Date 16 08 2023

U-1

CS409 - Neural Networks

In 401,

1.) Biological neuron vs computational neuron
2.) Neuron as a binary classifier - Percetron
3.) Brief introduction to NN - MLP

Topics

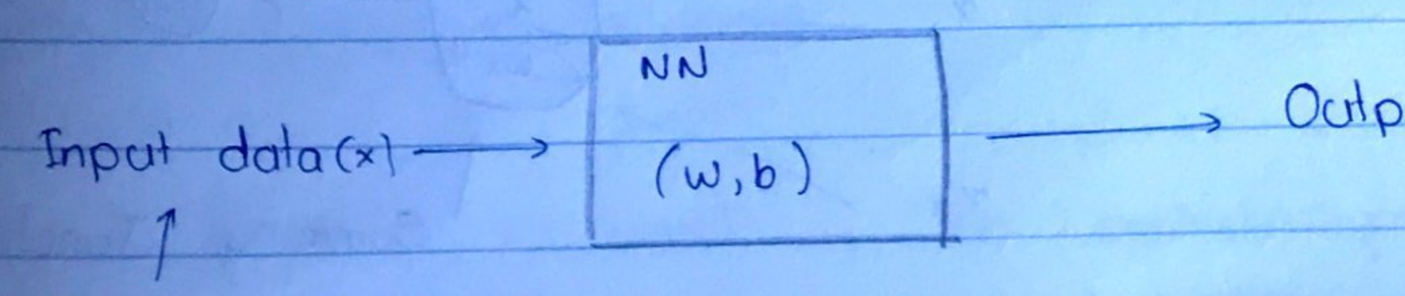
01.) Introduction to NN
02.) Perceptron
03.) Neural Networks and MLP
04.) Backpropagation learning algorithm (using MLP)
05.) Convolutional Neural Networks with Applications.
06.) Recurrent Neural Networks (RNNs) for sequential data
07.) Generative adversarial Network (GANs)

- Transfer learning
- Self-supervised learning

Book :- Ian Goodfellow, Yoshia Benshio

NN sucess stories :- object detection/recognition, speech recognition,

Traditional ML with NN (Multilayer Perceptron)



$x = \{x_1, x_2, \ldots, x_n\}$

$y = Wx + b$

we find w, b through learning

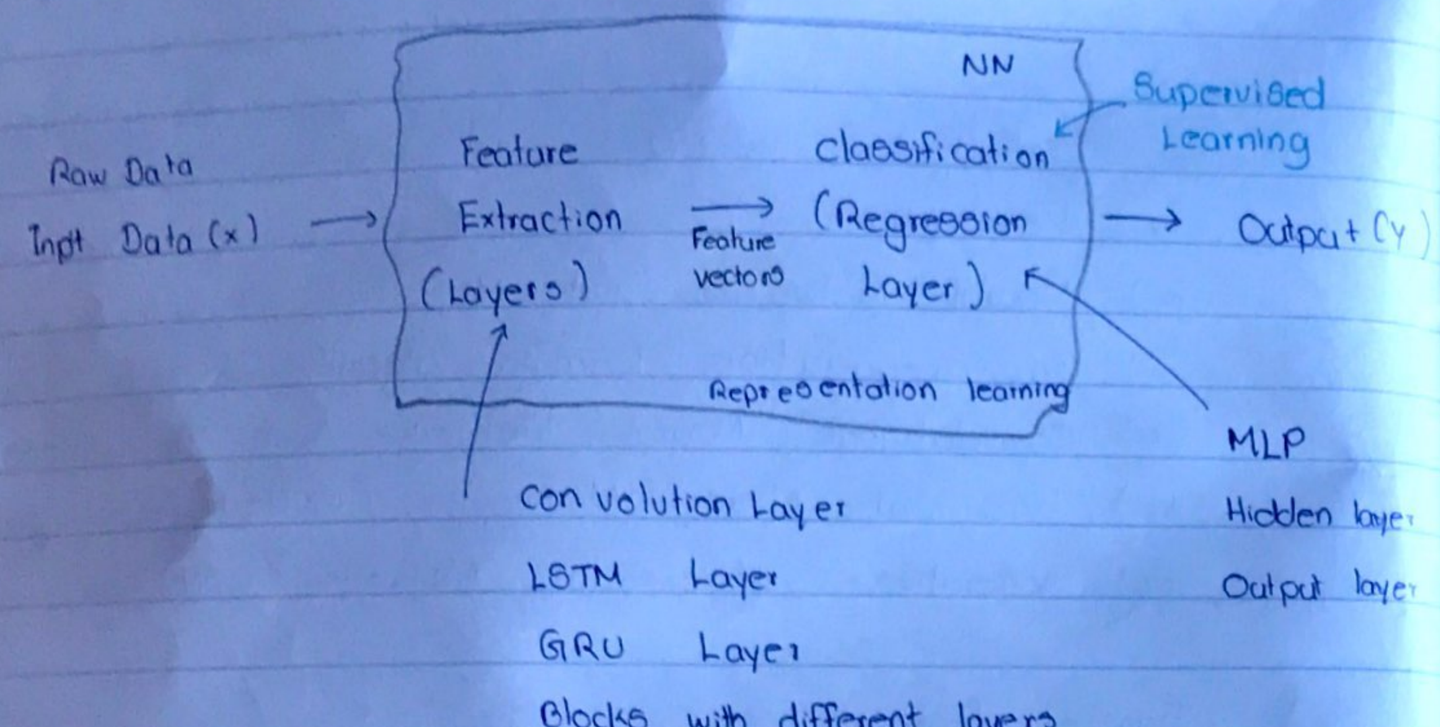
(w - weights, b - bias)

Scanned with CamScanner

MLP ← fully connected NN with input layer, hidden layer and output layer.

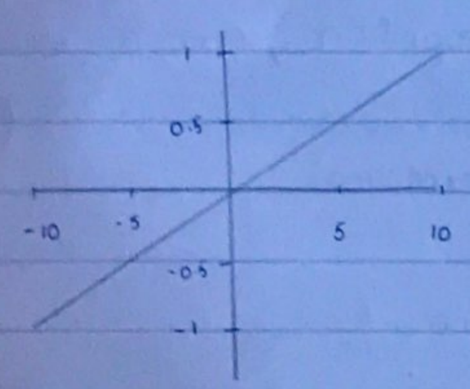
Shallow Networks

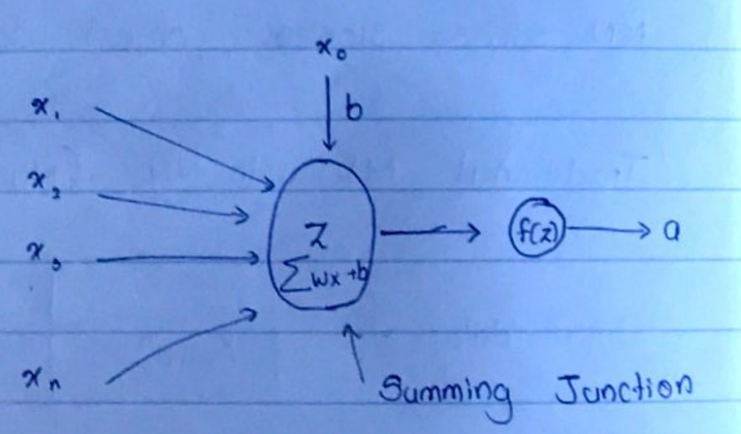
Deep Learning with Neural Network (Deep Networks)



Activation function

01.) Linear function $f(x) = x$

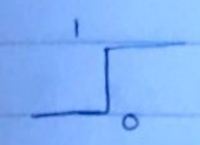




b is weight where $x_0 = 1$ (input of bias)

02.) Threshold (step) function - Unit function

→ output one of the two values, most of the time 0, 1



- step

step is centered at 0. This can be changed. ex: step at -3

Default step function

$$f(x) = \begin{cases} 1 & ; x \geq 0 \\ 0 & x < 0 \end{cases}$$ threshold 0

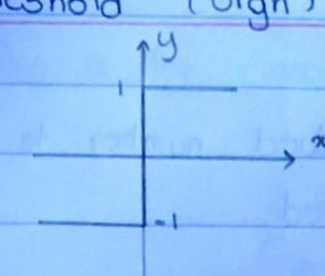
step at -3

$$f(x) = \begin{cases} 1 & ; x \geq -3 \\ 0 & ; x < -3 \end{cases}$$ threshold -3

Two levels represents 2 classes

- Can be use in classification problem

03.) Threshold (sign) function



$$f(x) = \begin{cases} 1 & ; x \geq 0 \\ -1 & ; x < 0 \end{cases}$$ 0

2023
08/17

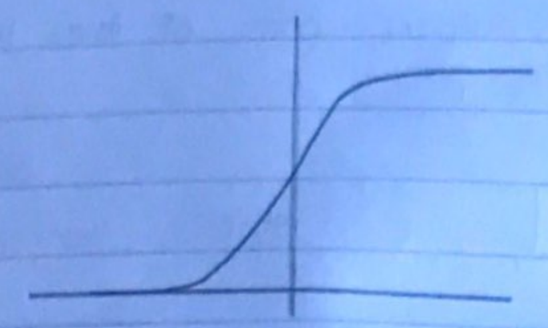
04.) Logistic or Sigmoid

Sigmoid is used in the backpropagation algorithm. The sigmoid non-linearity has mathematical form

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \left(\sigma(x) = \frac{1}{1+e^{-ax}} \; ; \text{ default } a = 1 \right)$$

Scanned with CamScanner

Cross at 0.5.
It takes real-valued number and "squashes" it into range between 0 and 1.



Sigmoid is differentiable
Use for continous output.

Differentiable is requirement for BPA

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right)$$

05.) Tanh (hyperbolic tangent)

$$\tanh(x) = 2\sigma(2x) - 1$$

$$= \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

non-linear function, squashes real-valued number to the range [-1, 1] and output is zero centered.

06.) The Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$

$$f(x) = \begin{cases} x & ; x > 0 \\ 0 & ; x \leq 0 \end{cases}$$

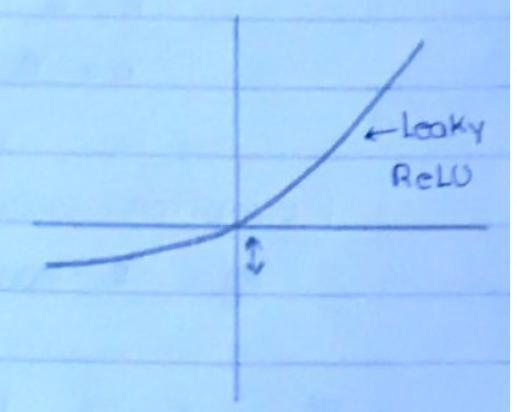
Activation is simply threshold at zero
important in CNN, image processing
Remove negative values.

Removing all negative could be a problem. As a solution we can allow small negative range (leak value). Then the function will be;

$$f(x) = \max(0.01x, x)$$

If $x = -3$

$$f(-3) = \max(-0.03, -3)$$
$$= -0.03$$



07.) Softmax - only for output neurons and classification problems

Softmax converts a vector of k real numbers into a probability distribution of k possible outcomes.

Softmax activation function. $\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ (different catogeries of cat family)

This is for classification multi class problem

Unit - 2
Perceptron

MP neuron + Learning algorithm
perceptron known as the first neuron with learning algorithm.
and it is a basic unit of Neural Network
learning algorithm : how to compute weights

Perceptron as a binary classifier
Data should be linear (separable)

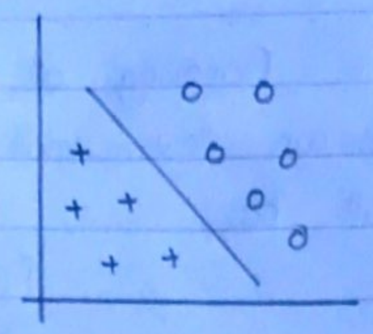
Scanned with CamScanner

problem size = n (# inputs)

$n = 2 \Rightarrow$ line

$n = 3 \Rightarrow$ plane

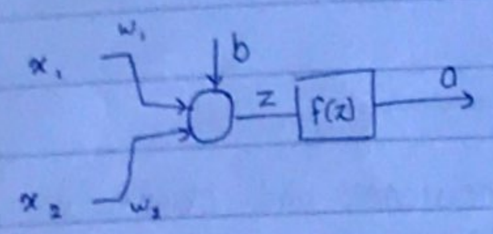
$n = n \Rightarrow$ Hyperplane



Perceptron

eg: 2-input perceptron

Proof by perceptron as binary classifier



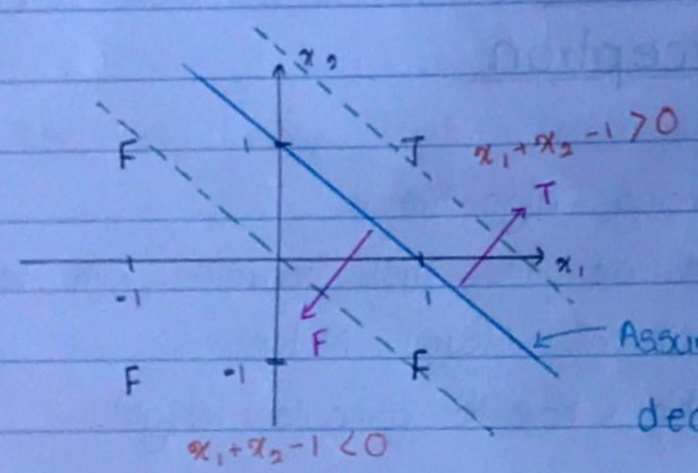
Class 1 = T

Class 2 = F

$$z = x_1 w_1 + x_2 w_2 + b = 0$$

Line in the $x_1 x_2$ space

x_1	x_2	y
-1	-1	F
-1	1	F
1	-1	F
1	1	T



Assume that this is the decision boundry.

~~$y = -x + 1$~~

$$x_1 + x_2 - 1 = 0$$

assume $w_1 = 1$, $w_2 = 1$, $b = -1$ by learning algorithm

$$z = 1 \times x_1 + 1 \times x_2 + (-1)$$
$$= x_1 + x_2 - 1 = 0$$

Proof of the peceptron as a binary classifier.

$$f(z) = \begin{cases} 1 & ; z > 0 \\ -1 & ; z < 0 \end{cases}$$

activation function give the 2 classes

z gives the line equation.

In general

$$z = b + x_1 w_1 + x_2 w_2$$

$$b + x_1 w_1 + x_2 w_2 = 0 \quad \text{or}$$

$$x_2 = -\frac{w_1}{w_2} \cdot x_1 - \frac{b}{w_2}$$

Peceptron as Learning Algorithm

Design to learn a linear decision boundary that seperates 2 classes in a feature space.

Two steps

- Training (fit) ← computing the weights using iterative algorithm
- Prediction (Predict) ← Applying the final weight and compute accuracy and loss (error)

Parameters : Input Data X, y ; $X = \{x_1, x_2, \dots, x_n\}$

y = output label (actual output)

Data : $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_9, y_9)$

Scanned with CamScanner

eg: Data set

	x_1	x_2	y
1	0	0	Y (1)
2	0	1	Y (1)
3	1	0	Y (1)
4	1	1	N (0)

$S = 4$

$n = 2$

Weights : $W = \{w_1, w_2, \ldots, w_n\}$

b = bias

m = no. of iterations

α = learning rate (value between $[0,1]$)

$\Rightarrow 0 < \alpha \leq 1$

z = net input

a = predicted output

Activation function : $f(z) = \begin{cases} 1 ; & z > 0 \\ 0 ; & z \leq 0 \end{cases}$

Algorithm

Initialize the weights, bias to small random values (e.g. zeros)

Until no. of iterations = m or convergence = true

for each input sample

compute $z = W^T x + b$

compute $a = f(z)$

Compute the error $e = y - a$ actual - predicted

update weights and bias

$w_i = w_i + \Delta w_i$

$b = b + \Delta b$

Calculate mean absolute error for all input samples.

Convergence = mean absolute error = 0 or less than threshold value

W^T = transpose of W

Perceptron weight Update rule (delta rule)

Weights

$w_i = w_i + \Delta w_i$

$\Delta w_i = \alpha . (y - a) . x_i$

error is not zero
prediction is incorrect.
∴ weights must be adjusted

Δw_i = learning rate × error of input sample × input (corresponding input)

bias = $\alpha . (y - a)$

Δb = learning rate × error of input sample

Dataset

x_1	x_2	y
0	0	Y (1)
0	0	Y (1)
0	1	Y (1)
1	0	Y (1)
1	1	N (0)

Activation function

$f(z) = \begin{cases} 1 ; & z > 0.5 \\ 0 ; & z \leq 0.5 \end{cases}$

Example

Initialize weights $w_1 = 0$, $w_2 = 0$, $b = 0$, $\alpha = 0.1$

$S = 1$ $x = (0,0)$, $y = 1$

$z = 0 + 0 + 0 = 0 < 0.5$

$a = 0$

$e = 1 - 0 = 1$

weight update

$w_1 = w_1 + (\alpha . e . x_1) = 0$

$w_2 = w_2 + (\alpha . e . x_2) = 0$

$b = b + (\alpha . e) = 0.1$

Scanned with CamScanner

$$MAE = \frac{|1|+|1|+|1|+0}{4} = \frac{3}{4} > 0$$

S.2 $x = (0,1)$ $y = 1$

$z = w_1x_1 + w_2x_2 + b$

$= 0 + 0 + 0.1$

$= 0.1 < 0.5$

$a = 0$

$e = 1 - 0 = 1$

weight update

$w_1 = w_1 + \alpha . e . x_1 = 0 + 0 = 0$

$w_2 = w_2 + \alpha . e . x_2 = 0 + 0.1 \times 1 \times 1 = 0.1$

$b = b + \alpha . e = 0.1 + 0.1 = 0.2$

S.3 $x = (1,0)$ $y = 1$

$z = w_1x_1 + w_2x_2 + b$

$= 0 + 0 + 0.2$

$= 0.2 < 0.5$

$a = 0$

$e = 1 - 0 = 1$

weight update

$w_1 = w_1 + \alpha . e . x_1 = 0.1$

$w_2 = w_2 + \alpha . e . x_2 = 0.1$

$b = b + \alpha . e = 0.3$

S.4 $x = (1,1)$ $y = 0$

$z = w_1x_1 + w_2x_2 + b$

$= 0.1 + 0.1 + 0.3$

$= 0.5$

$a = 0$

$e = 0 - 0$

run until it converge.

23 08 2023

Unit 3

Back propagation Algorithm

MLPs
CNNs
RNNs
Autoencoders → GANs

} BPA the learning algorithm for all these networks.

RNN → Sequential data → time series
Feedforward ↗ Text
Speech
Video

CNN → Topographical data → define with location value
(Image data)

Auto encoders :- Data transformation, Data Generation
GANs

Classical ML model :- Decesion trees, SVM

In 3-5-5-4 MLP (fully connected feedforward Network)

Total weights $= (3 \times 5) + (5 \times 5) + (5 \times 4)$

$= 15 + 25 + 20$

$= 60$

bias $= 5 + 5 + 4$

$= 14$

XOR problem - Classic non linear ~~problem~~ data

MLP works with non linear data

Scanned with CamScanner

Steps of BPA

Initialize the network - weight, learning rate, optimizer, # epochs, batch size

For each epoch

For each input example

1.) Forward propagation

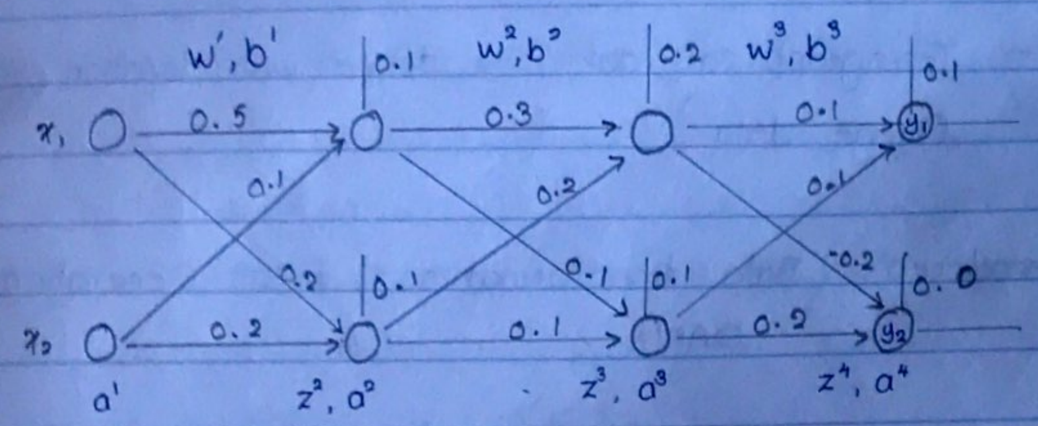
calculate the output

2.) Backward propagation

calculate the local gradient

amount of weights update

3.) Update the weights based on weight update method



$X = \{x_1, x_2\}$

$Y = \{y_1, y_2\}$

Trained using BPA with batch gradient descent.

Input example

	x_1	x_2	y_1	y_2
x^1	1	1	0	1

activation function ReLU

α - learning rate will not be required.
($\because$ batch gradient descent)

Data set

$x_1 \quad x_2 \quad x_3 \quad y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5$

$x' \quad y'$

(x^i, y^i) i^{th} input of example

$W^{(l)}_{ij}$ weight from j to i in l^{th} level

$a^1 = x$

$z^2 = w^1 a^1 + b^1$

Matrix multiplication

$a^2 = g(z^2) \qquad z^3 = w^2 a^2 + b^2$

$a^3 = g(z^3) \qquad z^4 = w^3 a^3 + b^3$

$a^4 = g(z^4)$ $\qquad$ g : activation function

$$\underset{j}{\overset{l}{\bullet}} \xrightarrow{W^l_{ij}} \underset{i}{\overset{l+1}{\bullet}}$$

l - Layer number

weights going out of layer l

W_{ij} - weights coming from j to i

2023
08/31

a^l - inputs to the neurons of layer $l+1$

$h_{w,b}(x) = a^4 = g(z^4)$ $\longleftarrow$ final output of network

hypothesis (prediction)

prediction of x when $w = W$ and $b = b$

$$\boxed{\begin{aligned} z^{(l+1)} &= W^{(l)} a^{(l)} + b^{(l)} \\ a^{(l+1)} &= g(z^{(l+1)}) \end{aligned}}$$

Scanned with CamScanner

we have to calculate loss after a_4

loss = actual - prediction

Loss function

01. Mean squared error (Average squared error)

$$\frac{\sum \text{error}^2}{n}$$

02. Cross entropy loss

Cost (Error/Loss) Function = $J(w, b; x, y)$

↑ single example

Loss when weight matrix w

bias is b

Squared error

$$J(w, b; x, y) = \frac{1}{2} \| h_{w,b}(x_i) - y \|^2$$

actual output

predicted output

In the future steps there is 2. ½ is there to cancel it out

↑ discuss later

Mean squared error

$$J(w, b) = \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \| h_{w,b}(x^i) - y^{(i)} \|^2 \right) \right]$$

Regularization term $\frac{\lambda}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(w_{ji}^{(l)} \right)^2$

to avoid too much fit on given data (overfitting)

λ - momentum / regularization coeffitien

↑ use previous weights

$J(w, b)$ = MSE + regularization

Cross Entropy

Single example (x, y)

$$J(w, b; x, y) = -\left[(1-y) \log (1 - h_{w,b}(x)) + y \log h_{w,b}(x) \right]$$

actual

predicted

entropy

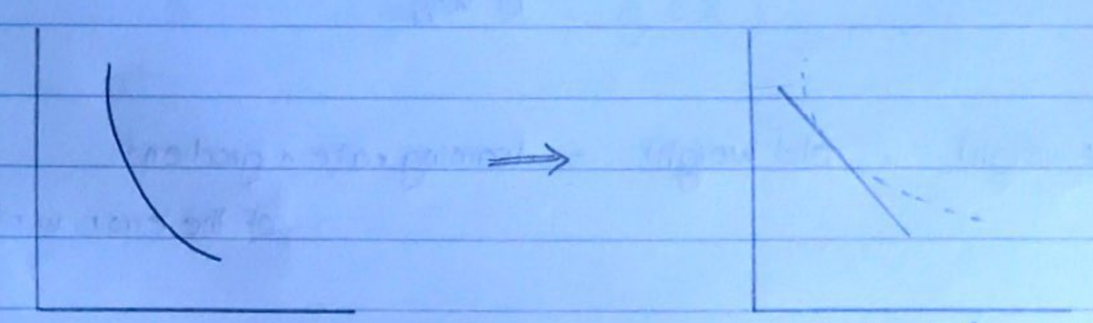
entropy ∴ how the predicted ~~de~~ diviated from actual

$$J(w, b) = \left[\frac{1}{m} \sum_{i=1}^{m} - \left[(1-y^i) \log (1 - h_{w,b}(x^i)) + y^i \log h_{w,b}(x^i) \right] \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} w_{ji}^{(l)\,2}$$

We have to ~~me~~ minimize the loss function and update weights and bias.

Gradient of the Error

$\Delta_{ij}^{(l)}$ = Gradient of the error with respect to the weight w_{ij} of layer l



Loss function

Gradient (tangent)

$$\Delta_{ij}^{(l)} = \frac{\partial}{\partial w_{ij}^{(l)}} J(w, b)$$

Scanned with CamScanner

$$j^{\,l} \xrightarrow[\Delta^{l}_{ij}]{w^{l}_{ij}} i^{\,l+1}$$

01. Stochastic gradient descent
02. Batch gradient descent
03. Mini-batch gradient descent

Weight update

$$w^{l}_{ij} = w^{l}_{ij} + \Delta^{l}_{ij}$$ ← This will be calculated for all training example anyway

These 3 techniques are when to do update

01 → for each example (after each example)

02 → wait until all example process, accumulate gradient after processing update weights with average accumulated gradient. (This for whole data set)

03 → break down the data set to batches. Then do the weight update by batch by batch.

Weight update rule

$$w^{(l)}_{ij} = w^{(l)}_{ij} - \alpha \frac{\partial}{\partial w^{(l)}_{ij}} J(w,b)$$ ← chain rule

new weight = old weight − learning rate × gradient of the error w.r.t w_{ij}

After applying chain rule in calculus

$$\Delta^{(l)}_{ij} = \frac{\partial J(w,b)}{\partial w^{(l)}_{ij}} = a^{(l)}_{j}\, \delta^{(l+1)}_{i}$$

where $\delta^{(l+1)}_{ij}$ = local gradient of ith neuron of layer $l+1$

a_j = activation fun^n

$$j^{\,l} \xrightarrow[\Delta^{l}_{ij}]{w^{l}_{ij}} i^{\,l+1}$$

neuron / input → j; neuron → i

$a^{(l)}_{j}$ = output (activation) of jth neuron

a^{l}_{j} $\quad a^{l+1}_{i}$

δ^{l+1}_{i}

$$\Rightarrow \quad w^{l}_{ij} = w^{l}_{ij} - \underbrace{\alpha\, a^{l}_{j}\, \delta^{l+1}_{i}}$$

Similar to perceptron learning rule (delta rule)

δ - contribution to the total error

In general $\delta^{L} = -(y - a^{(L)})\, g'(z^{L})$

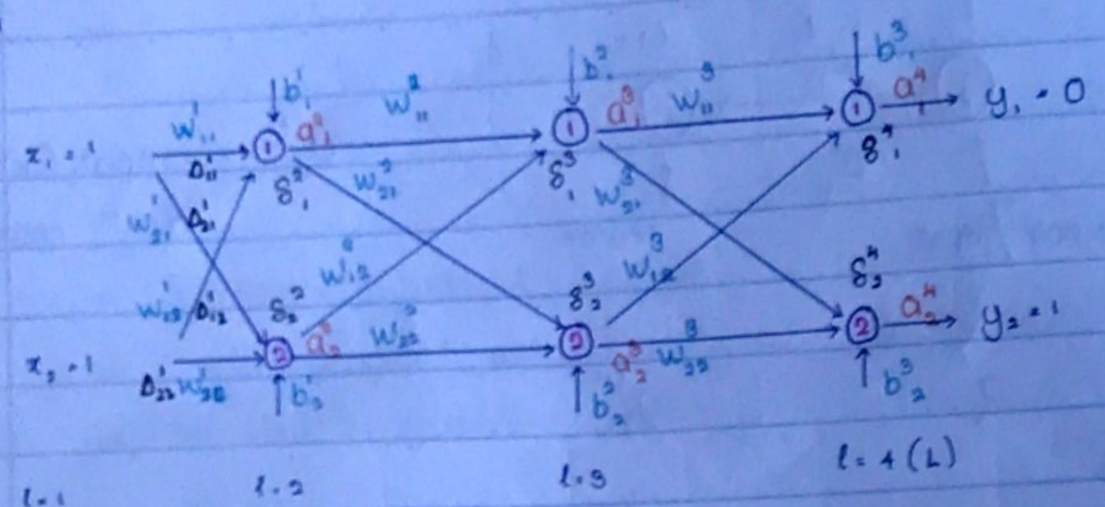
$$\delta^{4} = -(y - a^{4})\, g'(z^{(4)})$$
$$\delta^{3} = (w^{(3)})^{T} \delta^{4} \cdot g'(z^{3})$$
$$\delta^{2} = (w^{2})^{T} \delta^{3} \cdot g'(z^{2})$$

$$\delta^{l} = (w^{l})^{T} \delta^{l+1} \cdot g'(z^{l})$$

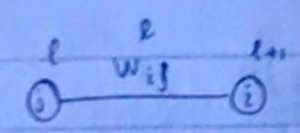
Understood up until now.

Scanned with CamScanner

2023
09/04



01.) Label the network



weight going out from l

$w_{11}^1 = 0.5$	$w_{11}^2 = 0.3$	$w_{11}^3 = 0.1$
$w_{21}^1 = 0.2$	$w_{21}^2 = 0.1$	$w_{21}^3 = -0.2$
$w_{12}^1 = 0.1$	$w_{12}^2 = 0.2$	$w_{12}^3 = 0.1$
$w_{22} = 0.2$	$w_{22} = 0.1$	$w_{22}^3 = 0.2$
$b_1^1 = 0.1$	$b_1^2 = 0.2$	$b_1^3 = 0.1$
$b_2^1 = 0.1$	$b_2^2 = -0.1$	$b_2^3 = 0.0$

02.) Forward propagation

Calculate z, a

$a_1^2 = g(z_1^2) = ReLU(z_1^2)$

$z_1^2 = 0.5 \times 1 + 1 \times 0.1 + 0.1 = 0.7$

$a_1^2 = ReLU(0.7)$

$= 0.7$

$a_2^2 = 0.5$

$a_1^3 = ReLU(z_1^3)$

$z_1^3 = 0.7 \times 0.3 + 0.5 \times 0.2 + 0.2$

$= 0.21 + 0.1 + 0.2 = 0.51$

$a_1^3 = 0.51$

Can use Matrix multiplication

$a^2 = g(z^2)$

$z^2 = \cancel{a^1 w^1 + b^1} \quad w^1 a^1 + b^1$

$$= \begin{bmatrix} 0.5 & 0.1 \\ 0.2 & 0.2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.5 \end{bmatrix}$$

$$a^2 = ReLU \begin{bmatrix} 0.7 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.5 \end{bmatrix}$$

$a^3 = g(z^3)$

$z^3 = w^3 a^2 + b^3$

$$= \begin{bmatrix} 0.3 & 0.2 \\ 0.1 & 0.1 \end{bmatrix} \begin{bmatrix} 0.7 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.31 \\ 0.12 \end{bmatrix} + \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.51 \\ 0.02 \end{bmatrix}$$

$$a^3 = ReLU \begin{bmatrix} 0.51 \\ 0.02 \end{bmatrix} = \begin{bmatrix} 0.51 \\ 0.02 \end{bmatrix}$$

Scanned with CamScanner

$a^4 = g(z^4)$

$$z^4 = w^3 a^3 + b^3$$

$$= \begin{bmatrix} 0.1 & 0.1 \\ -0.2 & 0.2 \end{bmatrix} \begin{bmatrix} 0.51 \\ 0.02 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.0 \end{bmatrix}$$

$$= \cancel{\begin{bmatrix} 0.072 \\ -0.060 \end{bmatrix}} + \cancel{\begin{bmatrix} 0.1 \\ 0.0 \end{bmatrix}} = \cancel{\begin{bmatrix} 0.172 \\ -0.06 \end{bmatrix}}$$

$$\cancel{a^4 = \mathrm{ReLU}\begin{bmatrix} 0.172 \\ -0.06 \end{bmatrix} = \begin{bmatrix} 0.172 \\ 0 \end{bmatrix}}$$

$$= \begin{bmatrix} 0.153 \\ \cancel{0.149} \\ -0.098 \end{bmatrix}$$

$$a_4 = \mathrm{ReLU}\begin{bmatrix} 0.153 \\ \cancel{0.149} \\ -0.1 \end{bmatrix} = \begin{bmatrix} 0.153 \\ \cancel{0.149} \\ 0 \end{bmatrix}$$

* Output Layer (Matrix form)

$$\delta^{(4)} = -(y - a^4)\, g'(z^4)$$

In general

$$\cancel{\delta^l \rightarrow w^{l\,(T)}} \qquad \delta^l = (w^l)^T \delta^{l+1} * g'(z^l)$$

This means element-wise multiplication

Backward Propagation

1.) Calculate local gradient

$$① \xrightarrow{a^4_1} y'_1 \qquad \delta^4_1 = -(y'_1 - a^4_1)\,.\,\mathrm{ReLU}'(z^4_1)$$

$$② \xrightarrow{a^4_2} y'_2 \qquad = -(0 - 0.153)\,.\,\mathrm{ReLU}'(0.153)$$

$$= 0.153 \times 1$$

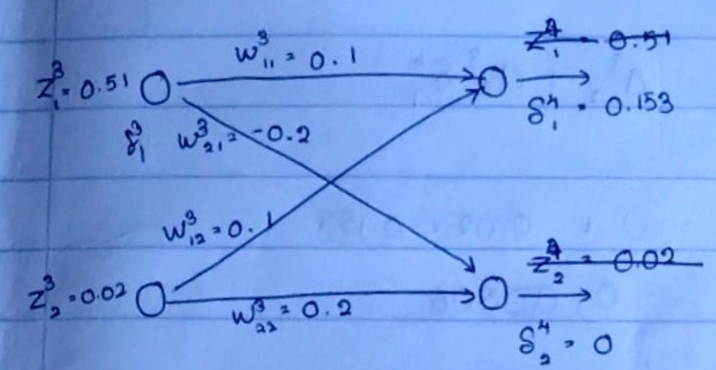
$$= 0.153$$

$$\mathrm{ReLU}'(z) = \begin{cases} 1 & ; z > 0 \\ 0 & ; z \le 0 \end{cases}$$

$$\delta^4_2 = -(1 - 0)\,.\,\mathrm{ReLU}'(0)$$

$$= 0$$

Local gradient of hidden layers



$$\delta^3_1 = \{ w^3_{11} \times \delta^4_1 + w^3_{21} \times \delta^4_2 \}\, \mathrm{ReLU}'(z^3_1)$$

$$\{ 0.1 \times 0.153 + (-0.2) \times 0 \}\, \mathrm{ReLU}'(0.51)$$

$$= 0.0153 \times 1$$

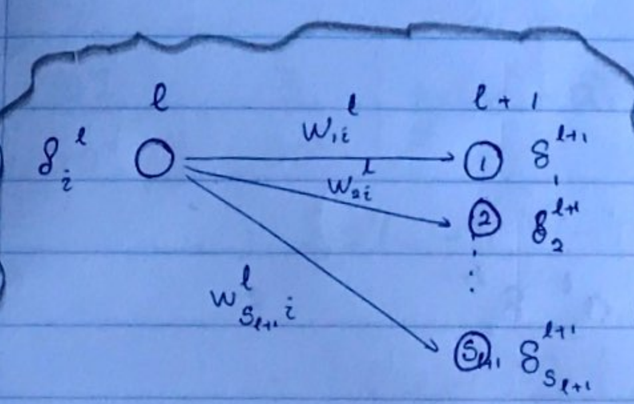
$$= 0.0153$$

$$\delta^3_2 = \{ w^3_{12} \times \delta^4_1 + w^3_{22} \times \delta^4_2 \}\, \mathrm{ReLU}'(z^3_2)$$

$$\{ 0.1 \times 0.153 + (0.2) \times 0 \}\, \mathrm{ReLU}'(0.02)$$

$$= 0.0153 \times 1$$

$$= 0.0153$$



$$\delta^l_i = \left[w^l_{1i} \cdot \delta^{l+1}_1 + w^l_{2i} \cdot \delta^{l+1}_2 + \dots + w^l_{S_{l+1}i}\, \delta^{l+1}_{S_{l+1}} \right] . \, g'(z^l)$$

$$= \left[\sum_{j=1}^{S_{l+1}} w_{ji}\, \delta^{l+1}_j \right] . \, g'(z^l)$$

$$\delta^3 = \begin{bmatrix} 0.1 & -0.2 \\ +0.1 & 0.2 \end{bmatrix} \begin{bmatrix} 0.153 \\ 0 \end{bmatrix} * \mathrm{ReLU}' \begin{bmatrix} 0.51 \\ 0.02 \end{bmatrix}$$

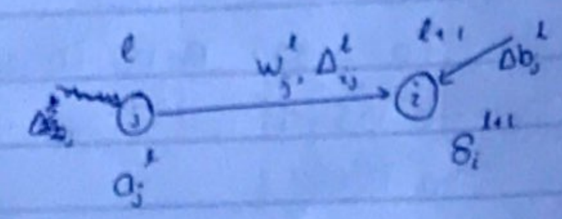
$$= \begin{bmatrix} 0.0153 \\ 0.0153 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.0153 \\ 0.0153 \end{bmatrix}$$

Scanned with CamScanner

$$W^{l} = \begin{bmatrix} w^{l}_{11} & w^{l}_{12} \\ w^{l}_{21} & w^{l}_{22} \end{bmatrix}$$

Update the gradients (Δ)



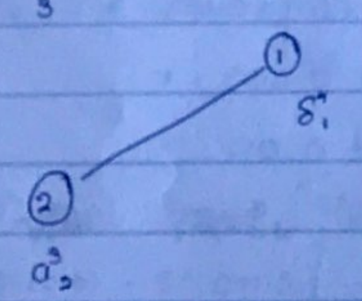
i - neuron

j - neuron / input

for weight $\Rightarrow$ $\Delta^{l}_{ij} = \Delta^{l}_{ij} + a^{l}_{j}\,\delta^{l+1}_{i}$

for bias $\Rightarrow$ $\Delta b^{l}_{j} = \Delta b^{l}_{j} + 1\,\delta^{l+1}_{i}$

Δ^{3}_{12}



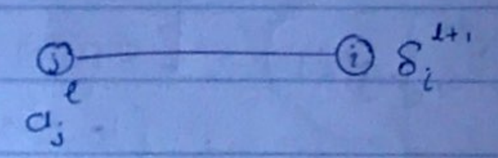
$$\Delta^{3}_{12} = \Delta^{3}_{12} + a^{3}_{2}\,\delta^{4}_{1}$$

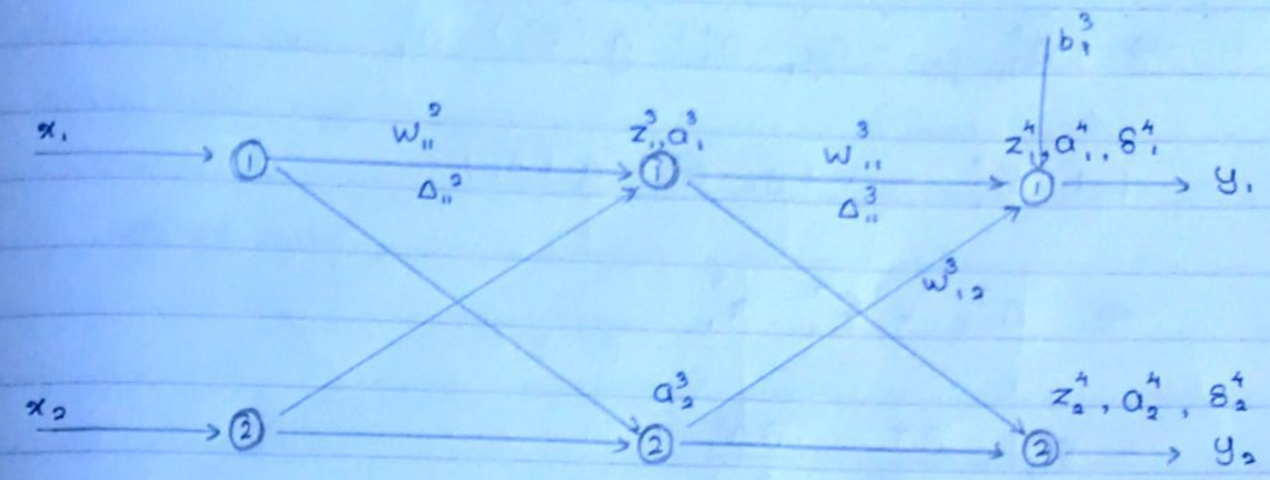
$$= 0 + 0.02 \times 0.153$$

$$= 0.00306$$

3/3 Proof of BPA

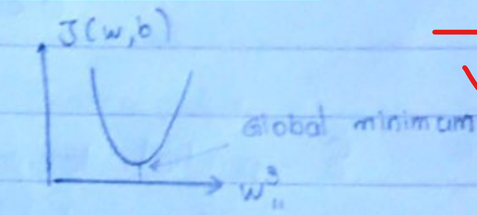
$$\Delta^{l}_{ij} = \frac{\partial}{\partial w^{l}_{ij}} J(w,b) = a^{l}_{j}\,\delta^{l+1}_{i}$$





$$\Delta^{3}_{11} = \delta^{4}_{1}\,a^{3}_{1} \qquad \Big| \qquad \delta^{4}_{1} = (y_1 - a^{4}_{1}) \cdot g'(z^{4}_{1})$$

$$\Delta^{3}_{11} = \frac{\partial}{\partial w^{3}_{11}} J(w,b)$$



$$J(w,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \| h_{w,b}(x^{i}) - y^{i} \|^{2} \right) \right]$$

$$\Delta^{3}_{11} = \frac{\partial J(w,b)}{\partial w^{3}_{11}} = \frac{\partial J}{\partial a^{4}_{1}} \cdot \frac{\partial a^{4}_{1}}{\partial z^{4}_{1}} \cdot \frac{\partial z^{4}_{1}}{\partial w^{3}_{11}}$$

$$J = \frac{1}{2}\left[(y_1 - a^{4}_{1})^{2} + (y_2 - a^{4}_{2})^{2} \right]$$

$$z^{4}_{1} = w^{3}_{11}\,a^{3}_{1} + \dots$$

$$\frac{\partial J}{\partial a^{4}_{1}} = \frac{\partial}{\partial a^{4}_{1}} \left\{ \frac{1}{2} (y_1 - a^{4}_{1})^{2} + (y_2 - a^{4}_{2})^{2} \right\}$$

$$= \frac{1}{2} \left\{ 2(y_1 - a^{4}_{1})(-1) + \cancel{2(y_2 - a^{4}_{2})(0)} \right\}$$

$$= -\{ (y_1 - a^{4}_{1}) \}$$

$$\frac{\partial a^{4}_{1}}{\partial z^{4}_{1}} = \cancel{\partial \{ \mathrm{ReLU}(z^{4}_{1}) } = \frac{\partial (g(z^{4}_{1}))}{\partial z^{4}_{1}} = g'(z^{4}_{1})$$

$$\frac{\partial z^{4}_{1}}{\partial w^{3}_{11}} = \frac{\partial (w^{3}_{11} a^{3}_{1} + w^{3}_{12} a^{3}_{2} + b^{3}_{1})}{\partial w^{3}_{11}} = a^{3}_{1}$$

Scanned with CamScanner

$$\rightarrow \Delta^3_{11} = -(y_1 - a^4_1) \cdot g'(z^4_1) \cdot a^3_1$$

Let $\delta^4_1 = -(y_1 - a^4_1) \cdot g'(z^4_1)$

$$\rightarrow \Delta^3_{11} = \delta^4_1 \cdot a^3_1$$

$$\delta^4_1 = \frac{\partial J}{\partial a^4_1} \cdot \frac{\partial a^4_1}{\partial z^4_1}$$

$$= \frac{\partial J}{\partial z^4_1}$$

δ = partial derivative of the loss function w.r.t. net input (z)

$$\Delta^3_{11} = \frac{\partial J}{\partial z^4_1} \cdot \frac{\partial z^4_1}{\partial w^3_{11}}$$

we can't use J for hidden layers. It can be only used in output layers.

Proof of Hidden layer Δs $\Rightarrow$ homework

RNN (Recurrent Neural Network)

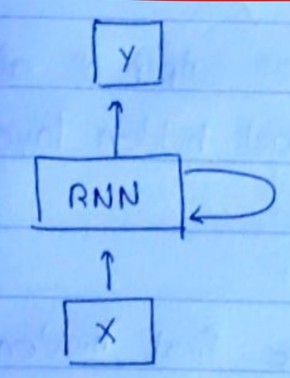
Sequential data

1.) Time series data

2.) Languages.

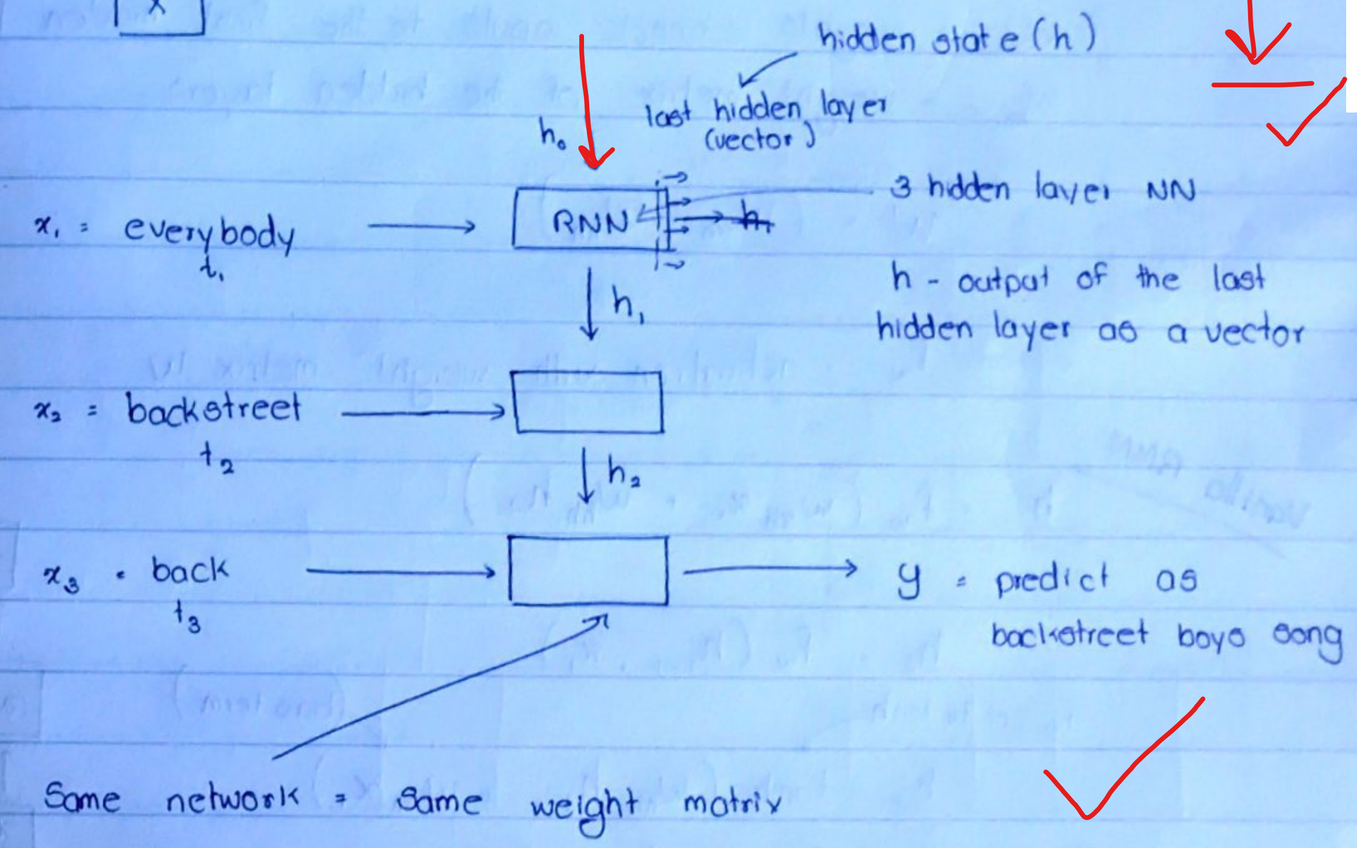
Sequential data
(non-fixed length)
video
weather data

Non-sequential data
(Fixed length)
An image

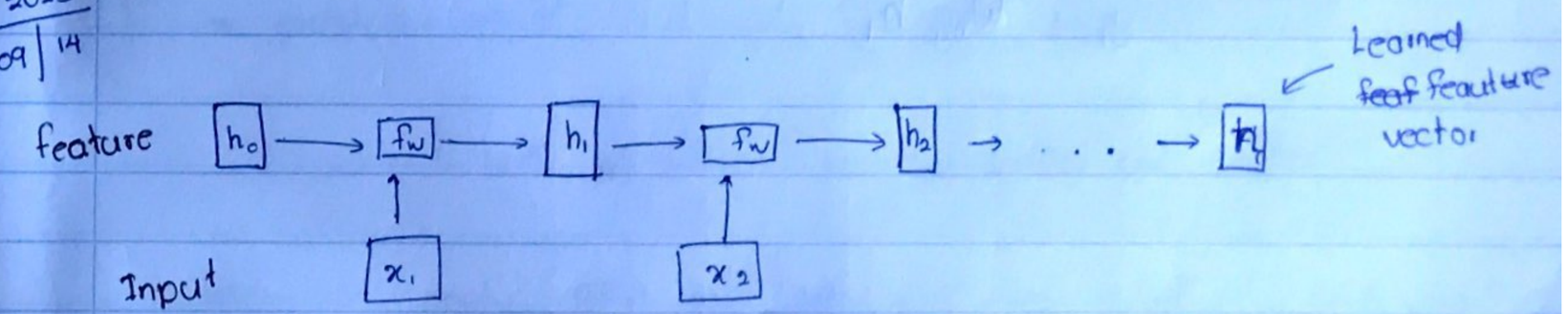


Key idea: RNNs have an "internal state" that is updated as a sequence is processed.

ex:

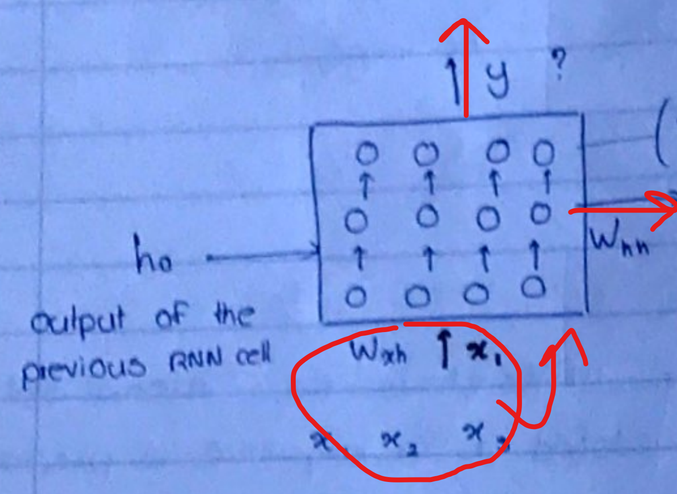


2023
09 | 14



feature Learning part of the RNN

RNN cell



$(h_1^1, h_1^2, h_1^3, h_1^4) = h_1$ vector

$h_1 = f_w(W_{xh} x_1 + W_{hh} h_0)$

combines output of all neuron of the last hidden layer

W_{xh} = weights connects inputs to the first hidden layer

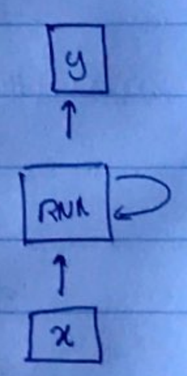
W_{hh} = weight matrix of the hidden layers.

$$W = (W_{xh}, W_{hh})$$

f_w = activation with weight matrix W

Vanilla RNN

$$h_1 = f_w(W_{xh} x_1 + W_{hh} h_0)$$



$$h_t = f_w(h_{t-1}, x_t)$$

f_w set to tanh

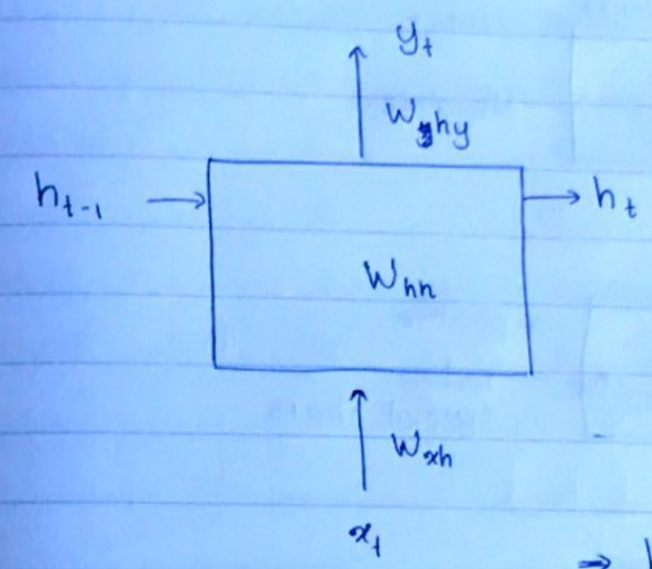
(bias term)

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

If you need to compute output

$$y_t = W_{hy} h_t$$

Vanilla RNN cell

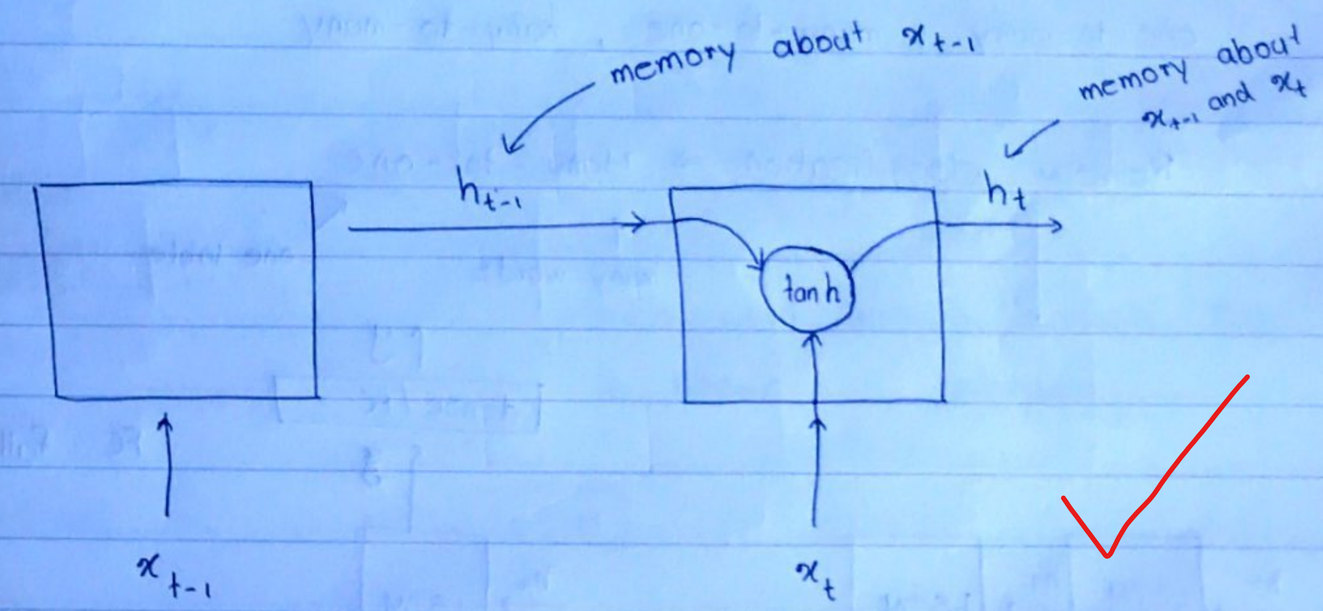


$$W = (\overset{\sigma}{W_{xh}}, \overset{\mu}{W_{hh}}, \overset{\nu}{W_{hy}})$$

$$h_t = f_w(W_{xh} \cdot x_t + W_{hh} h_{t-1})$$

critical activation function we use is tanh

$$\Rightarrow h_t = \tanh_w(W_{xh} x_t + W_{hh} h_{t-1})$$



This has a problem when we consider about backpropagation problem called 'vanishing gradient' problem.

δ, Δ values become smaller and smaller

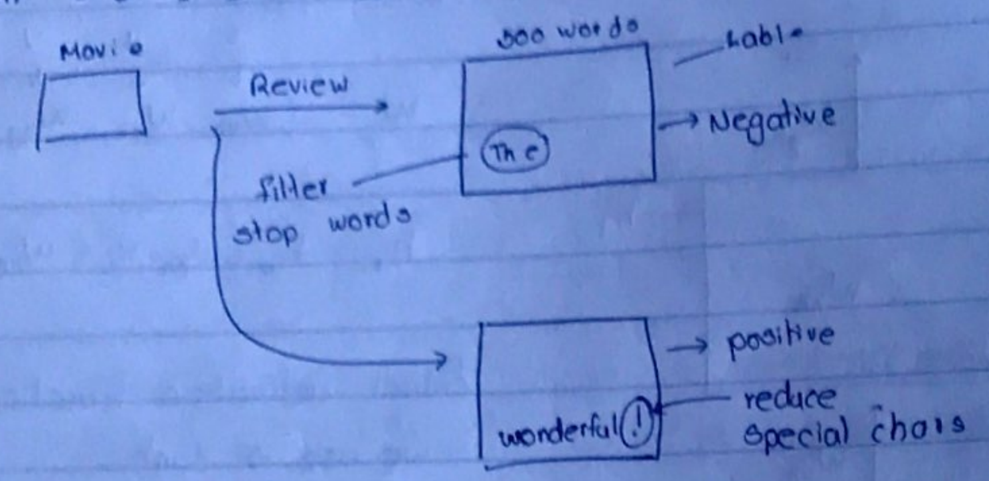
when we apply BPA to RNN cell we called it BPA with (BPTT).

To solve this problem we use LSTM cell.

Scanned with CamScanner

20 09 2023

In ~~IMB~~ IMDB Movie review example (classification model)



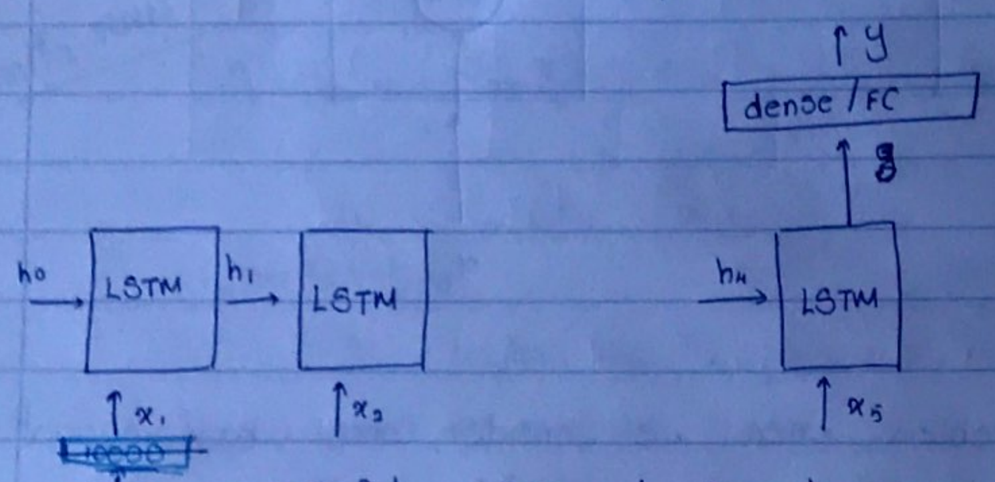
RNN Structure (Architecture

one-to-many , many-to-one , many-to-many

Review classification ⇒ Many-to-one

many words

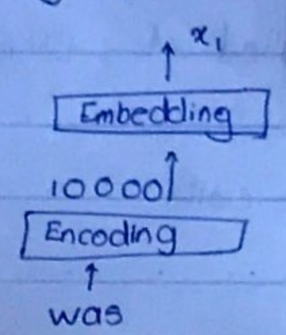
one lable



FC = fully connected

Language translation = many-to-many

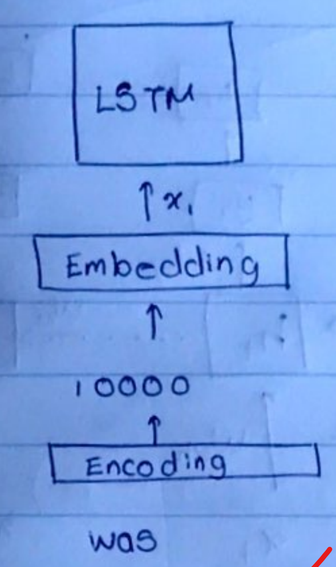
Language Classification = many-to-one



encoding = one-hot encoding

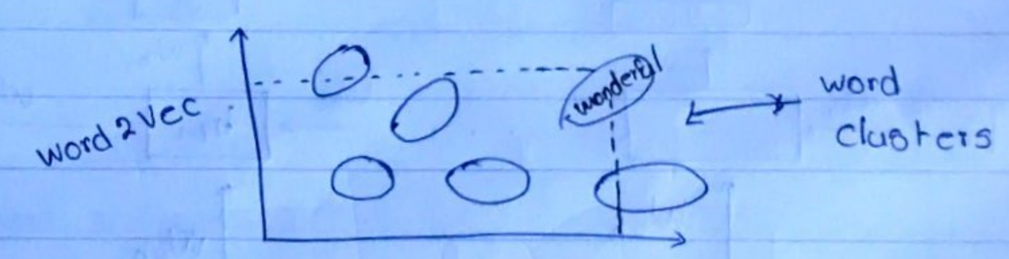
one-hot encoding

	was ~~wonderful~~	wonderful	movie	hands	down
was	1	0	0	0	0
wonderful	0	1	0	0	0
movie	0	0	1	0	0
hands	0	0	0	1	0
down	0	0	0	0	1



word Embedding

wonderful → 1 3 4 16 7 8 → what is this



each word has a location. This is known as embedding

word2Vec

BoW

TF - IDF

← change this to GRU

For RNN training : old backpropagation but with a slight twist

Loss function

MLP training

$L = J(w, b)$

$$\Delta w_{ij} = \frac{\partial J(w,b)}{\partial w_{ij}}$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

Scanned with CamScanner

In RNN $W \rightarrow W_{xh}$, $W \rightarrow W_{hh}$

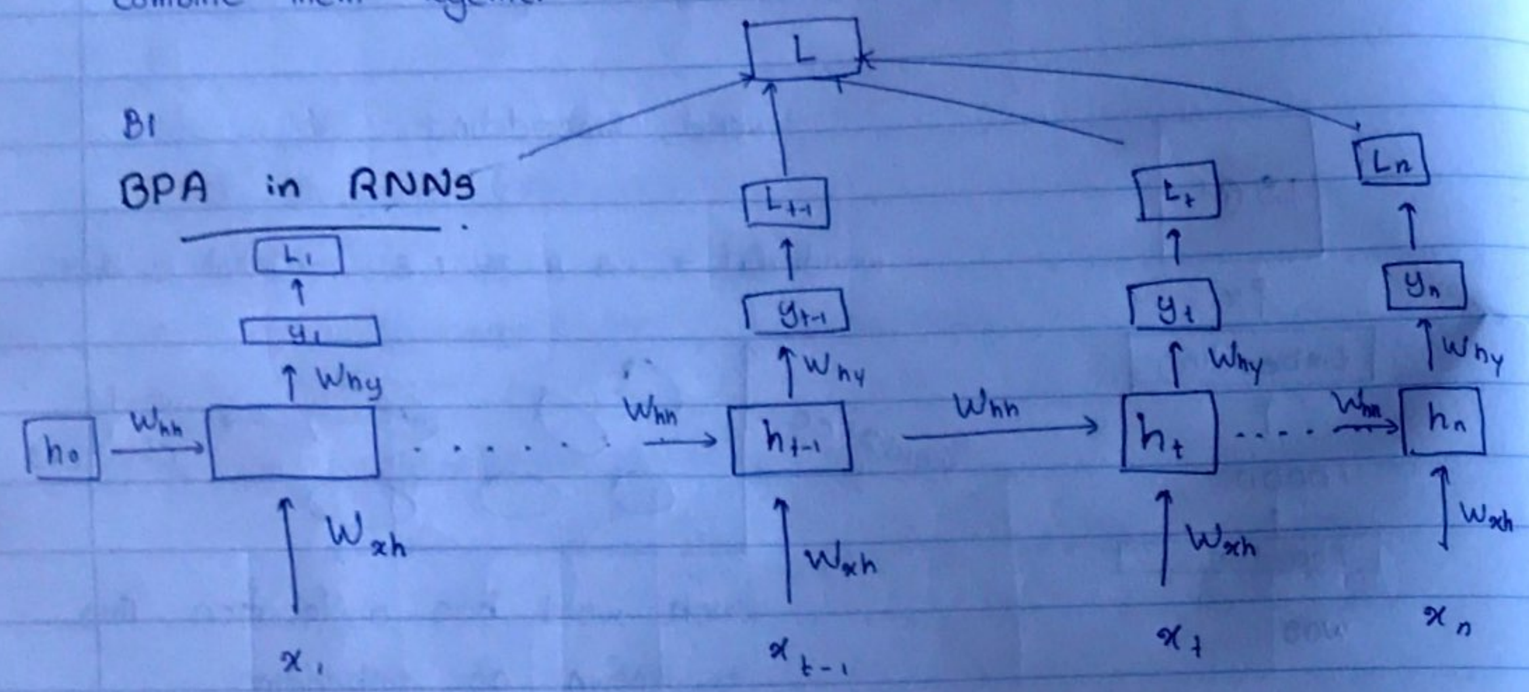
how to adjust the weight of these 2 metrices

(BPA in RNN)

We generate Loss function of each output then we combine them together

BI

BPA in RNNs



$$L = \sum_{i=1}^{n} L_i$$

We need to update the weight of W_{xh}, W_{hh}, W_{hy}

~~Ut~~ Update weights

Update each weight of W_{xh}, W_{hh}, W_{hy}

Matrix-wise (In practice we need this for each individual weight)

$$\frac{\partial L}{\partial W_{hh}} = \frac{\partial \left(\sum_{1}^{n} L_i\right)}{\partial W_{hh}}$$ If we apply chain rule

matrix

There is a partial derivative

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$ ← Multiplication of small (Large) gradient values of a long sequence

This long gradient sequences (multiplication) could result in two problems.

1.) Vanishing Gradient (Vanishing memory)

2.) Exploding Gradient

To overcome this ⟹ Gated RNN

1.) LSTM :- Vanishing Gradient

2.) GRU

2023 09/21

Sol^n for

1.) Exploding gradient :- Gradient clipping

2.) Truncated backpropagation through time (TBPTT)

Break the sequence into small parts and apply BPTT

3.) Long short term memory (LSTM) - vanishing gradient

↑ special RNN cell

Long (short term) memory (LSTM)

01.) Improving short term memory

02.) Forgetting content

Ex:

The clouds, which partly flow into each other and hang low, are at s

Vanishing gradient problem (Sentence is long)

Scanned with CamScanner

can forget part

Bob knows swimmig [He told me over the phone that he had] served the navy for four long years

In LSTM we use sigmoid function in forgeting

0 ← forget

1 ← remember

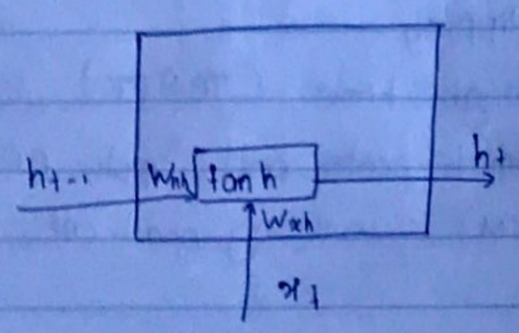
in between ← depends

In LSTM has 3 of these gates

Forget gate
Input gate
Output gate

⟸ + [RNN cell]

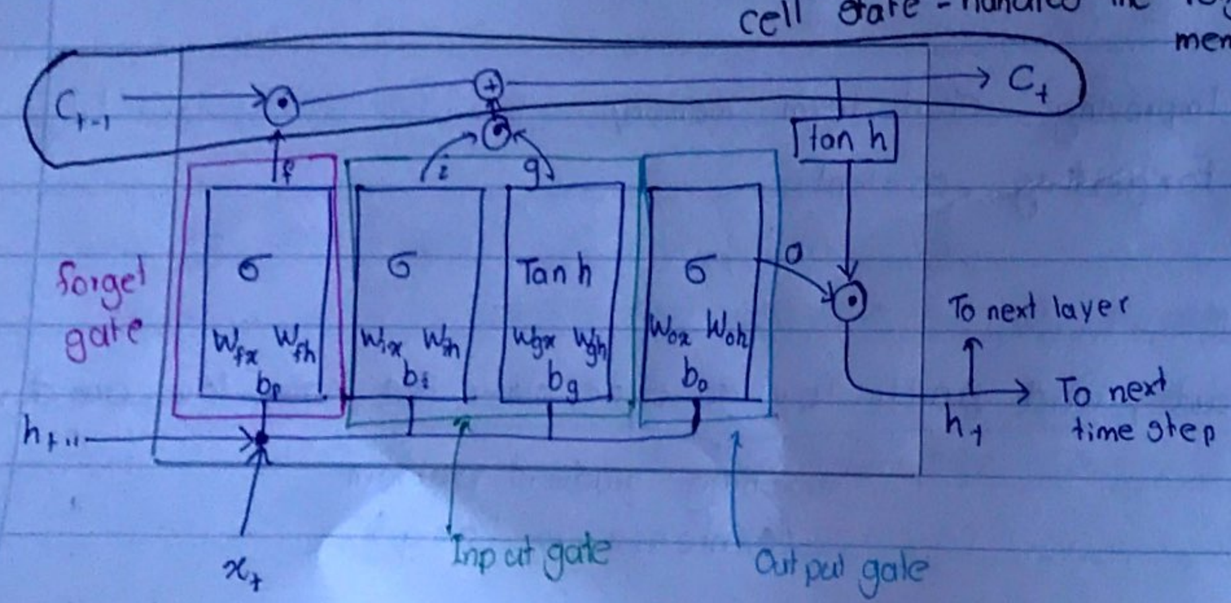
Vanilla RNN cell



$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b)$$

LSTM cell

cell state - handles the long-time memory



⊙ Element wise multiplication

⊕ Element wise addition

We need to learn 8 weight matrices and 4 bias matrices.

$$f_t = \sigma(W_{fx} x_t + W_{fh} h_{t-1} + b_f)$$

$$i_t = \sigma(W_{ix} x_t + W_{ih} h_{t-1} + b_i)$$

$$g_t = \text{Tanh}(W_{gx} x_t + W_{gh} h_{t-1} + b_g)$$

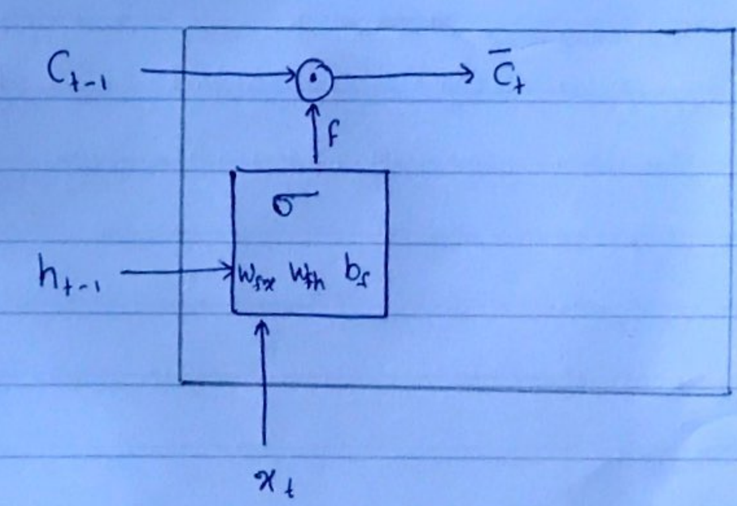
Forget has multiplication because output of f is value between 0-1. If we want to forget we can forget all the things.

$$C_t = (C_{t-1} \odot f_t) \oplus (i_t \odot g_t)$$

$$O_t = \sigma(W_{ox} x_t + W_{oh} h_{t-1} + b_o)$$

$$h_t = \cancel{(C_t \odot O_t)} \quad O_t \odot \tanh(C_t)$$ ← final state of the cell.

Forget Gate

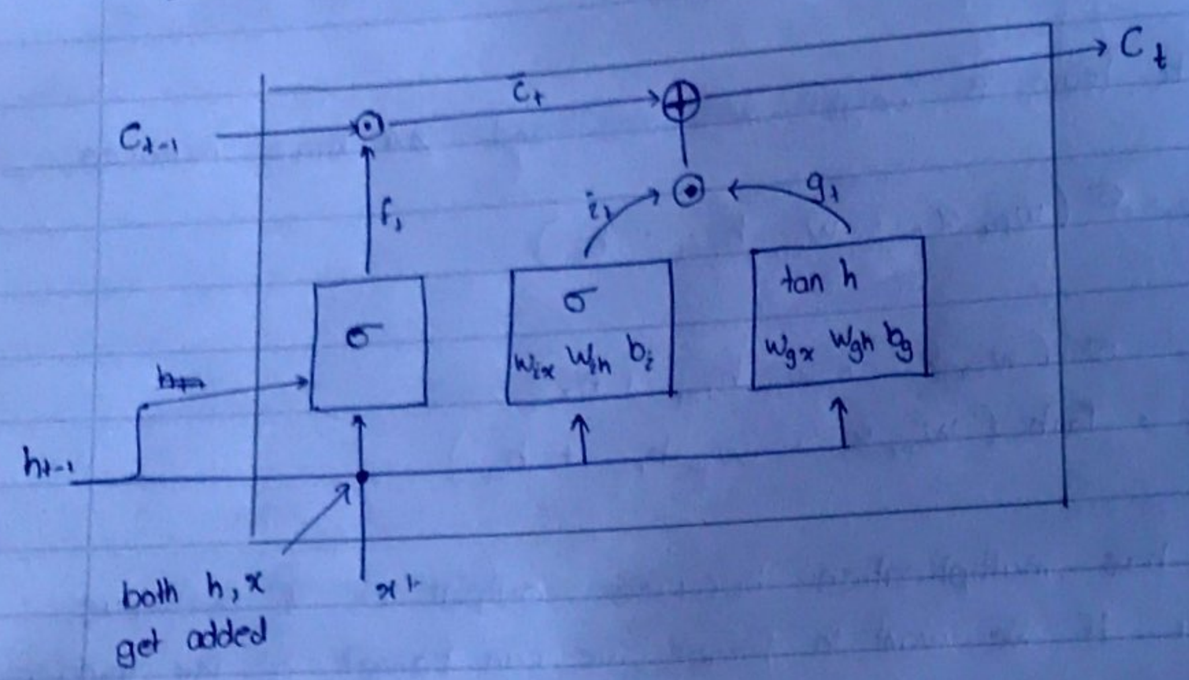


$$f_t = \sigma[W_{fx} x_t + W_{hx} h_{t-1} + b_f]$$

$$\bar{C}_t = f_t \odot C_{t-1}$$

Scanned with CamScanner

Input Gate



$$i_t = \sigma(w_{ix} \cdot x_t + w_{ih} h_{t-1} + b_i)$$

$$g_t = \tanh(w_{gx} \cdot x_t + w_{gh} \cdot h_{t-1} + b_g)$$

$$C_t = \bar{C}_t \oplus (i_t \odot g_t)$$