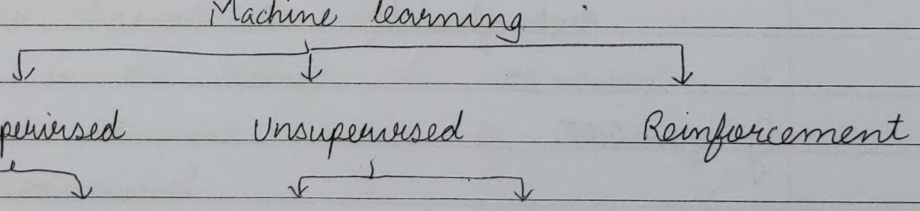
Deep learning

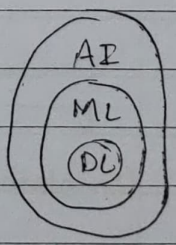
Neural Network

A NN is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experimental knowledge and making it available for use.

It resembles the brain in two respects:
- Knowledge is acquired by the network from its environment through a learning process
- Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.



Ensemble methods
Deep learning



Deep learning

Core idea: Instead of hand-crafting complex features, use increased computing

Works best with lots of homogeneous, spatially related features
→ Typically requires massive amounts of data and training resources

Machine learning vs. Deep learning

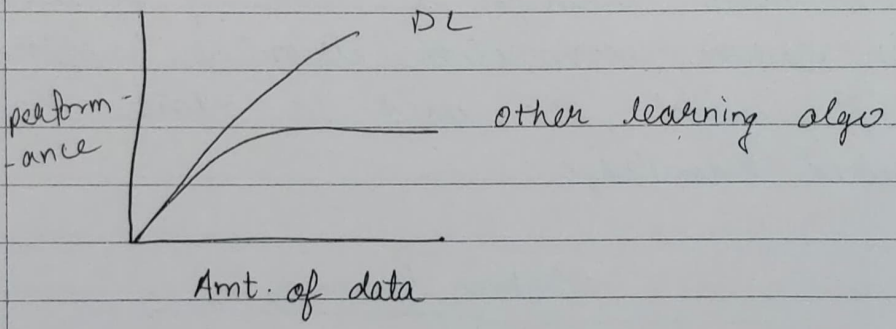
ML (programmer) (machine)

Input → feature extraction → classification → output

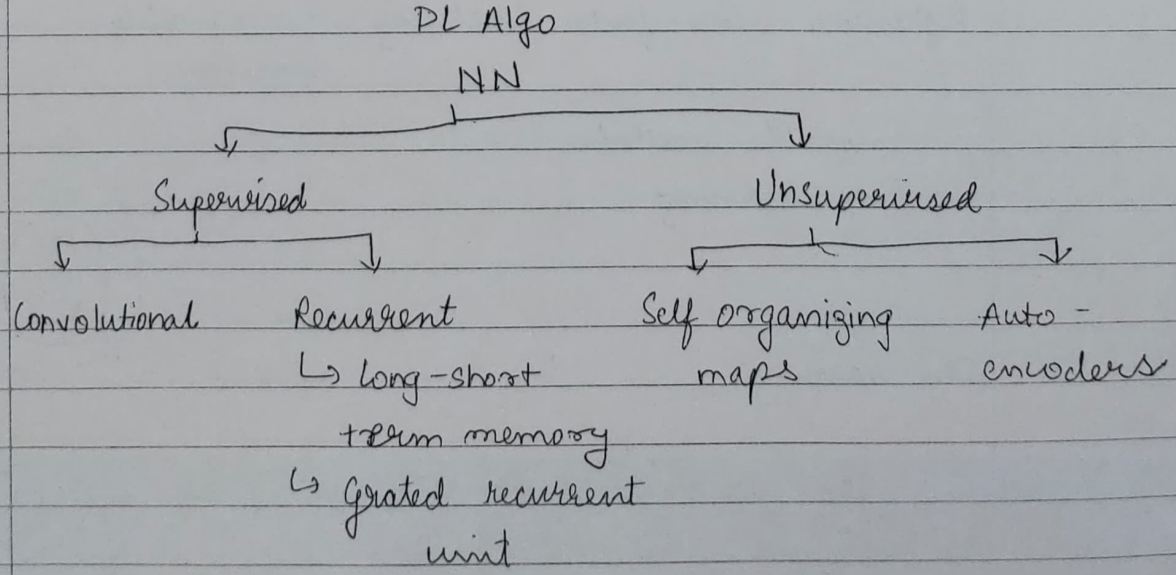
DL (machine)

Input → Feature extraction + Classification → Output

Why DL?



Eg: ChatGPT



NM

Learning

Any relatively permanent change in behaviour that occurs as a result of experience and practice

→ Learning occurs more rapidly when information is recieved frome one or more senses.

Sight → 75%
Hearing → 13%
Touch → 6%
Smell → 3%
Taste → 3%

Dendrites → Inputs
Synapse → weights
Axon → Output
Soma → Interconnections

Nature and characteristics of learning

→ Learning is the change in behaviour
→ Change in behaviour caused by learning is relatively permanent.
→ Learning is transferable from one situation to another
→ Learning is universal process
→ Learning is purposive and goal directive

→ Types of learning

↳ Motor learning
↳ Verbal learning
↳ Concept learning
↳ Discrimination learning
↳ Learning of principles.
↳ learning through association - Classical conditioning
↳ Learning through consequences - Operant conditioning.

↳ learning through observation - Modeling / Observational learning

Deep learning

Deep learning is an approach to learning where we can make a machine imitate the n/w of neurons in a human brain.

AI → ML → NN → DL
(Deep NN)
↓
3 or more hidden Layers.

ML	NN
- Feature based	- Example based
- Conscious learning	- Sub-conscious learning
- Eg: Fruit → Shape, color, texture	- Eg: Different pictures of fruits.
- Algo to parse data, learn from that data, and make informed decisions based on what it has learned	- Structures algorithms in layers to create an ANN that can learn and make intelligent decisions on its own.

Application (ANN)

Consider a real estate problem, we need to predict whether a house (flat) will get buyer or not. Need to make ANN model for the same.

[Missed Notes]

Activation function

An activation / threshold function decides whether a neuron should be activated or not.

It will decide whether the neuron's input to n/w is important to you.

Activation func^n is a mathematical 'gate'.

Necessity
- If not used, every neuron will only be performing a linear transformation on the inputs using the weights and bias.
- Hence, our model would just be a linear regression model.
- Used to enable limited output hence called squashing func^n.

~~① Step function~~

② Binary step func^n

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

- Cannot provide multi-valued output.

③ Linear activation func^n

$f(x) = ax + c$ ideal where interpretability is req.

Isn't much benficial bcoz NN would not improve error due to the same value of gradient.

③ Non-linear activation func^{n}

- They allow backprop
- ~~weights~~

ⓐ Sigmoid / Logistic activation function.

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

Any real No. as inp
Output in range 0 to 1.

Problem
- Sigmoid saturate and kills the gradient.
- Has vanishing gradient problem.
- Output not symmetric to zero. Hence training is difficult.

Gradients are ~~using~~ used during training to update the n/w weights.

ⓑ Tanh Activation function

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

Same as sigmoid but output is b/w -1 and 1.

- Preffered more than sigmoid its symmetric to zero.
- Helps centre the data.
- Usually used in hidden layer
- Has problem of vanishing gradient.

ⓒ ReLU Activation function
↳ Rectified linear unit.

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

Domain : $(-\infty, \infty)$
Range : $[0, \infty)$

Referred as piece-wise linear or hinge func^n

The func^n and its derivative both are monotonic
" " " " " allows for backprop
Also computationaly efficient

- Problem: Dying ReLU – Graphs become zero when negative. output graphs may not map to -ve value appropriately.

- Cause dead neuron : Can cause weight update which will make it never active

ⓓ Leaky ReLU Activation func^n

$$f_x = \begin{cases} 0.01x & x < 0 \\ x & x \geq 0 \end{cases}$$

Range $(-\infty, \infty)$

- Enables backprop for -ve value
- No dead neuron

Limitation:
- Prediction may not be accurate for -ve value
-

(e) Parametric ReLU

$$f(x) = \begin{cases} ax & x < 0 \\ x & x \geq 0 \end{cases}$$

we learn parameter a.

Limitation:
- Perform differently for different a.

(f) ELU

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

Avoid dead ReLU problem by introducing log curve for -ve value

Limitation:
- High computation Req.
- Exploding gradient problem.
  A problem where large error gradients accumulate and result

⑨ Softmax

$$S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$ Generally used in last layer.

Generates probability vector to indicate the probability of each of the classes.

The func^n, for every data point of all the individual class, returns the probability.

⑩ Swish

- Matches or outperforms ReLU

$$f(x) = x * \text{Sigmoid}(x) \qquad \sigma(x) = \frac{x}{1+e^{-x}}$$

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

Large -ve values are zeroed out for reasons of sparsity making it a win-win situation.

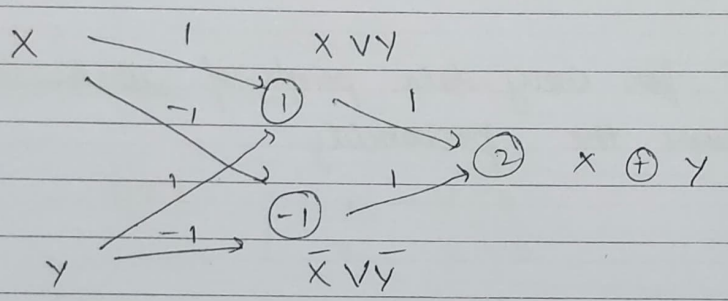
How to use?

→ Relu → Hidden layer
→ Sigmoid and tanh → Not used in hidden layers
→ Swish → NN depth > 40 layers
→ Regression → Linear activation func^n
→ Binary → Sigmoid
→ Multiclass → Softmax

Non - linearly seperable.

XOR problem can be solved by MLP (Multi layer perceptron).

Deep structure - depth → longest path from source to sink.



Convolutional Neural N/W (CNN)

CNN are a special type of FFNN (ANN) in which conectivity b/w its neuron is is inspired by visual cortex.

Consider learning an image:
Some patterns are much smaller than the whole image.

How can we represent smaller feature with few parameter.

Training a lot of such small detector and each detector must move around.
(different location but parameter are same)

A convolutional layer has a number of filters that does convolutional operation.

Aim is to reduce the images into a form that is easier to process, without losing features.

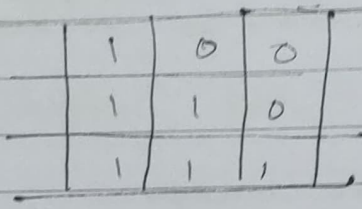
A CNN is a NN with some convolutional layers (and some other layers).

CNN generally has 3 layers

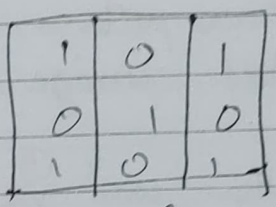
1. Input
2. Convolution → Apply filter
3. ReLU → Activation func^n
4. Pooling

Convolutional layer

Convolution is a mathematical op^n on two func^{n}s that produce a third func^n that expresses how the shape of one is modified by the other.



Part of image



filter / kernel.

$1 \times 1 = 1$
$0 \times 0 = 0$
⋮

} Multiplication of all corresponding places / cells.

4 } Add the results of multiplication.

This value is written in convolved matrix in place of 9 cells.

For 3 channel, we need 3 kernel and convolution matrix (the final one) would be one matrix with each corresponding cell would have addition of the channel output plus bias.

Pooling layer.

Pooling layer is responsible for reducing spatial size of convolved feature.

decrease power required to process data

Extract dominant feature

Dimensionality reduc^{n}

Two types
a) Max pooling
b) Average pooling.

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

20	30
112	37

13	8
79	20

Max pooling → Noise suppresant and dimensionality red^{n}

Avg. pooling → Dimensionality red^{n}

Max pooling is better than average pooling.

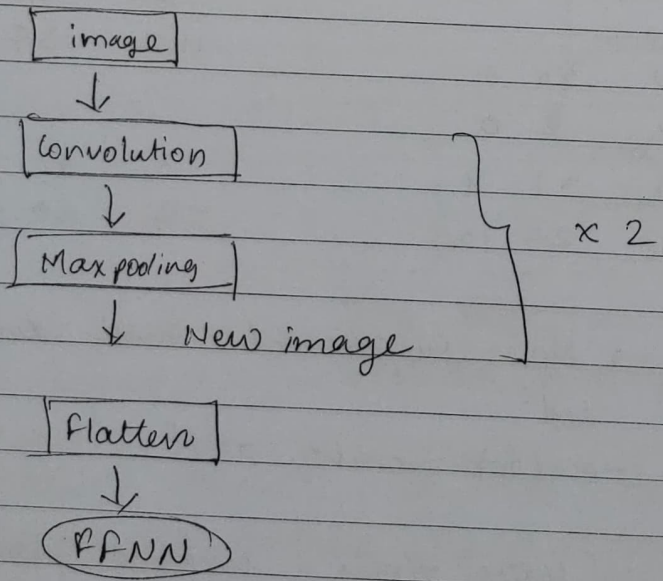
~~Flattening~~ Flattening

Flattening is used to convert all resultant 2-D array from pool feature maps into a single long array

- Pixel from image are fed to the Convolutional layer that perform convolution op^{n}
- Convolved map obtain
- ReLU applied
- Multiple convolution
- Pooling layer
- Flatten and fed to fully connected layer

CNN compresses the NN.

- Reduce no. of conn.
- Share weights on edges
- Max pooling further reduces the complexity



Advantage

- Good at detecting patterns and features in images, videos and audio signal
- Robust to translation, rotation & scaling invariance
- End to End training no need for manual feature extraction.
- Handle large data & have high accuracy.

Disadvantage

- Computationally expensive to train & require lot of memory.
- Can be prone to overfiting if not enough data or proper regulization is used
- Require large amount of labelled data.
- Interpretability is limited, it's hard to understand what the n/w has learned.

Parameter vs Hyperparameter

Parameter can be changed during execution
Hyperparameter cannot
↳ Set before execution

Gradient Descent

Key concept in Backprop.

→ commonly used iterative optimization func^n to train ML & DL Model

descent

$$W = W - \alpha \frac{\partial L}{\partial W}$$ ← gradient

↑ Learning rate

Loss func^n are used to calculate the error b/w the known predicted

Convex problems have only one minimum

Learning rate → Step size to take to reach minimum point

If LR is high then the GD might overshoot the minimum.

$x_1 \cdots x_n$ input → n dimension

L-1 hidden layer

1 output layer → k neuron

Each neuron (hidden & output) is split in two:
Summation and activation

=> Superscript → layer in N/W
Subscript → Sequence No. of the specific neuron in n/w.

for first neuron in first hidden layer.

↳ loss func^n helps to measure

Classification loss
↳ Cross - entropy loss (keep minimum)
Tries as predicted prob. diverges

-ve log likelihood or log loss func^n

$-(y_i$

$$= -\sum_{i=1}^{M} y_{i,j} \log \hat{y}_{i,j}$$

↑ actual ↑ predicted

Regression cost func^n

→ Deals with continuous value

Error = $y - \hat{y}$ ⟷ -ve error can occur

MSE / L2 / Quadratic loss

$$\text{MSE / L2 loss} = \frac{1}{n} \sum_{i=1}^{\hat{}} (y_i - \hat{y}_i)^2$$

MSE is less robust to outlier

MAE loss

$$MAE = \frac{1}{n} |y_i - \hat{y}_i|$$

AKA, L1 loss Not affected by noise or outlier

Weight initialization

deeplearning.ai/ai-notes/initialization/index.html.

Zero

Initialize all weights with zero leads the neurons to learn the same features during training.

All neurons would be symmetric and will receive same update.

We want each neuron to learn certain feature.

Same thing happen with a constant weight (& 0 bias)

Too-large

→ Leads to exploding gradients
↳ very big weight update which lead to large divergence of GD.
Leads the cost to oscillate around minimum

Too-small

→ Can have vanishing gradient problem.
→ Slow and unstable

Uniform initialization

Sigmoid distribution

$W =$

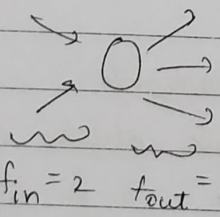
Xavier or Glorot initialization

Keeps variance same across every layer.
Normally distributed around zero.

Normal

$$W \approx N(\mu, \sigma)$$

$$\mu = 0 \qquad \sigma = \sqrt{\frac{2}{f_{in} + f_{out}}}$$



$f_{in} = 2 \quad f_{out} = 3$

Uniform

$$W \approx U(a, b)$$

$$a = -\sqrt{\frac{6}{f_{in} + f_{out}}} \qquad b = \sqrt{\frac{6}{f_{in} + f_{out}}}$$

He / Kaiming

→ Takes into account non-linearity

$$W = N(\mu, \sigma)$$

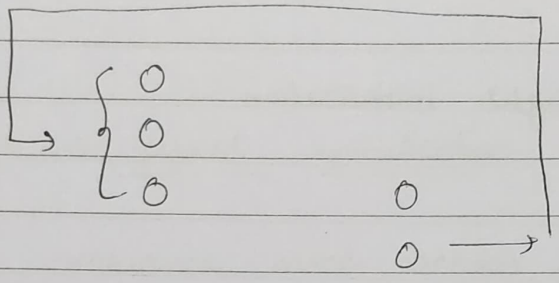
$$\mu = 0 \qquad \sigma = \sqrt{\frac{2}{f_{in}}}$$

Benefit of weight initialization

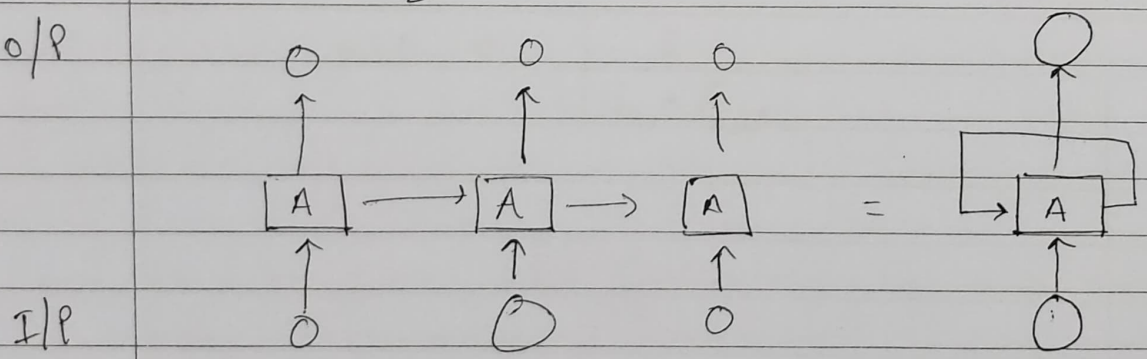
→

Recurrent NN (RNN)

No relation b/w 1st & 2nd output (cat & dog). So 't' is autonomous output to 't-1'.
There are scenario where current output depends on previous state.



New data → Predicted output



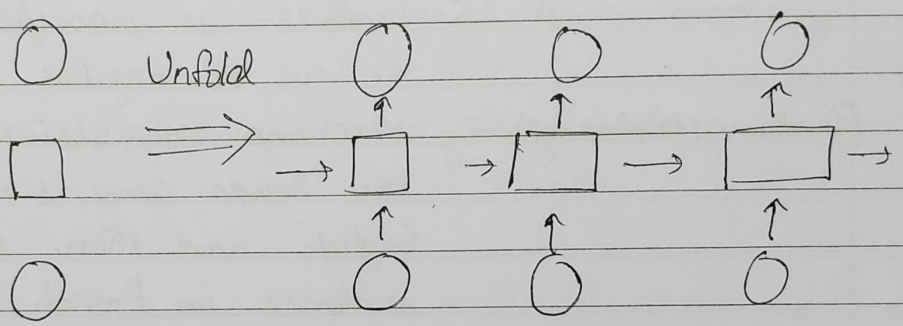
Recurrent NN (RNN)

RNN is a type of NN where output from the previous step are fed as input to the current step

The main & most imp. feature of RNN is Hidden State.
RNN also called STM (Short term memory)

Why?
Bcoz FFNN cannot :-
→ Handle sequential data
→ Memorize previous inputs
→ Considers only the current input

RNN is a special type of ANN adapted to work on for time series data that involves sequences.



Training through RNN.
Calculating current state

$$h_t = f(h_{t-1}, X_t)$$

Apply tanh activation func^n

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} X_t)$$

Calculating output

$$Y_t = W_{hy} h_t$$

Types of RNN

Four Types of RNN

① One to One (Vanila NN) Eg: Image Classification
② One to Many Eg. Image captioning (take image as I/P and output sequence of word)
③ Many to One Eg. Sentiment analysis where any sentence is classified as expressing the +ve or negative sentiment.
④ Many to Many Eg. Machine translation, where the RNN reads any sentence in English and then outputs the sentence in French.

↳ Bidirectional many to many
Eg. Video classification where we wish to label every frame of the video

Two issues of standard RNN.

① Exploding gradient problem
② Vanishing gradient problem

→ long training time, poor performance, and bad accuracy are the major issues in gradient problems.

$W = 2 \rightarrow 2^1 \quad 2^2 \quad 2^3 \quad 2^4$

$W = 0.5 \rightarrow 0.5^1 \quad 0.5^2 \quad 0.5^3 \quad 0.5^4$

W ~ Small → vanishing

W ~ large → Exploding

Solution

Exploding gradient → Truncated BTT (Backprop time)
→ Clip gradients at the threshold

Vanishing gradient → ReLU activation function
→ LSTM, GRU.
↓
Long short term memory.

Application, Advantage and Disadvantage