$$Au - \lambda u = 0$$

$$|A - \lambda I| = 0$$

PCA : Disadvantage :

1 Only used when linear transformation linear

1st Oct

DEEP LEARNING

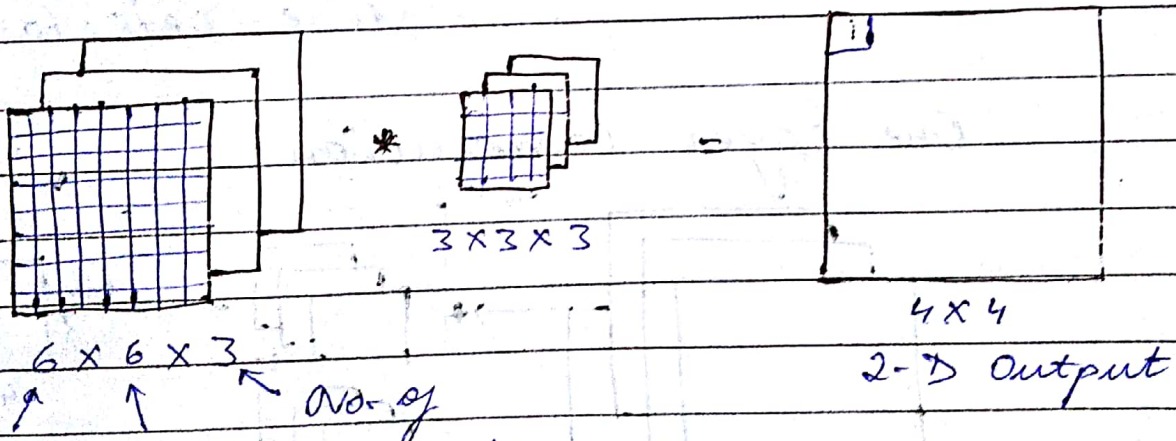
Boltzmann Machine

Auto Encoder

CNN

22nd Oct

Convolution Neural Network



2 → Output

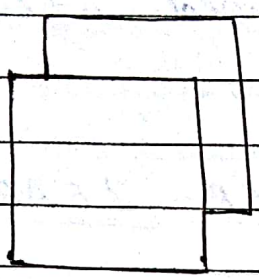
→ 27 elements are multiplied by 27 elements of filter and added to generate 1 element of output.

Vertical Edge :

Get info from all channels

R	G	B
1 0 -1	1 0 -1	1 0 -1
1 0 -1	1 0 -1	1 0 -1
1 0 -1	1 0 -1	1 0 -1

If we use select more no. of filters (vertical &, horizontal), those will be stacked of two 2D image 4×4×2



For 2 filters

$n \times n \times n_c * f \times f \times n_c$

$\Rightarrow (n-f+1) \times (n-f+1) \times n_c'$

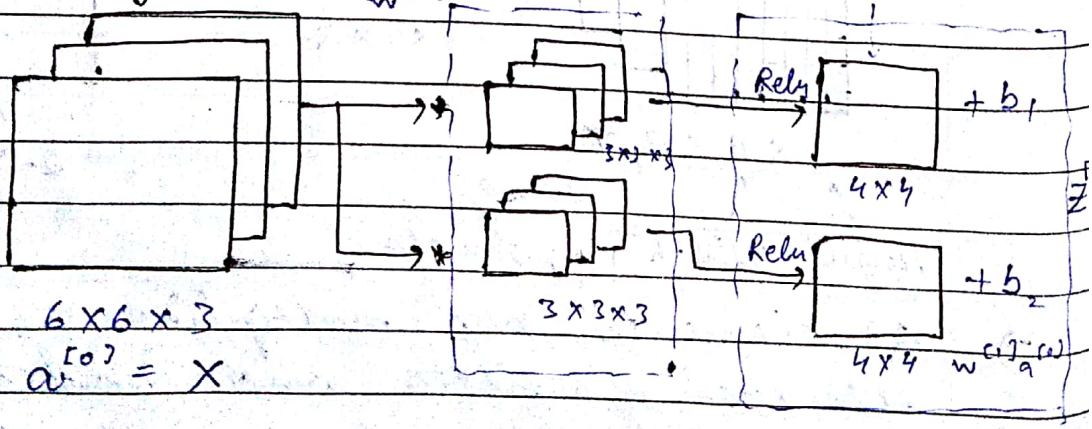
No. of filters

Padding = 0

stride = 1

Feature Maps / Feature Detectors

One Layer Convolution:



$z^{[1]} = w^{[1]} a^{[0]} + b_1$

$a^{[1]} = g(z^{[1]})$

$a^{[0]}$ to $a^{[2]}$

$6 \times 6 \times 3 \rightarrow 4 \times 4 \times 2$ After Convolution

Down Sampling

How many parameters ?

$4 \times 4 \times 10 \Rightarrow (27+1) \times 10$

$= 280$

⇒ Independent of Input image size.
Less Prone to overfitting

⇒ If l no. of layers are in Convolution layer :

$f^{[l]}$ = filter size of l^{th} layer

$p^{[l]}$ = padding " " " "

$s^{[l]}$ = stride " " " "

Input = $n_H^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$

Output : $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$$\therefore n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s} + 1 \right\rfloor$$

$$n_w^{[l]} = \left\lfloor \frac{n_w^{[l-1]} + 2p^{[l]} - f^{[l]}}{s} + 1 \right\rfloor$$

$n_c^{[l]}$ = no. of filters

Each filter is of size : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Activations : $a^{[l]} \longrightarrow n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$A^{[l]} \longrightarrow M \times n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

↓

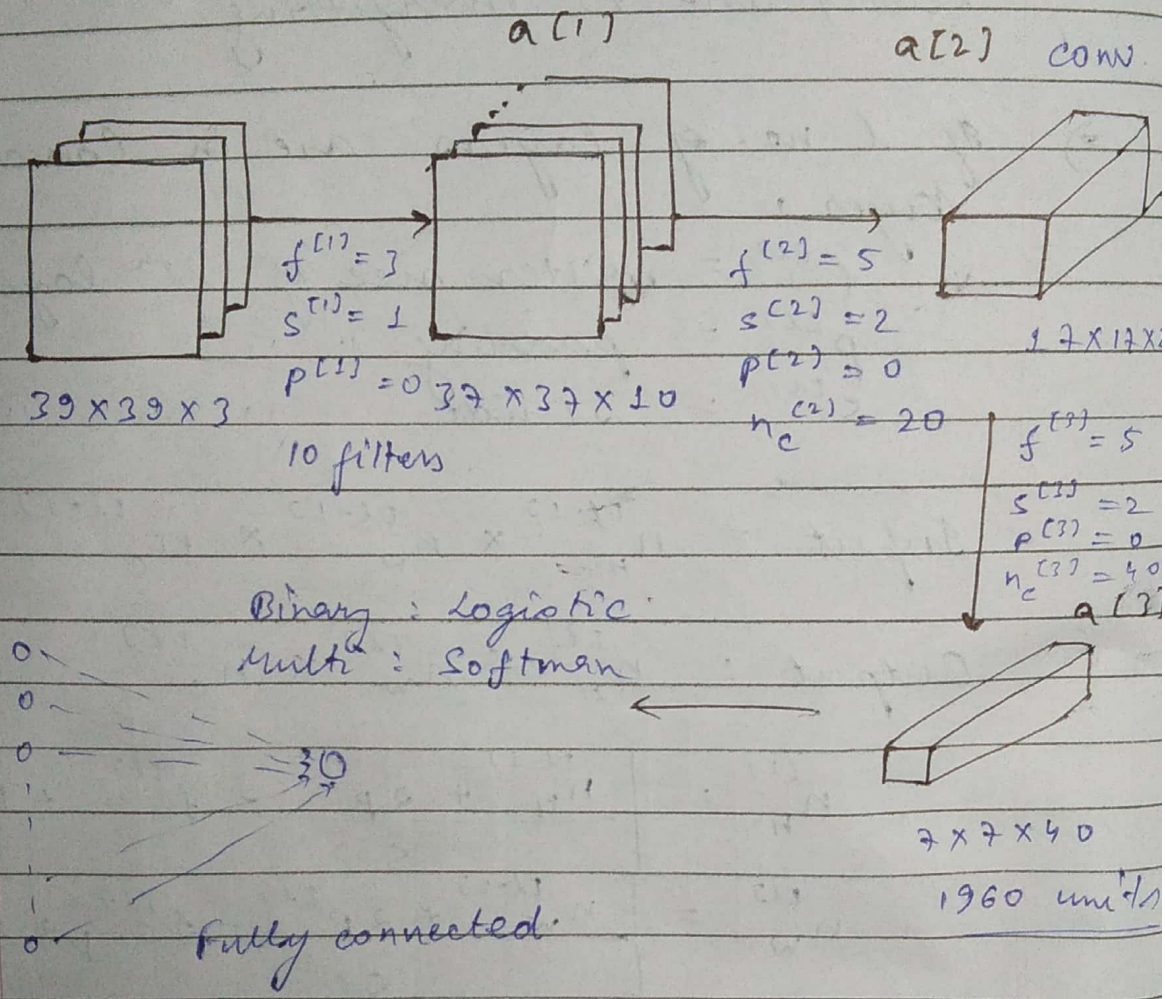
Vector notation

Weight = $f_H^{[l]} \times f_w^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

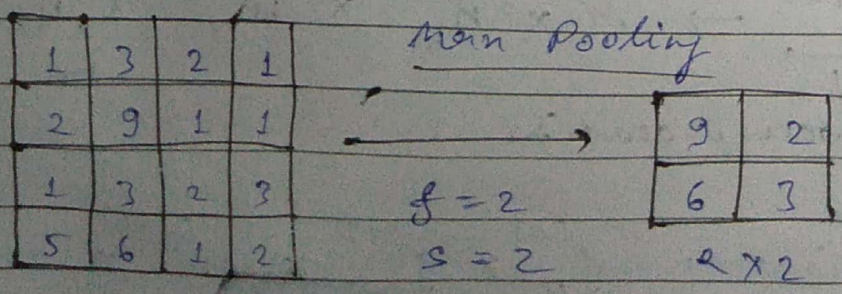
Bias : a vector

$[\,1 \quad 1 \quad 1 \quad n_c^{[1]}\,]$

Deeper Convolution NN :



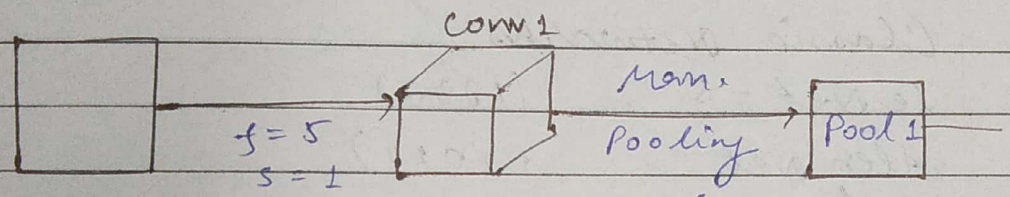
1. Conv layer
2. Pooling layer → Reduce dimension
   Robust feature detector
3. Fully connected layer

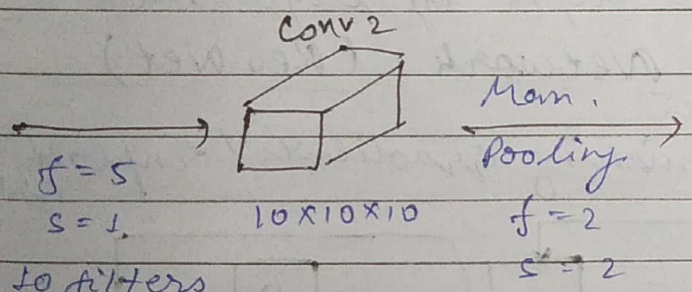


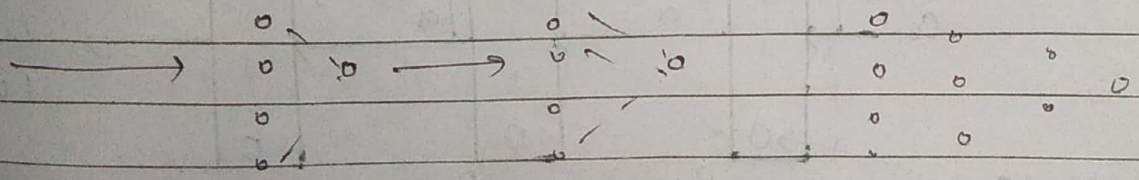
Pooling ⇒ Only reduction, no parameter learning.

→ Average Pooling is also there
↳ Used for very deep conv.

Fully connected layer:







n_H, n_W decreasing
n_c increasing with layer

	Activation	Activation size	# Parameters
Input	(32, 32, 3)	3072	0
Conv 1 (f=5, s=1)	(28, 28, 6)		
Pool 1	(14, 14, 6)		
⋮			
FC			

LOKENATH
Page No.
Date

Adv. of Conv. Net over MLP :

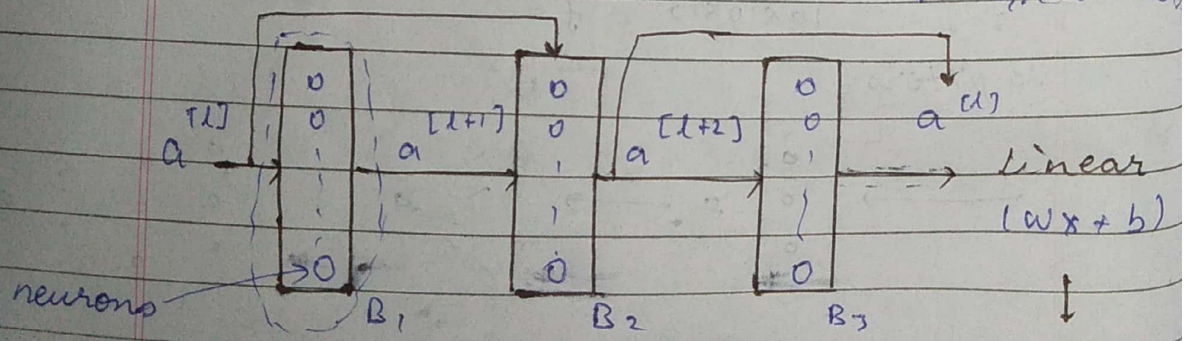
1) Sharing of weights / parameters
2) Sparsity of connection ∴ less no. of computations. ; depends on small no. of parameters.

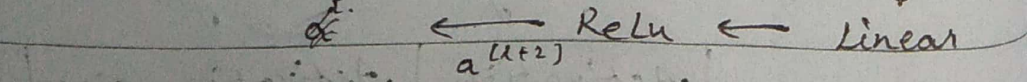
Classic Networks
LeNet - 5 (1998)
AlexNet (2012)
VGG - 16 (2015)

With more depth of layer, there is problem of Vanishing Gradients / Exploding Gradient

26th Oct

Residual Network (ResNet)





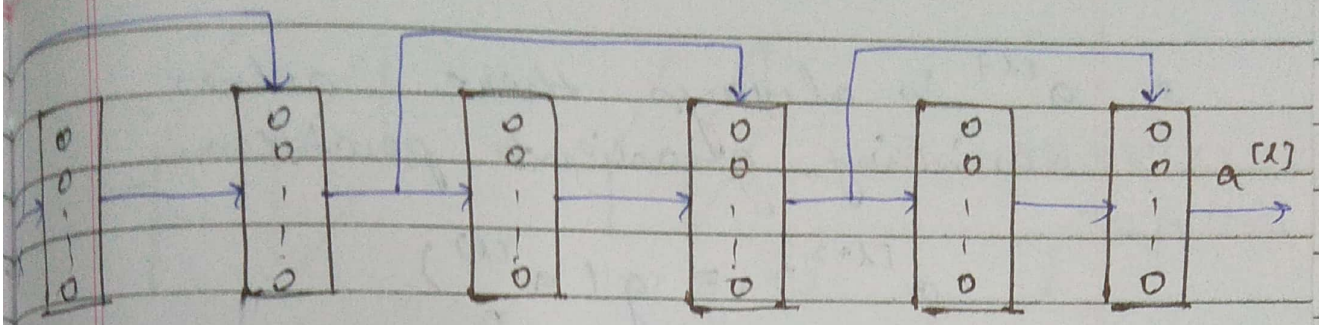
$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$$

$$a^{[l+1]} = g(z^{[l+1]})$$

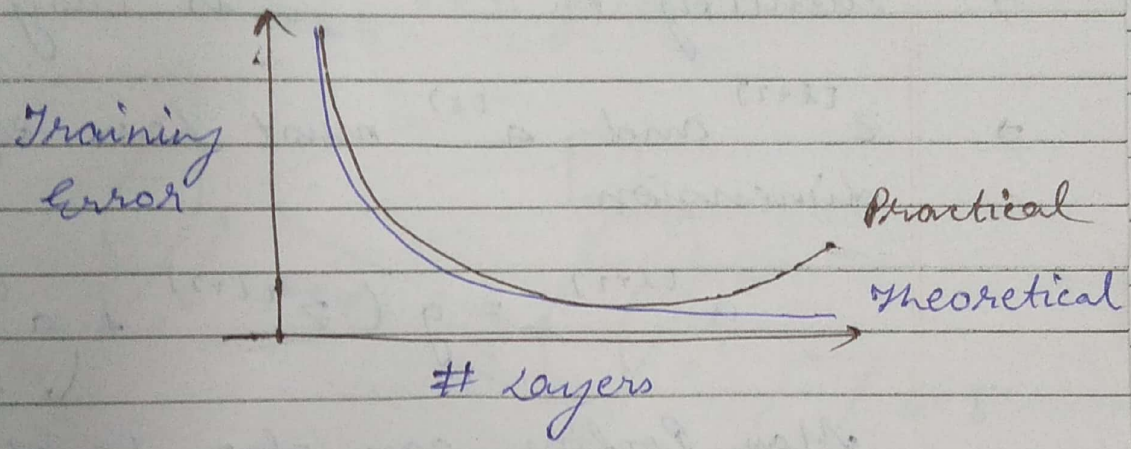
- In case of ResNet, $a^{[l]}$ is going through block B1 as well as directly to B2. Here, B2 is called Residual block $a^{[l]}$
- So, for ResNet :

$$a^{[l+2]} = g(z^{[l+2]})$$

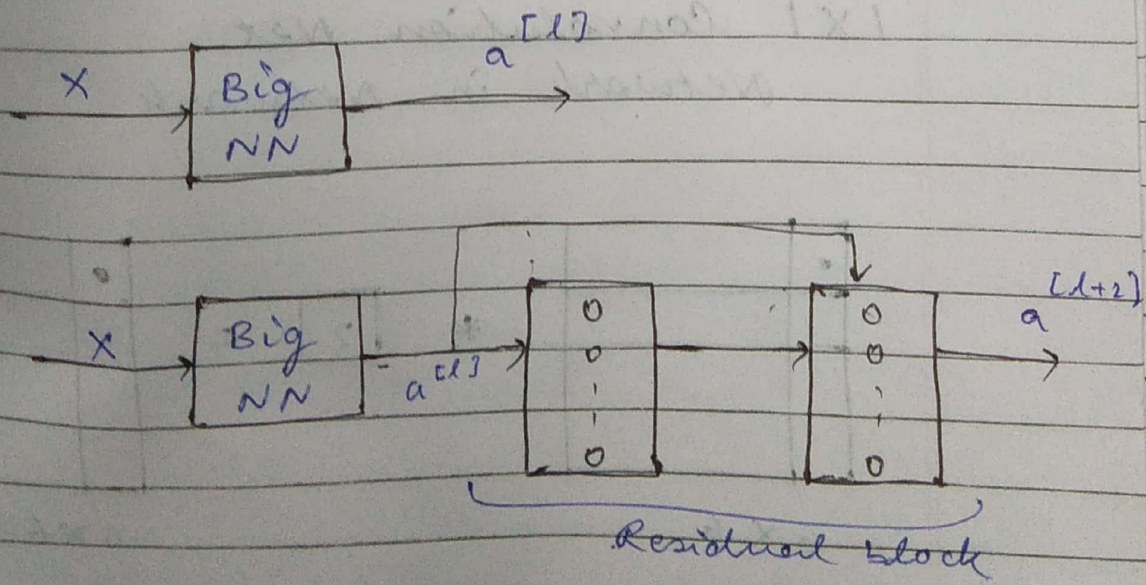
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$



Stacked of Residual block



Deep Conv. Neural Net



$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$
$$= g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

$w^{[l+2]} a^{[l+1]} \rightarrow 0$ (After certain)

$b^{[l+2]} \rightarrow$ Assume bias = 0

⇒ $a^{[l]}$ is always there, solves vanishing gradient problem.

$$a^{[l+2]} = g(a^{[l]})$$

Relu : $a \geq 0$

⇒ Identity fn : is easy to learn

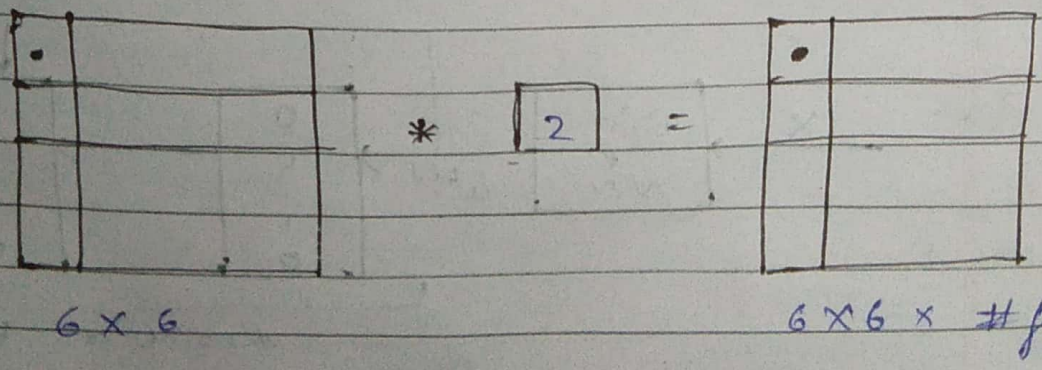
⇒ $z^{[l+2]}$ and $a^{[l]}$ must have equal dimension.

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

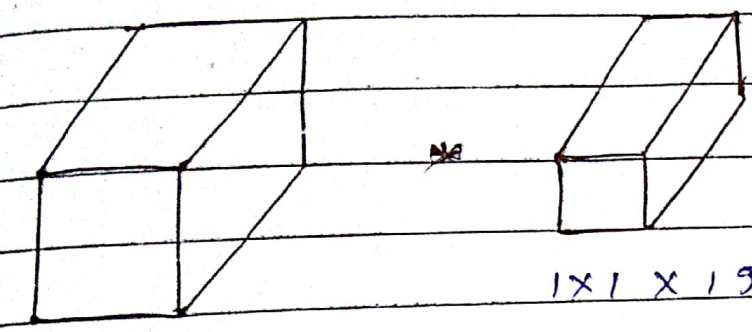
↳ ws

Max Pooling can also be applied.

1 X 1 Convolution Net
Network in Network

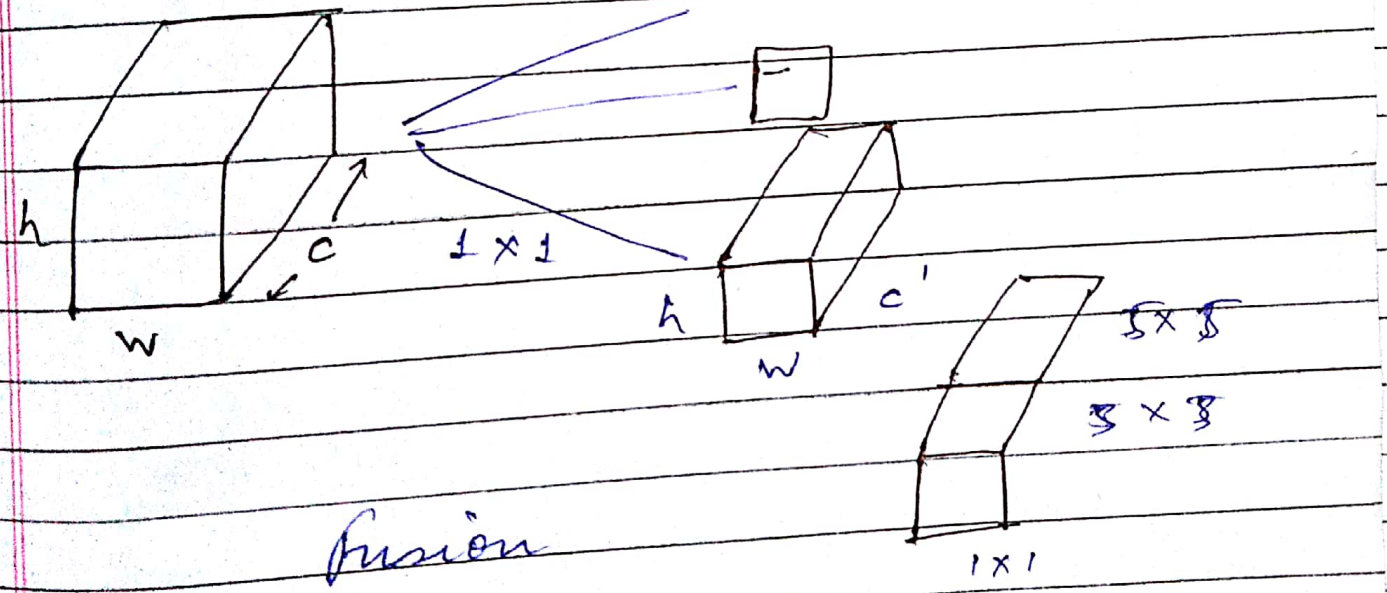


Full connected layers act as a Conv. net.



⇒ 28 x 28 x 192
32 filters
↓
28 x 28 x 32

Shrink → Another application of 1 x 1 filter



fusion

LOKENATH
Page No.
Date 23/10/18

Sequence Modelling

Feedforward ANN :
Inputs are given randomly.
(Not appropriate)

Bag of words : Vector of unique words in sentence
[1 0 0 0 . . . 1 . . .]^T

eg: Food is good, not so bad
Food is bad, not so good

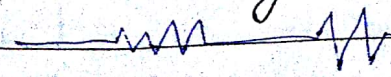
Markov Model : Predict $\hat{y}^{(t)}$ using state $t-1$

$\hat{y}^{(t-1)} \rightarrow \hat{y}^{(t)}$

Long term dependence
I was in France. time was beautiful.
I try to learn _____
predict (should be French)

⇒ This model fails here.

Recurrent ANN : It is transformed to sequence modelling

① Speech Recognition :

It is raining
X Y

② Music Generation :
♩ ♪♪ ♪♪♪

3) Sentiment classification

4) MIC Translation

5) DNA Sequence Analysis
ACCTGG ----→ Protein

6) Name Entity Recognition (NER)
Proper nouns.

⇒ RNN is supervised model.

Notation:

X :- Bill Clinton was president of America
$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<6>}$

Index :-

$T_x = 6$

Y :- [1 1 0 0 0 1] $T_y = 6$

NLP → how to represent the words

Vocabulary / Dictionary

a
an
and
:
Bill
chose
Clinton
10,000

One-hot vector $x^{<1>}$

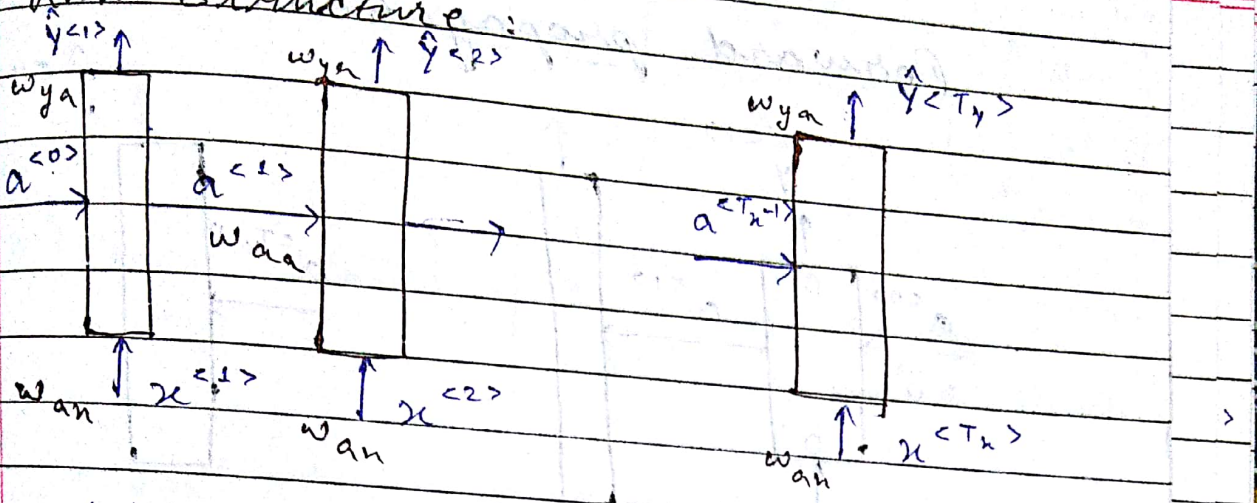
0
0
0
:
1
0
:
0

$X \rightarrow Y$

$(x^{<1>}, y^{<1>}), (x^{<2>}, y^{<2>})$
10,000 6

LDA : Latent Dirichlet Allocati

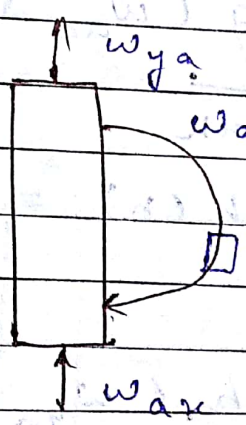
RNN structure:



Weights are shared.

$X^{(i)<t>}$: i^{th} instance at time t

$y^{(i)<t>}$:



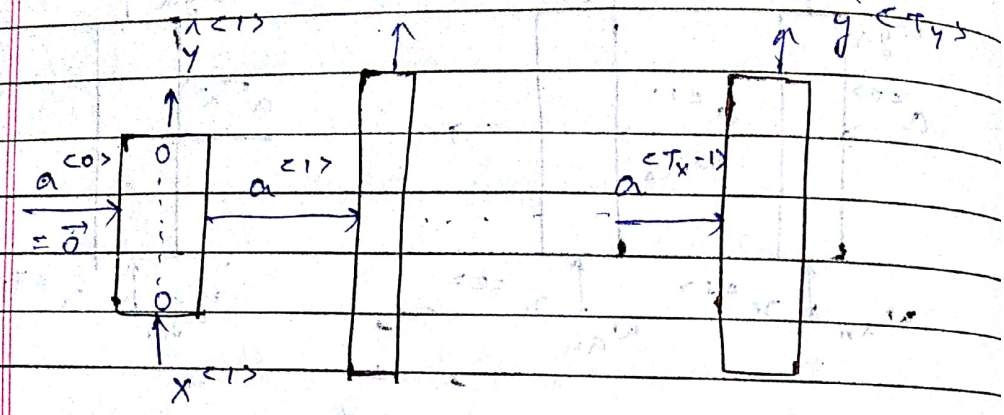
$\hat{y}^{<3>}$ requires $x^{<1>}$ $x^{<2>}$ $x^{<3>}$

Apple is a computer on discount
Apple is a fruit which is delicious

For predicting computer / fruit, we need to know the later part of the sentence. (Drawback of this basic RNN)

LOKENATH
Page No.
Date

Forward propagation:



$$a^{<1>} = g(w_{aa} a^{<0>} + w_{ax} x^{<1>} + b_a$$

tanh

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (-1 \text{ to } 1$$

$$\hat{y}^{<1>} = h(w_{ya} a^{<1>} + b_y)$$

sigmoidal / logistic or softmax

$$a^{<t>} = g(w_{aa} a^{<t-1>} + w_{ax} x^{<t>} + b_a$$

$$\hat{y}^{<t>} = g(w_a [a^{<t-1>}, x^{<t>}] + b_a$$

$$w_a = \begin{matrix} 100 \\ \updownarrow \end{matrix} \begin{bmatrix} w_{aa} & w_{ax} \end{bmatrix}$$

100 | 10,000

$w_{aa} = (100, 100)$

$w_{ax} = (100, 10000)$

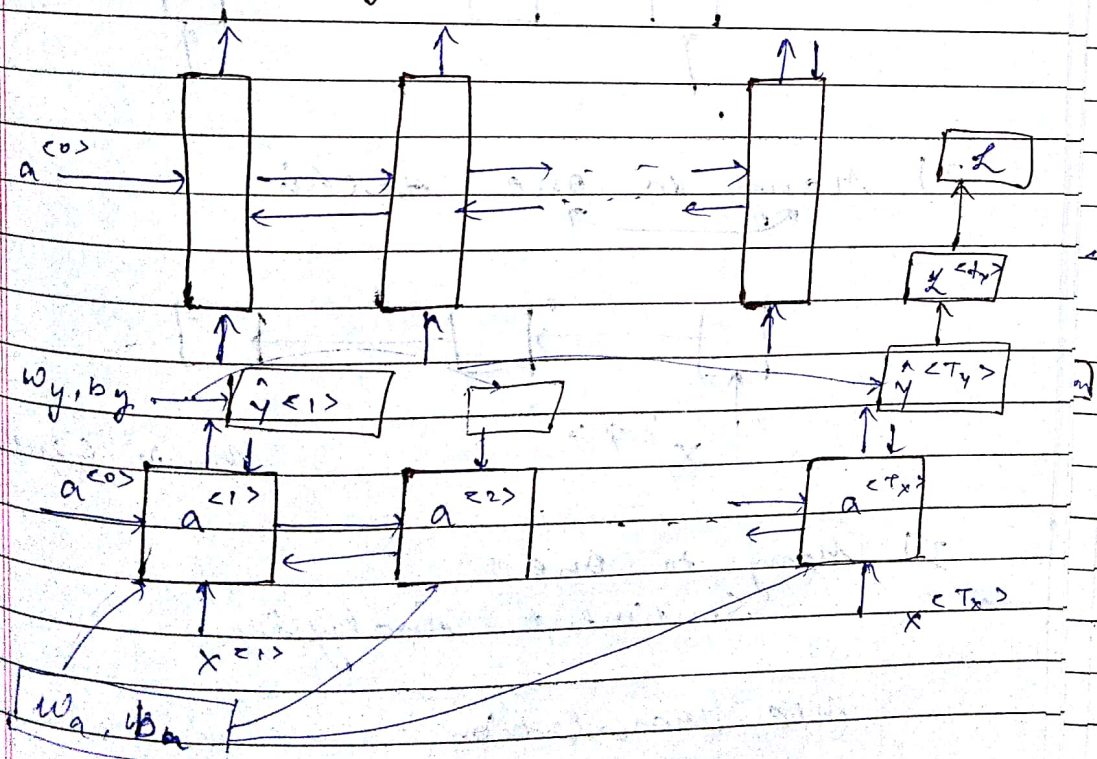
[Assume, $\dim(a^{<0>}) = 100$]

$$\left[a^{<t-1>}, x^{<t>} \right] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} \quad \begin{matrix} 100 \\ 10000 \end{matrix} \quad 10000$$

$$\begin{bmatrix} W_{aa} \mid W_{ax} \end{bmatrix} \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa} a^{<t-1>} + W_{ax} x^{<t>}$$

$$\hat{y}^{<t>} = h \left[W_a \left(W_y a^{<t>} \right) + b_y \right]$$

Back Propagation through time:



$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

Single Prediction

$$= -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log (1 - \hat{y}^{<t>})$$

Cross Entropy Loss

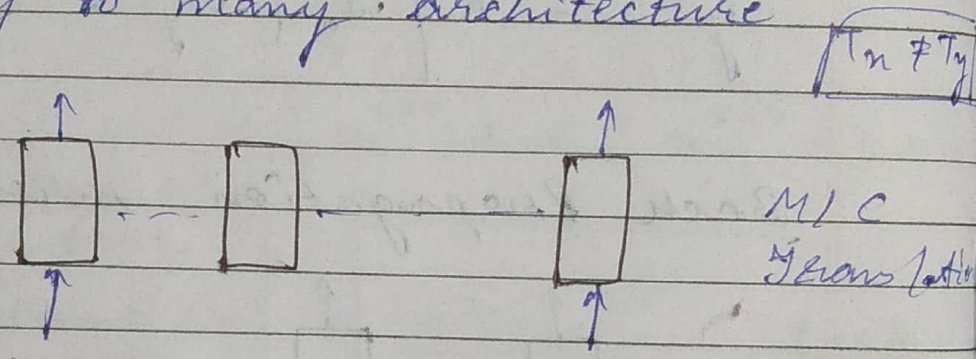
Overall Loss: T_y

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

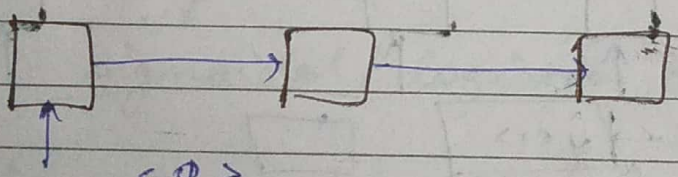
Main Feature of RNN:
Hidden State

Different Architecture:

1) Many to many architecture



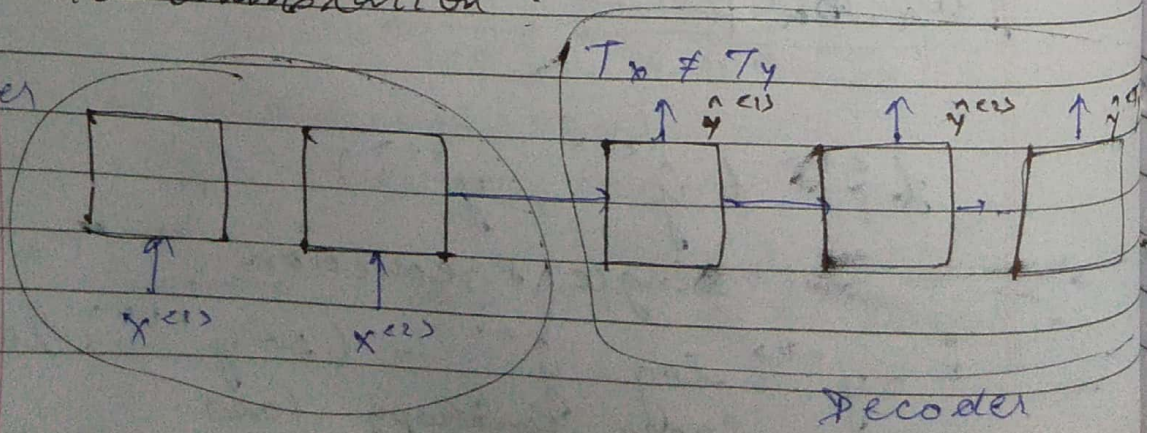
2) Many to one Archi:



Music Generation

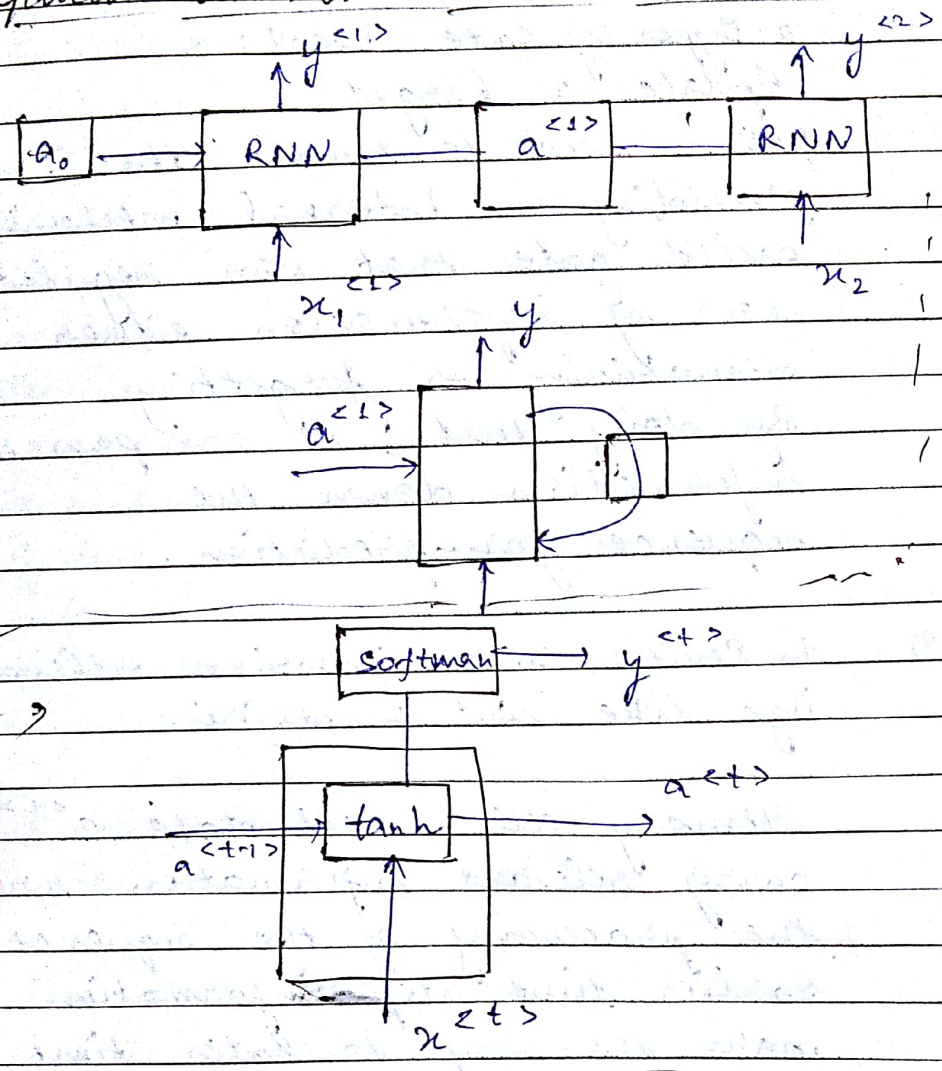
3) Many to one:
Sentiment Analysis

M/C Translation:



LOKENATH
Page No.
Date 12/11/18

Gated Recurrent Unit (GRU)



The cat ..... (was / were) running

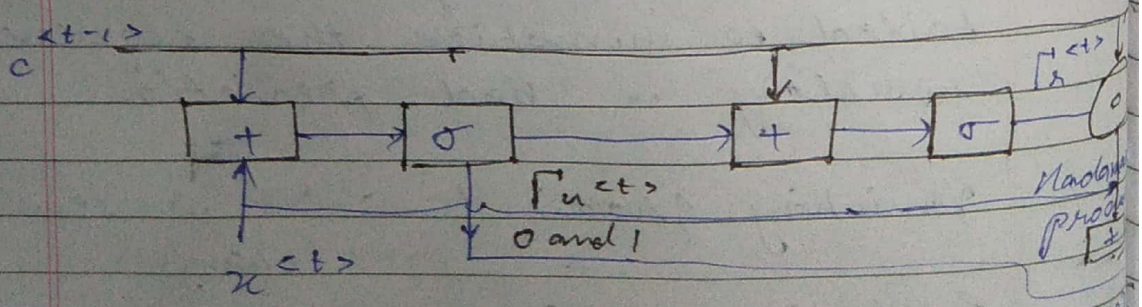
Back propagation training.
weight adjustment.
Failed to memorize the sequence, resulting in bad prediction.

Vanishing Gradient : No or less change.
Exploding Gradient : large change
⇒ Normal RNN → short term memory.

Solution is Gated Recurrent Unit.
2 Types of Gate used:
Update & Forget
GRU is used to handle the ... providing an internal mechanism called gate that can regulate the flow of information either memorizing or forgetting.
By doing that, it can pass relevant information down the sequence for prediction.

⇒ In Review, only memorize relevant info like cost & review.

Memory cell, cell state $c^{<t>}$ can carry relevant information throughout the processing of the sequence.
Earlier time step information can make its way to later time steps, reducing the effect of short term memory.
Gates are nothing but different neural network.



#1 Update Gate: $\Gamma_u^{<t>} = \sigma(W_u[c^{<t-1>}, x^{<t>}]$

Start calculating Update for time state t using

1) $x^{<t>}$ is plugged into network unit, multiplied by its own weight
2) Then $c^{<t-1>}$ is mult - with the activation weight.
3) Results is & a sigmoid function is applied to obtain results b/w 0 & 1.
4. Update gate determines how much info from the previous time state is learnt & passed along the future sequence cycle.

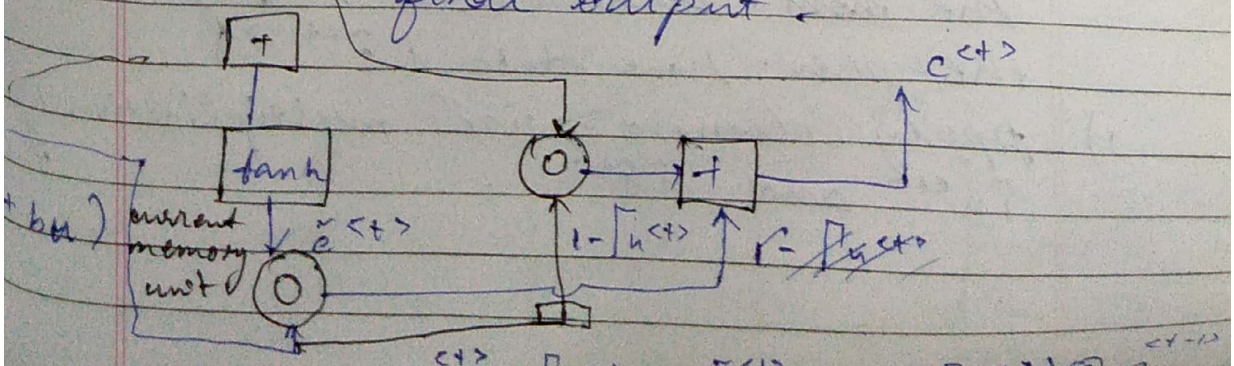
#2 Reset Gate:

$$\Gamma_r^{<t>} = \sigma(w_r[c^{<t-1>}, x^{<t>}] + b_r)$$

This gate is used to decide how much of the past information to forget.

#3. $$\tilde{c}^{<t>} = \tanh(w_c[\Gamma_r \odot c^{<t-1>}, x^{<t>}] + b_c)$$

Current implies how exactly the gates (update & reset) now influence the final output.



$$c^{<t>} = \Gamma_u^{<t>} \odot \tilde{c}^{<t>} + (1-\Gamma_u^{<t>}) \odot c^{<t-1>}$$

Reset Gate: How to use:

Next value of memory ($\tilde{c}^{<t>}$) which used the reset gate to store the relevant info. from past.

1) multiply the input $x^{<t>}$ with weight & $c^{<t-1>}$ with corresponding weight.
2) Calculate the Hadamard product b/w the reset gate & w. $c^{<t-1>}$. It determines what to remove from the previous time steps.
3) 1 & 2 are added & finally tanh is applied to produce $\tilde{c}^{<t>}$
4) Final memory at current time step:

$c^{<t>}$ which require $\tilde{c}^{<t>}$

So, the

The network needs to the current and (& passes it down to the network. To do that the update gate is needed. It determines what to collect from current memory content ($\tilde{c}^{<t>}$) & previous mem. cont $\tilde{c}^{<t-1>}$.

How to use Update Gate:

It determines what to collect from the mem - content ($\tilde{c}^{<t>}$) & from the prev - time state ($\tilde{c}^{<t-1>}$)

1) Apply element-wise multiplication of $\Gamma_u^{<t>}$ and $\tilde{c}^{<t>}$.

LSTM : It also use Output Gate.
LSTM is more powerful then GRU.
as it does not consider assumption

$$c^{<t>} = a^{<t>}$$

⇒ Long short Term Memory

$$\tilde{c}^{<t>} = \tanh\left(W_c\left[a^{<t-1>}, x^{<t>}\right] + b_c\right)$$

$$\Gamma_u^{<t>} = \sigma\left(W_u\left[a^{<t-1>}, x^{<t>}\right] + b_u\right)$$

$$\Gamma_f^{<t>} = \sigma\left(W_f\left[a^{<t-1>}, x^{<t>}\right] + b_f\right)$$

$$\Gamma_o^{<t>} = \sigma\left(W_o\left[a^{<t-1>}, x^{<t>}\right] + b_o\right)$$

$$c^{<t>} = \Gamma_u^{<t>} \odot \tilde{c}^{<t>} + \Gamma_f^{<t>} \odot c^{<t-1>}$$

$$a^{<t>} = \Gamma_o^{<t>} \odot \tanh c^{<t>}$$