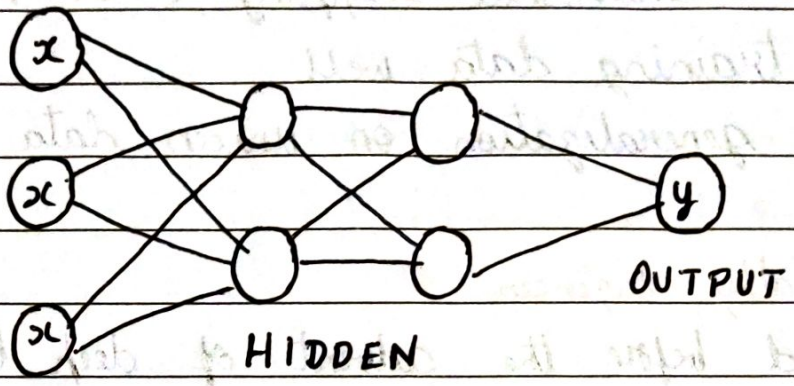
__/__/____

UNIT 1

* **Multi Layer Perceptrons**

- Feedforward Neural Network



- Aim : Approximate function f that maps input x to output y

- $y = f(x ; \theta)$ learns $\theta \rightarrow$ model parameters

- Linear function $\equiv$ Linear models $\equiv x$
- Non linear fn $\equiv$ Linear model $\equiv$ transformed input $\phi(x)$

$\phi \rightarrow$ non linear transformation

* **Activation Function :**

Used to compute hidden layer values

- Strategies to choose ϕ

1) Use generic ϕ
- high dimensional mapping (RBF kernel)
+ fits training data well
- poor generalization on unseen data

2) Manually engineer ϕ
- existed before the advent of deep learning
- required extensive human effort for each separate task

3) Learn ϕ (Deep learning strategy)

$$y = f(x : \theta, w) =$$
$$y = \phi(x : \theta)^T w$$

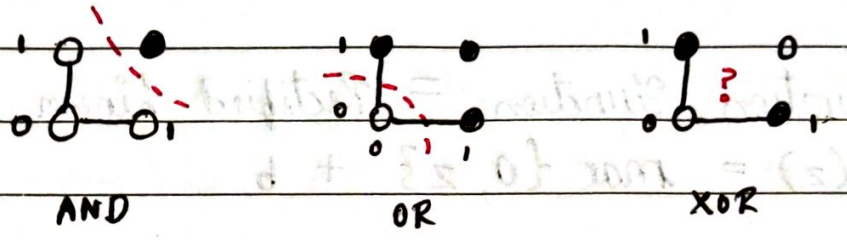
- Learns the mapping through parameter ϕ
+ Combines adv of generic and manual methods
- sacrifices convexity in training.

→ generic : broad family $\phi(x:\theta)$ is used
→ manual : humans can design

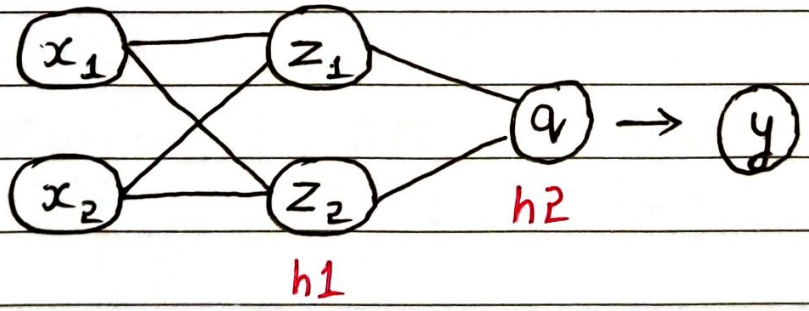
θ → Params that learn the transformation $\phi(x)$
w → Params that map the $\phi(x)$ to output

* Learning XOR

- Single layer Perceptron ≡ linearly Separable fn
- AND, OR, NOT, NAND, NOR



- Soln : Using Second layer of Units (MLP)
"Piecewise linear classification using MLP"



$$y = x_1 \bar{x}_2 + \bar{x}_1 x_2$$
$$y = z_1 + z_2$$

$$z_1 = \sum x * w$$

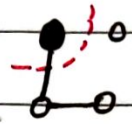
If $z_1 \neq e$

Update weights, $w = w + (e - o)\eta\, x$

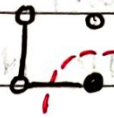
(back propogation)

__/__/____

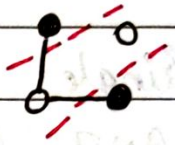
$z_1 \Rightarrow$



$z_2 \Rightarrow$



$y \Rightarrow$



* Activation Function = Rectified Linear Unit (ReLU)

$g(z) = \max\{0, z\} + b$

x — Input

z — Transformation of Input

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

y — Activation function applied to produce output, $y = g(z)$

* **Rectified Linear Unit [ReLU]**

$$g(z) = \max(0, z)$$

* **Maxout Units**

- Divides z into k group
- Each group outputs maximum element

$$g(z) = \max_{j \in G} z_j$$

* $$g(z, \alpha) = \max(0, z) + \alpha \min(0, z)$$

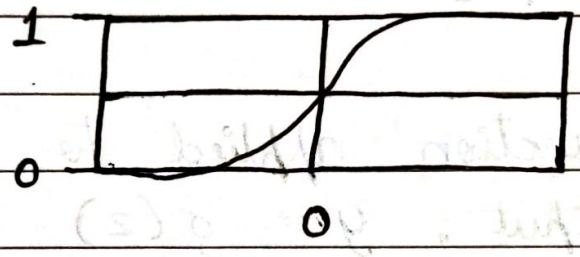
Leaky ReLU $\Rightarrow$ $\alpha = 0.01$ (small value)

Parametric ReLU $\Rightarrow$ α = Learnable Parameter

__/__/__

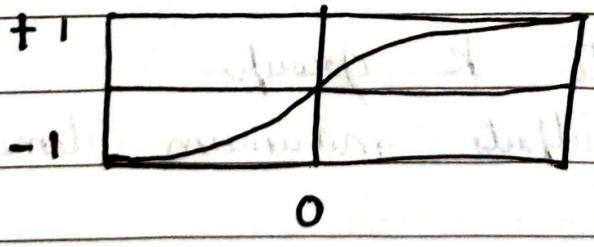
* Logistic Sigmoid

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



* Hyperbolic Tangent

$$g(z) = \tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$



* Gradient Based Learning

Training process of neural networks,

- Initialize weights with σ, γ values & biases
- SGD (use iterative gradient optimization algo)
- Compute gradients using backpropogation algorithm

- Cost Function

- It reflects how well the model predicts target y given input x

- Cost function based on principle of maximum likelihood. Cross entropy between $P(y \mid x; \theta)$ actual target & model's prediction

- Total Cost fn = Cost fn + Regularization

Regularization prevents overfitting
L1, L2, Dropout, Early stop

- Eg: Mean Squared Error, Cross Entropy

$$z = w^T h + b$$

__/__/____

* Loss fn for max like of Bernoulli activation

- Classification tasks of y with binary variable
- Bernoulli distribution defined by single number.

$$P(y=1 \mid x) = \max(0, \min(1, z))$$

problematic, wherever z is outside $[0, 1]$
output is 0 and no longer learn

- Soln: Combine Sigmoid output with max likelihd

$$\hat{y} = \sigma(z) \qquad \sigma \rightarrow \text{Sigmoid Activation}$$

$$P(y) = \sigma((2y-1)z)$$

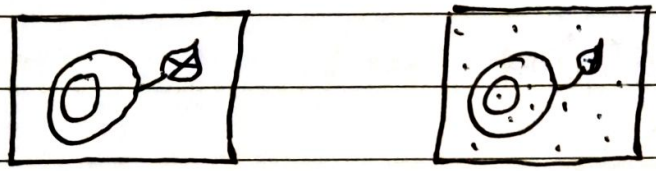
- Loss function

$$L = -\log P(y \mid x)$$
$$= -\log \sigma((2y-1)z)$$
$$= \text{softmax}(1-2y)z$$

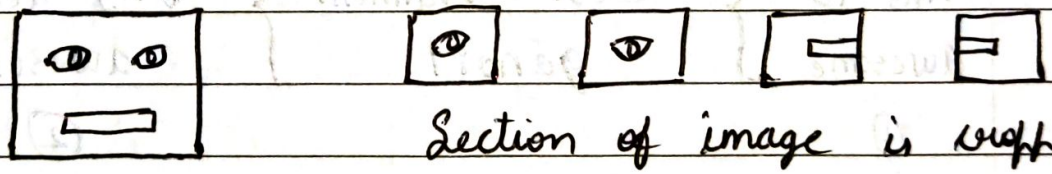
* Data Augmentation
- Generates different versions of a dataset
- More data to train
- Increases accuracy of the model

1) Adding Noise

Salt and Pepper noise



2) Cropping



Section of image is cropped

3) Flipping

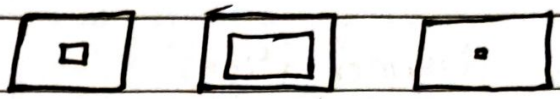
vertically

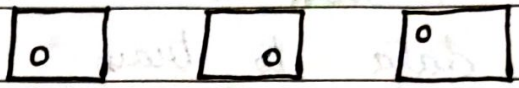
horizontally

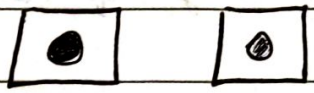
4) Rotation

Rotated bw 0 to 360°

__/__/____

5) Scaling 

6) Translation 

7) Brightness 

8) Contrast

9) Color Augmentation

— Back Translation for words

This is awesome } C'est vraiment génoit } It's really awesome

① ②

__/__/____

* Regularization

- avoids overfitting
- model generalizes better
- model performance on unseen data improved
- It is a technique that makes slight modifications to learning algorithms

- L2

$$\text{Cost fn} = \text{Loss} + \frac{\lambda}{2m} \sum w^2$$

- Weight Decay
- It forces weight to decay towards zero (but not exactly zero)
- λ is the regularization parameter, it is optimised for better results

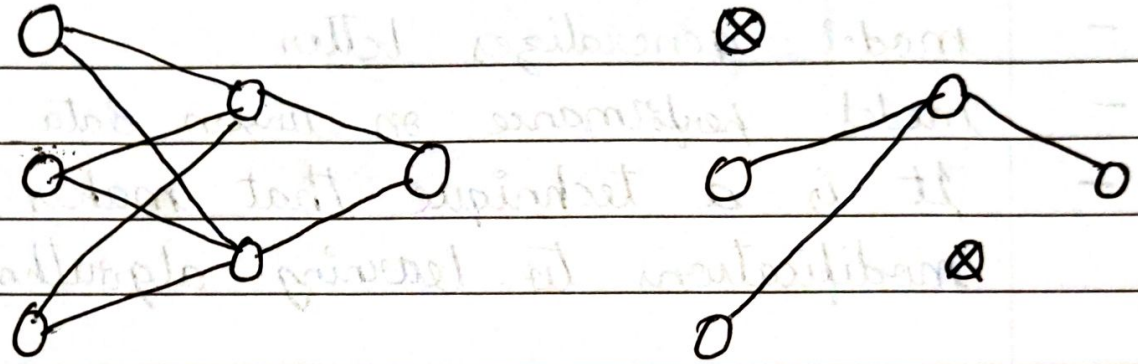
- L1

$$\text{Cost fn} = \text{Loss} + \frac{\lambda}{2m} \sum w$$

- Penalizes the absolute weights value
- Weights can be zero (compresses the model)

__/__/____

- Dropout



- Most frequently used
- At every iteration, some nodes are selected and removed

__/__/____

UNIT 2

- Convolution Operation
- Convolution Operation (for 2D input data)
- Core Component of CNN
- Three ideas : SI, PS, ER
- Visual Cortex is box for CNN
- Types of data used in CNN models
- Strides in CNN

__/__/__

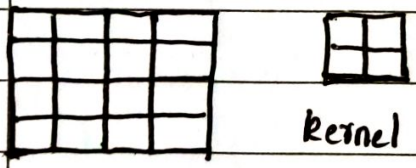
* Convolution Neural Network [CNN]

- Yann le Cunn {1989}

- Specialized type of nn for processing data that has [grid-like] topology

  → 2D grid ≡ images
  → 1D grid ≡ time series

- They employ mathematical operation called (Convolution) in place of general Matrix multiplication



input x

kernel k

Sliding dot product b/w x and k

convolution operation

$s(t)$ = Position at time t

$$s(t) = (x * k)(t) = \sum (x * k)(t)$$

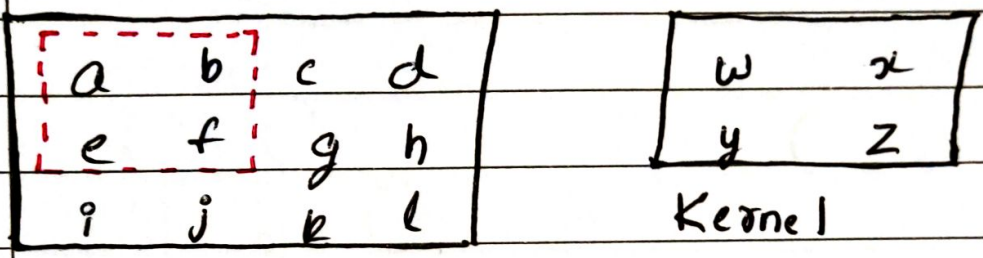
$$s(t) = (x * k)(i, j) = \sum_m \sum_n x(i+m, j+n)\, k(m, n)$$

output: feature map

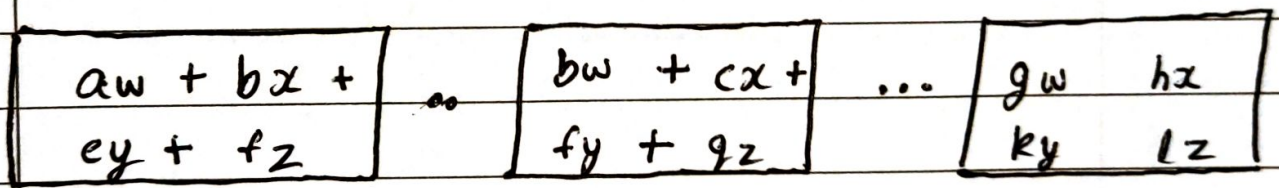
__/__/____

* Convolution to 2D tensor

Tensor : Multidimensional Array



Input Kernel



Output

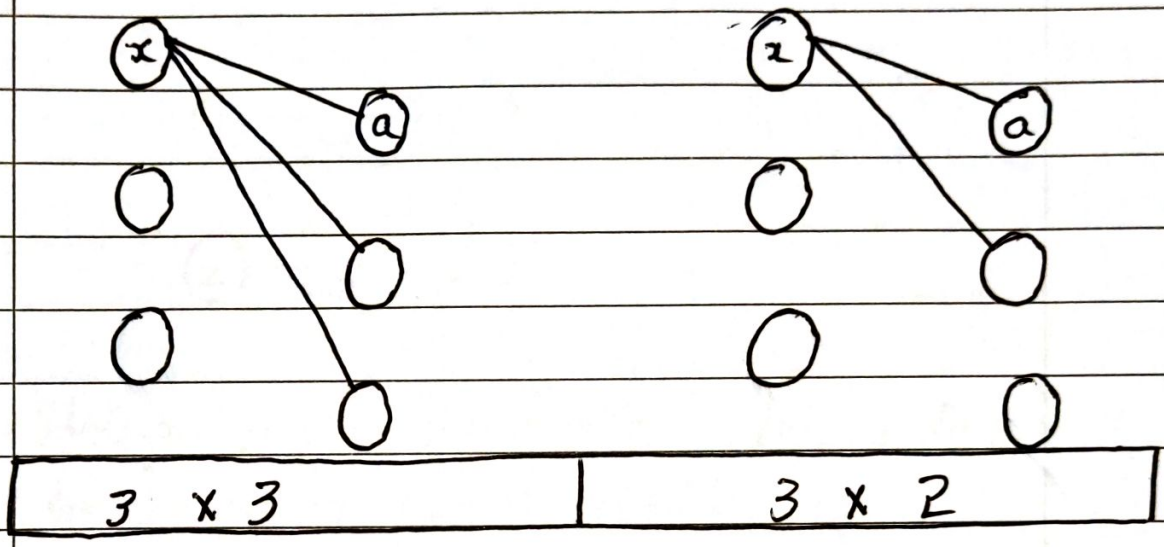
Sparse interaction

Parameter sharing

equivariant representation

__/__/____

* **Sparse Interaction**

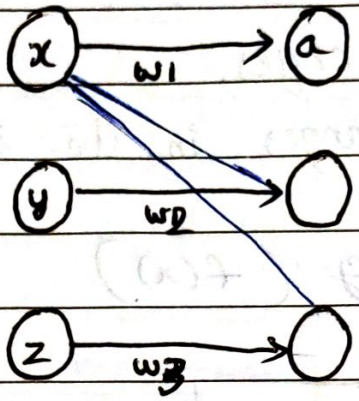


- SI ≡ SC ≡ SW
- Accomplished by making (kernel) < (input)
- Using fewer parameters
  - reduces memory
  - improves efficiency [computing output requires fewer operation]
- Input (m) — reduce to (k) — Output (n)

(m x n) params
(k x n) params

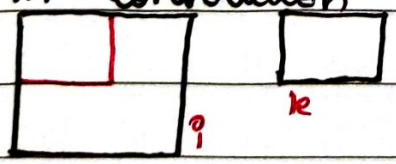
[matrix multiplication]

* Parameter Sharing



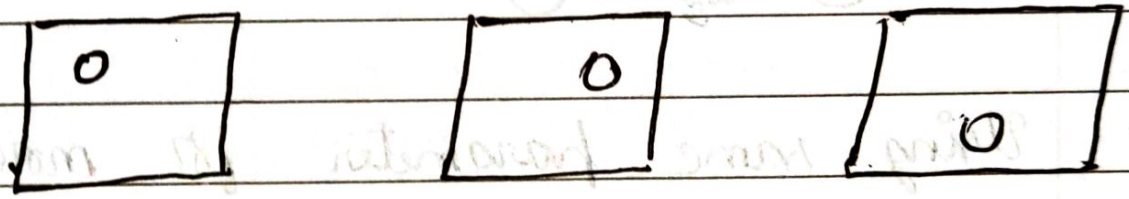
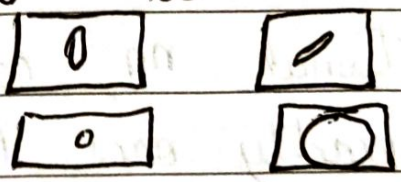
at x, w_1 is used
at y, w_2 is used
at z, w_3 is used

- Using same parameter for more than one function in a model.
- In traditional nn, each element of weight matrix is used exactly once to compute output.
  Eg: $w_{11} \times x_1$, $w_{21} \times x_1$, ...
  $w_{12} \times x_2$, $w_{22} \times x_2$, ...
- In convolution nn, kernel is used at each position of input



- Instead of learning separate set of parameters for every position, we learn only one set
- Parameters of model from $(m \times n)$ to (k)

* Equivariance representation

- If the input changes, output changes in the same way.
- $f(g(x)) = g(f(x))$
- 
- Not Equivariant to rotation / scale 

__/__/____

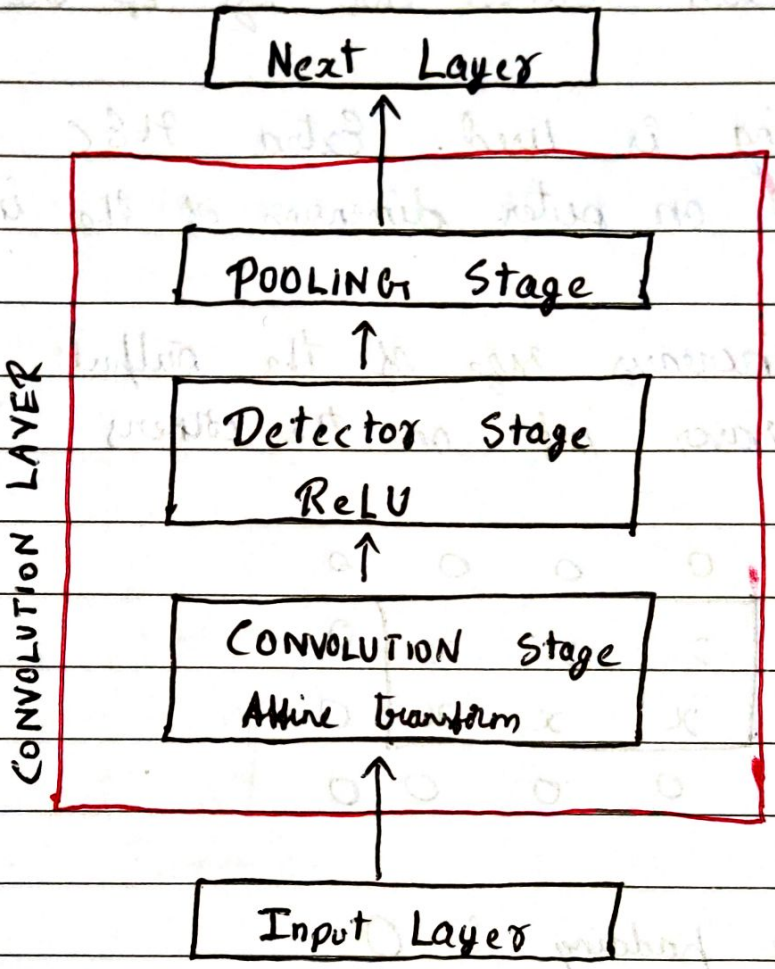
* **Pooling**

- down sampling technique
- reduces the size of feature maps (generated by convolution layer)
- retains important aspects / features

i) Dimensionality reduction
ii) Feature Extraction
iii) Translation Invariance (ER)
iv) Generalization
v) Reduction in computational time

- Max Pool, Avg Pool, Global Avg

* CORE Components of CNN



* Stride

No of pixel shifts over input matrix

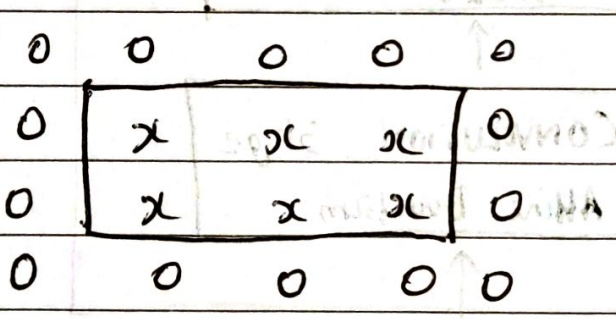
$$\left[\frac{n + 2p - (f+1)}{s} + 1\right] \left[\quad\right]$$

n → input
p → padding
f → filter

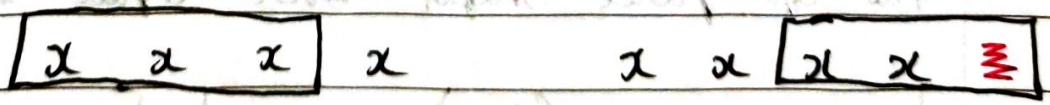
__/__/____

* **Padding**

- Convolution reduces the size of the output
- Padding is used. Extra 0's are added on outer dimension of the images
- It increases size of the output
- It saves info on the corners

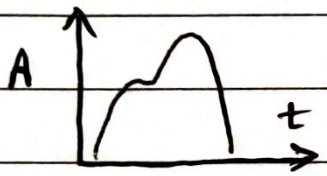
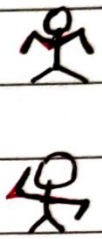


Same padding : 0
Casual padding : x
Valid padding : corner elements are not ignored



__/__/____

* DATA TYPES

	Single Channel	Multichannel
1D	Audio Waveform 	Skeleton Animation Data 
2D	Audio Data that has been processed with Fourier Transform 1D audio waveform ↳ 2D tensor	Color Image Data Ch1 Red, Ch2 Blue, Ch3 Green
3D	Volumetric Data Medical Imaging (CT Scan)	Color Video Data Ch1 time, Ch2 height, Ch3 width

* Neuroscientific basis for Convolutional Networks

- CN is inspired by the way mamalian visual system processes images

- findings from Hubel and Wiesel

  specific NEURONS in visual cortex respond to pattern of light (edges, bars)

  ↓

  different NEURONS are tuned to detect specific visual feature / pattern

- CN mimics the way brain processes visual info,
  i) spatial arrangement
  ii) simple and complex cell behaviours
  iii) hierarchial feature detection

i) V1 (Primary Visual Cortex) is a 2D structure.
   CN ≡ uses 2D feature maps

ii) Simple cells respond to local features (Filters in CN)
ii) Complex cells detect features equivariantly (Pooling in CN)

iii) Grandmother Cells. Certain cells are activated for specific stimulii, regardless of image presentation

DL

__/__/____

UNIT 3

* Sequence Modeling

- RNNs are neural networks for processing sequential data
- Sequence of values : $x^{(1)}, \ldots x^{(n)}$
- Eg : Time series data, Text, ...
  (Data where order of elements matter)
- Scalability : They can handle longer sequences
- Parameter Sharing : RNNs use same set of parameters (weights) across different time steps
  - "I went to Nepal in 2009"
  - "In 2009, I went to Nepal"

Using same parameter, network can generalize better across different sequence length

- In RNN, each output depends on previous output, not just the input
- This recurrent structure creates deep network (Deep Computational Graph)

* **Unfolding Computational Graphs**

- Computation Graph is a visual or mathematical representation of computations

- Unfolding is applied to recurrent computation
- It transforms recursive structure into non-recursive

* Dynamical system, $s^{(t)} = f(s^{(t-1)}; \theta)$

- $s^{(t)}$ → state of the system

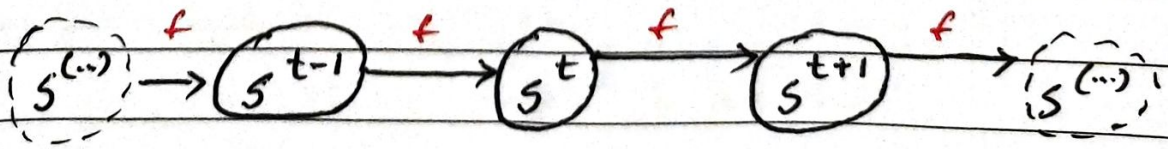
- It is recurrent, def^n of s at t refers back to def^n of s at $(t-1)$

- For time steps $T = 3$
Graph can be unfolded by

$$s^{(3)} = f(s^{(2)}; \theta)$$

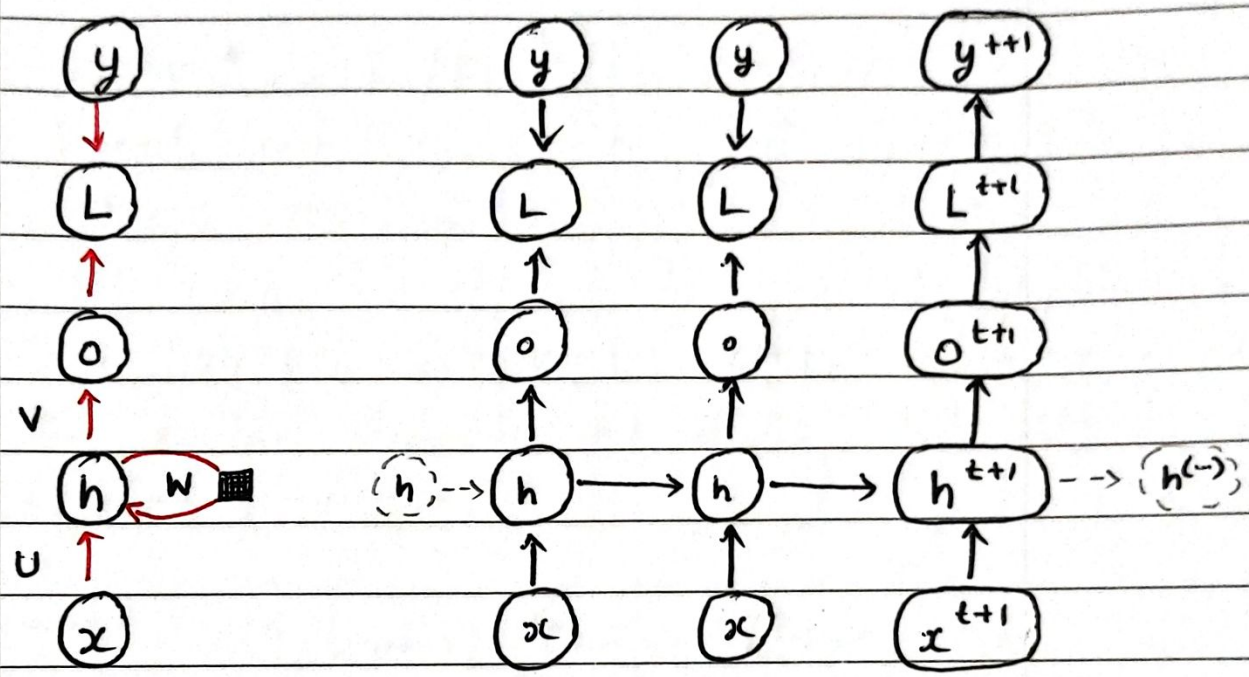
$$s^{(2)} = f(s^{(1)}; \theta)$$

$$s^{(3)} = f(f(s^{(1)}; \theta); \theta)$$



__/__/____

* Recurrent Neural Network



$$a^{(t)} = b + W h^{(t-1)} + U x^{(t)}$$

$$h^{(t)} = \tanh(a^{(t)})$$

$$o^{(t)} = c + V h^{(t)}$$

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$$

bias → b and c

weight matrices → U V W

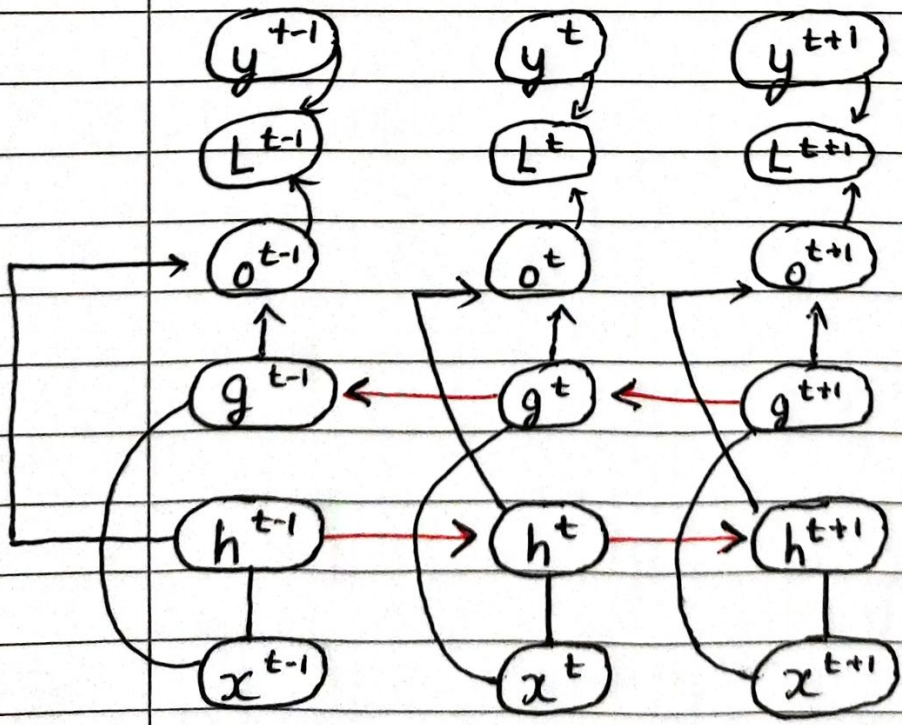
i-h h-o h-h

* Bidirectional RNN

- Recurrent network so far has casual structure. State at time t captures info from past, 1, t-1, t

- For Speech, Language recognition, bioinformatics, the output depends on whole input sequence (future and past)

- RNN that moves forward through time
  +
  RNN that moves backward through time

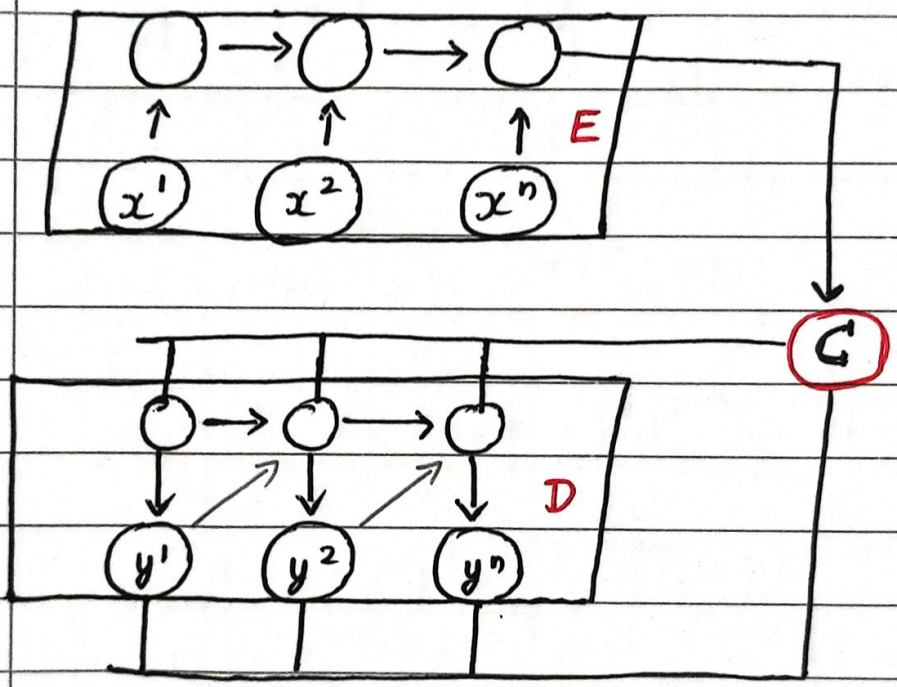


At each t, $o^{(t)}$ depends on
h^t (past)
g^t (future)
context

* Encoder - Decoder (Sequence to Sequence)

- To map input sequence ~~with~~ to output sequence of different lengths

- Speech translation, Question answering

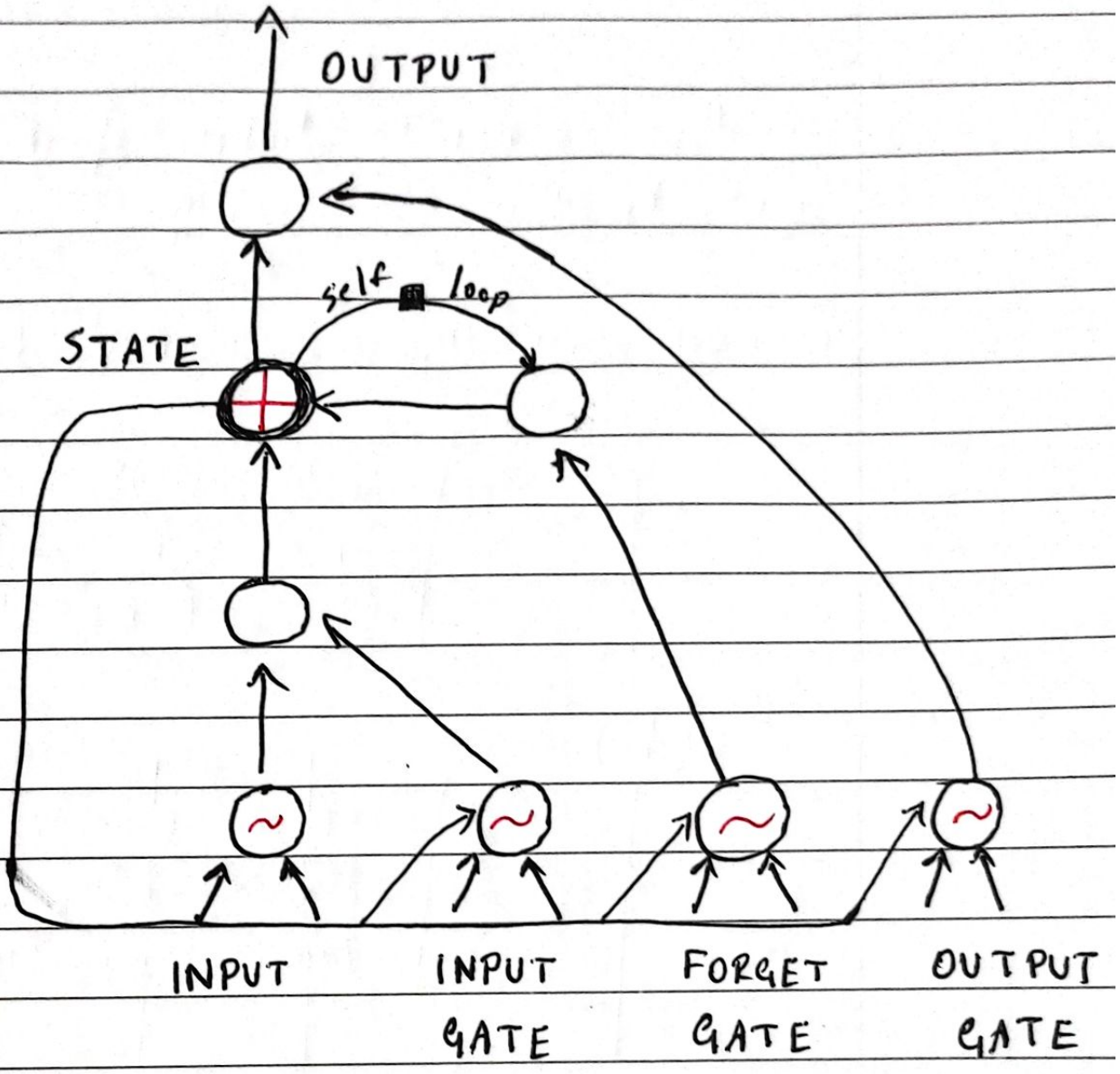


* Encoder processes the input sequence $X = x^1\ x^2 \dots x^n$ and generates context vector C

* Decoder uses this C to generate output $Y = y^1\ y^2 \dots y^n$ of varying length

__/__/____

* LSTM



- Gated recurrent network
- They can learn whento forget, whento retain information
- Use self-loops to maintain gradients over long sequence (weights controlled by f_g) (dynamic weight)

- Forget Gate

$$f = \left(b + \sum U\, x^t + \sum W\, h^{(t-1)}\right) \sigma$$

x^t → Current Input Vector
h^{t-1} → Previous Hidden Vector
b → Bias
U → Input Weights
W → Recurrent Weights

- Input Gate

$$i = f^{(t)}\, i^{(t-1)} + g^{(t)}\, \sigma \left(b + \sum U\, x^t + \sum W\, h^{t-1}\right)$$

- External Input Gate

$$g = \sigma\left(b + \sum U^g\, x^t + \sum W^g\, h^{t-1}\right)$$

- Output Gate

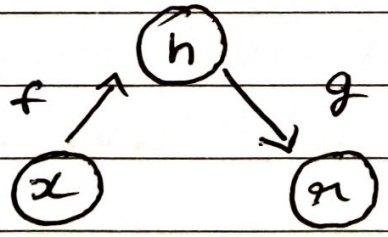
$$o = \sigma\left(b + \sum U^o\, x^t + \sum W^o\, h^{t-1}\right)$$

__/__/____

UNIT 4

* AUTOENCODER

- Neural network trained to attempt to copy its input to its output
- Goal : learn compact representation of input
- Encoder : $h = f(x)$
- Decoder : $r = g(h)$



- To avoid perfectly replicating input, they are restricted in some way so they can learn important features of data

(h) → internal representation

__/__/____

* Undercomplete Autoencoders

- Copying input to output is useless, but we are interested in that during training, (h) will learn useful properties

- By constraining (h) to have smaller dimension than (x), we can obtain useful features

- Autoencoder whose code dimension (h) is less than input dimension (x)

- Learning Process:
  Minimizing loss function, $L(x, g(f(x)))$

  loss function = MSE

- Linear function used ⇒ performs PCA
- Nonlinear function used ⇒ learn complex repr^{n}

* Regularized Autoencoders

- (h) < (x) but E and D have lot of capacity
- (h) = (x)
- (h) > x

↳ Autoencoders fail to learn useful features

* RA provide ability to train any {architecture of autoencoder}, choosing
  - code dimension
  - capacity of E and D

  } based on complexity of distribution

* They use LOSS FUNCTIONS that encourage models to have other PROPERTIES besides copying $i \to o$

  - Sparsity of the representation
  - Smallness of derivative of representation
  - Robustness to noise

* RA can be nL and OC but still learn

1) * Sparse Autoencoder

- It is autoencoder whose training criterion involves a sparsity penalty

$$\underbrace{L(x, g(f(x)))}_{\text{reconstruction error}} + \Omega(h)$$

- Used: learn features for another task (classification)

- $\Omega(h) \equiv$ regularizer term added to ffn
  - task
    - copy $i \to o$
    - perform supervised tasks

- Regularized Maximum likelihood

  maximizing $P(\theta \mid x) =$

  maximizing $\underbrace{\log P(x \mid \theta)}_{\text{likelihood}} + \underbrace{\log P(\theta)}_{\text{prior probability}}$

__/__/____

2) * **Denoising Autoencoder**

- Rather than adding penalty Ω, DA learns something useful by changing the reconstruction error

- $L(x, g(f(x)))$

- DA instead minimizes $L(x, g(f(\tilde{x})))$

$\tilde{x}$ → Copy of x that's been corrupted by some form of noise

- DA must undo this corruption

3) * **Regularizing by Penalizing Derivatives**

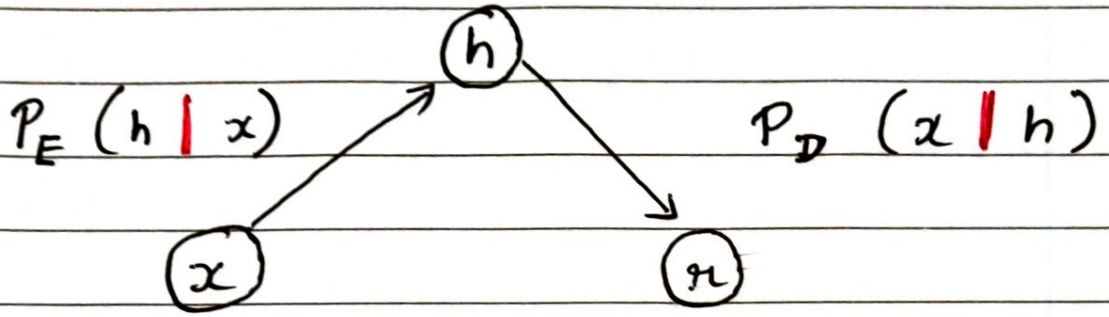
- CAE (Contractive Autoencoder)

- Penalty $\Omega(h)$ of different form

* $\Omega(h, x) = \lambda \sum_i \|\nabla_x h_i\|^2$

⇒ Model learns → fn ⟸ fn doesn't change when x slightly change

* Stochastic Encoder and Decoder



* We can generalize the notion of,

i) encoding function = $f(x)$

⇓

encoding DISTRIBUTION = $P_{encoder}(h \mid x)$

= Stochastic ENCODER

= $P_{model}(h \mid x)$

ii) III^{ly}

__/__/____

* Application of Autoencoder

1) Dimensionality Reduction

- Reduce no of features while preserving important characteristics
- Better than PCA (Hinton, 2006, RBM)
- Lower dimensional representation
  improve PERFORMANCE by ↓ compute burden
  ↓ memory usage
- Aid in generalization

2) Information Retrival

- Task of finding relevant entries in a database based on the query
- AE trained to produce low-dimensional, binary code

  input : { binary code → entries }

  Semantic hashing →

- Data can be image, text

3) Gen. Modelling
Boltzmann Machines