Joseph John Gerlach - proposed - nervous system is single continous network - (biology) - Reticular theory

Camillo Golgi - staining technique.

Neuron - coined by - Heir Rich Wilhelm Gottfried Von Waldeyer - Hartz.

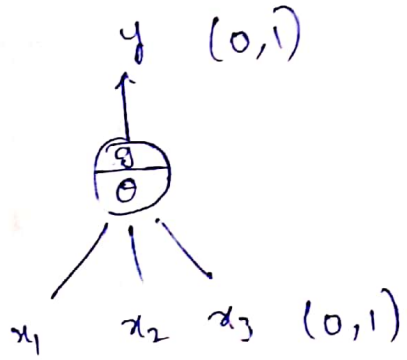
Perceptron - Frank Rosenblatt - idea initially pursued by Navy to translate languages

Minsky - Papert - Perceptrons paper - limitations highlighted

gradient descent - Cauchy.

Unsupervised pre training - 2006 - deep learning revived boomed again.

McCulloch Pitts Neuron.



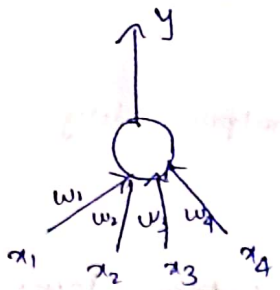
$g(x) = x_1 + x_2 + x_3$

$y = 1$ if $g(x) > \theta$

$= 0$ if $g(x) < \theta$

Single MPN can represent a boolean fn which is linearly separable.

Perceptron Model



$y = 1$ if $g(x) > \theta$

$g(x) = \sum w_i * x_i$

or

$y = 1$ if $\sum_{0}^{n} w_i * x_i \geq 0$

where $w_0 = -\theta$, $x_0 = 1$

w_0 is the bias.

Perceptron learning Algorithm.

P → input with label 1
N → input with label 0
w → weights

till convergence (no errors)

pick random $x \in P \cup N$

if $x \in P$ & $\sum x_i w_i \leq 0$

$x = x + w$

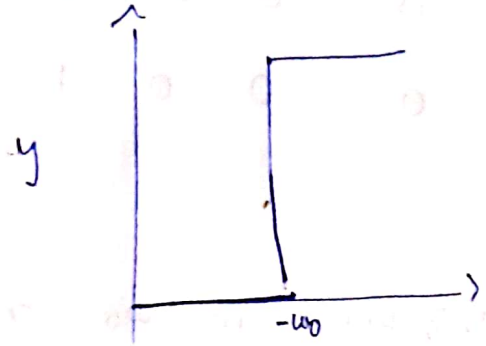
if

$x \in N$ & $\sum x_i w_i \geq 0$

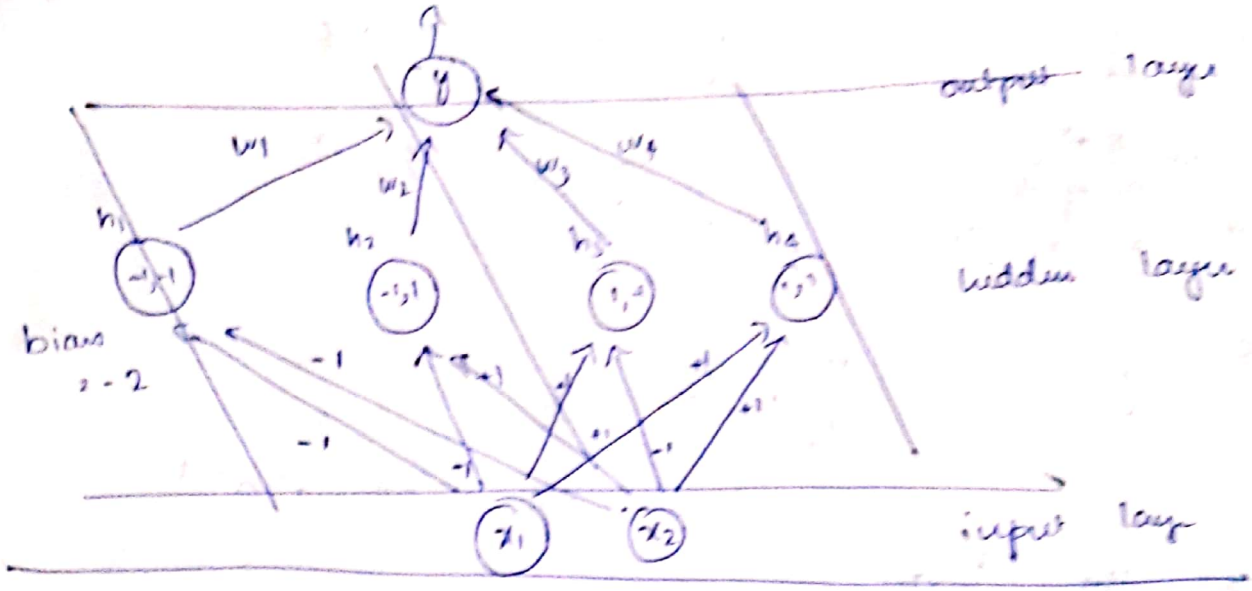
$x = x - w$

Network of perceptron can deal with non-linearly separable data.

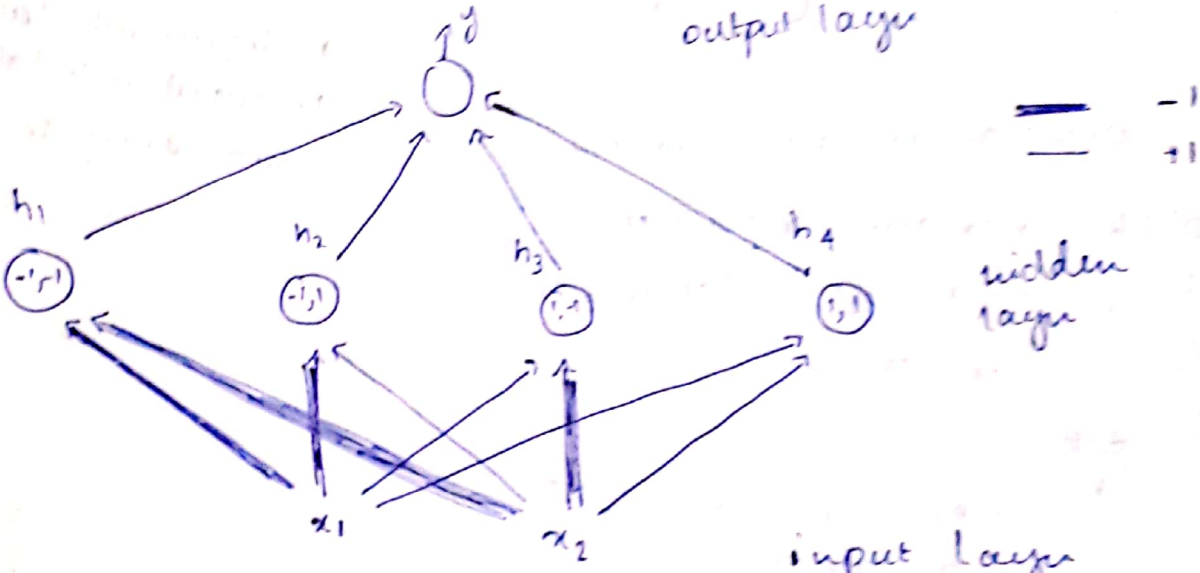
for two inputs, two input combinations are not linearly separable (XOR & !XOR)



Network of perceptrons



Network of perceptrons - Multilayer Perceptrons



XOR for this

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\Sigma w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

for XOR

$w_1 < w_0$ $\quad w_2 \geq w_0$

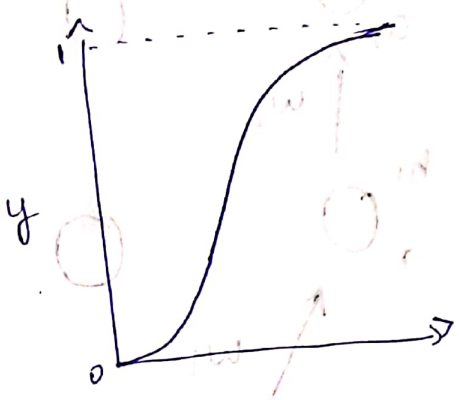
very neuron in hidden layer caters for each combination of the input and weights wi can adjusted to get the required output for the case.

there are n inputs, 2^n perceptrons in hidden layer. (for boolean functions)

Sigmoid neurons

logistic function

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



Output y is no longer binary.

Error function

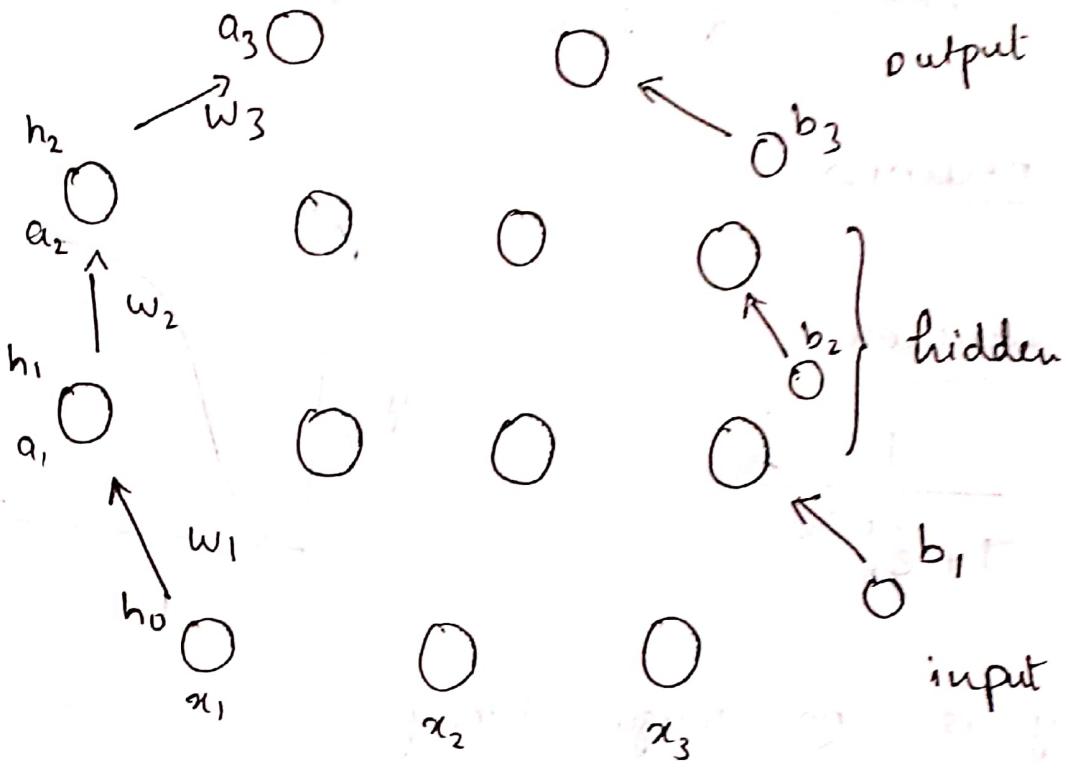
$$\text{loss} = \frac{1}{2}\left(\sum y_i - f(x_i)\right)^2$$

gradient descent

$$\theta_{t+1} = \theta_t - \eta(\text{loss})$$

Feed - fawad Neural Network.

Each neuron in hidden layer has 2 layers (pre activation & activation)



$$a_1 = b_1 + w_1 h_0$$

$$a_i = b_i + w_i h_{i-1}$$

$$h_i = g(a_i(x))$$

$g(x)$ is the activation function.

$f(x) = O(a_3(x))$ Output function

O is Output Activation function.

$$y = f(x)$$

$$= O(w^3 g(w^2 g(w^1 x + b_1) + b_2) + b_3)$$

Selecting o/p function:

when output needs to be between zero and one, we can use sigmoid function.

for large output values, we can use linear fn.

$f(x) = O(a_L)$ (used for regression in this case)

$= W_0 a_L + b_0$

for classification problems, Softmax function can be used as output function

In case when y is a probability distribution, we use a softmax function to keep $\hat{y}$ in a probability distribution

To find diff between two probability distributions, we can use cross entropy.

$L(\theta) = \text{Loss} = -\sum y \log \hat{y}$

to minimize this loss function,

we should $\min_{\theta} L(\theta) = -\log \hat{y}$

(or) $\max_{\theta} -L(\theta) = \log \hat{y}$

where $\hat{y}$ is the linear combination of all weights in the previous layers.

$\hat{y} = O(W_3 g(W_2 g(W_1 x + b_1) + b_2) + b_3)$

$\hat{y}$ is the probability x belongs to

$\log \hat{y}_i$ is the log likelihood of the data

		Outputs	
		Real Values	Probabilities
Output	Activation	Linear	Softmax
Loss	function	Squared error	Cross entropy

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

$g'(z) = g(z)(1-g(z))$

$g(z) = \tanh(z)$

$g'(z) = 1-(g(z))^2$

Nesterov Accelerated gradient descent : (NAG)

the next step gradients are found beforehand

So course correction is quicker

Oscillations are small, chances of missing minima is small

normal gradient descent → all data calculation is done and then weights updated, actual answer, no approx.

Stochastic G.D → weights updated after each data point

This is an approx gradient

mini-batch (say) updated for sd no of data points

To tune learning rate:

Try different values on log scale (0.0001, 0.001, ...)
Run few epochs of each & figure out what works best
do fine search around this value (if 0.1 then see for 0.05, 0.2 etc)

adaptive learning rate:

step decay:
- halve α every 5 epochs or
- halve α after an epoch if CV error is more than what it is at end of previous

exponential decay:

$\alpha = \alpha_0 e^{-kt}$

$t \rightarrow$ step no

1/t decay:

$$\alpha = \frac{\alpha_0}{1+kt}$$

$$\alpha = \min\left(1 - 2^{-1-\log_2(\lfloor t/250 \rfloor + 1)}, \alpha_{max}\right)$$

$\alpha_{max} = \{0.999, 0.995, 0.99, 0.9, 0\}$

adaptive gradient descent (adagrad)

$$v_t = v_{t-1} + (\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t$$

adagrad decays learning rate aggresively as the denominator grows.

to avoid this, denominator should be decayed.

$$v_t = \beta * v_{t-1} + (1-\beta)(\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t$$

Adam

$$m_t = \beta_1 * m_{t-1} + (1-\beta_1) * \nabla w_t$$

$$v_t = \beta_2 * v_{t-1} + (1-\beta_2) * (\nabla w_t)^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$\beta_1 = 0.9$ $\beta_2 = 0.999$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$$

Adam is best choice

Eigen vectors

$Ax = \lambda x$

Eigen vectors corresponding to different eigen values are linearly independant.

Eigen values of a square symmetric matrix are orthogonal

Principal Component Analysis

dimension reduction

The new dimension are non-redundant (low covariance) & not noisy (high variance)

Singular Value decomposition

used for data compression

SVD factorises a matrix into singular vectors and singular values.

Autoencoder

special type of feed forward NN.

encodes its input into a hidden rep

$$h = g(wx_i + b)$$

decodes again from its hidden rep

$$x_i = f(w^* h + c)$$

model is trained to minimin certain loss fn which will ensure $\hat{x}_i$ is close to x_i

if $\dim(h) < \dim(x)$

h is loss fue encodis of x_i

encodu with $\dim(h) < \dim(x)$ is called

undu complete autoencodu.

$\dim(h) \geq \dim(x)$ -> over complete.

---

g is typically chosen as sigmoid

---

encodu part = PCA if.

- linau encodi
- linau duodu.
- squand error lon fn.
- normalin i/p to

$$\hat{x}_{ij} = \frac{1}{rm}\left(x_{ij} - \frac{1}{m}\sum x_{kj}\right)$$

Sparse Autoencodu

tries to ensun neuon is inactive most of the tim

Contractive Autoencodu

tries to prevent an overcomplete autoencodu from learning identity fn.

generally;

Simple model : high bias, low variance
Complex model : low bias, high variance

Both b and v contribute to MSE

Different regularisation:

- l2
- Datant augmentation
- Paramutu shaing & tying
- Adding noin to i/p
- Adding noin to o/p.
- Early stopping
- Ensemble methods
- dropout

---

l2

$$\mathcal{L}(w) = \mathcal{L}(w) + \frac{\alpha}{2}\|w\|^2$$

$$w_{t+1} = w_t - \eta \nabla \mathcal{L}(w_t) - \eta \alpha w_t$$

Datast augmentation

Eg for image, rotate imagn in diffent angls

Paramutu sharing & tying

Used in CNNS

Same filtu applied at diff portions of imag

Same weight matrix acts on diff input neurons

Adding noise to i/p

Eg: Autoencoder

Adding noise to o/p

The true labels might be wrong

Use soft targets. taking ϵ → small +ve const

a class.

apple | orange | mango | banana

apple | 1 | 0 | 0 | 0 | hard target

apple | $1-\epsilon$ | $\epsilon/3$ | $\epsilon/3$ | $\epsilon/3$ | soft target

Early stopping

Track validation error

If there are no improvements in CV error in last p steps, stop training.

Ensemble methods

Combine Output of different models

Dropout

temporarily remove a node and all its connections resulting in thinned n/w

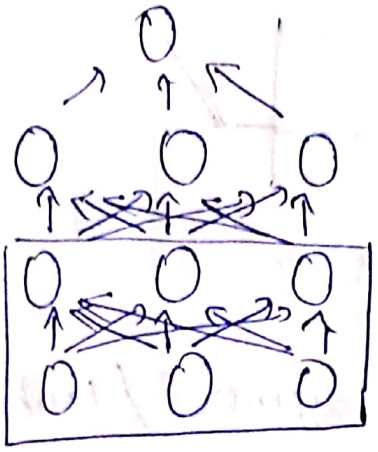
Share weights across all n/ws.

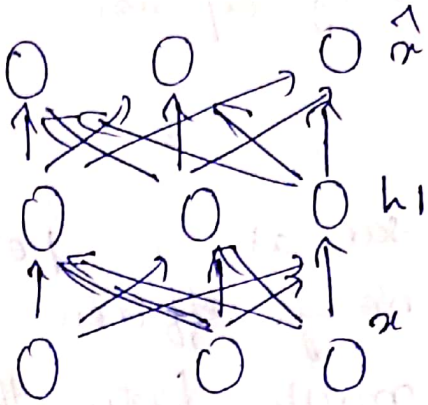
Sample different n/w for each x instance

Unsupervised Pre training

dep learning became popular again due to this.

for example





the weights in layer 1 are trained such that h1 captures an abstract rep of x.

Fix the weights in layer 1 and repeat it for layer 2.

repeat this process till last hidden layer.

y is not involved at all.

After this layer wise pre training, add the output layer.

Pre training leads to better weight initialisations

Sigmoids

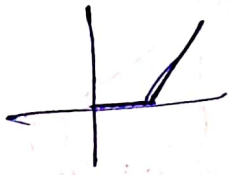
Saturated neurons cause gradient to vanish

Sigmoid and tanh units

Sigmoids are computationally expensive

tanh

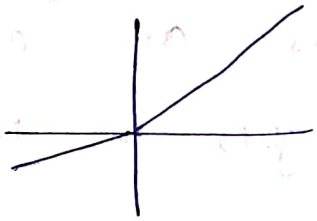
- Zero centred
- gradient vanishes at saturation
- Computationally expensive



ReLu

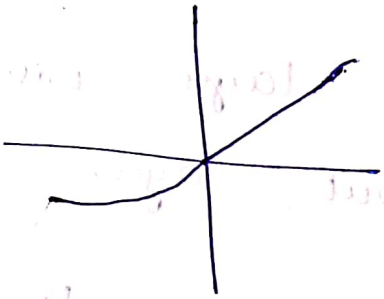
- does not saturate in positive region
- Computationally efficient
- Converges much faster than sigmoid / tanh
- units will die if α is high

Leaky ReLu



- no saturation
- no units will die
- computationally efficient
- close to zero centred o/p

Exponentially linear unit



- all benefits of ReLu
- computationally expensive

Maxout Neuron

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

- generalizes ReLU & Leaky ReLU
- No saturation
- doubles no of parameters

Professor's notes:

Sigmoids are bad
ReLU ~ Standard for CNNs
explore Leaky ReLU / Maxout / ELU
tanh ~ LSTMs / RNNs

---

initialisation strategies

if all weights are initialized zero, all neurons in layer 1 fire up.
this symmetry stays same for all layers.

Batch Normalisation:

Add a normalisation layer after a neuron layer (this is differentiable wrt backpropagation)

One hot encoding

representing single representation claims in terms of vector (mathematical models)

Co-occurance matrix

It is a terms x terms matrix which captures the number of times a term appears in the context of another term

SVD for learning word representations.

SVD gives best rank-k approx of original data

To obtain filter map, sum filter over the input

parameters involved:

width of input (w_1)

height of input (h_1)

depth of input (d_1)

Stride (S)

no of filters (K)

filter dimension (F)

output ($w_2 \times H_2 \times D_2$)

In general,

$$w_2 = w_1 - F + 1$$

$$H_2 = H_1 - F + 1$$

To have output same dim of input, use padding

Stride - area intervals at which filter is applied

$$D_2 = K$$

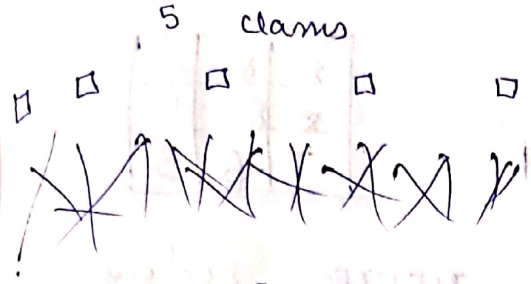
$$w_2 = \frac{w_1 - F + 2P}{S} + 1 \qquad P = \text{padding}$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

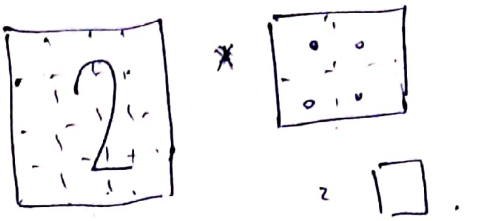
Convolutional Neural Networks:

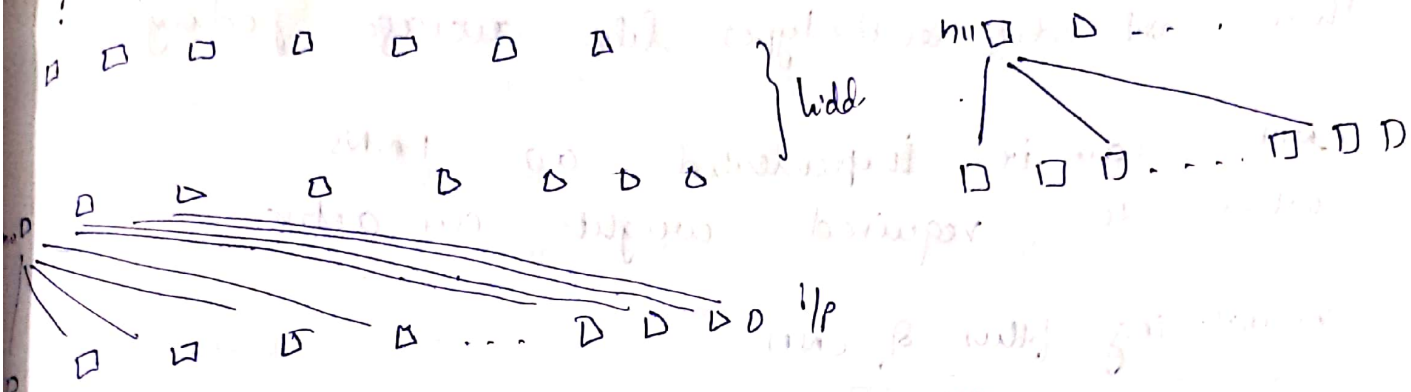
Along with learning weights of classifier, meaningful kernals, filters are learnt.

regular fed forward



CNNs.



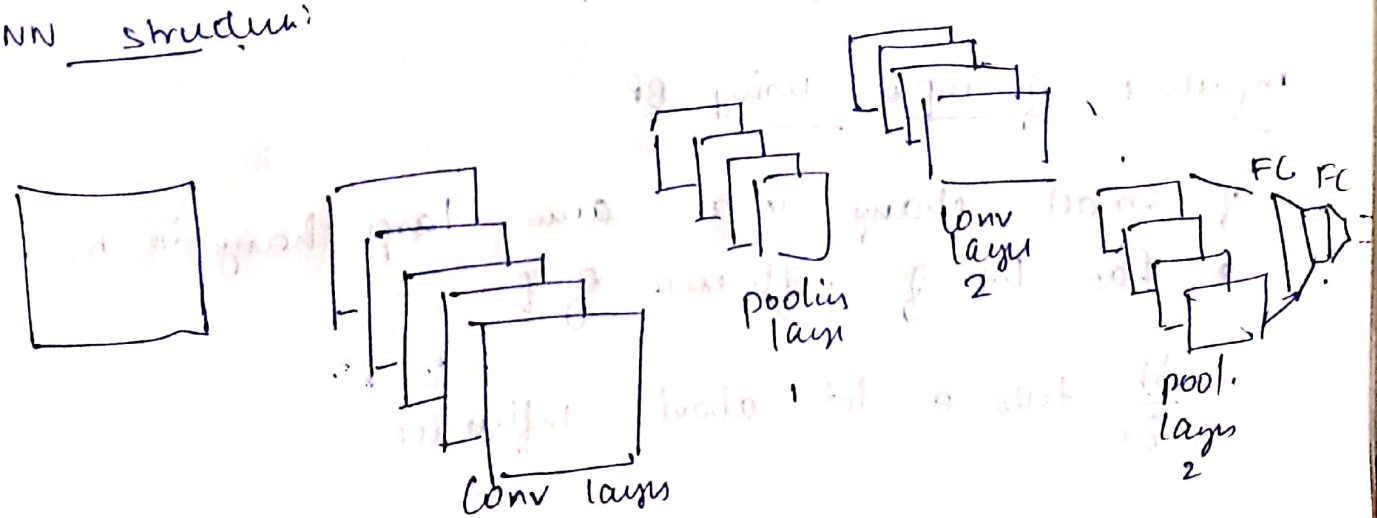


only few neurons contribute for hu

This reduces no of parameters in model

There are many kernels but they are shared by all locations in the image.
This is called weight sharing.

CNN structure:

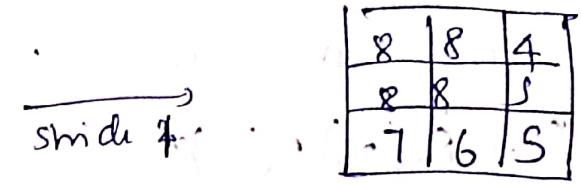


Input * filter

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

maxpool (2x2) (2 stride)

8	4
7	5



then add different types like average pooling

CNN can be implemented as FNNs when the required weights are active

Visualising filters of CNN

finding input which maximally excites a neuron

$x = \frac{w}{\|w\|}$ when w is the filter

so then filters are regarded as pattern detectors.

Occlusion Experiments

graying out certain regions to see how it affects probability of output.

influence of input using BP

if small change in x causes large change in h, x has lot of influence of h

$\frac{\partial h}{\partial x}$ tells a lot about influence

guided back propagation

- feed i/p & do forward pass
- consider some neuron in some feature map in some layer
- find influence of i/p at this neuron
- set all other neurons in region to 0
- back propagate to i/p
- During forward pass, ReLU pass only +ve units
- during back pass, no gradient passes through the dead relu units
- all -ve gradients are set to 0
- plotting this gives a better picture of true influence

Optimisation over images

keep weights fixed
vary input parameters till image is obtained

Creating image from embeddings.

reconstruction of image after different layers.

Deep Dream

Start with some image
Change the image so that some neurons fire even more

Deep art

add an image to another image to add colour and textures

fooling CNNs,

by adding some noise, CNN can be misguided to classify

Sequence Learning Problem

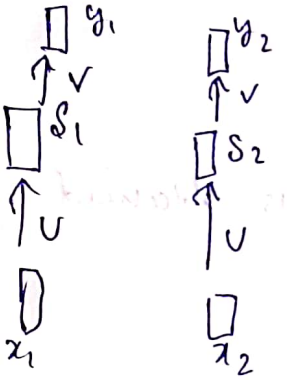
In FFNN, CNN, size of input was fixed.

each input was independent of previous

Examples for sequence learning.

predicting next word.

associating part of speech tag to each word



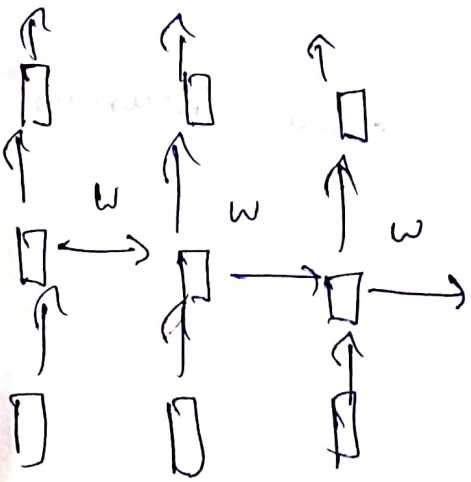
$s_i = \sigma(Ux_i + b)$

$y_i = O(Vs_i + c)$

i: timestep

~~at each time step, feed all previous inputs~~
(this won't work)

Add recurrent connection to n/w.



backpropagate over all previous time steps

Backpropagation through time (BPTT)

to avoid vanishing gradients, use truncated backpropagation

RNN has a finite state size, it needs to be selectively read, written & forgotten.

Long Short Term Memory (LSTM)

Selective write

decide what fraction of each element of previous state to be passed to next state decided by vector o varying b/w 0 & 1. RNN learns o along with other parameters

Selective read.

another function to eliminate some info passed over

Selectively forget:

another gate added to forget when needed

gates:

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

States:

$\tilde{s}_t = \sigma(Wh_{t-1} + Ux_t + b)$

$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$

$h_t = o_t \odot \sigma(s_t)$ & $rnn_{out} = h_t$

LSTM has many variants which include diff no of gates & also diff. arrangement of gates.

- a popular variant- GRU (gated Recurrent Unit)

No explit forgt gate (the forgt gate & input gates are tied)

LSTMs avoid vanishing gradients

During forward pass, gates control flow of info prevent irrelevant info from being written

During backward pass the gradients will get multiplied by gate.

Gradient vanish only when they should (when info is not needed)

But it does not deal with exploding gradients.

To overcome that, use gradient clipping.

while backpropagating, if norm of gradient exceeds a certain value, it is scaled to keep its norm within an acceptable threshold.

generating sentence given an image (encoder decoder)

we have find $P(y_t | y_1^{t-1}, I)$

where $P(y_t | y_1^{t-1})$ is the present output given previous outputs as input (sequence)

- CNN is used to encode the image,
- an RNN is used to decode into to a sentence

Applications:

- text entailment
- Machine translation
- Transliteration
- Image qn answering
- Document Summarisation
- Video captioning
- Video Classification
- Dialog

Attention mechanism

define a function f_{ATT} to specify which words need more weightage

Represent every location (time step) with a weight

for image, use output of one of the conv layers which has spatial information

Heirarchial attention

for sequence of sequences (eg: dialog between user and bot)

Use 2 level RNN.

first level operates on sequence of words in each utterance

another RNN which encodes this and gives single representation for sequences

document summarisation can be done using this