DEEP LEARNING

PAGE NO.
DATE / /

I 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 $y = f^*(x) = x_1 \oplus x_2$

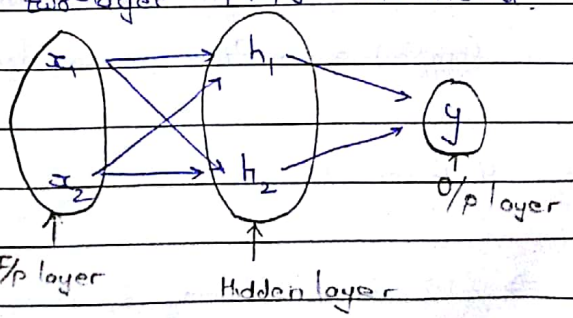
Training data has all four combinations of x_1, x_2

- Linear classifier $\hat{y} = \omega_1 x_1 + \omega_2 x_2 + b$, trained with least square loss function, gives $\omega_1 = \omega_2 = 0$, $b = \frac{1}{2}$

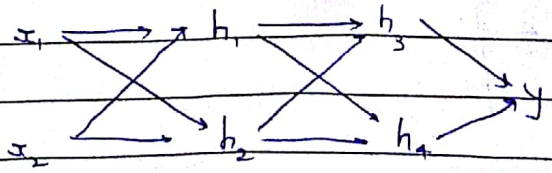
* $\sum ((y - \omega_1 x_1 - \omega_2 x_2 - b)^2)$ is convex in ω_i's

- Thus, linear classifier cannot discriminate.

- Use a two-layer FFN with ReLu.



- Three-layer FFN



TENSORFLOW TUTORIAL

- Good for fast computation

→ Installation

- Tensor Flow
- Google Colab
  - Edit → Notebook Settings → Hardware Acceleration → GPU

→ Tensor

- Multidimensional arrays

- Computational Graph

- Example program.
  * ~~Dimension = (3,) ⇒ 1×3~~
  - 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,) ⇒ 1D tensor having 3 values
  Shape = (3, 1) ⇒ 1D tensor having 3 single-valued tensors
  eg $[1,2,3]_{(3,)} + [[1],[2],[4]]_{(3,1)}$

$$= \begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 5 & 6 & 7 \end{bmatrix}_{(3,3)}$$

PAGE NO.
DATE / /

→ 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

eg $y = Ax + b$

A: Variable, b: Placeholder

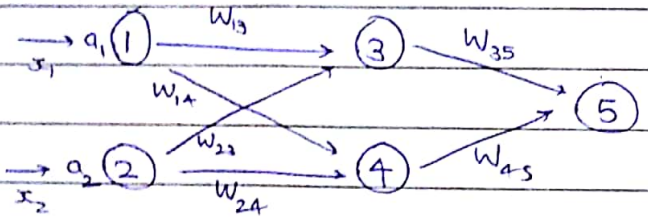
- Variables need to be declared with an initial value.
  The variable initializer needs to be run before further computations.

• Variable updation

- Variable is updated only after a session is run.

19/9 FFN continued

→ Parameterized Model



Bias terms (b_i) not shown

Every node could be a logistic classifier.

$a_3 = x_3 = g(W_{13}x_1 + W_{23}x_2 + b_3)$ where g is some function (TFT)

$x_4 = g(W_{14}x_1 + W_{24}x_2 + b_4)$

$x_5 = W_{35}x_3 + W_{45}x_4 + b_5$

- 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 $\vec{h}$, then

$$\underset{m\times 1}{\vec{h}} = \underset{m\times d}{W^T}\underset{d\times 1}{\vec{x}} + \underset{m\times 1}{\vec{b}}$$

- The value of m, and no. of hidden layers, are decided by designer.

- Simplest neural network is the perceptron

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

- A one-layer FFN has a form.

$$\text{MLP}(\vec{x}) = g(W^T\vec{x} + \vec{b})W_2 + \vec{b}_2$$

P.T.O.

PAGE NO.
DATE / /

A] Activation Function (g)

For one-layer FFN, $MLP(x) = g(xW_1 + b_1)W_2 + b_2$

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

Proof Let $g(z) = z$, $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $W_2 = \begin{bmatrix} W_{21} \\ W_{22} \end{bmatrix}$, $W_1 = \begin{bmatrix} W_{11} & W_{12} \\ W_{13} & W_{14} \end{bmatrix}$, $b_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $b_2 = 0$

$$g(xW_1 + b_1)W_2 + b_2$$

$$= (xW_1 + b_1)W_2 + b_2$$

$$\vdots$$

$$= W_{11}W_{21}x_1 + W_{13}W_{21}x_2 + \cdots \quad \text{(linear classifier)}$$

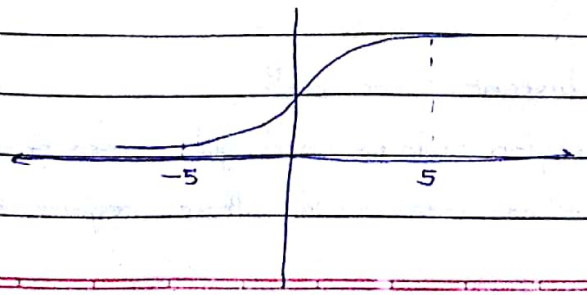
- We should choose a $g(z)$ that is efficient to compute, easy to optimize (should have an 'informative' gradient), almost linear

↓

Hidden layers should not lose characteristics of input

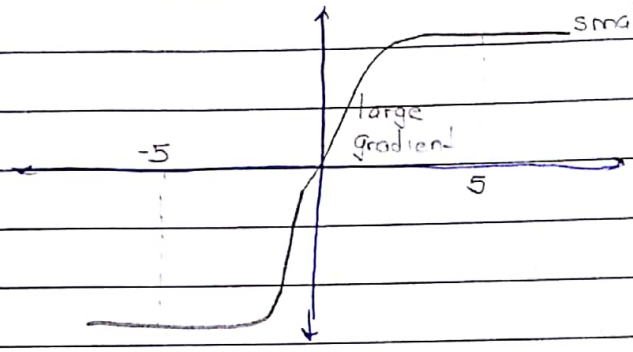
A Common Activation Functions.

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]

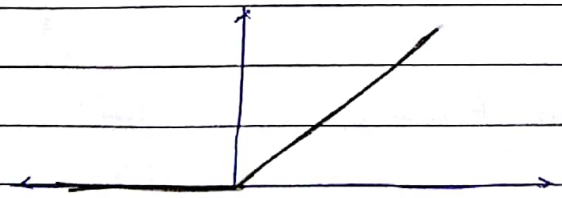
b Hyperbolic Tangent :- $\tanh = \frac{e^{2x} - 1}{e^{2x} + 1}$



- 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 are on either side

c RElectified Linear Unit :- $RELU(x) = \max(0, x)$

The RELU(0) = 0



☹ 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.

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

PAGE NO.
DATE / /

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

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

- Say, there is one-dimensional one hidden layer $\therefore h = g(\omega_1 x)$

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

Loss function :- $J(\omega_1, \omega_2, x, y) = L(\vec{h}\omega_2 y) = L(g(\omega_1 x)\omega_2 y)$

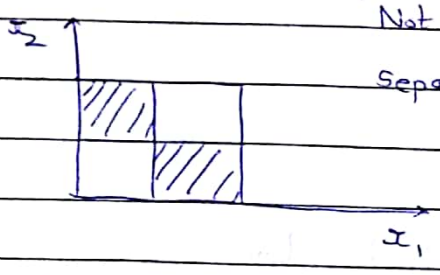
Gradient wrt ω_1 = $L' \omega_2 y g' x$

↑ Saturated if $g' = 0$

eg XOR

Neural networks can train decisions that conventional linear classifiers cannot.

$y = x_1 \oplus x_2$



- Linear classifier $\hat{y} = \omega_1 x_1 + \omega_2 x_2 + b$ with least square loss gives $\omega_1 = 0,\ \omega_2 = 0,\ b = \frac{1}{2}$

Useless

- Non-linear functions like $\hat{y} = \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_1 x_2 + b$ can work, but the designer will have to think which non-linear term to use.

• General one-layer FFN with ReLU

$$y = f(x) = W^2 \underbrace{\max(0, W^1 x + b^1)}_{g(W^1 x + b^1)} + b^2$$

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

- $[h_1, h_2] = [x_1, x_2] \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + [0 \quad -1]$

$= [x_1 + x_2 \quad x_1 + x_2 - 1]$

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

--- Element-wise max.

1) Put $[x_1 \; x_2] = [0 \; 0] \Rightarrow W^2 h + b^2 = 0$ ✓

$\max([0\,0], [0\,-1])$

$= [0\,0]$

2) Put $[x_1, x_2] = [1 \; 1] \Rightarrow W^2 h + b^2 = 0$ ✓

B] Output Layers

1° Depends on type of output

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

2) Multiclass class labels :- Softmax converts real numbers to 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

Input → h_1 → h_2 → Softmax → Prediction

For every class y_k, we have a set (w_k^o, b_k^o) $k \in \{1 \dots K\}$ → Parameters of output layer

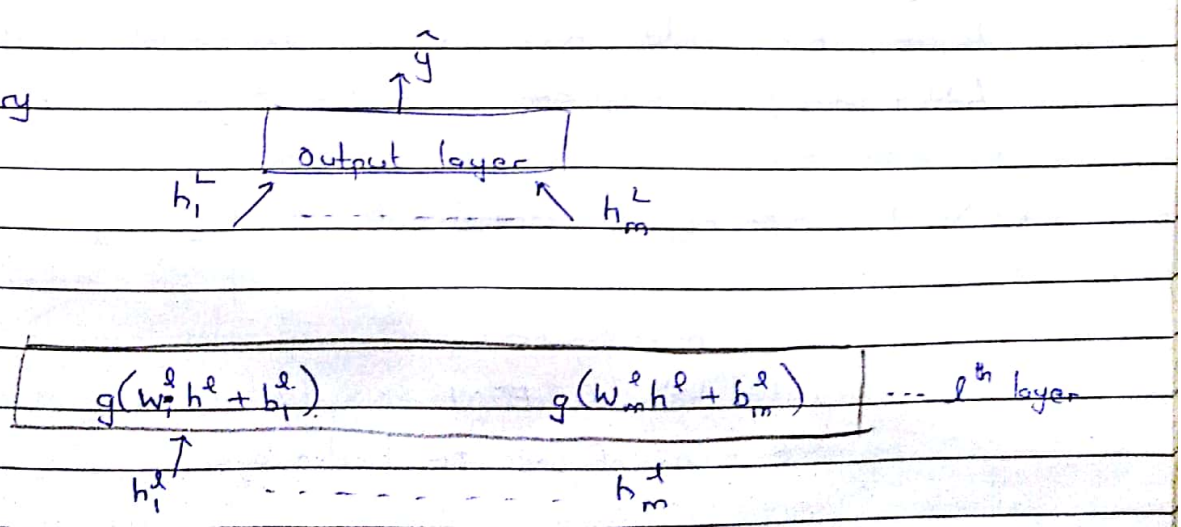
Softmax :- $P(y|x) = \frac{e^{w_y^o h_2 + b_y^o}}{\sum_y e^{w_y^o h_2 + b_y^o}}$

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

- Logit $(y) \triangleq w_y^o h_2 + b_y^o$

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

26/9 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 approx any continuous function from a closed and bounded set given enough hidden units.

- Works for ReLU activation as well.

☹ Not practical

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

A Effect of depth

- Many functions can be efficiently represent with multiple hidden layers, but would have required exponential width (no. of hidden units) with single hidden layer.
- No. of linear regions carved out $\sim O\left[\left({}^{c}C_{d}\right)^{dl} c^{d}\right]$ ★ Verify

d = no. of inputs

$l+1$ = depth of FFN

c = no. of units per hidden layer

PAGE NO.
DATE / /

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

D] Training an FFN

1. Define loss function $L(y, \tilde{y})$ :: Perceptron, hinge, logistic, etc.

(y: true value, $\tilde{y}$: predicted value.)

- $L(y, \tilde{y})$ gives non-negative score to the FFN
- L is minimized by gradient descent
- 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 $\forall i$

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 $NN(x|\theta)$

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 ∇L wrt θ

$\theta = \theta - \mu \nabla L$

}

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

↖ 'Learning rate'

Output :- θ.

- 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_ω is the node which uses weight 'ω' ← each of the weights for different inputs of that node

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

- Use chain rule :- $\frac{\partial L}{\partial \omega} = \frac{\partial L}{\partial h} \times \frac{\partial h}{\partial \omega}$

Compute recursively

easy, we know $h = g(\dots)$

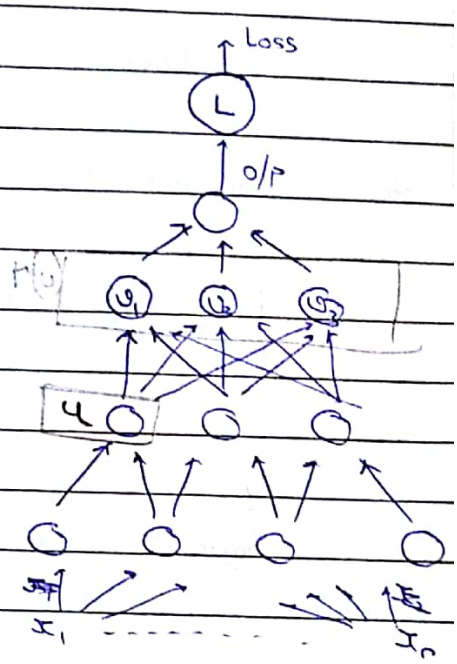
PAGE NO:
DATE: / /

- Chain rule on multiple variables

If L is a function of $v_1 \dots v_n$, which depend on variable u,

then $$\frac{\partial L}{\partial u} = \sum_i \frac{\partial L}{\partial v_i} \frac{\partial v_i}{\partial u}$$

→ Algorithm.



r(u) = All nodes taking u as input

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:- $\partial L / \partial \omega$

using :- $$\frac{\partial L}{\partial u} = \sum_{v \in r(u)} \frac{\partial L}{\partial v} \cdot \frac{\partial v}{\partial u}$$

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

for (each u from top to bottom) :-

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

Compute $\partial v / \partial u$.

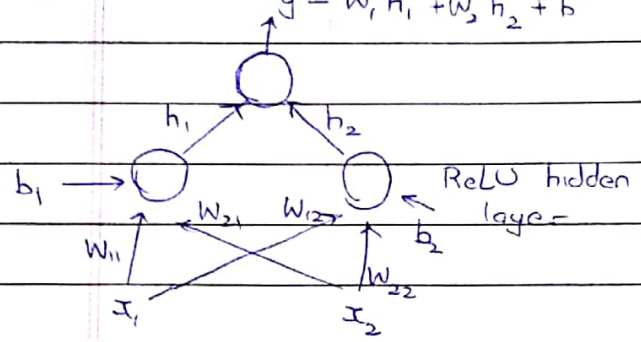
// Inductively, we have already calculated $\partial L / \partial v$

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

Compute $$\frac{\partial L}{\partial \omega} = \frac{\partial L}{\partial u} \cdot \frac{\partial u}{\partial \omega}$$

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

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

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

$h_2 = W_{12}\ W_{22}\ b_2 = -1 = 0$

$\hat{y} = W_1^0 h_1 + W_2^0 h_2 + b^0 = 0$

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

Step 2 Back Propagation

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

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

$$\frac{\partial L}{\partial W_1^0} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial W_1^0} = -2 h_1 = -2$$

Update W_1^0, W_2^0 similarly & b^0

2. $\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial h_1} = -2 W_1^0 = 0$

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

$= 0$

Similarly update W_{11} & W_{21} & b_1

3 Like step 2, find $\frac{\partial L}{\partial h_2}$, $\frac{\partial L}{\partial w_2}$, $\frac{\partial L}{\partial w_{22}}$, $\frac{\partial L}{\partial b_2}$ - Update w_{12}, w_{22}, b_2

F] Regularization

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

- Objective function = loss + α regularizer

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 - Sentiment 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/p :- $x_1 \dots x_{at}$

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.

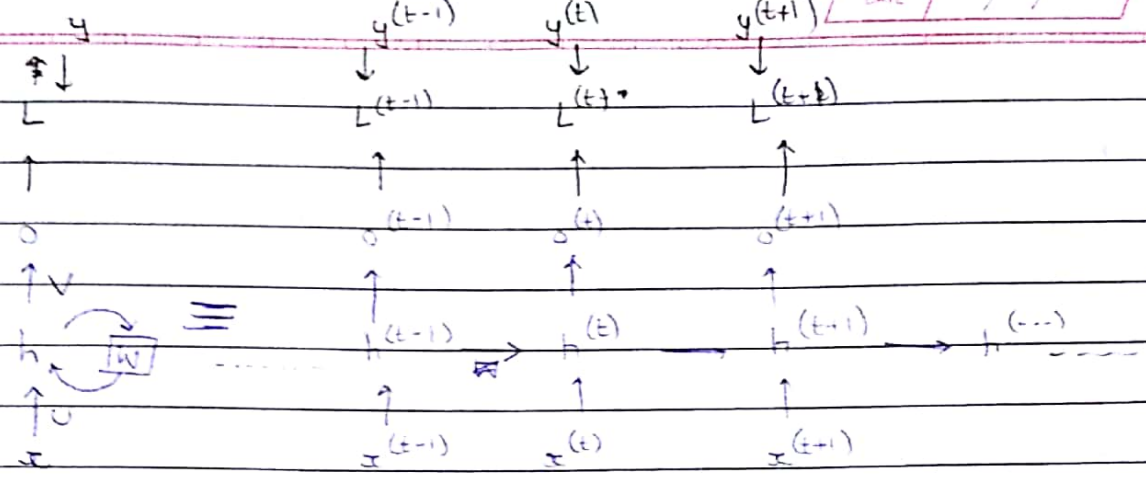
RNN

- 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.



$o^t = c + Vh^t$

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

↑ Sigmoid (convention)

* The end of sequence is represented by a 'token' 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 $h \in \mathbb{R}^4$, $\Rightarrow W \in \mathbb{R}^{4\times4}$, $V \in \mathbb{R}^{4\times1\times4}$, $U \in \mathbb{R}^{4\times1}$

$h \in \mathbb{R}^{4\times1}$, $c \in \mathbb{R}^{4}$

Consider $h^0 = 0$

Calculate $h^1 = \sigma(Wh^0 + Ux_1^1 + b)$

$o^1 = Vh^1 + c$

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 $\in [1 \quad \ldots \quad V]$

V = 'Vocabulary Size' $\approx$ 30000

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

Embedding matrix $S_{V \times d}$

where $d \sim 300$

Define ~~S_{x_t}~~ $S_{x_t} = x_t S$ = vector embedding of that word

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 corresp h^t

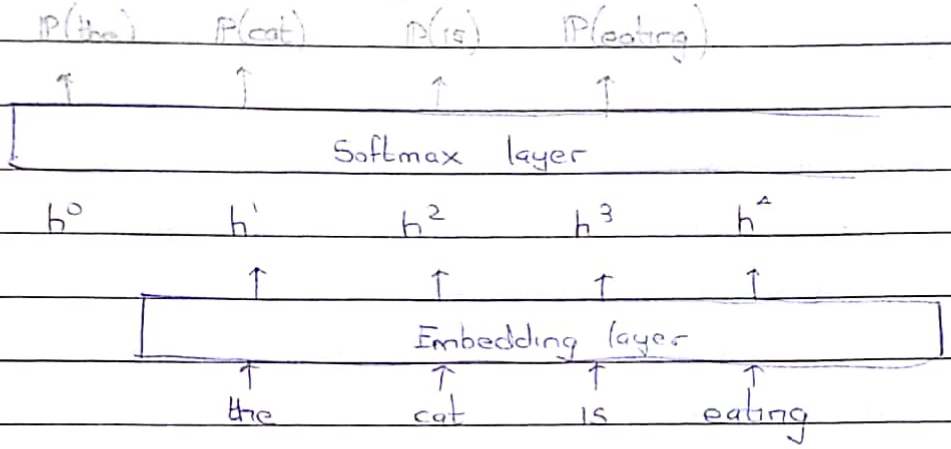
* Words of similar meaning are placed closer in the (standardized) 300-dimensional space

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

→ Training a sequence model

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

eg $\mathbb{P}$ (the, cat, is, eating)



- Training data

$D \equiv \{(x^1, y^1) \cdots (x^N, y^N)\}$

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

Training data :- $\vec{x}$:- The cat is ~~steep~~ eating

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

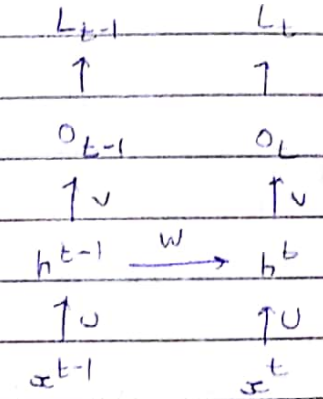
↳ end-of-sequence.

→ Training RNN parameters

See following example

→ Backpropagation through time

Total loss = $L = L_{t-1} + L_t$ ... add loss at all time



$$-\frac{\partial L}{\partial V} = \frac{\partial L_{t-1}}{\partial V} + \frac{\partial L_t}{\partial V} \quad \cdots$$

$$= \frac{\partial L_{t-1}}{\partial o_{t-1}} h^{t-1} + \frac{\partial L_t}{\partial o_t} h^t \quad \cdots$$

$$-\frac{\partial L}{\partial W} = \frac{\partial L_{t-1}}{\partial W} + \frac{\partial L_t}{\partial W}$$

$$= \frac{\partial L_{t-1}}{\partial o_{t-1}} \times \frac{\partial o_{t-1}}{\partial h_{t-1}} \times \frac{\partial h^{t-1}}{\partial W} + \cdots$$

$$= \frac{\partial L_{t-1}}{\partial o_{t-1}} \times V\sigma'\left(h^{t-2} + W\frac{\partial h^{t-2}}{\partial W}\right) + \cdots$$

This will have all terms till $\frac{\partial h^{t-1}}{\partial W}$ with coefficient W^t (can explode or vanish)

∴ We have to go back over all h^n to compute gradient

'Exploding or Vanishing Gradient'

Gradient is either small or large

PAGE NO.
DATE / /

- 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)

8/10

8] ⟶ Sequence Prediction

$\vec{x} = (x_1 \cdots x_n) \longrightarrow \vec{y} = (y_1 \cdots y_n)$

eg - sentences, image, audio

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

eg Translation :- <English sentence> ⟶ <Hindi sentence>

eg Image captioning :- <Image> ⟶ <Describing sentence>

eg Conversation assistance

eg Speech recognition :- <Speech spectrogram> ⟶ <Phoneme sequence>

- Challenges

1) Long range dependency
   - Does not assume conditional independence
   eg Ram eats. He sings.

2) Highly open-ended prediction space
   - Even length of output is variable

When argmax $\mathbb{P}(y_t | \dots) = EOS$, stop the sentence.

A Encoder - Decoder Model

1. Encode input elements into a fixed-dimension vector X. (embedding)
2. Decode $\vec{y}$ 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

While $\mathbb{P}(y_t | y^{(t-1)}, \vec{z}, X, \theta) = \mathbb{P}(y_t | h_t, \theta)$

↑ state vector implemented by RNN

with $h_0 = X$.

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

Stop RNN if token = EOS

- We are doing greedy search :- Word prediction #n does not depend on word # k for $k > n$

→ 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

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

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

'Attention-weighted input states'

$\Sigma a_{tj} h_j$ 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
   $d(x^i, x^j) \in R$

→ 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 → +ve 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 $C_i(i)$ to each instance 'i'

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

$$C_i \cap C_j = \emptyset$$

Minimize $\sum_{k=1}^{t} \left(\sum_{(i,j) \in C_k} d(x^i, x^j) \right)$ ---- pairwise distances in each cluster.

2. Cluster centers $m_1 \dots m_k$

- Most popular distance function ≡ Euclidean

K-MEANS ALGORITHM

Algorithm Begin at $t = 0$

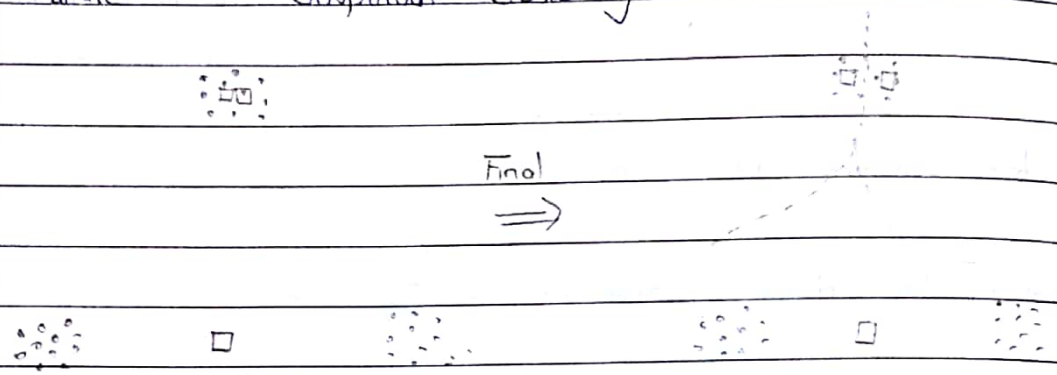
$m_1^t \dots m_k^t$ ≡ random k points from $x^1 \dots x^N$

// representative of each cluster

$c^t(x^i)$ → randomly from $1 \dots k$

```
while (changing) {
    for (i = 1 to N) {
        c^{t+1}(x^i) = argmin_{k ∈ {1...k}} d(x^i, m_k^t)
    }
    for (k = 1 to k) { // update representatives
        m_k^{t+1} = min_m Σ_{x^i in C_i} d(x^i, m)
        // If d is Euclidean, then m_k^{t+1} will be equal
        to mean of all points in that cluster
        (Central Limit Theorem)
    }
}
```

eg Failure :- Unoptimal clustering



3 clusters

□ → representatives

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

→ Proof of convergence of K-means algorithm

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 equality occurs.

∴ F is always decreasing.

∴ We will never repeat any iteration. ☺

∴ Converges to locally optimal solution

PAGE NO.
DATE / /

A] → Probabilistic View of Clustering — EM ALGORITHM

Each cluster is a distribution.

Given 1) No. of clusters = k

2) Parametric form of distribution characterizing each cluster
(eg - Gaussian)

eg - $f(x^i, \theta_k) \equiv$ density for cluster k

$\equiv N(x^i, \mu_k, \Sigma_k)$

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

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

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

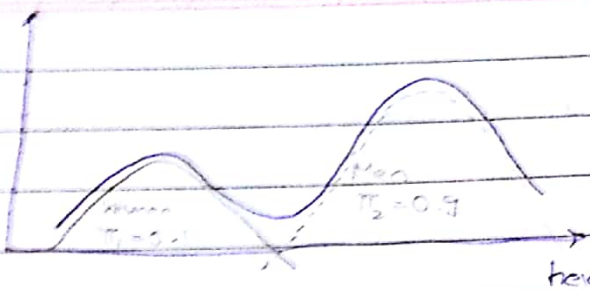
such that $\pi_k \geq 0$, $\sum \pi_k = 1$

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

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

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

P.T.O.



We want to find out π_1 & π_2 and θ_1 & θ_2

Note

* $\sum \log \mathbb{P}$

Objective functions:-

Generative classifiers	Clustering
$\sum_{(x^i, y^i) \in D} \log \mathbb{P}(x^i, y^i)$	$\sum_{x^i \in D} \log \sum_y \mathbb{P}(x^i, y)$
	$= \mathbb{P}(x^i \mid y)\, \mathbb{P}(y)$
	Class labels (y) are not available

- Training

$$\max_{\theta_1, \ldots \theta_k, \pi_1, \ldots \pi_k} \sum_{i=1}^{N} \log \left(\sum_{y=1}^{k} \pi_y f(x^i, \theta_y) \right) \quad \text{..... not concave wrt parameters}$$

$$\sum \pi_i = 1$$

☹

* In contrast, for generative classifier,

$$D \equiv \{(x^1, y^1) \ldots (x^N, y^N)\}$$

$$\max_{\theta_1 \ldots \theta_k, \pi_1 \ldots \pi_k} \sum \log (\pi_y f(x^i, \theta_y))$$

If f was multivariate Gaussian $(\forall x^i, \mu, \Sigma)$, this function was concave in $\pi_1 \ldots \pi_k, \mu_1 \ldots \mu_k, \Sigma_1 \ldots \Sigma_k$

PAGE No.
DATE / /

HW Show that objective is not concave in (μ_1, μ_2)
for $k=2$, $N=1$, f = Gaussian, $d=1$, $\sigma_1 = \sigma_2 = 1$

Hint – $\log\left(e^{-g(\mu_1)} + e^{-h(\mu_2)}\right)$ 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, $\log \omega$ is concave in ω.

$$\log \sum_{y=1}^{k} \omega_y = \log \sum_{y=1}^{k} p_y \frac{\omega_y}{p_y} \quad \text{such that } \sum p_i = 1 \text{ and } p_i \geq 0$$

$$\geq \sum p_y \log\left(\frac{\omega_y}{p_y}\right)$$

(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

$$\sum \log\left(\sum \underbrace{\pi_y f(x^i, \theta_y)}_{\omega_y}\right) = \sum_i \log \sum_y \boxed{z_{iy}}^{p_y} \frac{\pi_y f(x^i, \theta_y)}{z_{iy}}$$

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

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

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 $\{z_{iy}\}$ that make this lower bound as close to original as possible.

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

$$\max_{\{z_{iy}\}} \sum_i \sum_y z_{iy} \log \frac{\pi_y f(x^i, \theta_y)}{z_{iy}} \quad \text{.... concave in } \{z_{iy}\} \text{ for fixed } \{\theta_y, \pi_y\}$$

$$\sum_y z_{iy} = 1, \quad z_{iy} \geq 0$$

→ use Lagrange to enforce the constraint

eq - $\sum_y z_{iy} = 1$ $\forall i$

$$\tilde{L} = \sum_i \sum_y z_{iy} \log \left(\frac{\pi_y f(x^i, \theta_y)}{z_{iy}} \right) + \sum_i \lambda_i \left(\sum_y z_{iy} - 1 \right)$$

$$\frac{\partial \tilde{L}}{\partial z_{ij}} = \log \pi_j f(x^i, \theta_j) - \log(z_{ij}) - 1 + \lambda_i$$

$$= 0$$

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

To choose λ_i, $\sum_j z_{ij} = 1$

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

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

$$= \frac{P(y=j)\, P(x^i, y=j)}{\sum (\quad)}$$

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

New objective :-

$$\max_{\theta_1 \dots \theta_k, \pi_1 \dots \pi_k} \; \max_{\substack{z_{iy} \geq 0 \\ \sum_y z_{iy} = 1 \\ \{z_{iy}\}}} \sum_{i=1}^{N} \sum_{y=1}^{k} \left[z_{iy} \log \frac{\pi_y f(x^i, \theta_y)}{z_{iy}} + \sum_i \lambda_i \left(\sum_y z_{iy} - 1 \right) \right]$$

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

"Expectation - Maximization (EM) Algorithm"

- Initially chosen arbitrary values for $\{\theta_i\}\{\pi_i\}\{z_{iy}\}$
  eg - $\pi_i = \frac{1}{k} \quad \forall i$.
- while (change) {
  E - step :-
  for (i = 1 to N) {
  for (y = 1 to k) {

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

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

M-step :- (maximization)

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

$\sum z_{iy}^t$ because max w.r.t θ & π only

P.T.O.

- First consider the π variables

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

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

$$\therefore \pi_y = \frac{-\sum z_{iy}^t}{\lambda}$$

$$= \frac{\sum_i z_{iy}^t}{\sum_k \sum_i z_{iy}^i}$$

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

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

- For the θ variable,

For instance, f is Gaussian $\frac{1}{2}(x^i-\mu_y)^T \Sigma_y^{-1}(x^i-\mu_y)$

$$f(x^i, \theta_y) = \frac{1}{(2\pi)^{d/2}|\Sigma_y|} e$$

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

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

$$\mu_y^{t+1} = \frac{\sum_{i=1}^{N} z_{iy}^t x^i}{\sum_i z_{iy}^t}$$