_{t-1} + (0.05)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

⑦ Adam Optimizer (Best optimizer)

Momentum + RMS Prop (Adaptive Learning rate)

$V_{dw} = 0 \quad V_{db} = 0 \quad S_{dw} = 0 \quad S_{db} = 0$

$$w_t = w_{t-1} - \eta' V_{dw} \qquad \eta' = \frac{\eta}{\sqrt{S_{dw} + \epsilon}}$$

$$b_t = w_{bt-1} - \eta' V_{db}$$

$$V_{dw} = \beta * V_{dw_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)$$

$$V_{db_t} = \beta * V_{db_{t-1}} + (1-\beta)\frac{\partial L}{\partial w_{bt-1}}$$

→ Smoothening

→ Learning rate becomes adaptive

open-sourced by

Tensorflow => Google, Deepminds

Tensorflow is used for developing complex neural nlos.

< 2.0

Tensorflow — [Keras (Keras: Wrapper)

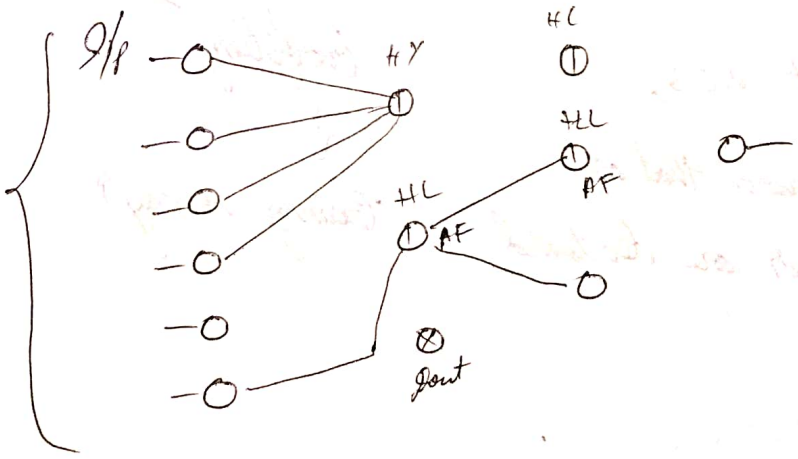
≥ 2.0

Tensorflow + Keras (Keras: Wrapper)

Keras is a high-level API built on top of TensorFlow that make it easier & faster to build deep learning models with simple code.

```
from tensorflow.keras.models import Sequential
     layers import Dense
     layers import ReLU, PReLU, LeakyReLU, ELU
     layers import Dropout
```

→ { Sequential...



←

Sequential is a class from Keras (part of TensorFlow) used to build feed forward neural n/w's where layers are added one after another, in sequence.

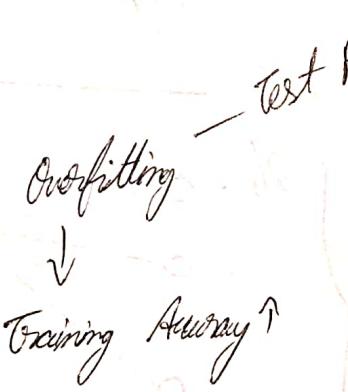
→ Where each layer has exactly one input tensor & one output tensor.
We can also perform forward & backward propagation.

Dense is used to create layers where each neuron is connected to every neuron in previous layer.
→ Using Dense we will be used to create i/p layer, hidden layer, o/p l.

AF, introduces non-linearity to the model, allowing it to learn complex patterns.

Dropout is a regularization technique used during training the model to prevent overfitting in NN.

If I specify dropout is 0.3,
30% of entire neuron that are present in those layer are deactivated while training.

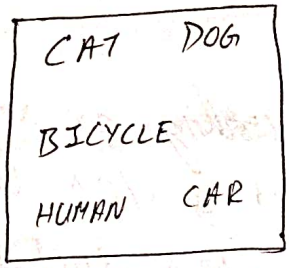


Early Stopping is a regularization technique used during training to stop the model early when the performance on the validation set starts to degrade, to prevent overfitting.

Black box model is a ML model whose internal workings are not easily interpretable by humans. We can see the i/p's & o/p's but not how model make its decision. Ex:- Neural n/w's, SVM etc

White box model is fully transparent & interpretable, where we can understand how the model make decisions. Ex:- Linear Regression.

CNN vs Human Brain → Always understand how does it work



Cerebral cortex

↓

Visual cortex → responsible for identify the items.

V_1 → Moving objs

↓

V_2 → Animal

↓

V_3 → Mach the one

↓

V_4 V_5 V_6

↓

V_7

o/p

CNN

Convolution

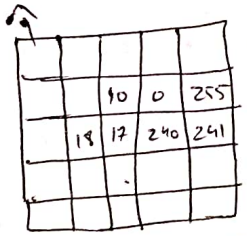
1 channel ← B & W

&

RGB

Images =

0-255



5×5

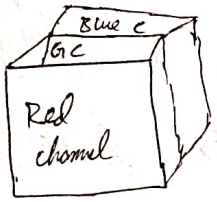
0 - Black

255 - white color

6×6 3×3 4×4 , what happening here is image size is decreasing. There losing some kind of info. So to prevent that will can apply padding to it.

Padding is about protecting the image by adding another layer on top of it. We can infill 0's or apply nearest value.

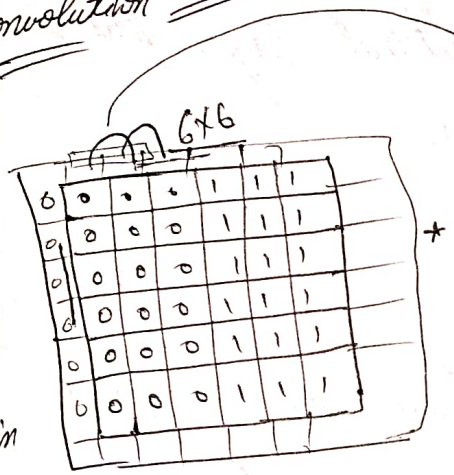
RGB → 3 channel



5×5×3

0-255 each channel, when combine these we will getting diff color.

① Convolution operation



3×3 filter / kernel

1	2	1
0	0	0
-1	-2	-1

Horizontal edge filter

⇒

4×4 o/p

0	0	0	0
-4	-4	-4	-4
-4	-4	-4	-4
0	0	0	0

① Min - Max scaling ***

1. 0+0+0+0+0+0 = 0
2. 0+0+1+0+0+0 -1 = 0
3. 0+0+0+2+1-2-1 = 0

Stride = 1, to move filter 1 step to right

n = 6, f = 3

6×6 → 3×3 → 4×4 o/p

B4 padding $n - f + 1 = 6 + 1 - 3 = 4$

After padding $8 - 3 + 1 = 6$

After padding = $n + 2p - f + 1$

B4 padding = $n - f + 1$

1	0	-1
2	0	-2
1	0	-1

n = 6, p = 1, f = 3

$n + 2p - f + 1$

$6 + 2(1) - 3 + 1 = 6$

6×6 o/p

⇒

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

Vertical edge filter

filters can take out the information from the specific image.

Importance of padding is that to prevent the loss of image we make use of info padding.

ANN is the basic form of NN, inspired by how human brain works. It is made of input, hidden & output layers & each neuron is connected with a weight. Ex:- house price prediction.

CNN (convolutional NN) is deep learning algo that can automatically learn spatial data & image. It uses convolutional layers that can detect features like edges, shapes, textures.

CNN Architecture

1. Convolutional layer :- extracts features from image & applies filter (kernel) that slide over image & detect patterns.

---

2. Apply AF, to learn complex patterns. converts -ve value to 0. ReLU.

---

3. Pooling layer is used to downsamples the features imp to reduce spatial size & computation. Keeps most imp feature.

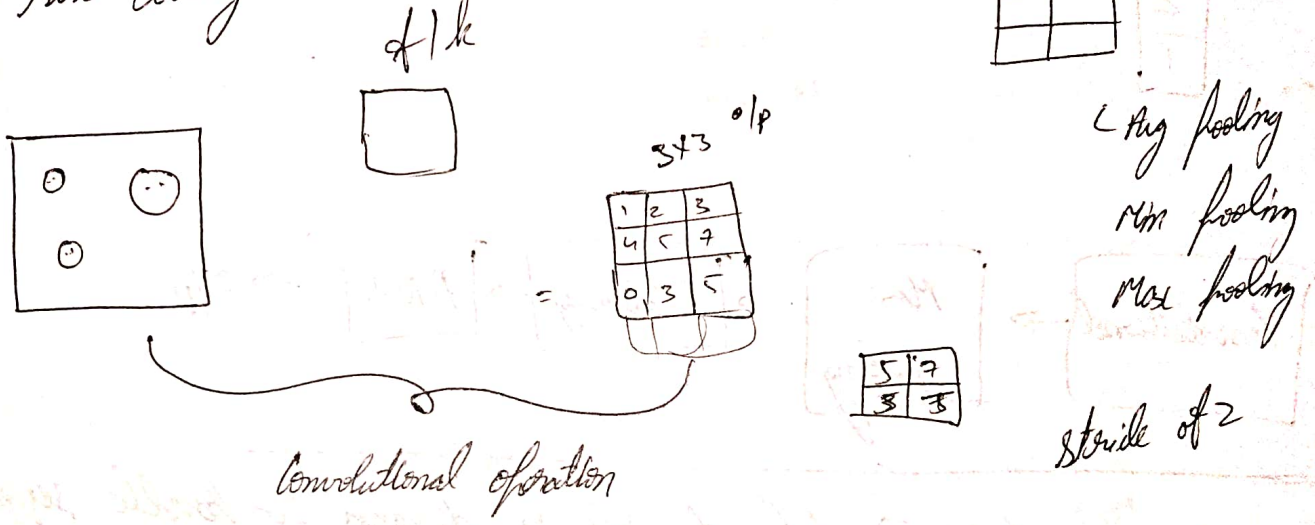
Reduces
dimensions
computational cost
overfitting

4. Flatten layer, converts the multi-dimensional tensor (2D or 3D) into 1D vector. After flattening the vector is then passed to fully connected layer.

5. Fully connected (Dense) layer :- performs classification based on extracted features.

---

(2) Max Pooling



location invariant refers to the ability to recognize features like edges, shapes, objects regardless of their position of image.
Basically means if I have multiple obj also if 1 of the filter can determine that object.

Flattening Layer - similar to ANN Dense layer

Max Pool

Max

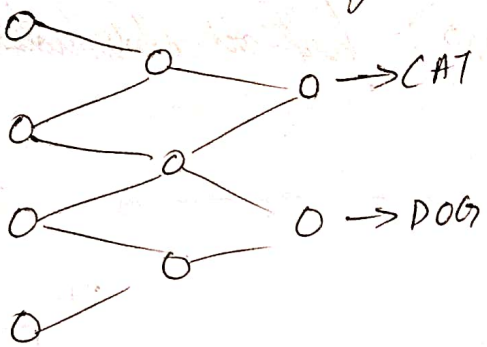
5	7
3	5

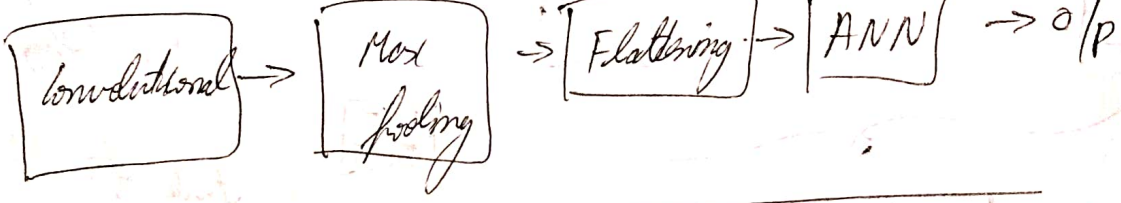
7	9
2	1

6	5
3	1

5
7
3
5
7
9
2
1

Image classification





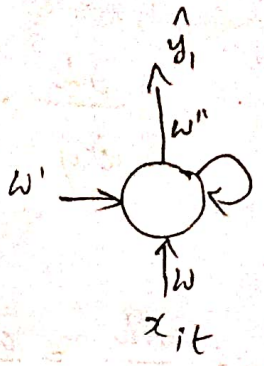
RNN (Recurrent NN) type of NN is designed to handle sequential data, such as:-

- → Time series data (eg stock prices)
- → Natural language (eg text or speech)
- → Video frames or sensor data over time.

RNN

① NLP → 9/p data

↳ BOW, TF-IDF, WORD2VEC



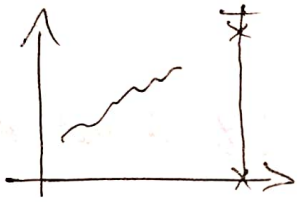
Vectors

	W1	W2	W3	....
S1	1	0	1	0
	0.3	0.75	.	....

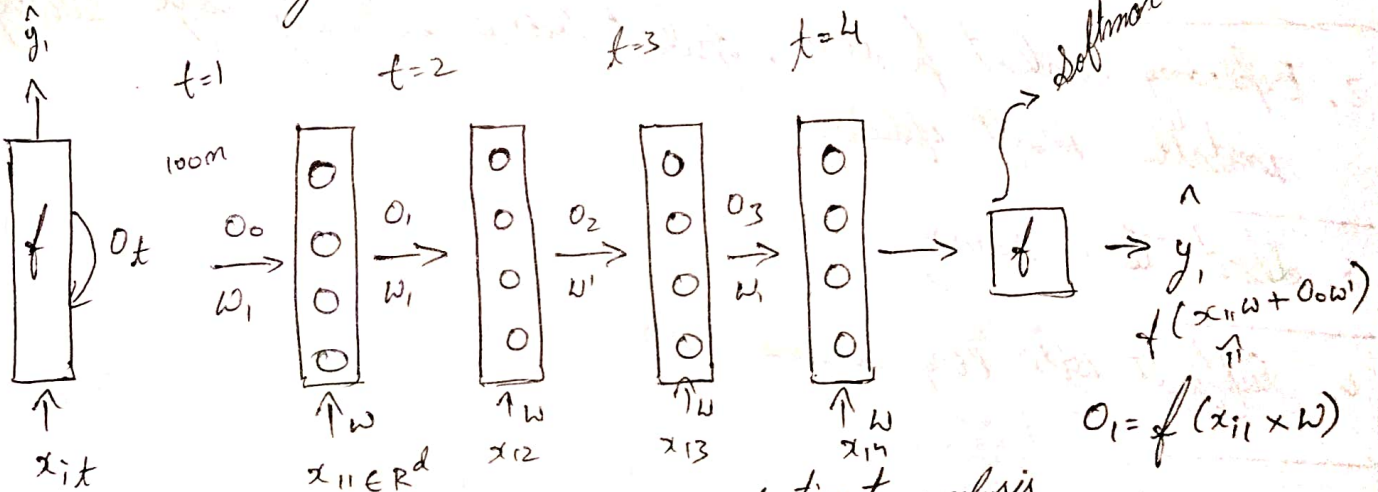
Time series

ARIMA

SARIMAX



Google Translation



NLP Usecase sentiment analysis

$x_1 = < x_{11}, x_{12}, x_{13}, x_{14} >$

$O_2 = f(x_{12}W + O_1W_1)$

$O_3 = f(x_{13}W + O_2W')$

$O_4 = f(x_{14}W + O_3W_1)$

→ Forward propagation

P.T.O

$$Loss = (\hat{y} - y)$$

$$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial L}{\partial w''} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w''}$$

$$w'' = w'' - \frac{\partial L}{\partial w''}$$

$$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial o_4} \cdot \frac{\partial o_4}{\partial w}$$

The RNN has many problem like

$$w = w - \frac{\partial L}{\partial w}$$

1. Vanishing gradient problem
   During backward propagation, gradients becomes very small (almost 0) as they are multiplied each step.

2. Exploding Gradient problem, gradient become extremely large, causing unstable weight updates.

3. Short-Term Memory.

4. Difficult with long sequences.

LSTM - RNN (It decides what to remember, what to forget & what to update at each step.)

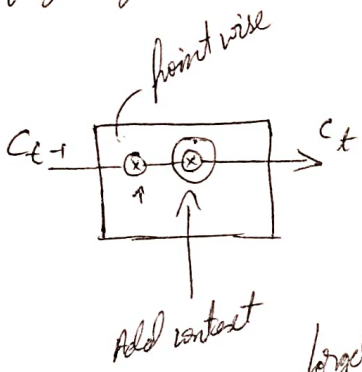
Long-short term memory, its a special kind of RNN having the capable to learn long-term dependencies.

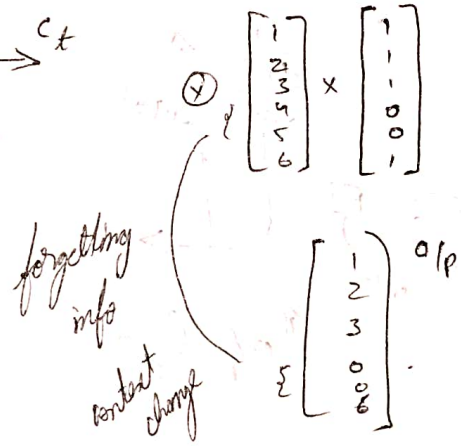
Introduced by Hochreiter & Schmidhuber (1997).

It resolves the vanishing gradient problem which occurs in RNN.

1. Memory Cell
2. Forget Gate
3. I/P Gate
4. O/P Gate

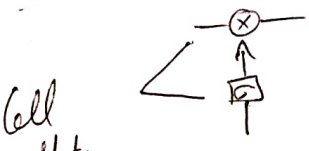
① Memory Cell, basically used for remembering & forgetting based on context of i/p.





---

② Forget Gate, decides what to forget from old memory.



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$y = mx + c$$

$$y = W^T x + b$$

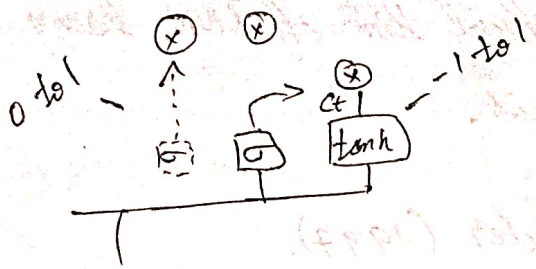
King Queen

[ ] [ ]

King Poor

[ ] [ ]

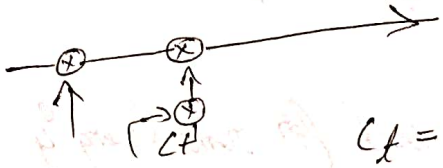
③ Input Gate, decides what new info to add.



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$C_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

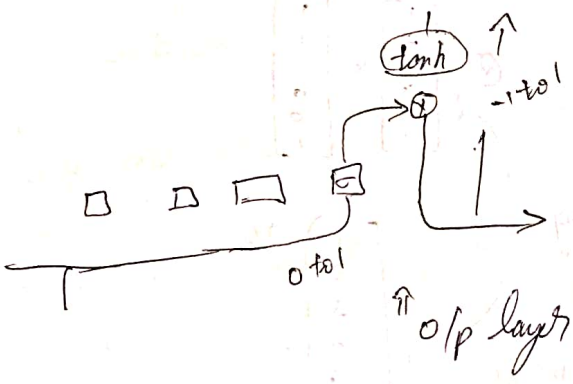
Adding information



$$C_t = f_t * C_{t-1} + i_t * C_t$$

---

④ Output Gate, decides what info to pass to the next step.



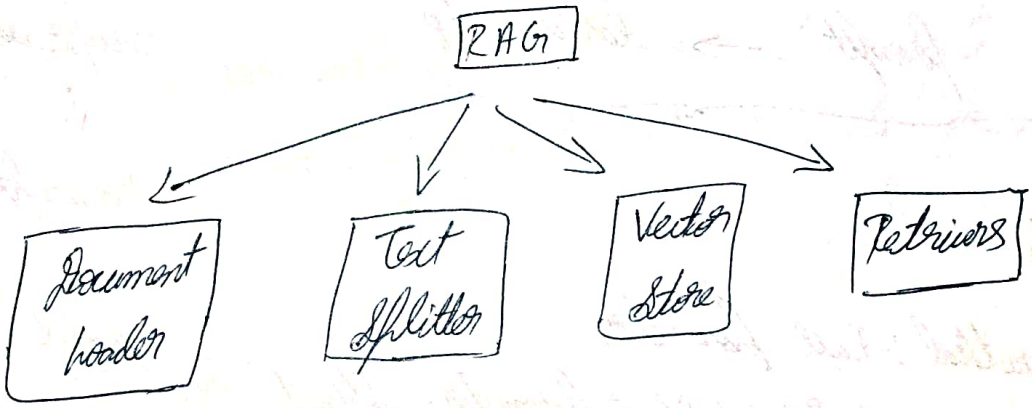
cell state is a long-term memory of LSTM. It keeps track of all 3 gates.

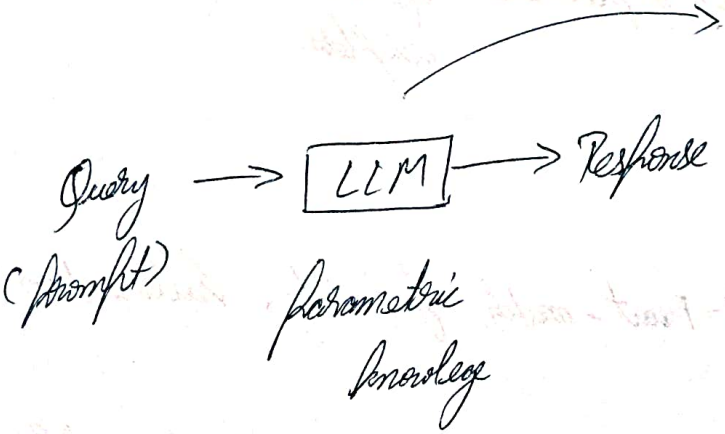
RAG (Retrieval - Augmented Generation) a modern technique of NLP that combines retrieval based models with generative models to provide more accurate & informative responses.

=> It retrieve relevant documents from knowledge source & generate answers based on both, retrived content & i/p query.

Tools commonly used in RAG systems :-

+ Vector DB's :- FAISS, Pinecone, Weaviate
+ LLM's :- OpenAI's GPT, Hugging face Transformer.
+ Framework :- Langchain, LlamaIndex.





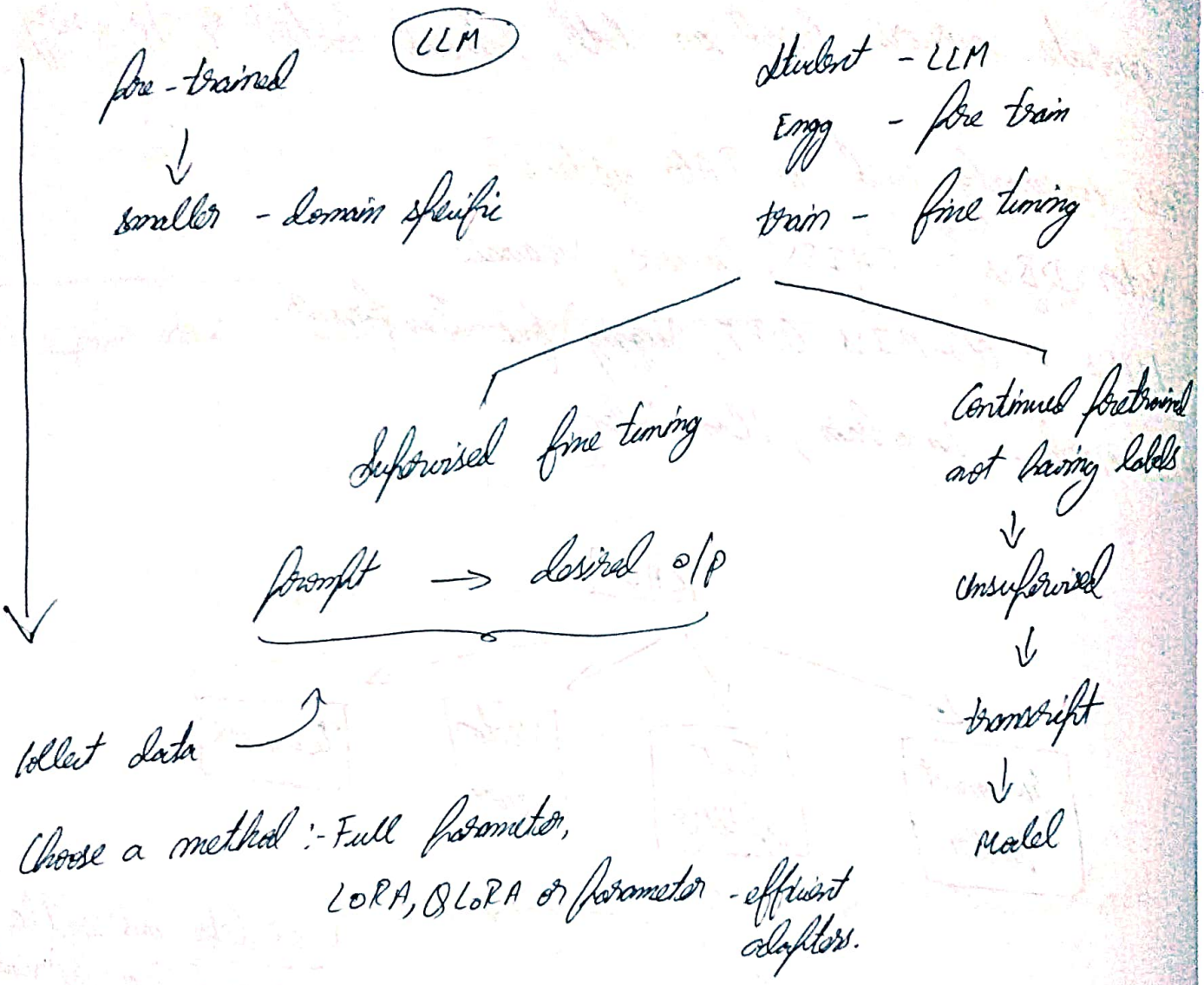
1. Pvt data ans can't be accessed. Bcz pretrained it not seen that data.

2. Earlier or recent data we can't get response.

3. Hallucination, it factually gives incorrect answer using parameter by confident.

We can solve that 3 LLM problem atleast using fine-tuning.

Fine-tuning is the process in ML, where a pretrained model is further trained on specific dataset to adapt new domain.



Hallucination solved giving examples.

↳ I don't know

Problems in FT

1. LLM -> train (computationally expensive).
2. Strong technical expertise.
3. High data rate; updating data again & again.

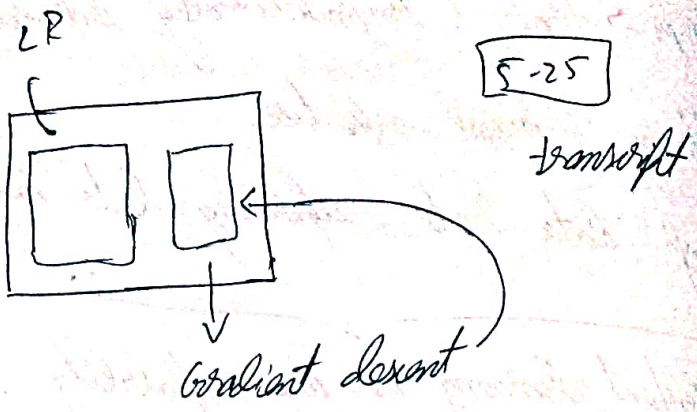
In context learning is one capability of LLM like GPT 3/4, where models learn to solve a task purely by seeing examples in the prompt - without updating its weights.

→ few shot prompting.

In context-learning is a LLMs emergent property
(is ability that suddenly appears in a system when it reaches a certain scale. Even though it was not explicitly programmed or expected from individual components.

GPT-3 - 175B parameters.

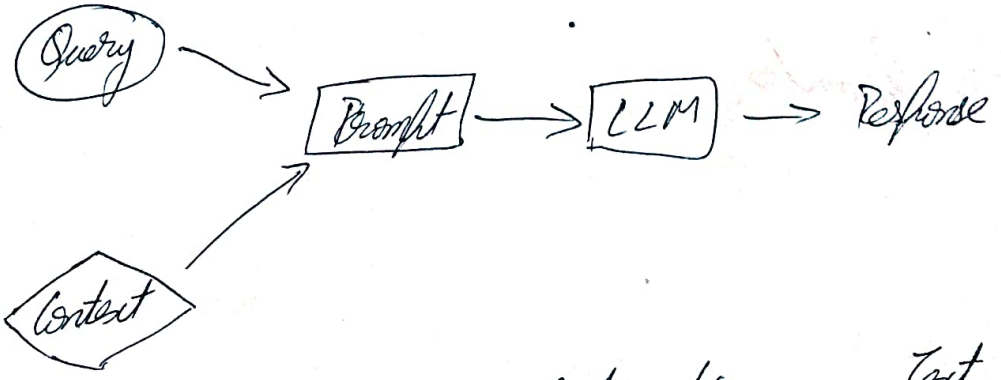
context



→ Instead of just ex tasks, retrieve background info, facts, documents, product manuals etc.

→ Inject that into prompt to augmented the model's knowledge.

RAG is a way to make language model smarter by giving it extra information at the time we ask the question.



RAG → Information retrieval + Text generation

* Indexing
* Retrieval
* Augmentation
* Generation

Indexing is a preprocessing step, large datasets are broken down into smaller chunks, Each chunk is converted into a vector embedding using a model (like BERT). These embedding are stored in vector database.

Retrieval at query time, the user qry also embedded into a vector. & The system retrieves the most similar chunks from indexed database using similarity search (like cosine similarity).

Augmentation, the retrived documents are combined with user qry. This augmented i/p is then sent to language model. The model now has contextual information from the knowledge base to answer more accurately.

Generation, This is where actual o/p is created. Generates informed response.

---

* Indexing takes 4 steps :-
  - -> Document Ingestion :- load our knowledge into memory.
  - -> Text Chunking :- Break large doc into small, semantically meaningful chunks.
  - -> Embedding generation :- converts each chunk into dense vector that capture its meaning.
  - -> Storage in Vector store :- store the vectors along with original chunk text + metadata in a vector database.