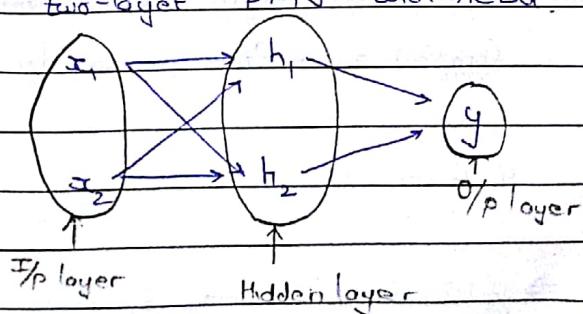


# DEEP LEARNING

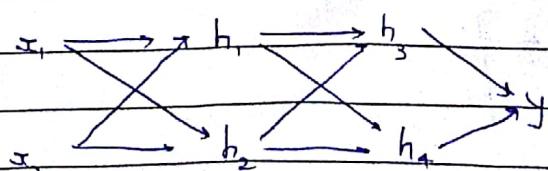
PAGE NO.	/ /
DATE	/ /

## T FEED FORWARD NETWORK

- Multiple layers :- I/p layer, few hidden layers, O/p layers
  - Every layer outputs a set of real numbers.
  - The last layer will employ a conventional classifier on the transformed data.
  - Every layer linearly transforms input, followed by unit-wise non-linearity. (eg - ReLU)
- Neural Networks can model decisions that conventional classifiers cannot  
 eg  $y = f^*(x) = x_1 \oplus x_2$   
 Training data has all four combinations of  $x_1, x_2$ 
  - Linear classifier  $\hat{y} = w_1x_1 + w_2x_2 + b$ , trained with least square loss function, gives  $w_1 = w_2 = 0, b = \frac{1}{2}$
  - \*  $\sum ((y - w_1x_1 - w_2x_2 - b)^2)$  is convex in  $w_i$ 's.  
 - Thus, linear classifier cannot discriminate.
  - Use a two-layer FFN with ReLU.



- Three-layer FFN



# TENSORFLOW

## TUTORIAL

- Good for fast computation.

### → Installation

- TensorFlow
- Google Colab
  - Edit → Notebook Settings → Hardware Acceleration → GPU

### → Tensor

- Multidimensional arrays

### • Computational Graph

- Example program.

\* Dimension =  $(3,)$   $\Rightarrow 1 \times 3$

-  $c = a + b$

- Just makes a computational graph.

- Evaluates only after running `tf.Session.run(c)`

-  $c$  is a tensor. `tf.Session.run(c)` is an ndarray.

- Every program has a (graph) construction phase, followed by an execution phase.

- Graph is made by `tf.Graph`.

\* Shape =  $(3,)$   $\Rightarrow$  1D tensor having 3 values

Shape =  $(3, 1)$   $\Rightarrow$  1D tensor having 3 single-valued tensors

$$\begin{aligned} \text{eg } [1, 2, 3] &+ [[1], [2], [4]] \\ &= \begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 5 & 6 & 7 \end{bmatrix}_{(3, 3)} \end{aligned}$$

→ Placeholders = Non-constant tensors,

- When declared, produce a dummy node in graph
- Feed dictionary will assign values to placeholders whenever a session requiring them is run
- Shape is not specified during declaration

→ Variable

- Variables are tensors that hold their value across session runs
- Can be used as updating parameters for gradient descent

$$\text{eg } y = Ax + b$$

↑   
 Variable Placeholder

- Variables need to be declared with an initial value.  
The variable initializer needs to be run before further computations.

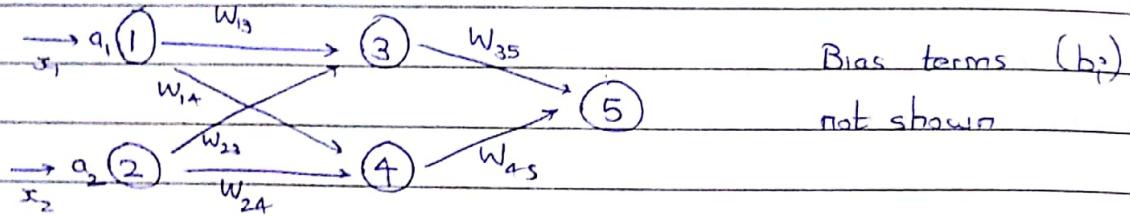
• Variable update

- Variable is updated only after a session is run.

19/g

## FFN continued

## → Parametrized Model



Every node could be a logistic classifier.

$$q_3 = \vec{x}_3 = g(W_{13}x_1 + W_{23}x_2 + b_3)$$

where  $g$  is some function

$$\vec{x}_4 = g(W_{14}x_1 + W_{24}x_2 + b_4)$$

(TFT)

$$\vec{x}_5 = W_{35}\vec{x}_3 + W_{45}\vec{x}_4 + b_5$$

- The same  $g()$  is typically used for all hidden layers
- Say  $\vec{x}$  is  $d$ -dimensional, output of hidden layer whose input is  $\vec{x}$  is  $m$ -dimensional  $\vec{b}$ , then  

$$\vec{h} = \underset{m \times 1}{W^T} \underset{m \times d}{\vec{x}} + \underset{m \times 1}{\vec{b}}$$
- The value of  $m$ , and no. of hidden layers, are decided by designer.

- \* Simplest neural network is the perceptron

$$\text{Perceptron } (\vec{x}) = \vec{w}^T \vec{x} + \vec{b}$$

- A one-layer FFN has a form.

$$\text{MLP}(\vec{x}) = g(\vec{w}_1^T \vec{x} + \vec{b}_1) \vec{w}_2 + \vec{b}_2$$

P.T.O.

## A] Activation Function ( $g$ )

For one-layer FFN,  $MLP(\mathbf{x}) = g(\mathbf{x} \mathbf{w}_1 + \mathbf{b}_1) \mathbf{w}_2 + \mathbf{b}_2$

- The activation function cannot be linear. Otherwise, we will get a linear classifier back.

Proof Let  $g(z) = z$ ,  $\mathbf{z} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$ ,  $\mathbf{w}_2 = \begin{bmatrix} \mathbf{w}_{21} \\ \mathbf{w}_{22} \end{bmatrix}$ ,  $\mathbf{w}_1 = \begin{bmatrix} \mathbf{w}_{11} & \mathbf{w}_{12} \\ \mathbf{w}_{13} & \mathbf{w}_{14} \end{bmatrix}$ ,  $\mathbf{b}_1 = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}$ ,  $\mathbf{b}_2 = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}$

$$g(\mathbf{x} \mathbf{w}_1 + \mathbf{b}_1) \mathbf{w}_2 + \mathbf{b}_2$$

$$= (\mathbf{x} \mathbf{w}_1 + \mathbf{b}_1) \mathbf{w}_2 + \mathbf{b}_2$$

⋮

$$= w_{11} w_{21} x_1 + w_{13} w_{21} x_2 + \dots \quad (\text{Linear classifier})$$

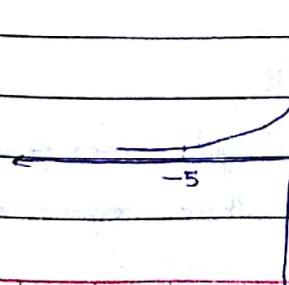
- We should choose a  $g(z)$  that is efficient to compute, easy to optimize (should have an 'informative' gradient), almost linear

↓  
Hidden layers should not lose characteristics of input

## A

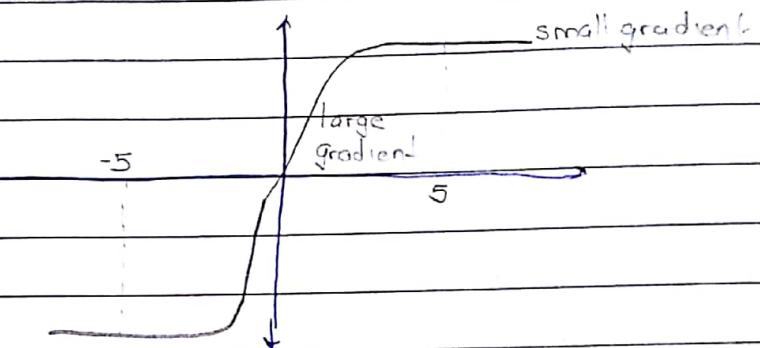
### Common Activation Functions.

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



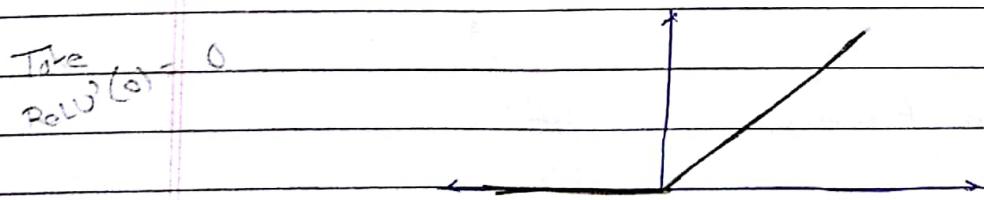
This works well if the input lies in the (almost) linear region of sigmoid function  $[-1, 1]$

b Hyperbolic Tangent :  $\tanh = \frac{e^{2x} - 1}{e^{2x} + 1}$



- Large and small gradients lead to convergence problems
- Because none of these functions is convex or concave, the gradient descent becomes tricky as the zero-slope regions on either side

c Rectified Linear Unit :  $\text{RELU}(x) = \max(0, x)$



∴ Non-differentiable

∴ We can use gradient descent very well.

This is because the region where you get zero gradient is continuous and only on one side. The other region is perfectly linear.

$$\tanh(z) = 2\text{sigmoid}(2z) - 1$$

Page No.	/ /
Date	/ /

Gradient becomes zero in some hidden layer (loss of gradient info)  
↑  
Subsequent layers become inactive

- To avoid 'ReLU saturation', it is suggested to keep bias term slightly positive, so that you begin iterating in the linear region.

- Say, there is one-dimensional one hidden layer  $h = g(w, x)$

$$x \xrightarrow{g} h \xrightarrow{L} y \in \{-1, 1\}$$

$$\text{Loss function: } L(w_1, w_2, x, y) = L(h w_2 y) = L(g(w, x) w_2 y)$$

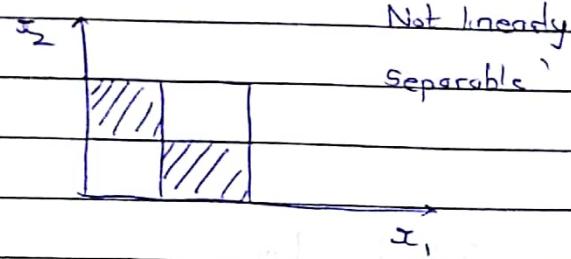
$$\text{Gradient wrt } w_1 = L' w_2 y g' x$$

Saturated if  $g' = 0$

e.g. XOR

Neural networks can train decisions that conventional linear classifiers cannot.

$$y = x_1 \oplus x_2$$



- Linear classifier  $\hat{y} = w_1 x_1 + w_2 x_2 + b$  with least square loss

gives  $w_1 = 0, w_2 = 0, b = \frac{1}{2}$

Useless

- Non-linear functions like  $\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + b$  can work, but the designer will have to think which non-linear term to use.

- Generate one-layer FFN with ReLU

$$y = f(x) = w^2 \underbrace{\max(a, w^1 x + b^1)}_{g(w^1 x + b^1)} + b^2$$

where  $w^1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$   $b^1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$   $w^2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$   $b^2 = 0$

$$\begin{aligned} - [h, h_2] &= [x_1, x_2] \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + [0, -1] \\ &= [x_1 + x_2, x_1 + x_2 - 1] \end{aligned}$$

$$- g(w^1 x + b^1) = \max([x_1 + x_2, x_1 + x_2 - 1], 0)$$

--- Element wise max.

$$\begin{aligned} 1) \text{ Put } [x_1, x_2] &= [0, 0] \Rightarrow w^2 h + b^2 = 0 \quad \checkmark \\ &\downarrow \\ &\max([0, 0], [0, -1]) \\ &= [0, 0] \end{aligned}$$

$$2) \text{ Put } [x_1, x_2] = [1, 1] \Rightarrow w^2 h + b^2 = 0 \quad \checkmark$$

## B] Output Layers

- Depends on type of output

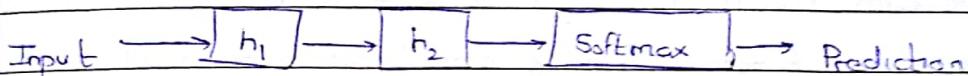
1) Binary class labels :- Sigmoid will convert real number into Bernoulli probability parameter.

2) Multiclass class labels :- Softmax converts real numbers into multinomial probabilities.

3) Real :- Output = mean of Gaussian distribution,

- Advantage of above three :- We get a probability distribution over output. Maximum likelihood training loss is convex in parameters of outermost layer.

### \* Softmax output layer



For every class  $y_k$ , we have a set  $(w_{y_k}^o, b_{y_k}^o)$   $k \in \{1, \dots, k\}$

$$\text{Softmax} \text{ :- } P(y|x) = \frac{e^{w_y^o h_2 + b_y^o}}{\sum e^{w_y^o h_2 + b_y^o}} \quad \begin{matrix} \text{Parameters of} \\ \text{output layer} \end{matrix}$$

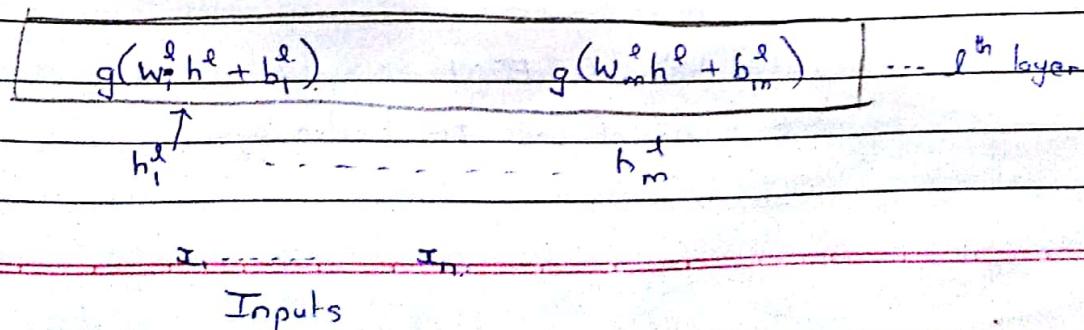
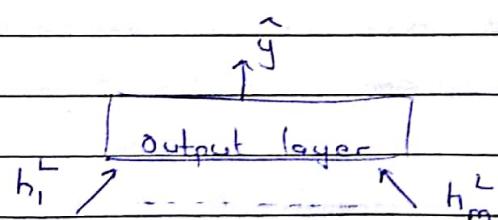
$$\text{Prediction} = \arg \max_y P(y|x)$$

$$\cdot \text{Logit}(y) \triangleq w_y^o h_2 + b_y^o$$

$$\cdot \text{Hard max} = \arg \max_y (\text{Logit}(y)) \dots \text{Non-differentiable}$$

26/g

Summary



### C] Network Architecture

- Choosing no. of layers, width of network, connection between

#### Theorem Universal approximation theorem.

A network with one hidden layer of sigmoid activation can approximate any continuous function from a closed and bounded set given enough hidden units.

- Works for ReLU activation as well.

#### ii) Not practical

- No. of hidden units in the single layer may be huge
- Parameters are not easily learnable. Might overfit.

### A Effect of depth

- Many functions can be efficiently represented with multiple hidden layers, but would have required exponential width (no. of hidden units) with single hidden layer.

- No. of linear regions carved out  $\sim O\left[\left(\frac{c}{C_d}\right)^{d_l} c^d\right]$

\* Verify

$d$  = no. of inputs

$l+1$  = depth of FFN

$c$  = no. of units per hidden layer

- Empirically, larger depth leads to better generalization, lower error

### D] Training an FFN

1. Define loss function  $L(y, \hat{y})$   
 $\uparrow$   
 true value      predicted value. ⇒ Perceptron, hinge, logistic, etc

- $L(y, \hat{y})$  gives non-negative score to the FFN
- $L$  is minimized by gradient descent
- ~~E~~ Initial values of  $W_i$  are very important for neural networks
  - Ensure all  $\vec{W}_i$  are distinct, otherwise hidden units in every layer become identical
  - For example, choose  $\vec{W}_i$  from Gaussian  $(0, \text{some small } \sigma^2)$  for initialization

### E] Stochastic Gradient Descent.

- Not calculating gradient over all instances
- Calculates approximate gradient over a single batch of instances
  - Batches must be containing uniformly random instances
    - Shuffle your instances first, choose required no. of instances. These will come out to be random.
- Expectation of gradient over single batch = True gradient over all instances.

Inputs :- Function  $NN(\vec{x}|\theta)$

Training examples  $\{\vec{x}_1, \dots, \vec{x}_N, \vec{y}_1\} \dots \vec{y}_N\}$

Loss function  $L$ .

Algorithm :-

while (!stopping criterion) {

Pick a batch of  $b$  examples

Compute loss  $L(NN(\vec{x}_i|\theta), \vec{y}_i)$

Compute  $\nabla L$  wrt  $\theta$

$$\theta = \theta - \eta \nabla L$$

$\theta = \vec{w}, b$   
in our case

} *(Learning rate)*

Output :-  $\theta$ .

- Possible stopping criterion :-

Use a validation dataset. Stop when accuracy over validation dataset starts decreasing.

## A Back Propagation Algorithm.

- To compute gradient efficiently.

- Loss function =  $L$

$h$ , is the node which uses weight ' $w$ ' each of the weights (for different inputs) of that

- To do :- Efficiently compute  $\frac{\partial L}{\partial w} \forall w$

- Use chain rule :-

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h} \times \frac{\partial h}{\partial w}$$

compute recursively

easy, we know  $h = g$

- Chain rule on multiple variables

If  $L$  is a function of  $u_1, \dots, u_n$ , which depend on variable  $v_i$ ,

then

$$\frac{\partial L}{\partial v_i} = \sum_j \frac{\partial L}{\partial u_j} \frac{\partial u_j}{\partial v_i}$$

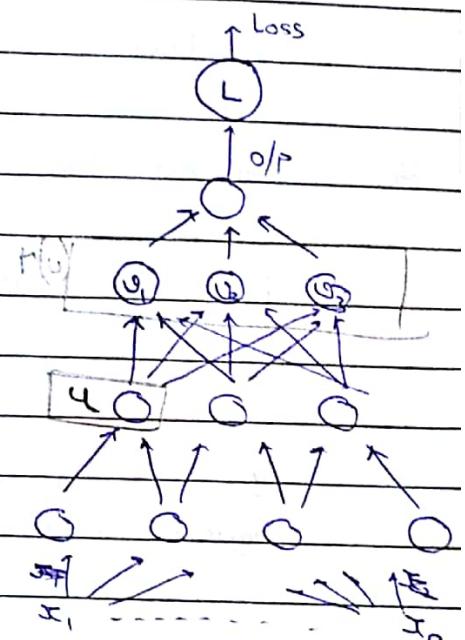
→ Algorithm.

Step 1 Forward Pass

Given the input  $\vec{x}$ , compute outputs of all nodes. The values of nodes will be needed for step 2

Step 2 Backpropagation :- TFT :-  $\frac{\partial L}{\partial w}$

$$\text{using } \frac{\partial L}{\partial w} = \sum_{v \in \Gamma(u)} \frac{\partial L}{\partial v} \cdot \frac{\partial v}{\partial u}$$



Base case :-  $\frac{\partial L}{\partial L} = 1$

$\Gamma(u) = \text{All nodes taking } u \text{ as input}$

for (each  $u$  from top to bottom) :-

for (each  $v \in \Gamma(u)$  = layer above layer containing  $u$ ) :

Compute  $\frac{\partial v}{\partial u}$ .

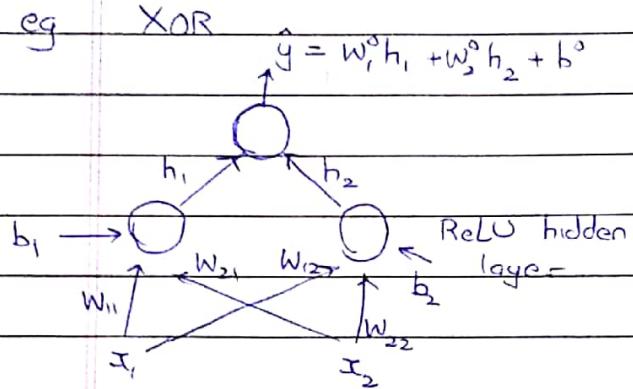
// Inductively, we have already calculated  $\frac{\partial L}{\partial v}$

$$\text{Compute } \frac{\partial L}{\partial u} = \sum_{v \in \Gamma(u)} \frac{\partial L}{\partial v} \cdot \frac{\partial v}{\partial u}$$

$$\text{Compute } \frac{\partial L}{\partial w} = \frac{\partial L}{\partial u} \cdot \frac{\partial u}{\partial w}$$

Optimize → Update every  $w_i$  by  $\bar{w}_i = \bar{w}_i - \eta \frac{\partial L}{\partial w_i}$

eg XOR



Consider the case for 'square loss'

Initially, all  $w = 0$

$$b^0 = 0, b_1 = 1, b_2 = -1$$

One iteration : over one training example  $(x_1, x_2, y) \equiv (1, 0, 1)$

### Step 1 Forward Pass

$$h_1 = \text{ReLU}(w_{11}x_1 + w_{21}x_2 + b_1) = \max(0, 1) = 1$$

$$h_2 = w_{12}x_1 + w_{22}x_2 + b_2 = -1 = 0$$

$$\hat{y} = w_1^0 h_1 + w_2^0 h_2 + b^0 = 0$$

$$\text{Square loss} = L = (y - \hat{y})^2 = 1$$

### Step 2 Back Propagation

$$L = (y - \hat{y})^2$$

$$\frac{\partial L}{\partial \hat{y}} = -2$$

$$\frac{\partial \hat{y}}{\partial y}$$

$$\frac{\partial L}{\partial w_1^0} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_1^0} = -2 h_1 = -2$$

Update  $w_1^0, w_2^0$  similarly &  $b^0$

$$2. \frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial h_1} = -2 w_1^0 = 0$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial h_1} \times \frac{\partial h_1}{\partial w_{11}} = 0 \cdot \partial \text{ReLU}^*(w_{11}x_1 + w_{21}x_2 + b_1) = 0 \cdot 1 \cdot x_1 \\ = 0$$

Similarly update  $w_{11}, w_{21}$  &  $b_1$

FACE NO.	
DATE	/ /

3 Like step 2, find  $\frac{\partial L}{\partial h_2}$ ,  $\frac{\partial L}{\partial w_b}$ ,  $\frac{\partial L}{\partial w_{22}}$ ,  $\frac{\partial L}{\partial b_2}$  - Update  $w_{12}, w_{22}, b_2$

## F] Regularization

- Reducing generalization error of a model, even at the cost of training error

A      Regularization :- Early Stopping

- Training error always decreases, validation error decreases, then increases in case of overfitting.
  - Stop training when validation set increases more than a certain number of times

## RECURRENT NEURAL NETWORKS

→ For sequential inputs,

1 Classifying sequences

$$x_1 \dots x_n \rightarrow y$$

eg - Sentence classification

2 Next word in a sequence

$$x_1 \dots x_n \rightarrow x_{n+1}$$

eg - Language modeling, predictive text

3 Label per token in a sequence

$$x_1 \dots x_n \rightarrow y_1 \dots y_n$$

eg - Parts of speech

4 Sequence prediction

$$x \rightarrow y_1 \dots y_m$$

eg - Translation

- Examples :-

eg Forecasting (in a time series)

$$I/O : x_1 \dots x_{at}$$

where  $x_i$  = day of the week/holiday or not/temperatures...

$$O/P : y_1 \dots y_t$$

where  $y_i$  = demand for an item.

### T A RNN

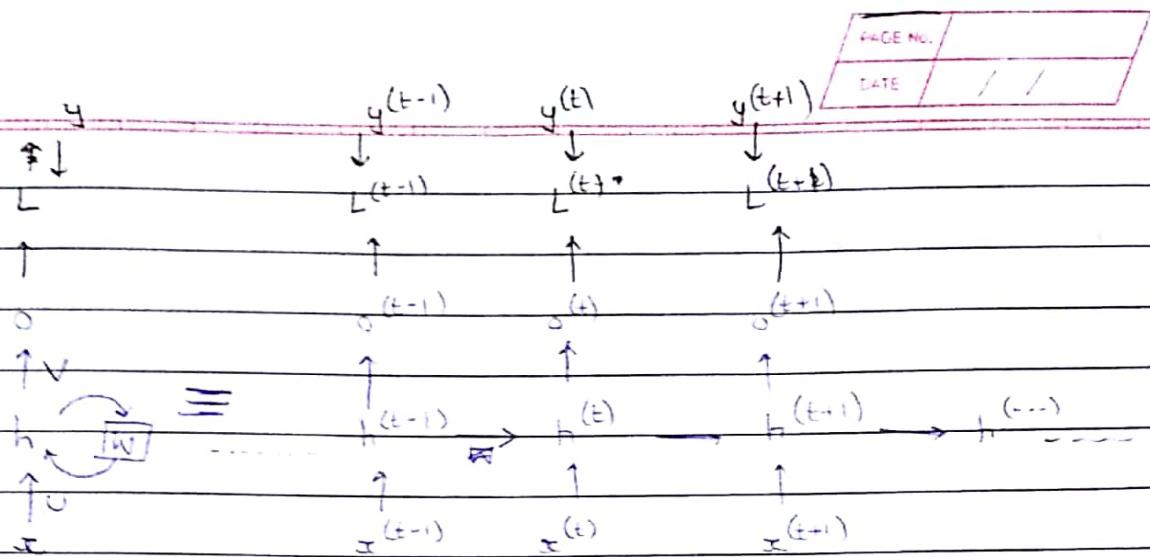
- Processing 1D input of variable length

- \* In CNN, each hidden output is a function of corresponding input and some immediate neighbours

- In RNN, each output is a function of a 'state' summarizing all previous inputs and current input.

- State summary is computed recursively.

- RNN allows deeper, longer-range interaction among parameters than CNNs for the same cost.



$$o^t = C + Vh^t$$

$$\hookrightarrow h^t = \sigma(b + Wh^{t-1} + Ux^t)$$

$\hookrightarrow$  Sigmoid (convention)

\* The end of sequence is represented by a 'None' in the input

- Parameters :-  $U, V, W$

### → RNN :: Forward Computation Example

Sequence  $(x_1, y_1), \dots, (x_t, y_t)$

where  $x_i, y_i \in \mathbb{R}$

Say  $b \in \mathbb{R}^4$ ,  $\Rightarrow w \in \mathbb{R}^{4 \times 4}$ ,  $v \in \mathbb{R}^{4 \times 4}$ ,  $u \in \mathbb{R}^{4 \times 1}$   
 $h \in \mathbb{R}^{4 \times 1}$ ,  $c \in \mathbb{R}^4$

Consider  $h^0 = 0$

Calculate  $h^1 = \sigma(wh^0 + ux_1 + b)$

$$o^1 = Vh^1 + c$$

Consider square loss :-  $L_1 = (y_1 - o^1)^2$

Similarly find  $h^2, o^2, L_2$

\* Practically, we actually lengthen input sequence ('padding') by ~~as~~ junk data. While calculating loss, we must mask the padding contribution.

### A] → RNN for Text (Word Prediction)

- We 'embed' ~~as~~ distinct words to the real number space

Every  $x_t$  denotes a word  $\in [1 \dots V]$

$V = \text{'Vocabulary Size'} \approx 30000$

For each of the 30000 words, we make a vector embedding

Embedding matrix  $S_{V \times d}$

where  $d \approx 300$

Define  $s_{x_t} = x_t S$  = vector embedding of that word  
 $\downarrow$   
 $x_t$  is a  $1 \times V$  one-hot vector

(1 at one position depending on word input, 0 at all other elements)

We now give  $s_{x_t} = x_t S$  as input to RNN to compute  $h_t$  resp

\* Words of similar meaning are placed closer in the (standardized) 300-dimensional space.

\*  $\rightarrow S$  is a look-up table that was learned

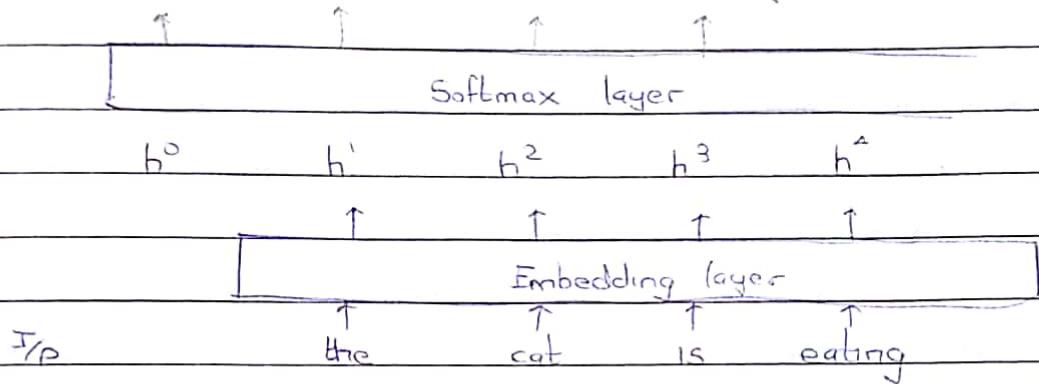
PAGE NO.	
DATE	/ /

→ Training a sequence model

- Maximum Likelihood :-  $P(y | x, \theta) = \prod P(y_t | y_1, \dots, y_{t-1}, x_t, \theta)$

e.g.  $P(\text{the, cat, is, eating})$

$$P(\text{the}) \quad P(\text{cat}) \quad P(\text{is}) \quad P(\text{eating})$$



- Training data

$$D \equiv \{(x^1, y^1), \dots, (x^n, y^n)\}$$

Language model :- We want  $y^{t-1} = x^{t-1}$  (prediction)

Training data :-  $\vec{x}$  :- The cat is sleeping

$\vec{y}$  :- cat is eating EOS

↳ end-of-sequence

→ Training RNN parameters

See following example

→ Backpropagation through time

$$\begin{aligned} \text{Total loss} &= L = L_{t-1} + L_t \quad \text{--- add diff loss at all time} \\ - \frac{\partial L}{\partial v} &= \frac{\partial L_{t-1}}{\partial v} + \frac{\partial L_t}{\partial v} \quad \text{--- } h^{t-1} \xrightarrow{w} h^t \\ &= \frac{\partial L_{t-1}}{\partial o_{t-1}} h^{t-1} + \frac{\partial L_t}{\partial o_t} h^t \quad \text{--- } \alpha^{t-1} \xrightarrow{z} z^t \end{aligned}$$

$$-\underline{\quad \text{LE} \quad} = \underline{\quad \text{LE} \quad} + \underline{\quad \text{LE} \quad}$$

$\text{ME}$        $\text{ME}$        $\text{ME}$

$$= \underline{e_{t+1}} \times \underline{e_{t-1+0e}} \times \underline{e_{t-1-4e}} + \dots$$

$$= \partial L_{t-1} \times \nabla_{\theta} \left( h^{t-2} + M_2 h^{t-2} \right) + \dots$$

This will have all terms till  $\partial b^*$   
 $\partial w$

with coefficient  $W^t$  (can explode or vanish)

$\hat{w}$ : We have to go back over all  $h^n$  to compute gradient

## Exploding or Vanishing Gradient

Gradient is either small or large

- Solutions for vanishing/exploding gradient

Multiple time scales :-

Add direct connection from far past inputs to output instead of depending on state to capture all past inputs.

∴ Cannot change how far back we look at different times or for different inputs.

Solution :- Gated RNNs, (eg - LSTMs)

Q10

R] → Sequence Prediction

$$= (\vec{x}_1, \dots, \vec{x}_n) \longrightarrow \vec{y} = (y_1, \dots, y_n)$$

e.g. - sentences, image, audio

Every  $y_i$  is called a 'token' (any term from a huge vocabulary)

?

e.g. Translation :- < English sentence > → < Hindi sentence >

e.g. Image captioning. < Image > → < Describing sentence >

e.g. Conversation assistance

e.g. Speech recognition :- < Speech spectrogram > → < Phoneme sequence >

- Challenges

- 1) Long range dependency

- Does not assume conditional independence
- ↗ Run cats. He dessings

- 2) Highly open-ended prediction space

- Even length of output is variable

When dragon →  $y_1, \dots, y_t = \text{EOS}$   
at size  $t$ , the sentence

## A Encoder - Decoder Model

- 1) Encode input  $x$  <sup>tokens</sup> into a fixed-dimension vector  $X$ . (contracting)
- 2) Decode  $\rightarrow y$  token by token using an RNN
  - Initialize RNN with state  $x$ .
  - Repeat until RNN generates an EOS token :-
  - Feed previously generated token as input
  - Get a distribution over output tokens and choose the best

Write  $P(y_t | y^{(t-1)}, X, \theta) = P(y_t | h_t, \theta)$

↑  
state vector implemented by RNN

with  $h_0 = X$ .

$\underset{y_t}{\operatorname{argmax}} P(y_t | h_t, \theta)$  is a word known as 'token'

Stop RNN if token = EOS

- We are doing greedy search:- word prediction #<sub>n</sub> does not depend on word #<sub>k</sub> for  $k \geq n$

→ Attention-based sequence learning :-

↳ Alignment b/w input sequence & output sequence

(↔ correspondence b/w English words and Hindi words)

- An RNN can attend to inputs output of another RNN

$$\text{Logit} = \text{'Attention'} = A_{tj} = z_{t-1} \cdot h_j \quad (\text{dot product})$$

Take Softmax ( $A_{tj}$ ) for producing  $t_j^{\text{th}}$  output-element.

'Attention-weighted input states'

$$\sum a_{tj} h_j \quad \text{to produce } z_t$$

## CLUSTERING

### Unsupervised Learning

- Data is unlabelled. We want to group similar elements.

Input:  $D \equiv \{x^1, \dots, x^n\}$

1. Dimensional Representation:-

Let  $x^i \in \mathbb{R}^d$ . Find Euclidean distance

OR

2. Distance function

$$d(x^i, x^j) \in \mathbb{R}$$

→ Parameters

1.  $k \equiv$  No. of clusters (taken as input)

2. Hierarchy of clusters

3. Correlation clustering :- 'signed' distances

\* Deduplication :- Remove duplicate addresses

If instances belong to same address → true distance  
different → -ve

- Based on clusters :-

Interested in:-

- 1) Centroid of each cluster

- 2) Partitioning data into clusters.

→ Objectives

1. Assign one of  $k$  clusters  $C_i(p)$  to each instance ' $i$ '

$$C_1 \cup C_2 \cup \dots \cup C_k = D$$

$$C_i \cap C_j = \emptyset$$

$$\text{Minimize} \quad \sum_{k=1}^K \left( \sum_{(i,j) \in C_k} d(x_i^k, x_j^k) \right) \quad \dots \text{pairwise distances in each cluster.}$$

2: Cluster centers  $m_1, \dots, m_k$

- Most popular distance function  $\equiv$  Euclidean

Algorithm. Begin at  $t = 0$

## K-MEANS ALGORITHM

$\underbrace{m_1^t \dots m_k^t}_{\text{representatives of each cluster.}} = \text{random } k \text{ points from } x^1 \dots x^N$

while (changing) {

for ( $i = 1$  to  $N$ ) {

$$c^{t+1}(x^i) = \underset{k \in \{1 \dots k\}}{\operatorname{argmin}} d(x^i, m_k^t)$$

for ( $k = 1$  to  $K$ ) {      $H$  undate  $\gamma_{\text{max}}^{(k)}$  in  $\gamma_{\text{min}}^{(k)}$

$$m_k^{t+1} = \min_m \sum_{x^i \in S_i} w(x^i, m)$$

// If d is Euclidean, then  $m_1^{(t+1)}$  will be b equal

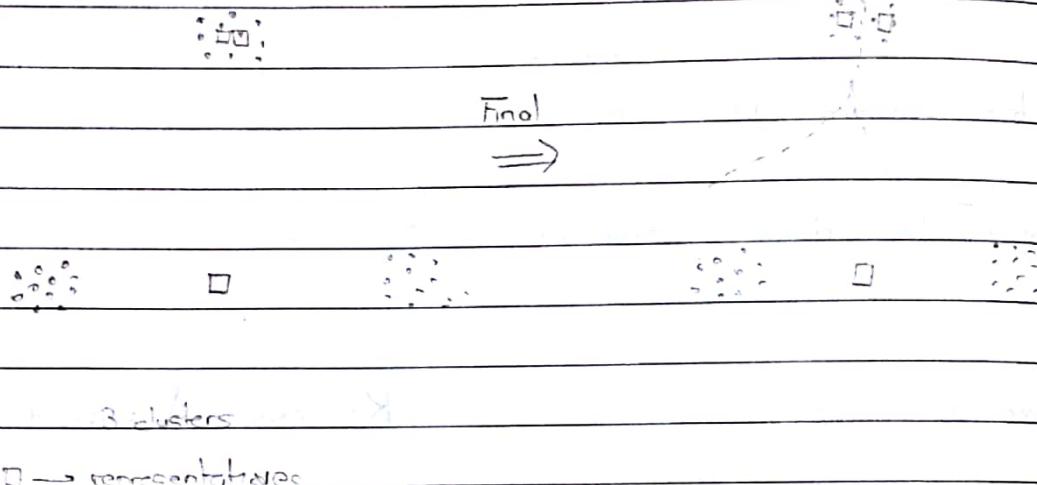
to mean of all points in that cluster.

## (Central Limit Theorem)

3

3

eg Failure :- Unoptimal clustering



- Solution :- Run multiple times, with different initial clustering

→ Proof of convergence of K-means algorithm

$$\text{Define } F(\{c(x_i)\}, \{m_k\}) = \sum_{k=1}^K \sum_{\substack{i=1 \\ c(x_i)=k}}^N d(x_i, m_k)$$

Step 1 :- By design,

$$F(\{c^{t+1}\}, \{m_k^t\}) \leq F(\{c^t\}, \{m_k^t\})$$

Step 2 :- By design,

$$F(\{c^{t+1}\}, \{m_k^{t+1}\}) \leq F(\{c^{t+1}\}, \{m_k^{t+1}\})$$

We will exit the while loop if ~~not~~ equality occurs.

∴ F is always decreasing.

∴ We will never repeat any iteration. ∴

∴ Converges to locally optimal solution

# A] → Probabilistic View of Clustering

## EM ALGORITHM

Each cluster is a distribution.

Given i) No. of clusters =  $k$

ii) Parametric form of distribution characterizing each cluster  
(e.g - Gaussian)

$$c_k = f(x^i, \theta_k) \equiv \text{density for cluster } k \\ \equiv N(x^i, \mu_k, \Sigma_k)$$

$$= \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2} (x^i - \mu_k)^T \Sigma_k^{-1} (x^i - \mu_k)}$$

$$3) D \equiv (x^1 \dots x^N)$$

$$\text{Mixture Distribution} \equiv P(x) = \sum_{k=1}^K \pi_k P(x|k) \quad \text{Proof on next page}$$

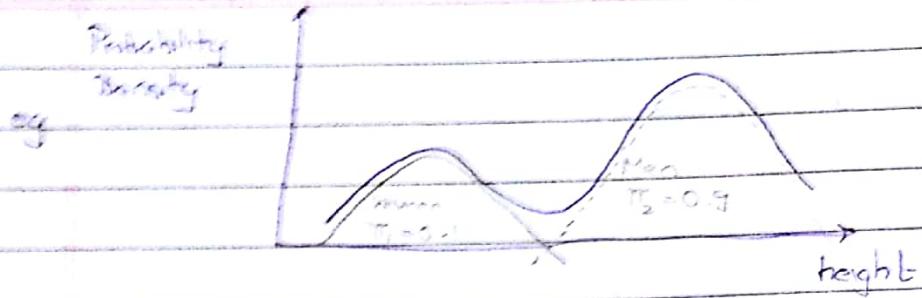
such that  $\pi_k > 0$ ,  $\sum \pi_k = 1$

$$= \sum_{k=1}^K \pi_k f(x^i, \theta_k)$$

Values of  $\pi_i$  and  $\theta_i$  for  $i \in \{1 \dots k\}$  have to be discovered during clustering by MLE estimation.

→ Training objective :- Maximize  $\sum_{j=1}^n \log P(x^j)$   
 $\pi_i, \theta_i$   
 for  $i \in \{1 \dots k\}$

P-T.O.



We want to come find out  $\pi_1, 2\pi_2$  and  $\theta_1, 2\theta_2$

This

$$\star \sum \log P$$

Objective Functions:-

Generative classifiers

Clustering

$$\sum_{(x^i, y)} \log P(x^i, y)$$

$$\sum_{x^i \in D} \log \sum_y P(x^i, y)$$

$$= P(x^i|y) P(y)$$

If we know labels

Class labels if not available

• Training

$$\max_{\theta_1, \theta_2, \pi_1, \dots, \pi_k} \sum_{i=1}^N \log \left( \sum_{g=1}^k \pi_g f(x^i, \theta_g) \right) \quad \dots \quad \text{not concave wrt parameters}$$

$\sum \pi_g = 1$

\* In contrast, for generative classifier,  
 $D \equiv \{(x^1, y^1), \dots, (x^N, y^N)\}$

$$\max_{\theta_1, \theta_2, \pi_1, \dots, \pi_k} \sum \log (\pi_g f(x^i, \theta_g))$$

If  $f$  was multivariate Gaussian ( $Bx^i, \mu, \Sigma$ ),

this function was concave in  $\pi_1, \dots, \pi_k, \mu_1, \dots, \mu_k, \Sigma, \dots, \Sigma_k$

HW Show that objective is not concave in  $(\mu_1, \mu_2)$

$\Rightarrow k=2, N=1, f = \text{Gaussian}, d=1, \sigma_1^2 = \sigma_2^2 = 1$

Hint -  $\log(e^{-g(\mu_1)} + e^{-h(\mu_2)})$  is not concave

- We will use a trick to rewrite the objective so that we get an efficient locally optimum solution.
  - We could have directly used gradient (like in deep learning) but it is not efficient

We know,  $\log w$  is concave in  $w$ .

$$\begin{aligned} \log \sum_{y=1}^k w_y &= \log \sum_{y=1}^k p_y \frac{w_y}{p_y} \quad \text{such that } \sum p_y = 1 \\ &\geq \sum p_y \log \left( \frac{w_y}{p_y} \right) \end{aligned}$$

(Jensen's inequality :- Convex combination of values of function is greater less than value of function at convex combination of data points)

- Apply Jensen's inequality on objective function

$$\sum_i \log \left( \sum_y z_{iy} \pi_y f(x^i, \theta_y) \right) = \sum_i \log \sum_y \frac{z_{iy}}{\sum_j z_{ij}} \pi_y f(x^i, \theta_y)$$

such that  $\forall i, \sum_y z_{iy} = 1$   
 $z_{iy} \geq 0$

$$\geq \sum_i \sum_y z_{iy} \log \pi_y f(x^i, \theta_y)$$

For fixed  $\{z_{iy}\}$ , this 'lower bound' is concave in  $\theta_1, \dots, \theta_k$  and  $\pi_1, \dots, \pi_k$

We will maximize this lower bound.

For best results, we have to choose  $\{z_{ij}\}$  that make this lower bound as close to original as possible.

Choose  $\{z_{ij}\}$  such that

$$\max_{\{z_{ij}\}} \sum_i \sum_j z_{ij} \log \frac{\pi_i f(x^i, \theta_j)}{z_{ij}} \quad \dots \text{concave in } \{z_{ij}\}$$

for fixed  $\{\theta_j, \pi_i\}$

$\sum z_{ij} = 1$

$z_{ij} \geq 0 \rightarrow$  use Lagrange to enforce this constraint

$$g_j = \sum_i z_{ij} - 1 = 0 \Rightarrow g_j = 1$$

$$\hat{L} = \sum_i \sum_j z_{ij} \log \left( \frac{\pi_i f(x^i, \theta_j)}{z_{ij}} \right) + \sum_j \lambda_j (\sum_i z_{ij} - 1)$$

$$\frac{\partial \hat{L}}{\partial z_{ij}} = \log \pi_i f(x^i, \theta_j) - \log(z_{ij}) - 1 + \lambda_j \\ = 0$$

$$\therefore z_{ij} = e^{\lambda_j - 1} \pi_i f(x^i, \theta_j)$$

$$\text{To eliminate } \lambda_j, \quad \sum_j z_{ij} = 1$$

$$\sum_j e^{\lambda_j - 1} \pi_i f(x^i, \theta_j) = 1$$

$$\therefore z_{ij} = \frac{\pi_i f(x^i, \theta_j)}{\sum_j \pi_i f(x^i, \theta_j)}$$

$$= \frac{P(y=1) P(x^i, y=1)}{\sum_i P(x^i, y=1)}$$

$$\therefore z_{iy} = P(y | x^i)$$

New objective :-

$$\max_{\theta_1, \dots, \theta_k, \pi_1, \dots, \pi_k} \sum_{i=1}^N \left[ \sum_{y=1}^k z_{iy} \log \frac{\pi_y f(x^i, \theta_y)}{z_{iy}} + \sum_i \lambda_i (\sum_y z_{iy} - 1) \right]$$

..... concave in  $\{z_{iy}\}$  for fixed  $\{\theta_i\}, \{\pi_i\}$   
 $\{\theta_i\}, \{\pi_i\}$                      $\{z_{iy}\}$

### "Expectation - Maximization (EM) Algorithm"

- \* Initially chosen arbitrary values for  $\{\theta_i\}, \{\pi_i\}, \{z_{iy}\}$   
 eg -  $\pi_i = \frac{1}{k} \forall i$ .

- \* while (change) {

E-step :-

for ( $i = 1$  to  $N$ ) {

    for ( $y = 1$  to  $k$ ) {

$$z_{iy}^t = P(y | x^i, \theta^t, \pi^t) \quad \dots \text{soft assignment of points to clusters}$$

$$= \frac{\pi_y^t f(x^i, \theta_y^t)}{\sum_{y=1}^k \pi_y^t f(x^i, \theta_y^t)}$$

M-step :- (maximization)

$$\{\pi^{t+1}\}, \{\theta^{t+1}\} = \arg \max \underbrace{\sum_i \sum_y z_{iy}^t \log \pi_y f(x^i, \theta_y)}_{\geq z_{iy}^t \text{ because max wrt } \theta_y} \triangleq Q^t(\theta, \pi)$$

only

P.T.O.

- First consider the  $\pi$  variables

$$\frac{\partial \theta^t}{\partial \pi_y} = \frac{\partial}{\partial \pi_y} \left[ \sum_i \sum_j z_{ij}^t \log(\pi_y f(x^i; \theta_y)) + \lambda (\sum_k \pi_k - 1) \right]$$

$$= \sum_i \frac{z_{ij}^t}{\pi_y} + \lambda = 0$$

$$\therefore \pi_y = - \frac{\sum z_{ij}^t}{N}$$

$$= \frac{\sum z_{ij}^t}{\sum_k \sum_i z_{ij}^t}$$

$$= \frac{\sum z_{ij}^t}{\sum_i \underbrace{\left( \sum_k z_{ij}^t \right)}_{=1}}$$

$$= \boxed{\frac{\sum z_{ij}^t}{N}}$$

- For the  $\theta$  variables,

For instance,  $f$  is Gaussian  $\frac{1}{(2\pi)^{d/2} |\Sigma|} e^{-\frac{1}{2} (x^i - \mu_y)^T \Sigma^{-1} (x^i - \mu_y)}$

$$f(x^i; \theta_y) = \frac{1}{(2\pi)^{d/2} |\Sigma|} e^{-\frac{1}{2} (x^i - \mu_y)^T \Sigma^{-1} (x^i - \mu_y)}$$

$$\rightarrow \frac{\partial \theta^t}{\partial \mu_y} = \frac{\partial}{\partial \mu_y} \left( \sum_i \sum_j z_{ij}^t \left( \log \pi_y - \frac{1}{2} (x^i - \mu_y)^T \Sigma^{-1} (x^i - \mu_y) - \log |\Sigma| \right) \right)$$

$$= - \sum_{i=1}^N z_{ij}^t (x^i - \mu_y) = 0$$

$$\mu_y^{t+1} = \frac{\sum_i z_{ij}^t x^i}{\sum_i z_{ij}^t}$$