CSE 425

Rajin Ibna Rajuanur Rahman

Perceptron

* Neuron যাদেরকে control করতে পারে সেগুলোই Neuron এর Parameter.

* আমি Neuron কে যা Provide করলো যা control mechanism কে help করবে সেগুলোকে বলা হয় Hyperparameters

Example → Weight – Neuron ইচ্ছামতো weights generate করতে পারে তাই weights একটি Parameter. এখন ধরি এই weights এর value কখনও 100 এর বেশি cross করতে পারবে না. এই 100 এর বেশি যে cross করতে পারবে না. এটা neuron কে বুঝানো Hyper-parameter এর কাজ।

∴ Hyperparameter neuron কে control করে আর neuron যাদেরকে control করে তারা হচ্ছে Parameters.

* Transfer function → কোন function কে আরেকটা state এ transfer করা, সেই function এর মাধ্যমে এটা হয় সেটাকে বলা হয় transfer function.

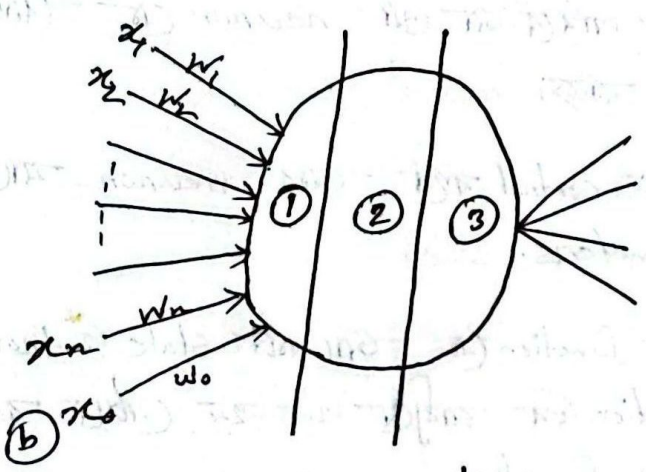
ধরি, input and weights আমরা external হিসেবে, এদেরকে process করে neuron এর ভেতরে Properly information pass করার procedure কোনো হচ্ছে transfer function.

* Compartment 1 এ weighted sum compute হবে এবং সেটাকে transfer করে compartment 2 তে নিয়ে যাওয়া হয়,

Weights and inputs গুলো multiply করা হয়, তখন এর impect factor টা বুঝা যায়, Positive কে Positive দিয়ে multiply করলে +ve হয়, Positive কে negative দিয়ে multiply করলে value কমানোর চেষ্টা করবে, -ve and -ve multiply করলে +ve হয়ে যাবে, তাহলে আবুমাত্র +ve and -ve ই যখন হবে তখনই weighted sum এর value reduction হতে পারে। Negative input কে negative weight দিলে তার importance টা আরও কমিয়ে দেওয়া হয়, এই সবকিছু কাজে control করে transfer function. তারপর এই সমস্ত কিছু

transfer বায়া হয় Activation function ও।

* Activation function অনেকটা security guard এর মতো, এটাও সমস্ত calculation এর condition check না করে Activation function information গুলোকে compartment-3 তে যেতে দিবে না, 3no compartment ও যদি যেতে না দেয় তাহলে 3 no compartment থেকে যে outcome টা পাবো সেটা zero. তারমানে Neuron activate হয় নাই, Neuron Activate হলে Outcome একটা Non-zero value থাকে,



Neuron – j

3 major components

(1) Transfer function / weighted sum function

Weighted sum $V_j = \sum_{i=1}^{n} x_i w_i$

i → num of inputs

V_j ← Value of jth neuron

(2) Activation function

$$\phi(V_j) = \begin{cases} 1; & \sum_{i=1}^{n} x_i w_i \geq \theta_j \\ 0; & \sum_{i=1}^{n} x_i w_i < \theta_j \end{cases}$$

If transfer function includes bias,

$V_j = \sum_{i=1}^{n} x_i w_i + b_j$

আমরা Perceptron এর ক্ষেত্রে – bias is inverse of threshold

IF Threshold = 0, Bios কে তার Threshold হিসাবে consider করতে হবে।

$b_j = x_0 w_0$ ← bias

$\therefore V_j = \sum_{i=0}^{n} x_i w_i$

* প্রতি neuron এ একটাই Threshold এবং একটাই bias থাকে, এরা control হয় neuron এর দ্বারা

(3) Output function

$y_j = \phi(V_j)$ ← Weighted sum

Activation function

Data set কে Excel sheet এ represent করা হয়, যেই Data এ যেই columns গুলো থাকে সেখানে অনেক গুলো category এর values গুলোকে বাদ দেয়া হয়, এগুলোই হচ্ছে neuron এর inputs x. সেখানে আমরা importance দিয়ে কিছু x কে include করি, আবার কিছু x কে বাদ দিয়ে দেই,

Excel sheet এর একদম last column এ যদি output টা বলে দেওয়া থাকে সেই Data set কে বলা হয় supervised data set.

*Neuron গুলো একেকটি row অনুসারে learn করতে থাকে থাকে।

*Normally column গুলোকে একটা করা হয়, আর rows গুলো এর value গুলোকে neitherate করতে থাকে।

* One epok → আমার যে training example থাকবে, সেই training examples এর starting থেকে আবার কয়বার ending পর্যন্ত full revision একবার দেওয়াকে বলা হয় one epok.

* এক epok থেকে next epok এ যেতে shaffling করে নিতে হয়। কারণ শেষবার revision দেওয়ার style আর তার পরের বার revision দেওয়ার style same হবে না, এই process গুলো neuron এর ভেতরে হয় না, Training টা শেষ হচ্ছে কিনা বের test টাকে evaluate করতে হয়। Training টা শেষ এবং এই শেষানোর জন্য epok এর process টা চলে।

*Training proper way তে যাচ্ছে কিনা, এটা check করার জন্য আমরা validation করে থাকি।

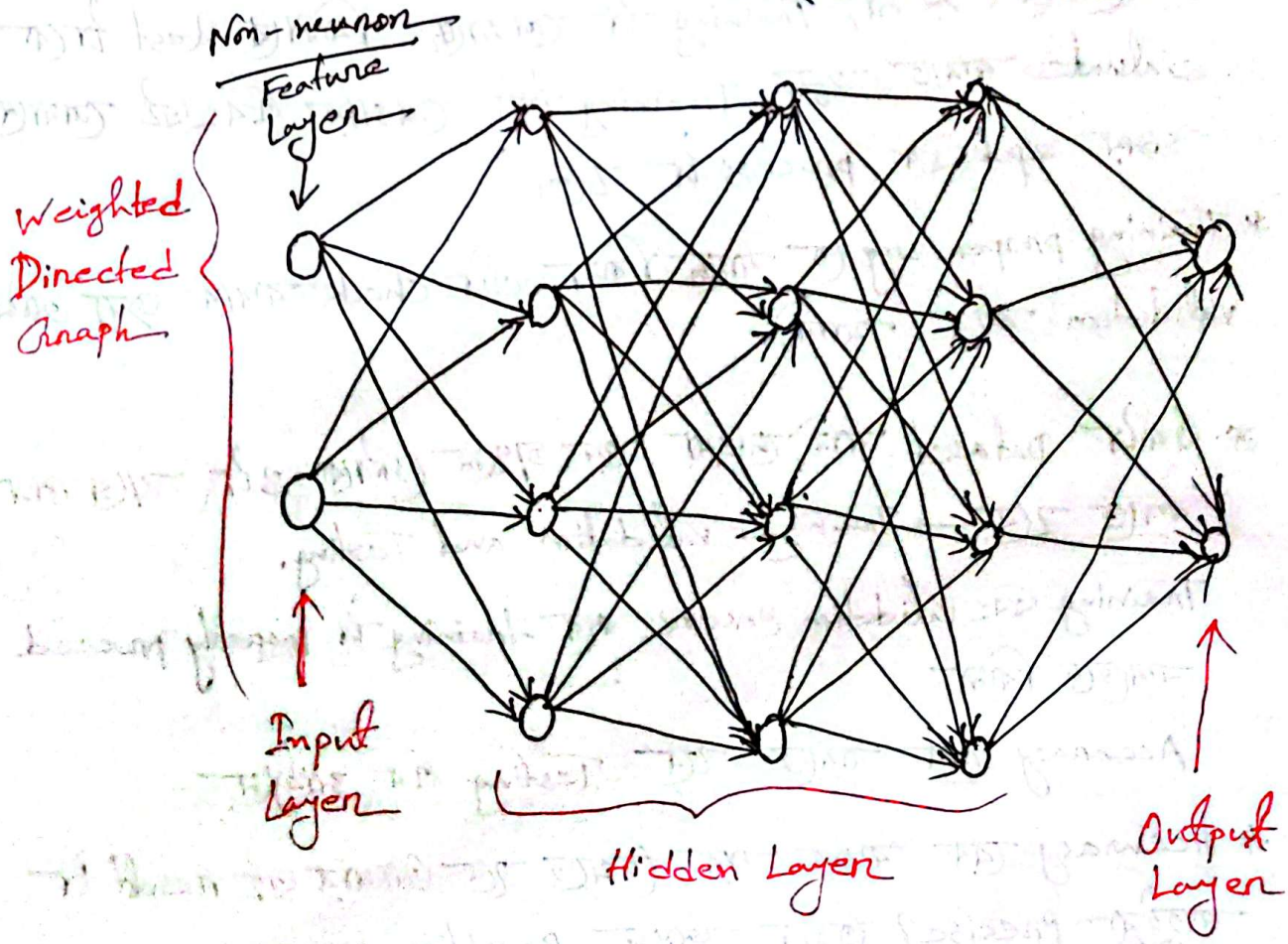
* একটা Dataset যদি দেওয়া হয় তাহলে সেটাকে ৩ টা ভাগে ভাগ করতে হবে → Training, validation and Testing.
Training এবং Validation ensure করে training টা properly proceed করছে কিনা,
Accuracy বের করতে হবে Testing এর মাধ্যমে।

* Accuracy বের করার পর দেখতে হবে আমার এই result টা কতটুকু precise? তখন আসে prediction.

* বতগুলোতে Pit falls বারবিল সেই ক্ষেত্রে আমরা recall
* তারপর আমরা statistical measurement score (Accuracy, precision এবং Recall এর উপর base করে), তাকে বলা হয় F1 - Measurement.
* তারপর আমরা Explainable AI. আমি যেটাকে ভালো বলতেছি আমার use করা model সেটাকে ~~ভালো~~ বলবে? কিভাবে ভালো বলবে, model নিজে explain করবে, ঐ particular accuracy ও কিভাবে আনছে?
* উপরোক্ত এইসবকিছু Perception perform করবে.

Parameters → Bias, Weights, Threshold

একটা Neuron এ একটা Bias এবং একটাই Threshold



How many parameters are there?

① This is a fully connected neural network

Each neuron of the next layer is connected with every other neuron of its previous layer: No connection in own layer's neuron.

If fully connected

Input layer	2	(not considering as neurons)
H_1	4	neurons
H_2	4	neurons
H_3	4	neurons
Output layer	2	neurons

1 bias and 1 Threshold at each neuron.

So, 4+4+4+2 → 14 biases
14 Thresholds

So for parameters = 14+14 = 28

What about weights?

$(4\times2)_{H_1\ \text{input}} + (4\times4)_{H_2\ H_1} + (4\times4)_{H_3\ H_2} + (2\times4)_{\text{output}\ H_3}$

= 48

∴ Total parameters = 48 + 28 = 76

Matrix ~~array~~ ও 76 টি variables ~~ধরে~~ store করার হয়।
Matrix operations is much faster than normal variable operations

* যদি বলা হয় Neural network four layer, তাহলে 5 টি layer বোঝাতে হবে। Input layer বাদে বাকি 4 টি layer এ neuron থাকবে।

5 layers of neural networks, including inputs.

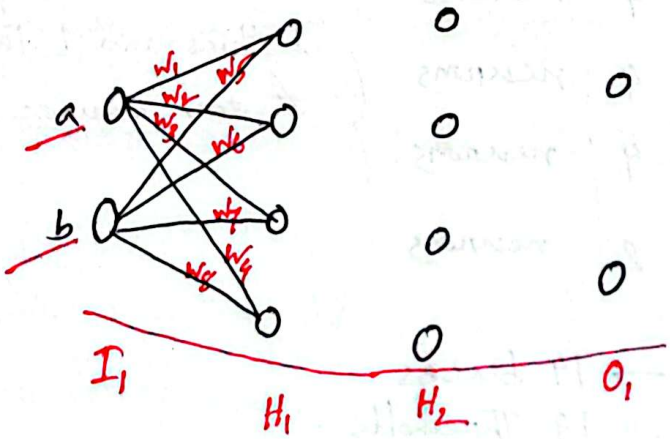
Row এর dimension 1 → Row vector

Column —" — " 1 → Column vector

Matrix representation এর main target computation faster করা

3 টা Matrix যা দিয়ে neuron control করা → Weight matrix,
Threshold matrix and
Bias matrix

Matrix Representation (Fully connected NN)



Z_i^l ← lth layer, ith neuron

Weighted sum

H_1

Layer, Num of neuron → H_{11} H_{12} H_{13} H_{14}

$$\begin{matrix} a \\ b \end{matrix}\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_5 & w_6 & w_7 & w_8 \end{bmatrix} \xrightarrow{\text{Transpose}} \begin{bmatrix} w_1 & w_5 \\ w_2 & w_6 \\ w_3 & w_7 \\ w_4 & w_8 \end{bmatrix}$$

(2×4) (4×2)

Weighted sum depend করে a এবং b

Layer, Weighted sum, Neuron no

$$\begin{aligned} Z_1^1 &= w_1 a + w_5 b \\ Z_2^1 &= w_2 a + w_6 b \\ Z_3^1 &= w_3 a + w_7 b \\ Z_4^1 &= w_4 a + w_8 b \end{aligned}$$

Column vector (4×1)

$$\begin{bmatrix} Z_1^1 \\ Z_2^1 \\ Z_3^1 \\ Z_4^1 \end{bmatrix} = \begin{bmatrix} w_1 & w_5 \\ w_2 & w_6 \\ w_3 & w_7 \\ w_4 & w_8 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} + \text{bias}$$

(4×1) (4×2) (2×1) (4×1)

যে কয়টা Neuron সে কয়টা bias হবে।

$$\Rightarrow Z_i^1 = W_1^T \begin{bmatrix} a \\ b \end{bmatrix}$$

$$F(z_i^1) = f\left(\begin{bmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \\ z_4^1 \end{bmatrix}\right) = \begin{bmatrix} f(z_1^1) \\ f(z_2^1) \\ f(z_3^1) \\ f(z_4^1) \end{bmatrix} = \begin{bmatrix} O_1 \\ O_2 \\ O_3 \\ O_4 \end{bmatrix}$$

(4×1)

Activation function

input এর Matrix size যদি হয় → (2×1)

Hidden layer 1 এর output size (4×1) এবং এটি 2টি Hidden layer 2 এর জন্য input.

$$\begin{bmatrix} a \\ b \end{bmatrix} \quad \begin{bmatrix} \\ \\ \\ \end{bmatrix} \quad \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \quad \begin{bmatrix} \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

(2×1) (4×2) (4×4) (3×4) (3×1)

output (final)

Pretrained Model → Weights গুলো আগে থেকে জানা থাকে এবং neuron এর সংখ্যা train করানো থাকে, Last layer বা output layer টা check করে Accuracy বা validation বের করা হয়,

Exploring Boolean Function Through Perceptron

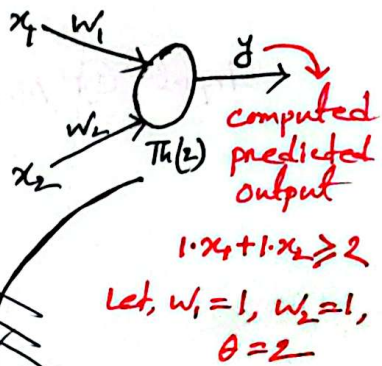
AND

Data set এর Rang (0–1)

Training data set এর যেই result টার সাথে যোগ 2 এলে desired result

Threshold

x_1	x_2	$x_1 \cdot x_2$ = d	Weighted sum $V_j = 1 \cdot x_1 + 1 \cdot x_2$	$V_j \geq 2$ = y	error d − y
0	0	0	0	0	0
0	1	0	1	0	0
1	0	0	1	0	0
1	1	1	2	1	0



$1 \cdot x_1 + 1 \cdot x_2 \geq 2$

Let, $w_1 = 1$, $w_2 = 1$, $\theta = 2$

OR

$$\begin{cases} w_1 == Th(2) \\ w_2 == Th(2) \end{cases}$$

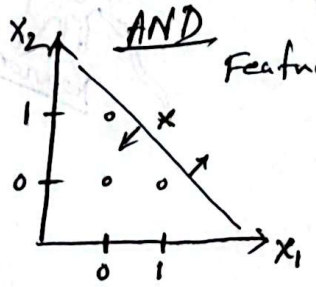
AND

$$\begin{cases} w_1 < Th(2) \\ w_2 < Th(2) \\ (w_1 + w_2) == Th(2) \end{cases}$$

If Th = 1

* Threshold কে যদি weighted sum এর equal বলে ধরা হয় তাহলে OR gate এর feature show করবে।

AND Featured Map

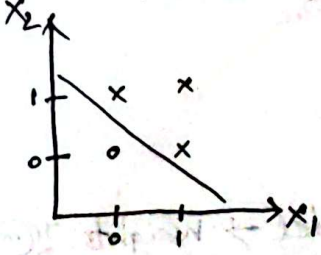


Data point গুলোকে একটি single line দ্বারা divide করা গেলো।

এই ধরনের Dataset টিকে বলে linearly separable dataset.

এমন linearly separable dataset গুলোকে single perceptron দিয়ে তৈরি করা সম্ভব/ predict করা সম্ভব।

OR Featured Map



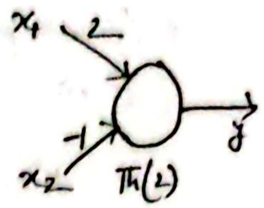
Linearly separable dataset.

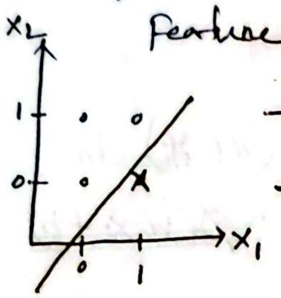
Single perceptron দিয়ে predict করা সম্ভব।

* Featured Map automatically তৈরি হলে সেটাকে বলা হয় Deep learning.

* $x_1 \overline{x}_2$

x_1	x_2	$\overline{x}_2$	$x_1 \cdot \overline{x}_2$ d	Weighted sum $V_j = 2 \cdot x_1 + (-1) \cdot x_2$	$V_j \geq 2$ y	d-y error
0	0	1	0	0	0	0
0	1	0	0	-1	0	0
1	0	1	1	2	1	0
1	1	0	0	1	0	0



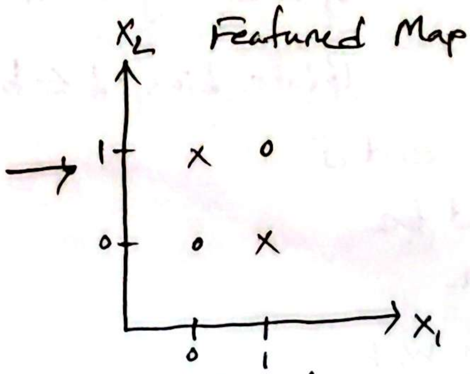


→ Linearly Separable Dataset

→ single perceptron দিয়ে solve করা possible।

* Exclusive OR

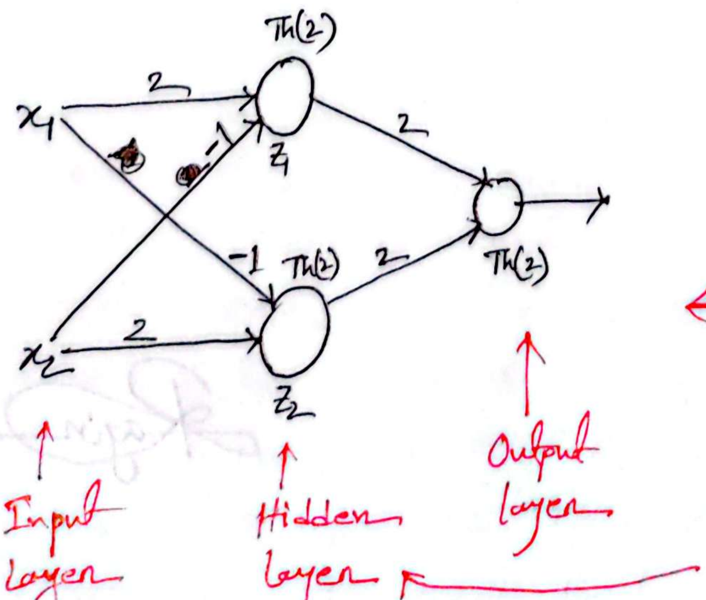
x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0



$$x_1 \oplus x_2 = \underbrace{x_1 \overline{x}_2}_{Z_1} + \underbrace{x_2 \overline{x}_1}_{Z_2}$$

Not linearly separable dataset

singleperceptron দিয়ে Mapping করা যাবে না।



← Multilayer Perceptron

Multiple line দিয়ে, সেই multiple line এর through এ boundary region তৈরি করতে পারি।

* Error column নিয়ে কাজ Learning করা।

* Error produce হওয়া নিয়ে Error সমাধান Correct করার Process নিয়ে কাজ হবে Perceptron Learning algorithm

Perceptron Learning Algorithm → Slide

Weigh vector w and bias b

initialize $w \leftarrow 0$, $b \leftarrow 0$

for $t = 1$ to T do → max num of iteration

  for each training example (x_i, y_i) do

    Compute prediction: $\hat{y}_i = \phi(w \cdot x_i + b)$

    if $\hat{y}_i \neq y_i$ then

      Update weights: $w \leftarrow w + \eta \cdot y_i x_i$

      Update bias: $b \leftarrow b + \eta \cdot y_i$

    end if

  end for

end for

return w, b

Hyperparameter
→ Learning rate (η)
→ Activation function

error boolean

Ragin

Multilayer Perceptron

Sigmoidal Function

* Differentiable
* Support non-linearity

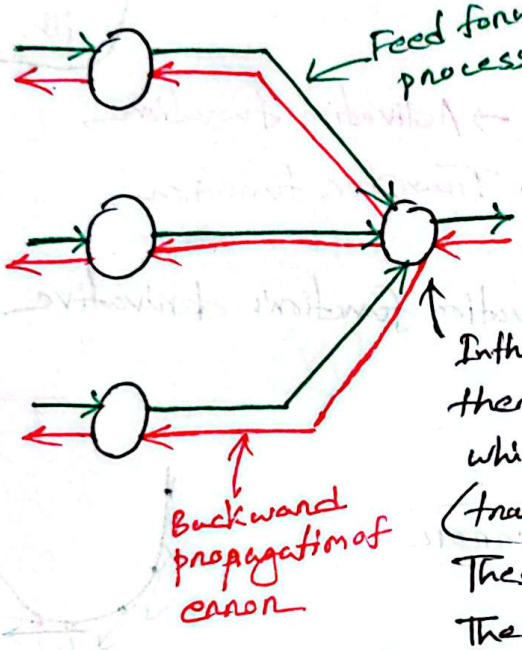
$$v_j = \sum_{i=0,\dots,m} w_{ji} y_i$$

$$\varphi(v_j) = \frac{1}{1+e^{-av_j}}$$

$$\varphi'(v_j) = a\varphi(v_j)[1-\varphi(v_j)]$$

if $a \to \infty \Rightarrow \varphi \to$ Threshold function

Learning Algorithm



In the output layer, there is a loss function which takes two inputs — (training value, computed output value). These computes error. The errors the repropagated with red lines

Back propagation algorithm

ওই NN এর weights adjust করার জন্য যার যেন loss function এর value minimize হয়।

ওনারা Loss function নিচ্ছে → The avg. square error

* Error signal of output neuron j

n-th training example

Total energy at time n: $e_j(n) = d_j(n) - y_j(n)$

(Desired training value, Computed output of the neuron)

Avg. Square error: $E(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$

(C ← set of neurons in output layer)

Measure of learning performance: $E_{AV} = \frac{1}{N}\sum_{n=1}^{N} E(n)$

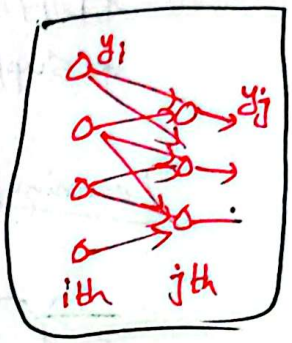
Our Goal: Adjust weights of NN to minimize E_{AV}

Minimize the avg. square error which is our loss function, so we need to adjust our NN's weights in such a way so, our loss function can be minimized.

$e_j \rightarrow$ Error at output of neuron j.

$y_j \rightarrow$ Output of neuron j.

$v_j \rightarrow \sum_{i=0,\cdots m} w_{ji} y_i$ Induced local field of neuron j.



$e_j = d_j - y_j$

jth neuron

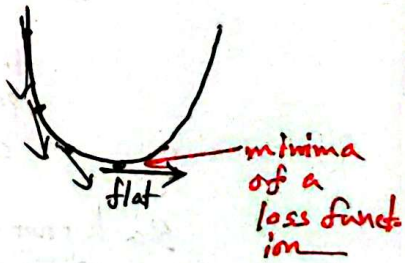
$y_j = \varphi(v_j)$ $\varphi(\cdot) \rightarrow$ Activation function

$v_j \rightarrow$ Transfer function

$\frac{dy_j}{dv_j} = \varphi'(v_j) \rightarrow$ Activation function's derivative

Hebbs rule

$w_{new} = w_{old} + \underbrace{LR * I_j * error}_{\Delta w}$



$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}}$

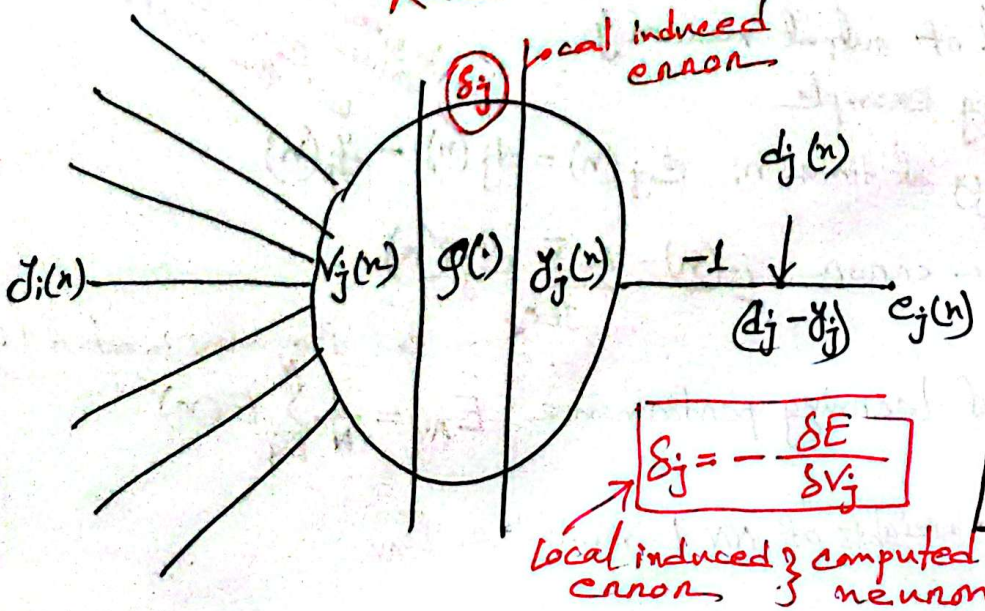
Learning Rate

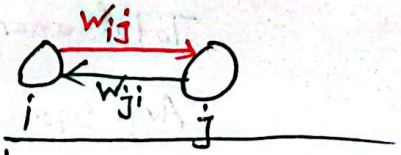
Forward passing

$\rightarrow w_{From-To}$

Error Passing

$\rightarrow w_{To-From}$





$v_j = \sum w_{ji} y_i$

$\varphi(v_j) =$ ____

$y_j = \varphi(v_j)$

$e_j = d_j - y_j$

$\delta_j = -\frac{\delta E}{\delta v_j}$

Local induced error } computed inside the neuron itself

$$E = \frac{1}{2}\sum e_j^2$$

$$= \frac{1}{2}\left(e_1^2 + e_2^2 + \cdots + e_j^2 + \cdots e_n^2\right)$$

$$\frac{\delta E}{\delta e_j} = 0+0+0+\frac{1}{2}\cdot 2e_j = e_j$$

$$e_j = d_j - y_j$$

$$= 0 - 1 = -1$$

$$\frac{\delta y_j}{\delta v_j} = \varphi'(v_j)$$

$$\frac{\delta E}{\delta v_j} = \frac{\delta E}{\delta e_j}\cdot\frac{\delta e_j}{\delta y_j}\cdot\frac{\delta y_j}{\delta v_j}$$

$$= e_j \cdot (-1)\cdot \varphi'(v_j)$$

$$\therefore \delta_j = -\frac{\delta E}{\delta v_j}$$

$$= -\{e_j \cdot (-1)\cdot \varphi'(v_j)\}$$

$$= e_j \varphi'(v_j)$$

Local ~~induced~~ induced error of the output layer

$$\delta_j = e_j \varphi'(v_j)$$

$$\delta_j = -\frac{\delta E}{\delta v_j}$$

$$v_j = \sum_{i=0,\cdots m} w_{ji} y_i$$

$$\frac{\delta v_j}{\delta w_{ji}} = y_i$$

$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}}$$

$$= -\eta \cdot \frac{\delta E}{\delta v_j}\frac{\delta v_j}{\delta w_{ji}}$$

$$= \eta \cdot \delta_j \cdot y_i$$

$$\therefore \Delta w_{ji} = \eta * \delta_j * y_i$$

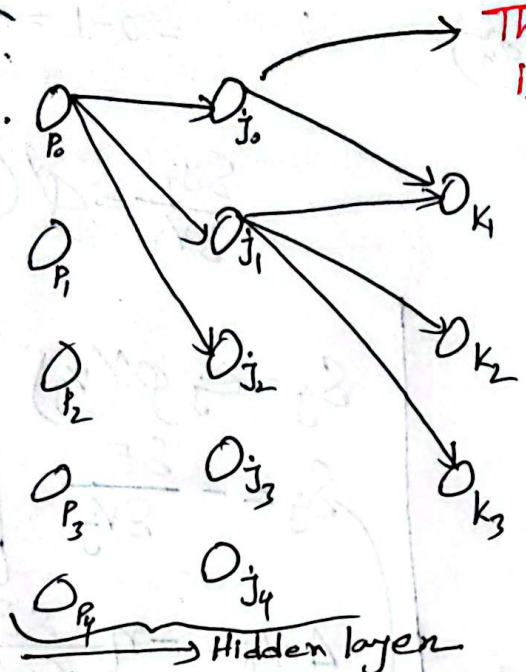
Case:1 $j \rightarrow$ output neuron

$$e_j = d_j - y_j$$

$$\delta_j = (d_j - y_j)\varphi'(v_j)$$

Case2: J → Hidden neuron

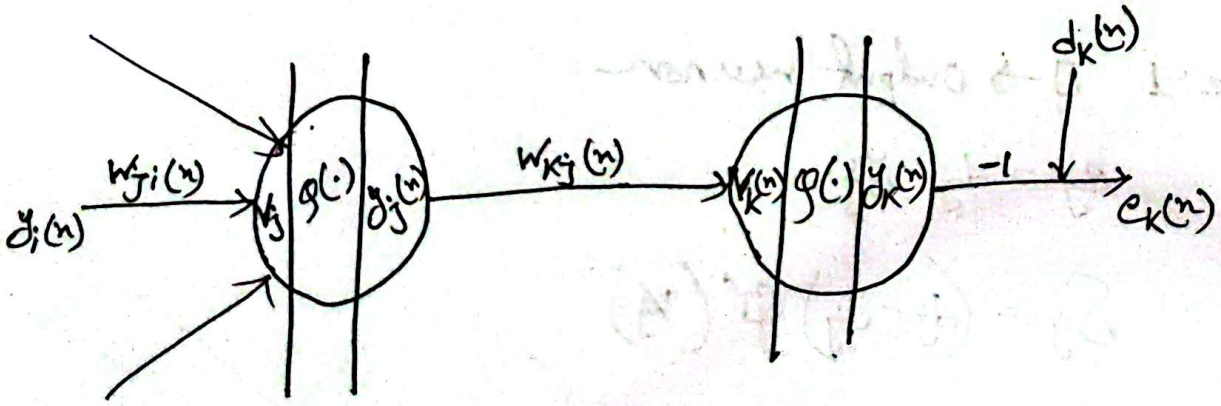
The local gradient for neuron j is recursively determined in terms of the local gradients of all neurons to which neuron j is directly connected.



$$\delta_{J_1} = f_1(\delta_{K_1}, \delta_{K_2}, \delta_{K_3})$$

$$\delta_{P_0} = f_2(\delta_{J_0}, \delta_{J_1}, \delta_{J_2})$$

$$= f_2\left(f(\delta_{K_1}), f_1(\delta_{K_1}, \delta_{K_2}, \delta_{K_3}), f\text{---}\right)$$



$$\delta_j = -\frac{\delta E}{\delta v_j} = -\frac{\delta E}{\delta y_j} \cdot \frac{\delta y_j}{\delta v_j}$$

$$= -\frac{\delta E}{\delta y_j} \cdot \varphi'(v_j) \quad \left[\frac{\delta y_j}{\delta v_j} = \varphi'(v_j)\right]$$

$$\delta_k = e_k \varphi'(v_k)$$

$$\varphi(v_k) = y_k$$

$$e_k = d_k - y_k$$

$$\frac{\delta E}{\delta y_j} = ?$$

$$E(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n)$$

$$\frac{\delta E}{\delta y_j} = \sum_{k \in C} e_k \cdot \frac{\delta e_k}{\delta y_j}$$

$$= -\sum_{k \in C} e_k \cdot \left(-\frac{\delta e_k}{\delta y_j}\right)$$

$$= -\sum_{k \in C} e_k \cdot \left(\frac{-\delta e_k}{\delta v_k}\right)\frac{\delta v_k}{\delta y_j}$$

$$= -\sum_{k \in C} e_k \varphi'(v_k) w_{kj}$$

$$\delta_j = -\frac{\delta E}{\delta v_j}$$

$$= -\frac{\delta E}{\delta y_j}\varphi'(v_j)$$

$$= \sum_{k \in C} \delta_k w_{kj} \cdot \varphi'(v_j)$$

$$v_k = \sum_{k=0,\cdots m} w_{kj} y_j$$

$$\frac{\delta v_k}{\delta y_j} = w_{kj}$$

$$-\frac{\delta e_k}{\delta v_k} = -\frac{\delta(d_k - y_k)}{\delta v_k}$$

$$= \frac{\delta y_k}{\delta v_k}$$

$$= \varphi'(v_k)$$

$$[y_k = \varphi(v_k)]$$

$$\therefore \frac{\delta E}{\delta y_j} = -\sum_{k \in C} e_k \varphi'(v_k) w_{kj}$$

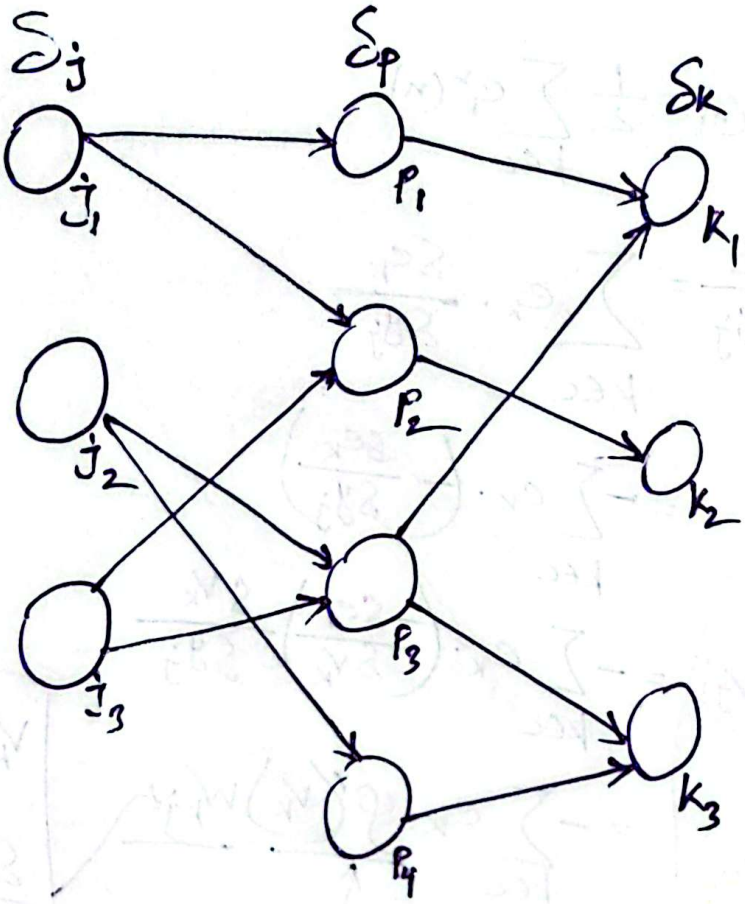
$$= -\sum_{k \in C} \delta_k w_{kj}$$

Local induced error of output layer

Output layer (j) → $\delta_j = e_j \varphi'(v_j)$

Hidden layer (j) → $\delta_j = \sum_{k \in C} \delta_k w_{kj} \cdot \varphi'(v_j)$

Example



Connections

$j_1 \rightarrow (P_1, P_2)$

$j_2 \rightarrow (P_3, P_4)$

$j_3 \rightarrow (P_3, P_3)$

$P_1 \rightarrow K_1$

$P_2 \rightarrow K_2$

$P_3 \rightarrow (K_1, K_3)$

$P_4 \rightarrow K_3$

$\delta_{K_1} = \varphi'(v_{K_1})\, e_{K_1}$

$\delta_{K_2} = \varphi'(v_{K_2})\, e_{K_2}$

$\delta_{K_3} = \varphi'(v_{K_3})\, e_{K_3}$

$$\delta_{P_1} = \varphi'(v_{P_1})\, \delta_{K_1} w_{K_1 P_1}$$

$$\delta_{P_2} = \varphi'(v_{P_2})\, \delta_{K_2} w_{K_2 P_2}$$

$$\delta_{P_3} = \varphi'(v_{P_3})\left(\delta_{K_1} w_{K_1 P_3} + \delta_{K_3} w_{K_3 P_3}\right)$$

$$\delta_{P_4} = \varphi'(v_{P_4})\, \delta_{K_3} w_{K_3 P_4}$$

$$\delta_{j_1} = \varphi'(v_{j_1})\left(\delta_{P_1} w_{P_1 j_1} + \delta_{P_2} w_{P_2 j_1}\right)$$

$$= \varphi'(v_{j_1})\left\{\left(\varphi'(v_{P_1})\, \delta_{K_1} w_{K_1 P_1}\right) w_{P_1 j_1} + \left(\varphi'(v_{P_2})\, \delta_{K_2} w_{K_2 P_2}\right) w_{P_2 j_1}\right\}$$

$$\delta_{j_2} = \varphi(v_{j_2})\left(\sum_{P_3, P_4 \in P_i} \delta_{P_i} w_{P_i j_2}\right)$$

← Short form in the context of computing

$\delta_{j_3} =$ — —

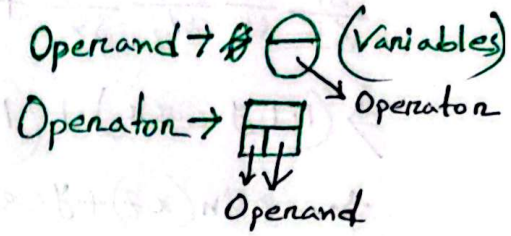
Local gradient of induced error inside the cell

Computational Graph

$f(x, y, z) = y \sin(x) + \cos(x \cdot z)$

$\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}, \frac{\delta f}{\delta z}$

$t_1 = x \cdot z$
$t_2 = \cos(t_1)$
$t_3 = \sin(x)$
$t_4 = y \cdot t_3$
$t_5 = t_4 + t_2$

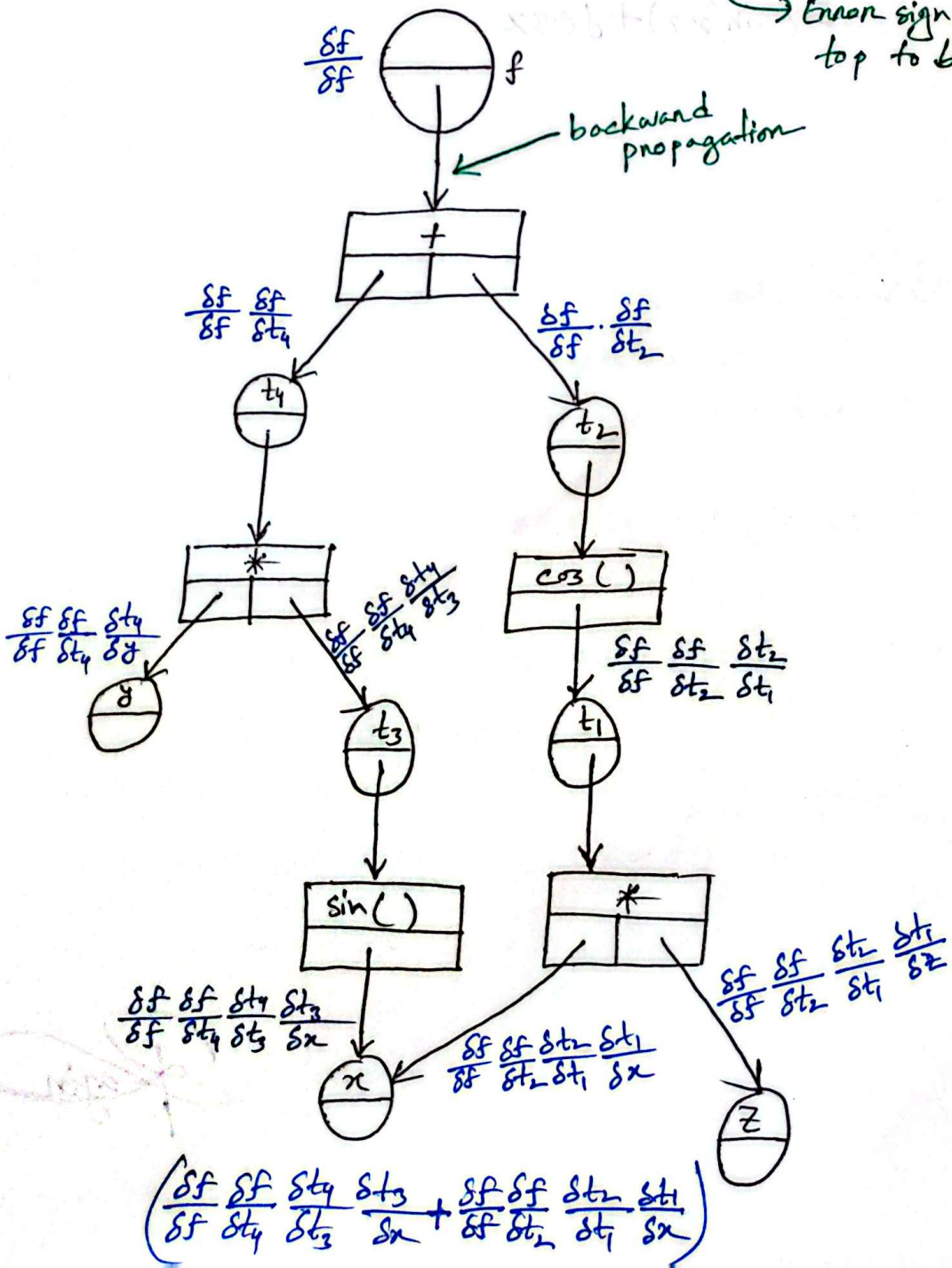


Operaton → No differentiation

↳ Feed forwarding process
↳ Computational result

↳ Backward process
↳ Error signal from top to bottom

Computational Graph



$$\frac{\delta f}{\delta f}\frac{\delta f}{\delta t_4}\frac{\delta t_4}{\delta t_3}\frac{\delta t_3}{\delta x} + \frac{\delta f}{\delta f}\frac{\delta f}{\delta t_2}\frac{\delta t_2}{\delta t_1}\frac{\delta t_1}{\delta x}$$

$$\Rightarrow (1 \cdot 1 \cdot y \cdot \cos x) + (1 \cdot 1 \cdot (-\sin(xz))z)$$

$$\Rightarrow -z\sin(xz) + y\cos x$$

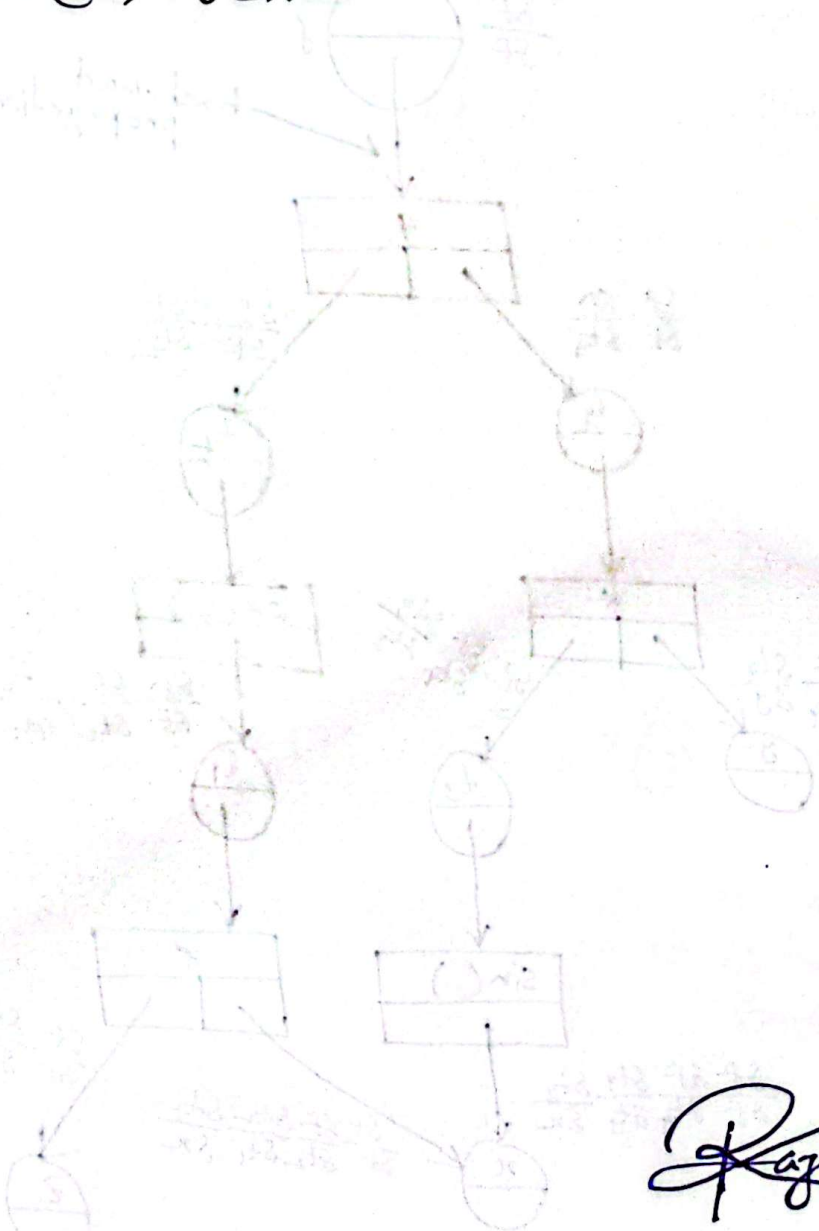
Check

$$f = y\sin(x) + \cos(xz)$$

$$\frac{df}{dx} = y \cdot \frac{\delta \sin(x)}{\delta x} + (-\sin(x \cdot z)) \cdot \frac{\delta(xz)}{\delta x}$$

$$= y\cos x + (-\sin(xz))z$$

$$= -z\sin(xz) + y\cos x$$



Rajin

Recurrent Neural Network (RNN)

① Sequence or pattern, we can apply the RNN

```
if (weather == sunny)
    eat chicken
else if (weather == Rainy)
    eat Burger
```

} Control flow program
No sequence is getting generated
We can not apply RNN

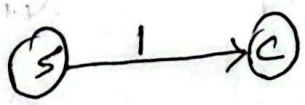
input: Weather (sunny, rainy)
Output: Food (Chicken, Burger, Pizza)

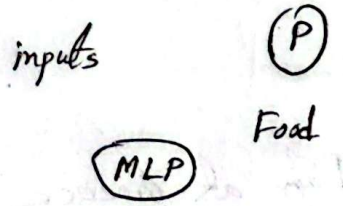
Weather → Food

MLP → One hot vector representation — Only 1's value could be 1.

$$W = \begin{bmatrix} S \\ R \end{bmatrix} \Rightarrow S = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad R = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$F = \begin{bmatrix} C \\ B \\ P \end{bmatrix} \Rightarrow C = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad P = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$





Adj. Matrix

	C	B	P
S	1	0	0
R	0	1	0

(2×3)

Transpose →

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}_{(3\times2)}$$

This problem shows how a boolean functionality can be represented with the help of MLP

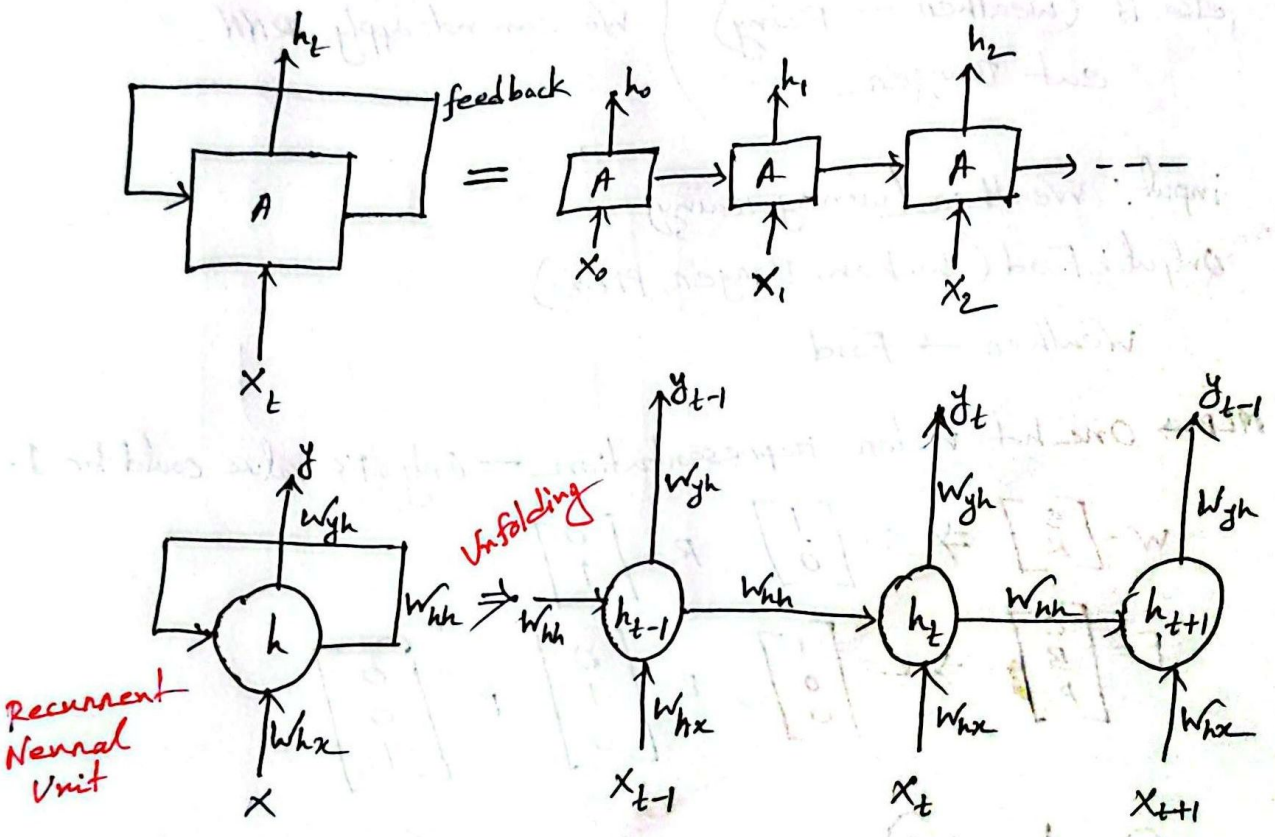
This weight matrix basically computed by the learning procedure of MLP

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}_{(3\times2)} \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{S} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}_{C\ (3\times1)}$$

This not applicable for RNN, because it demands sequence and also needs to produce sequence or pattern.

* How it is different from MLP

① Anahiteatural difference



② MLP can not be applied upon sequence
all the information passes in single pass so. thats why we can not apply Long contentual sequences in MLP for prediction.

Example

I am travelling to europe. Now I'am at Greece, after travelling from Germany to Rome, Fourance and Switzerland. In France, I need to communicate with people in - - - - - - language.

contextual information

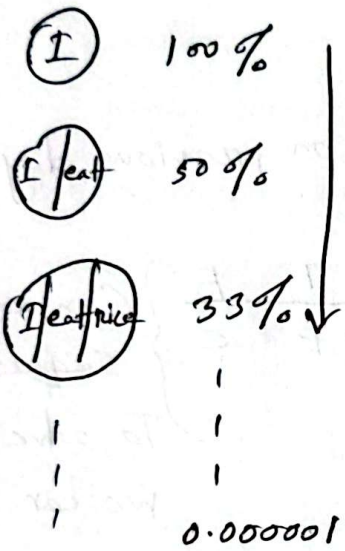
Upper hand RNN is used to capture the sequence which hold the context.

There is a limitation

It can not process on provide connect prediction if the sequence become too too long.

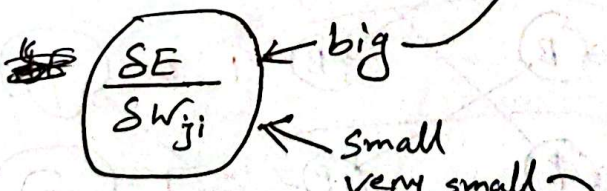
Human memories are work like shuttle points.

Example : I eat nice everyday.



Longer sequences will wipeout incideats from the context and that's why RNN suffers from Vanishing Gradient Problem

Exploded Gradient Problem



Vanishing Gradient Problem

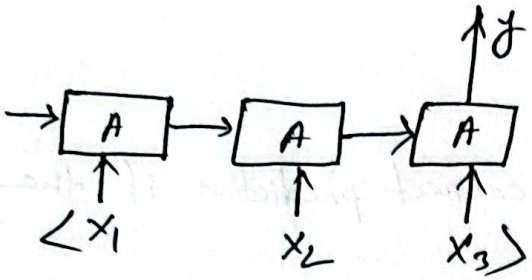
For removing vanishing gradient problem

RNN takes help of LSTM (Long short Term Memory)

Short term memory will be recalled for Long period

* 4 different formats of RNN

One to one	Word to word
One to many	Word to sentence
Many to one	Sentence to word
Many to Many	Sentence to sentence



⇒ Sentence to word

Weather → Food using MLP

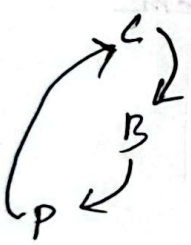
Let's think about a scenario

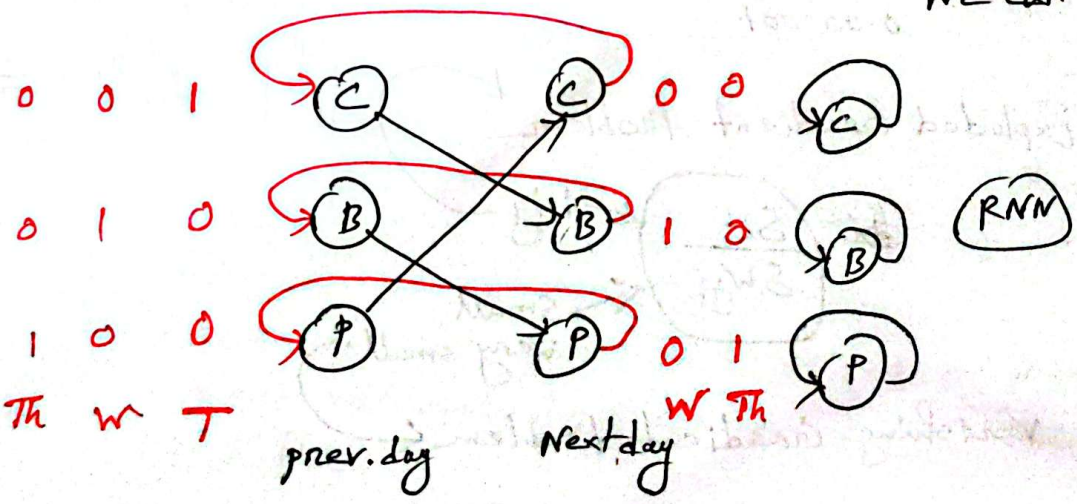
→ what you had on previous day?

S	S	M	T	W	T	F
C	B	P	C	B	P	C
B						

Generation of sequence.

To solve this generation we can apply RNN





	C	B	P
C	0	1	0
B	0	0	1
P	1	0	0

(3×3)

Transpose ⇒

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

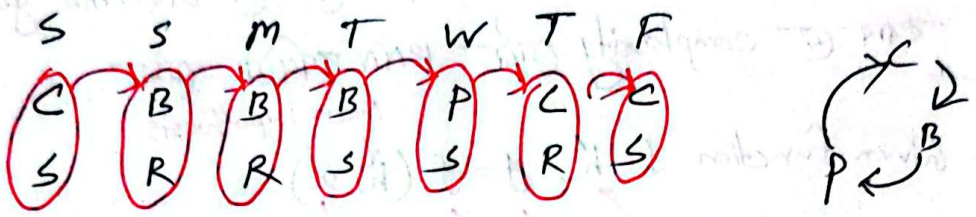
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

(3×3) (3×1) (3×1)

C B

This weight matrix are computed similar fasion like MLP

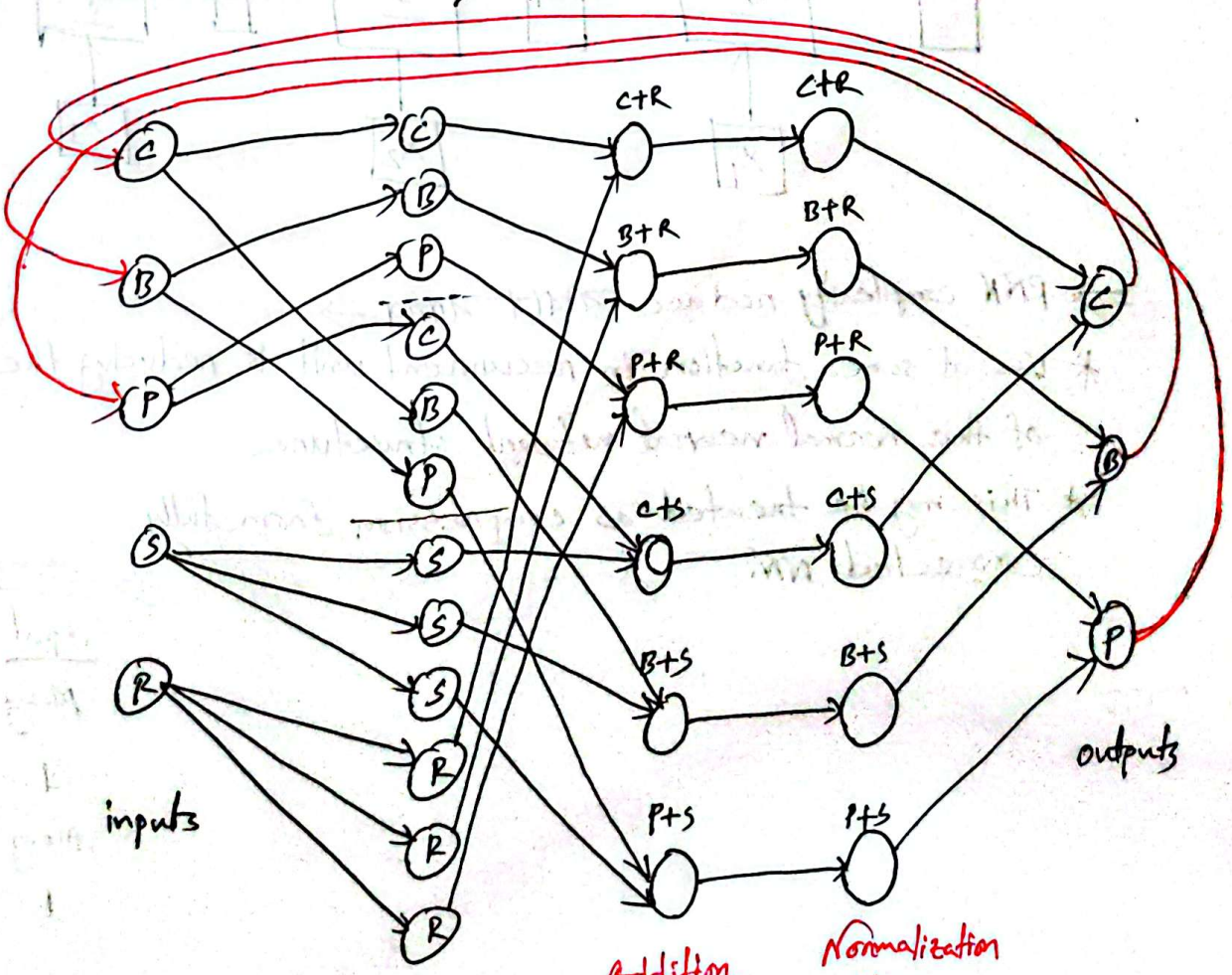
Lets introduced impact of weather in the problem



(1) If weather is sunny i can go to shopkeeper and he will provide me the next dish on the cycle

(2) If weather is Raing I can not go, so I had to eat previous dish.

Weather + Food → Food

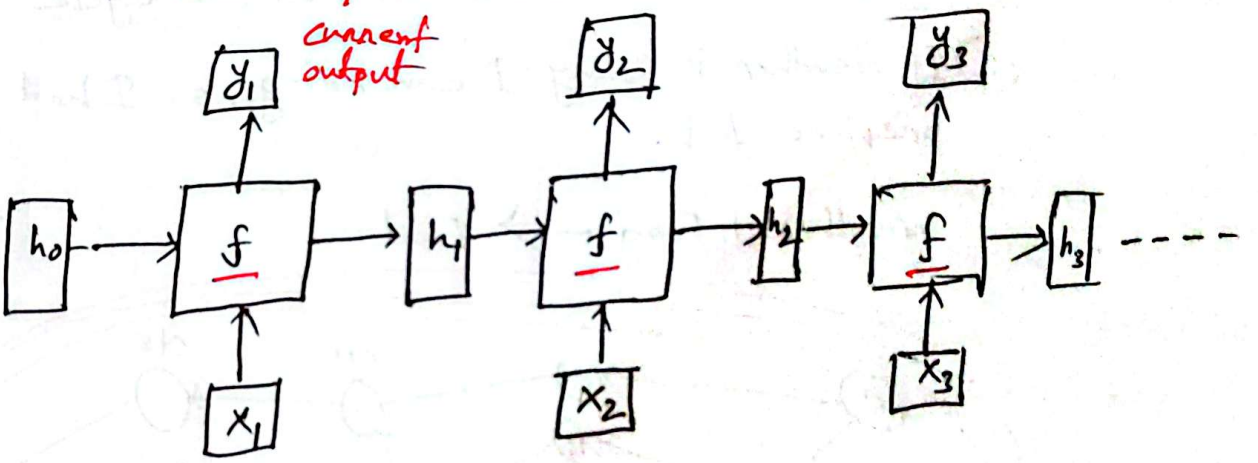


Problem Friend + Weather + Food

* RNN হলো normal MLP এর চেয়ে একটু complex structure ~~balag balong~~ বলা। Normal MLP এর চেয়ে RNN এর calculation complexity বাড়তে পারে কিন্তু এর computation এর মাধ্যমে যে outcome generate ~~হচ্ছে~~ করার যে complexity সেটা RNN বাদিয়ে দেয়।

Given function $f: h', y = f(h, x)$

prev. hypothesis, current input, current hypothesis, RNN function, current output



⇒ RNN complexity reduce ~~করার~~ করার →

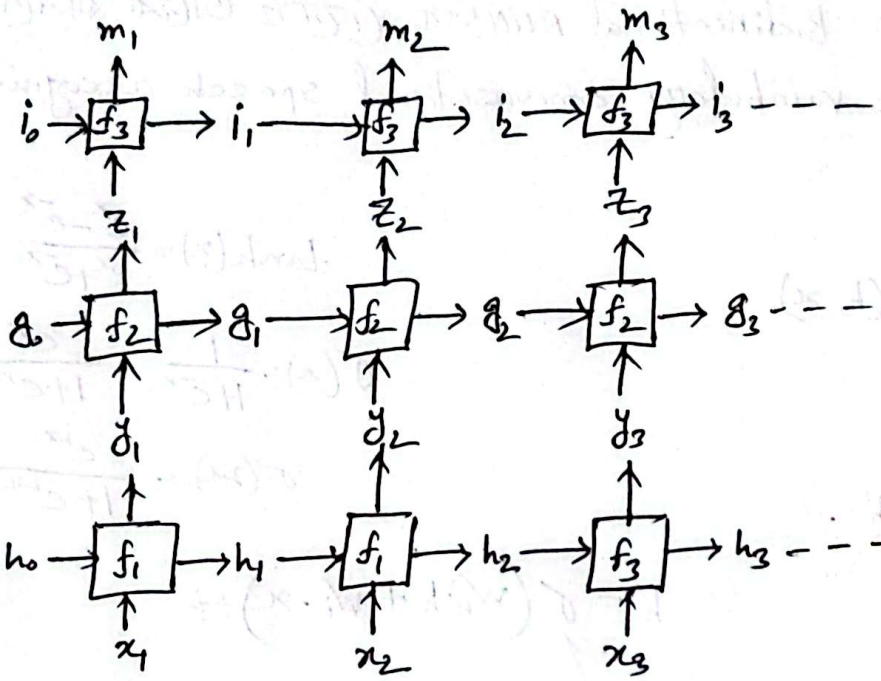
* Use of same function in recurrent unit is reducing the complexity of the normal neural network structure
* This may be treated as compression from fully connected NN.

Sequences

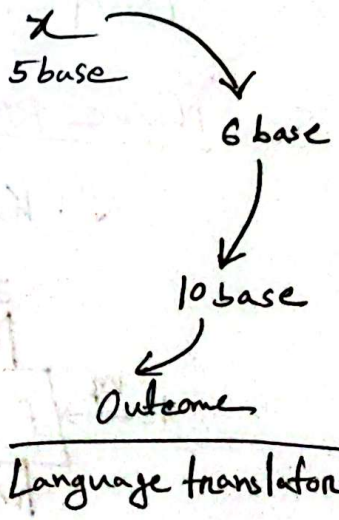
input	Output
Many to	Many
1 to	Many
Many to	1
1 to	1

Deep RNN

$h', y = f_1(h, x)$ $\quad g', z = f_2(g, y)$ $\quad i', m = f_3(i, z)$



Use case



Language translator

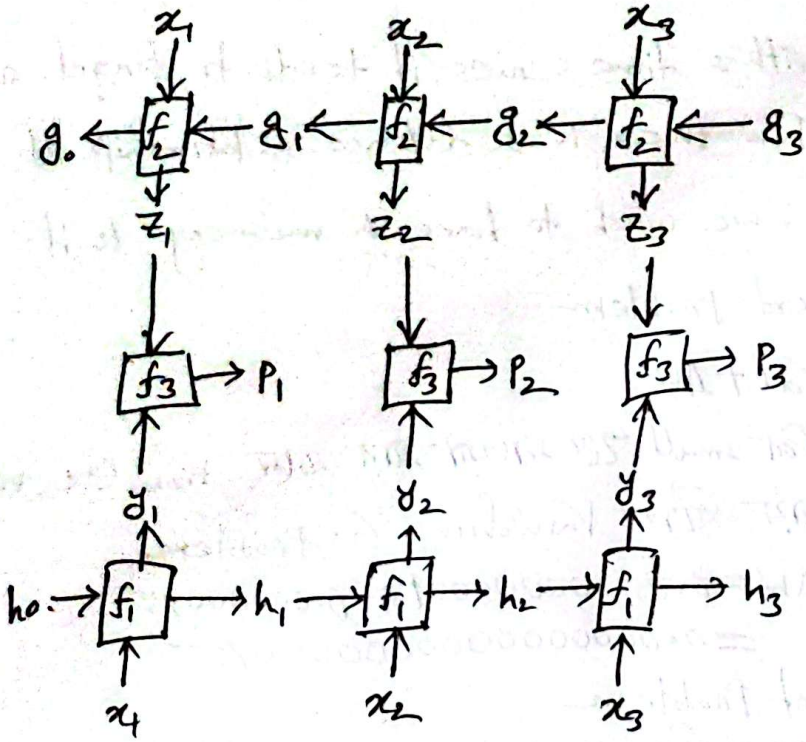
Bidirectional RNN

Use case → Palindrome checking

Normal RNN এর Complexity এর চেয়ে Bidirectional RNN এর Complexity অনেক বেশি,

A* search

huge depth এর problem এর Complexity কে solve করা যায়



Pyramid RNN

Use case → Sequence to word generation

এর learning/training speed significantly faster

Recurrent steps Bidirectional RNN এর চেয়েও অনেক কমিয়ে দেয়,

Example: Large vocabulary conversational speech recognition

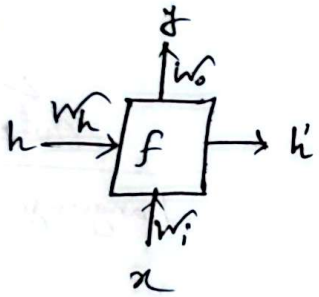
Naive RNN

$f : h', y = f(h, x)$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \sigma(2x)$$

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

$$\sigma(2x) = \frac{e^{2x}}{1+e^{2x}}$$



$$h' = \sigma\left(W_h h + W_i \cdot x\right) + b$$

Activation function

y is computed from h'

$$y = \sigma\left(W_o h'\right) + b$$

Softmax process

Problem

- When dealing with a time series, it tends to forget old information. When there is a distance relationship of unknown length, we wish to have a memory to it.
- Vanishing Gradient Problem

  $W_{new} = W_{old} + \Delta W$

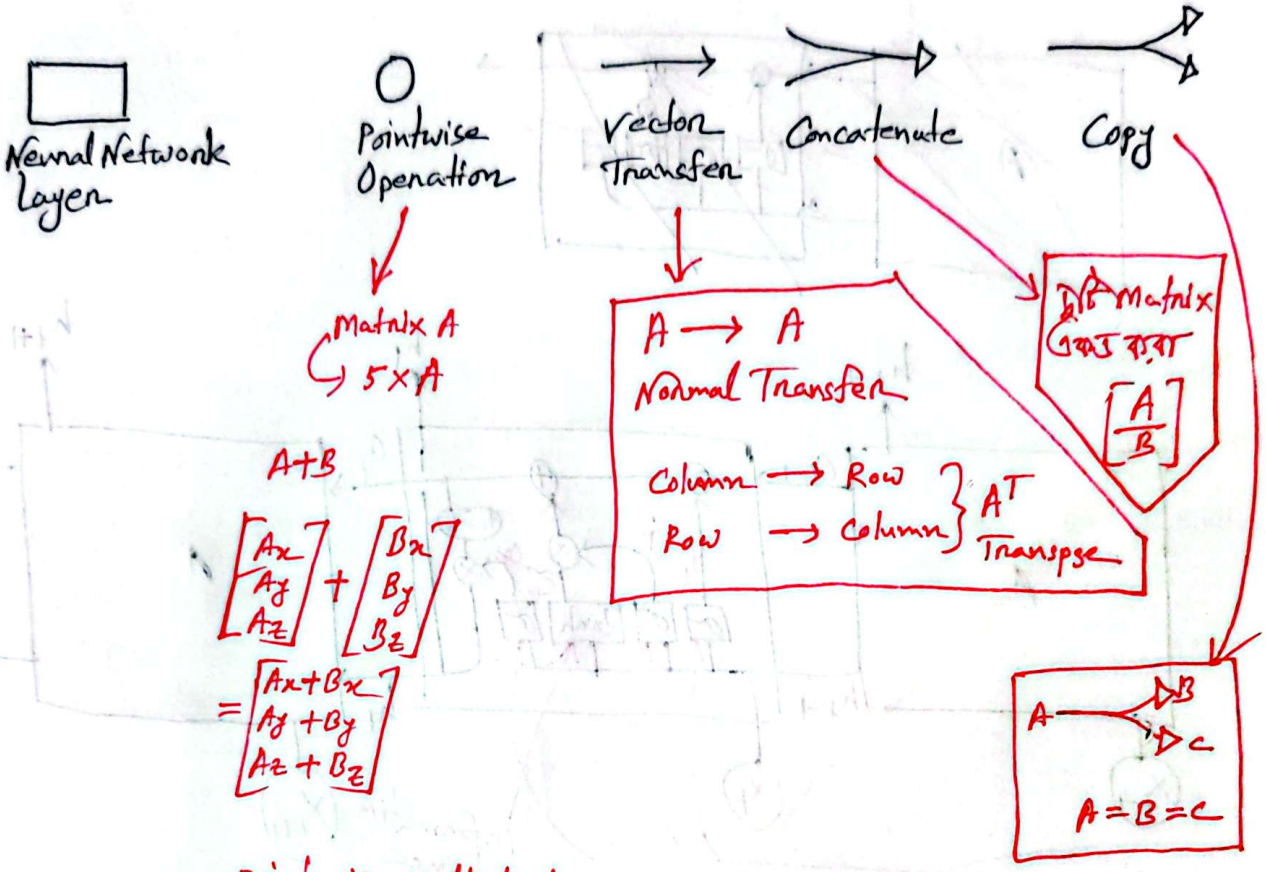
  ΔW অনেক বেশি small হয়ে গেলে গেলে W_{new} এর W_{old} same হয়ে যায়, তাহলে Vanishing G. Problem.

  Example: $\Delta W = 0.0000000001 * 0.000000075$

  $= 0.0000000000000000075$
- Exploding Gradient Problem

  ΔW যদি অনেক অনেক বেশি বড় হয়ে যায় তখন এই Problem arise হয়,

LSTM



A+B

$$\begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} + \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = \begin{bmatrix} A_x+B_x \\ A_y+B_y \\ A_z+B_z \end{bmatrix}$$

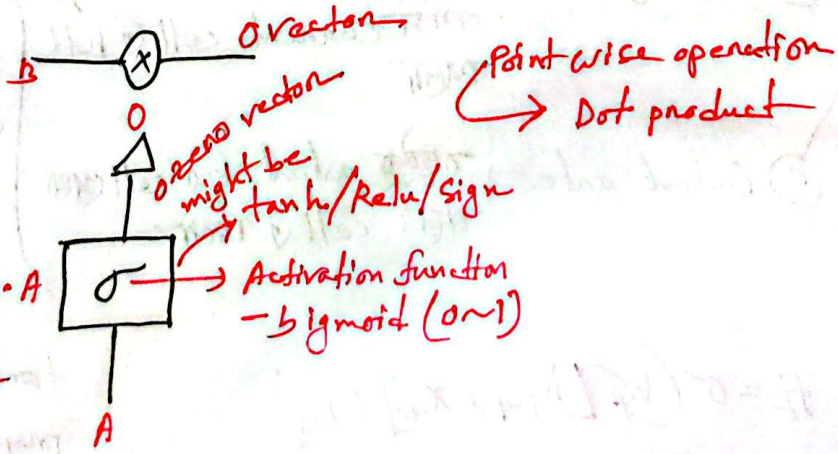
Point wise Multiplication

$$\begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} \odot \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = \begin{bmatrix} A_xB_x \\ A_yB_y \\ A_zB_z \end{bmatrix}$$

(3×1) (3×1)

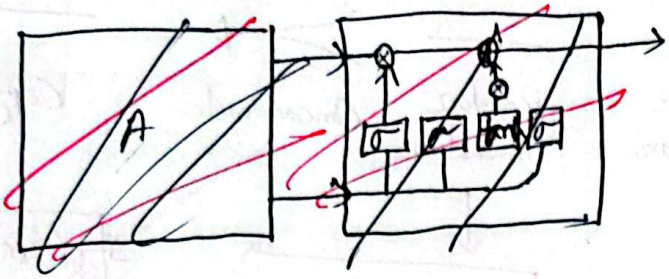
→ Matrix multiplication possible এর. (3×1).(3×1) wrong

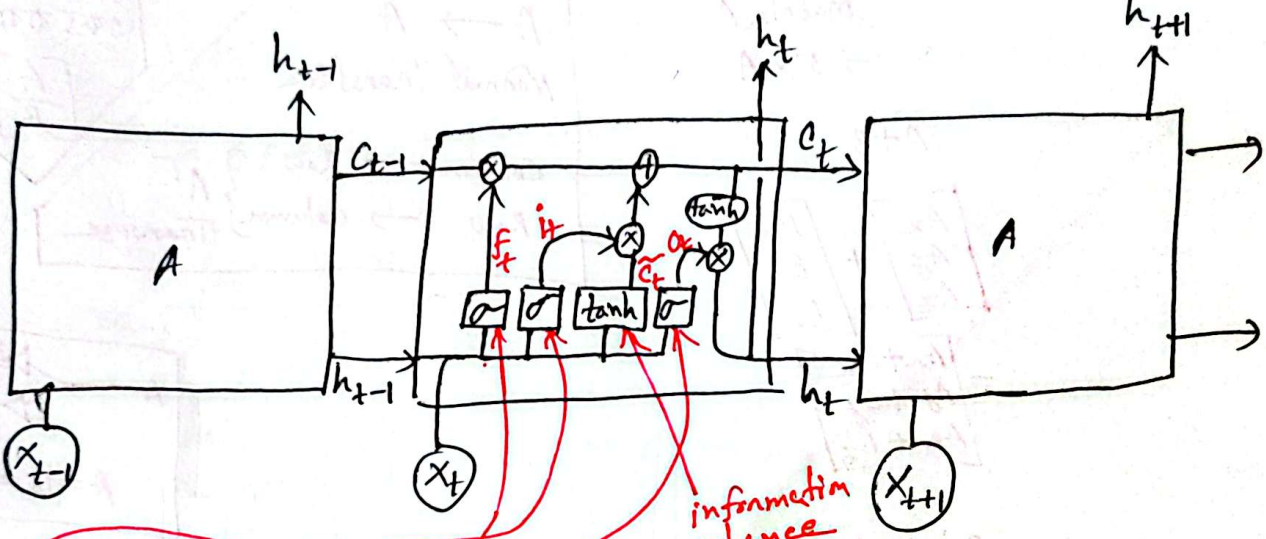
কিন্তু Pointwise multip. possible



The sigmoid layer outputs numbers between 0-1 determine how much each component should be let through.

LSTM





① Forget gate → কোনার বাছে যা info আসবে সবগুলো কিছু Proceed করবো না, কিছু info forget করে দিবো

② Input gate → যেই info আসতেছে তার কত % আমি current cell এ hold করবো

③ Output gate → কতটুকু output যাবে cell থেকে পরের cell এ যাবে

gate/NN layer with Activation function σ - ③

3 σ Activation function গুলো control করতেছে

$$f_t = \sigma\left(W_f [h_{t-1}, x_t] + b_f\right)$$

$$i_t = \sigma\left(W_i [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh\left(W_{information} [h_{t-1}, x_t] + b_{information}\right)$$

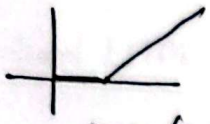
$$C_t = f_t * C_{t-1} + i_t \times \tilde{C}_t$$

$$h_t = O_t * \tanh(C_t)$$

tanh এর সবচেয়ে বড় property non-linearity

Information enhanced and shrink দুটিই করতে পারে।

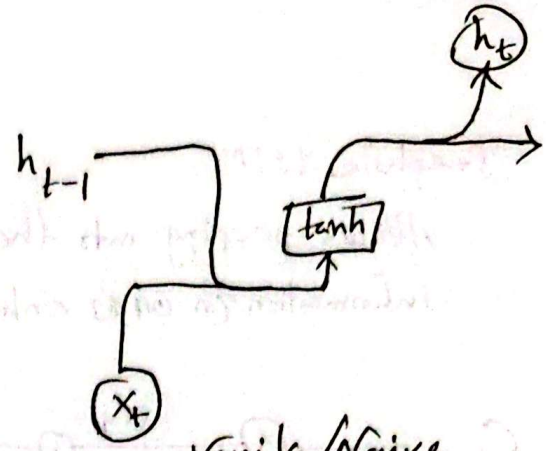
ReLU



$\max(z, 0)$

Linear

in manner

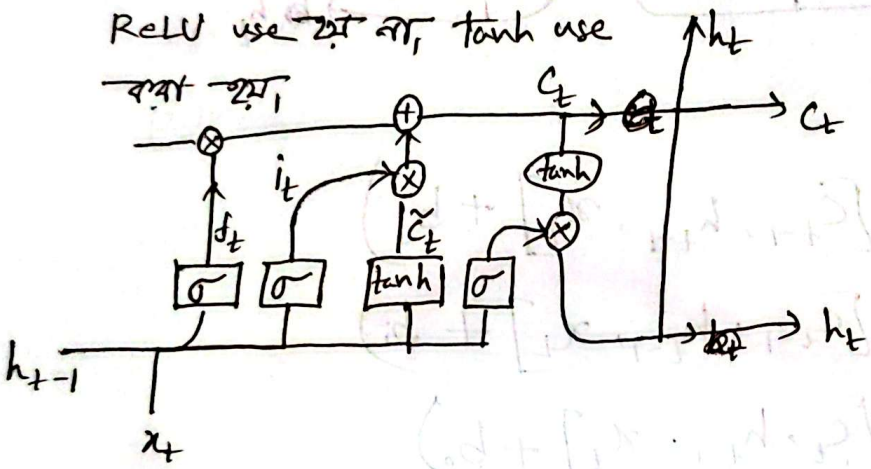


Vanila / Naive RNN

① No cell state in RNN

② There is no sigmoid locking key.

Information কে খুব তাড়াতাড়ি enhanced / shrink করতে পারে না, তাই LSTM এর structure এ ReLU use হয় না, tanh use করা হয়।



LSTM

① Computationally LSTM একটু expensive but LSTM can hold a longer memory

② It is able to solve vanishing gradient problem

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

key of forget gate

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

key

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

Information Update

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

updating the cell state

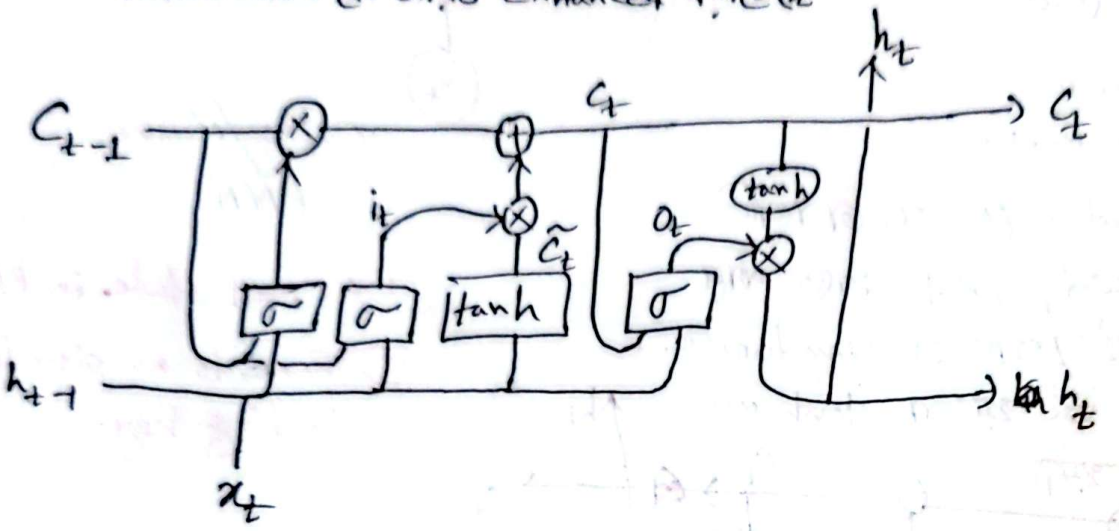
Decide what part of the cell state to output

$$O_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = O_t * \tanh(C_t)$$

Peephole LSTM

Allows peeping into the memory.
information ক আরও enhanced করা যায়



$$f_t = \sigma\left(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f\right)$$

$$i_t = \sigma\left(W_i [C_{t-1}, h_{t-1}, x_t] + b_i\right)$$

$$O_t = \sigma\left(W_o [C_t, h_{t-1}, x_t] + b_o\right)$$

$$C_t = f_t * C_{t-1} + i_t \times \tilde{C}_t$$

$$h_t = O_t * \tanh(C_t)$$

Naive RNN VS LSTM

C changes slowly → C^t is C^{t-1} added by something

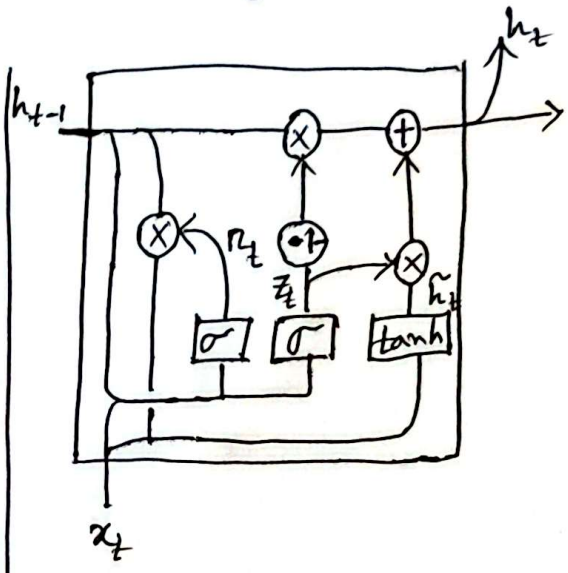
h changes faster → h^t and h^{t-1} can be very different

GRU

* ~~Lost~~ LSTM এর চেয়ে Computationally অনেক faster.

* GRU and RNN এ আলাদা Cell memory নাই।
  GRU and RNN দুটি Hypothesis এর উপর ভিত্তি করে তৈরি।

* To omit Vanishing Gradient Problem, GRU is being introduced.



2 gates in GRU

1. Update gate (z_t)
2. Reset gate (r_t)

Variables

$x_t \rightarrow$ Input vector
$h_t \rightarrow$ Output vector
$\tilde{h}_t \rightarrow$ Candidate activation vector
$z_t \rightarrow$ Update gate vector
$r_t \rightarrow$ Reset gate vector
$W, U \rightarrow$ Weight matrices, $b \rightarrow$ bias

Update gate

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right)$$

Reset gate

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right)$$

$$\tilde{h}_t = \tanh\left(W_h x_t + U_h (h_{t-1} \odot r_t) + b_h\right)$$

$$h_t = \left((1 - z_t) \cdot h_{t-1}\right) + \left(z_t \cdot \tilde{h}_t\right)$$

$x_t \rightarrow (5 \times 1)$

$W_z x_t$, $U_z h_{t-1}$, b_z } dimension same example: (5×1)

z_t, r_t } (5×1)

$\tilde{h}_t$ এর h_{t-1} এবং h_t এর dimension same হবে।

Ragin