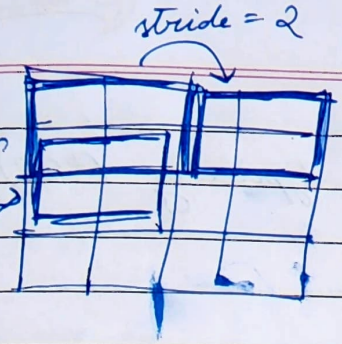
MNIST 0-9

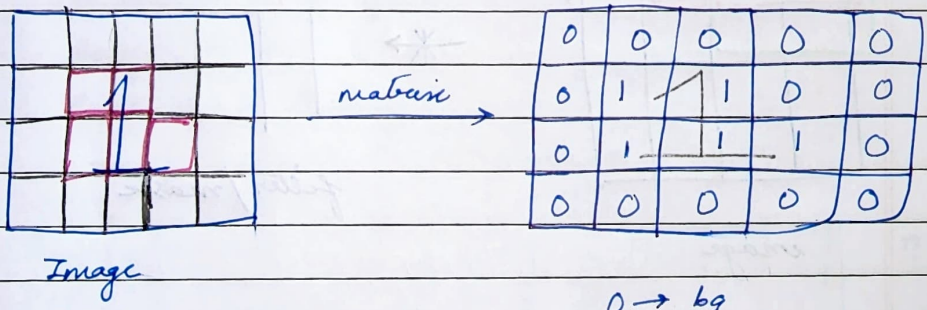


Book — Deep learning by Ian Goodfellow

12/09/22

Convolutional Neural Network

Image – basically a signal represented in the form of a matrix of discrete values



Image

0 → bg

1 → has parts of the img of 1

Why CNN instead of ANN ?

1. Computationally expensive to use ANN for images

~~For images have many features, say 1024.~~

Say image is 1024 × 1024.

⇒ if there are ~~1024 neurons in~~ 1000 neurons in H1 and 1000 in H2, then already 1024 × 1024 × 1000 computations are being carried out. (flatten the image and pass it as an ip)

Grayscale image :

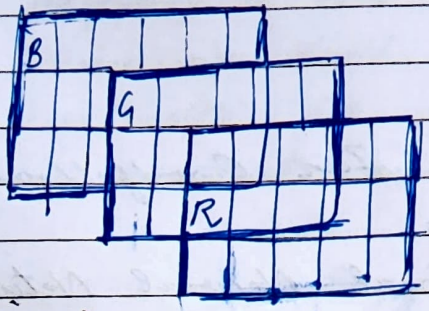
0 - black

255 - white

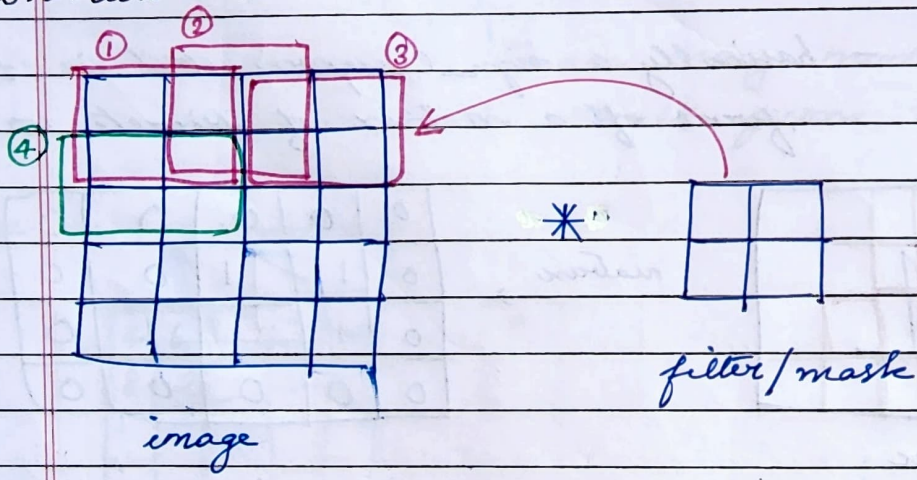
any value bw 0 & 255 - gray

Color images :

3 channels typically (RGB).



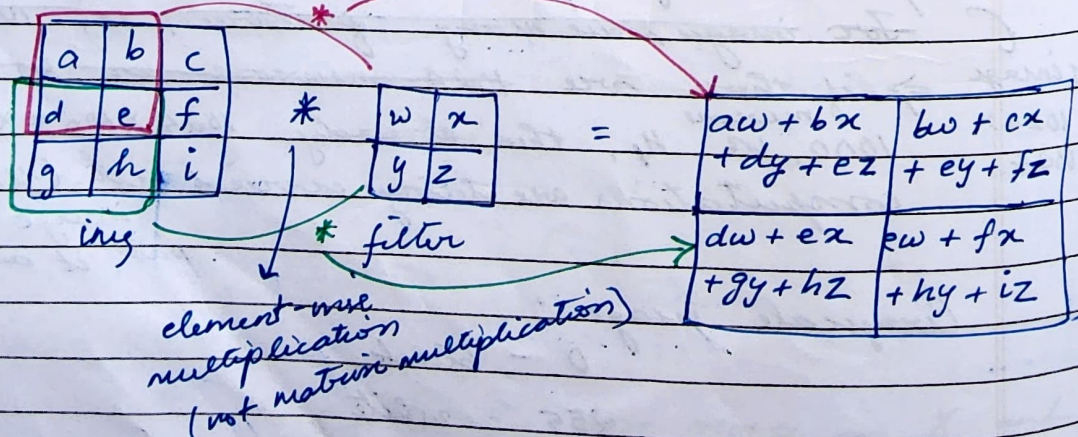
Convolutions :



Place the filter on the left end of the img do some computation & move the filter 1 step to the right (say stride is 1)

once a row is complete, move to the next row's left end and repeat (assuming stride is 1)

(Eg):



(3×3) * (2×2) → ~~(2×2)~~ (2×2)

Typically, the feature image is convolved many times with the filter before activation fn is applied

Keras eg:

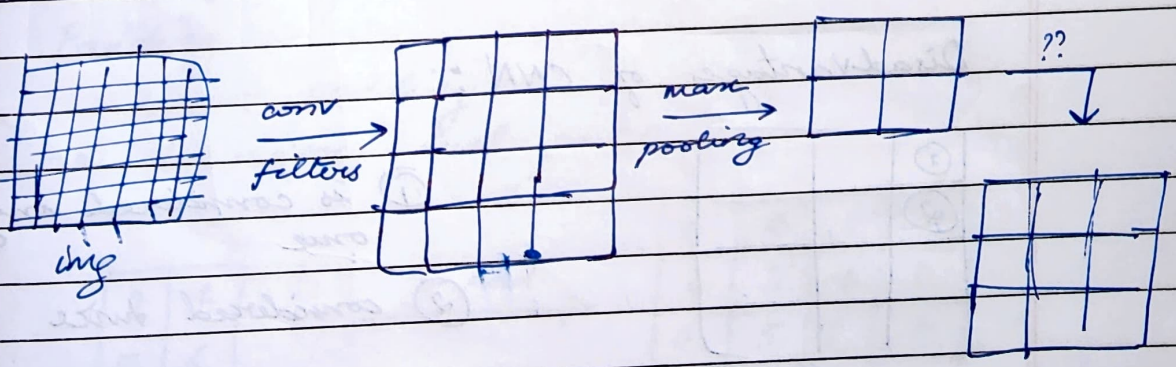
Dense (filters = 16, kernel = (3,3), input_shape = (240, 240), channels = 3).

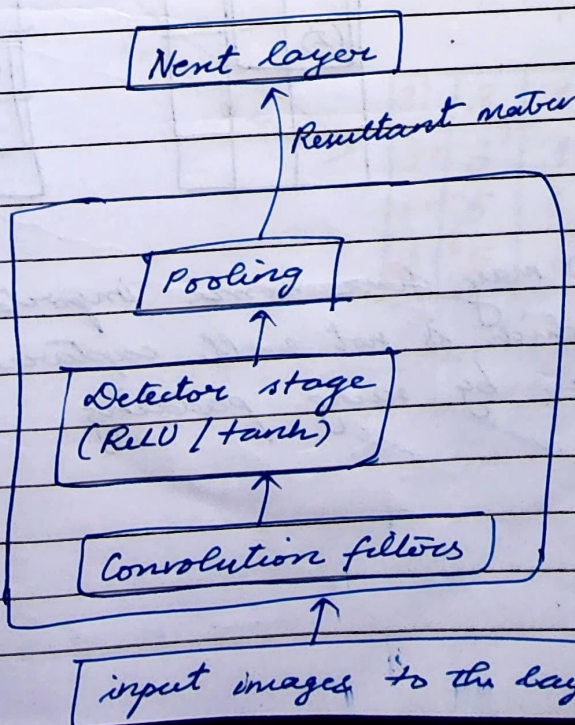
Pooling: done to reduce matrix size

① Pass over input img to Conv layer (with a number of filters) ~~after which we a~~

② Apply ReLU

③ Apply max Pooling





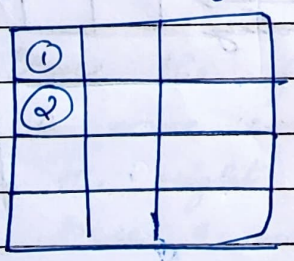
Feature learning via convolutions

1st layer (convolution) detects edges / outline
next detects little more complex features like 2D shapes (circles, squares, etc..)
3rd layer detects ~~shapes~~ eyes, etc..
and so on
(each subsequent layer detects more & more complex features in the ~~the~~ image).

*. Number of parameters in CNN is much less than that of ANN

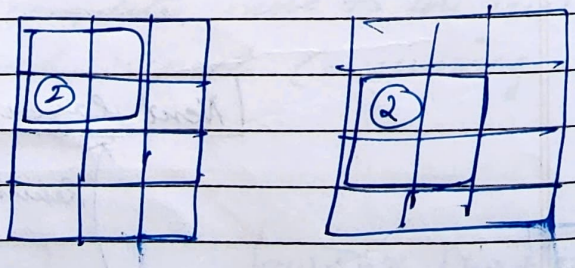
4X4 img → 3X3 filter → CNN ⇒ 16 X 3X3

Disadvantages of CNN:

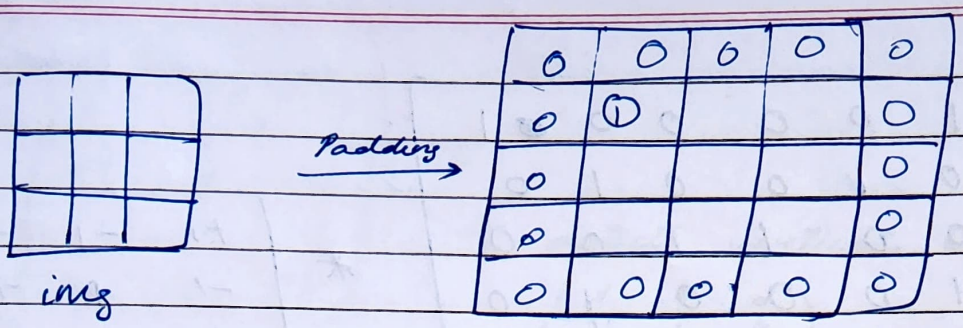


① is considered only one

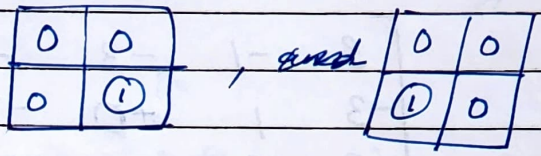
② considered twice

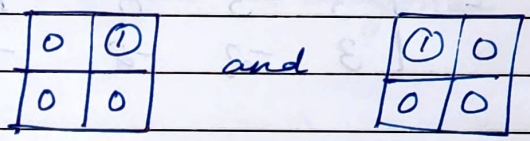


But ① may have some important feature as well (which is not well captured in the ↑).
→ overcome by using padding



Now ① is also considered ~~more~~ >1

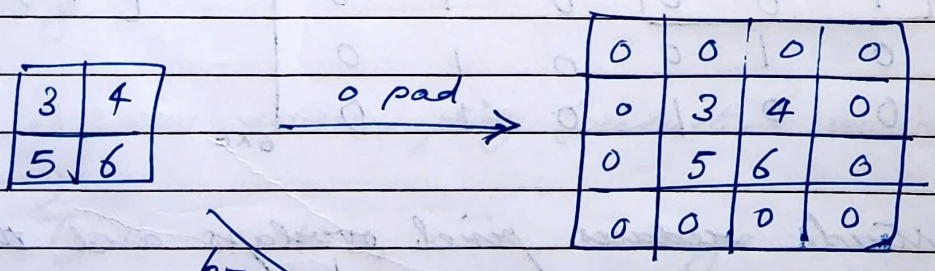


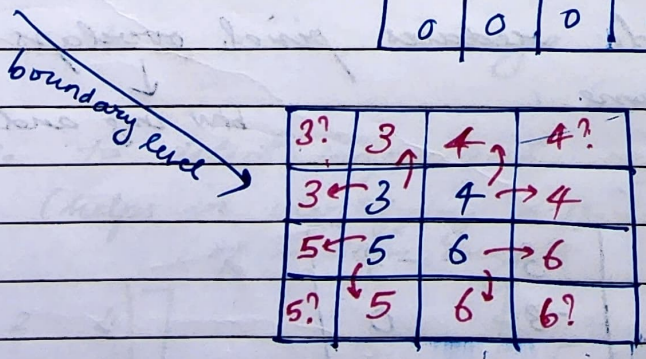


=> ① considered 4 times like all other pixels.

Padding :

1. Zero padding
2. boundary level padding





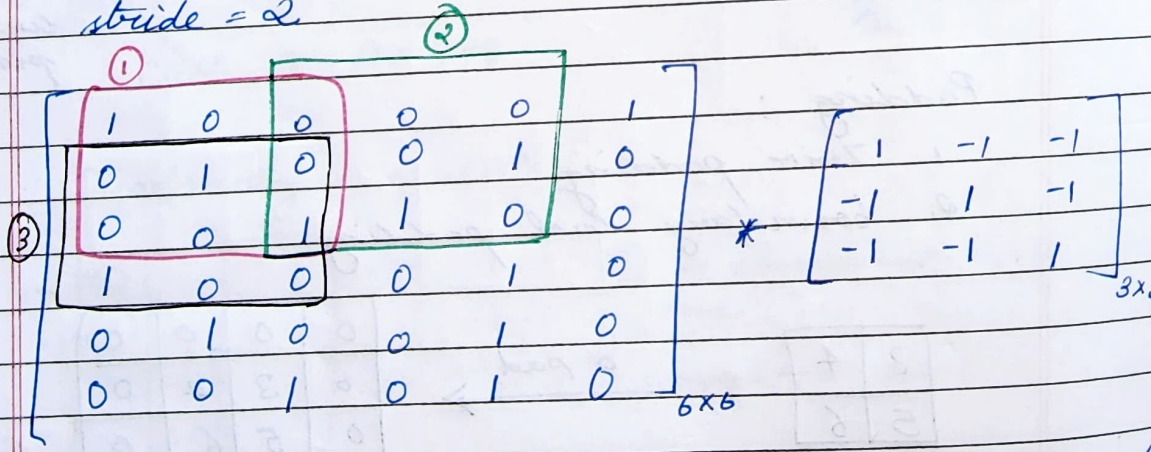
(Just copy the values at the boundary).

13/09/22

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}_{6\times6} * \begin{bmatrix} +1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}_{3\times3}$$

$$= \begin{bmatrix} 3 & -1 & -3 & -1 \\ -3 & 1 & 0 & -3 \\ -3 & -3 & 0 & 1 \\ 3 & -2 & -2 & -1 \end{bmatrix}_{4\times4} \quad (\text{stride} = 1)$$

stride = 2



stride reduces pixel overlaps and the output volume

↓

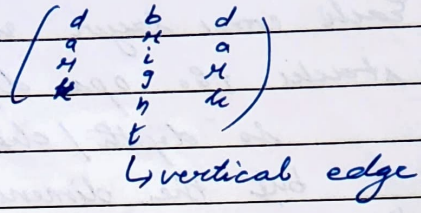
btw img and filter

$$\begin{bmatrix} 3 & -3 \\ -3 & 0 \\ -3 & 0 \\ 3 & -2 \end{bmatrix}_{4\times2}$$

After ②, if we move the filter by a stride of 2, it goes beyond the image, so we skip it & ~~more~~ move to the next

row.

$$\begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix}$$ ⇒ detects vertical boundaries



↳ vertical edge

$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$ ⇒ detects a diagonal boundary

To detect outline / edges

⇒ use kernel $\begin{bmatrix} 1 & -1 \end{bmatrix}_{1\times 2}$

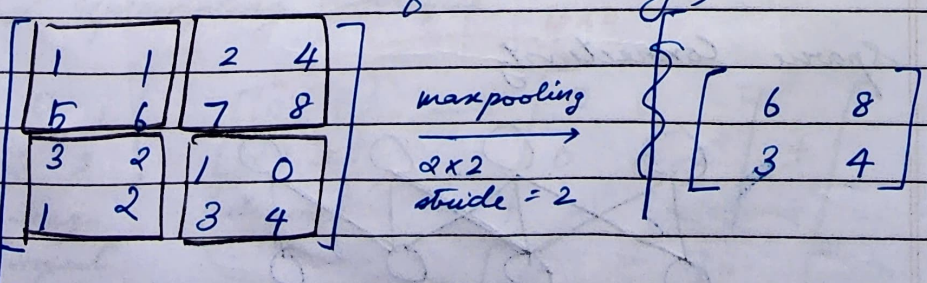
↓ edge ↓ not an edge

Pooling :

Performs downsampling (reduction in image size)

Types - Max, min, average

Responsible to detect important information (helps in identifying different variants of similar images)



Padding - performs upsampling

5 can be written in ~~ten~~ different handwritings ⇒ Pooling helps in identifying that all are similar (belong to class 5).

Each conv layer uses many filters and stacks the ops depthwise
so depth / channels ↑s.
but the dimension of each channel ↓

(Eg): $160 \times 160 \times 3 \rightarrow 80 \times 80 \times 9$

due to 3 filters?

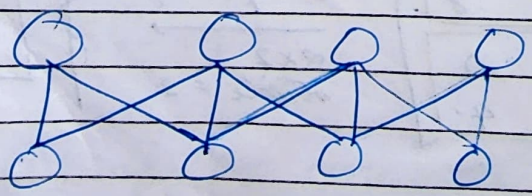
* CNN & fully-connected network

After using convolution, we flatten the matrix and pass it to a fully connected (why?) nn network

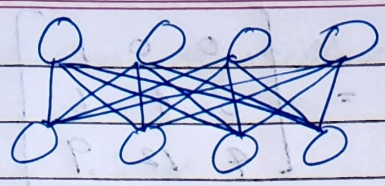
Activation → generally softmax to predict the class with that the img belongs to

Sparse matrix

Sparse Connectivity



⇒ only some connections & not all ⇗ dense



⇒ dense

Img size = $N \times N$

kernel size = $K \times K$

padding = P , stride = S

output image size = $\left(\frac{N-K+2P}{S}+1\right) \times \left(\frac{N-K+2P}{S}+1\right)$

16/09/22

28×28 —(Padding = 1, stride = 1, kernel = 3×3)→ 28×28.

$$\frac{N-K+2P}{S}+1 = \frac{28-3+2}{1}+1$$

$$= 28$$

(If you get fractions, take the floor value)

Transposed Convolution :

✓ Upsampling technique

(eg):

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$$

input

kernel

$$\begin{bmatrix} \textcircled{0} & 1 \\ \textcircled{2} & 3 \end{bmatrix}$$

ip × 0 → $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ + $\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$

ip × 2 → + $\begin{bmatrix} 0 & 2 \\ 4 & 6 \end{bmatrix}$ = + $\begin{bmatrix} 0 & 3 \\ 6 & 9 \end{bmatrix}$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 4 & 6 \\ 4 & 12 & 9 \end{bmatrix}$$

1. Apply padding
2. Do convolution

(Eg): ip = 3×3

filter = 3×3.

reqd op size = 5×5.

$$\frac{N - K + 2P}{S} + 1 = 5.$$

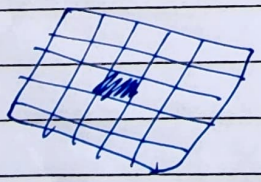
$$\frac{3 - 3 + 2P}{1} + 1 = 5.$$

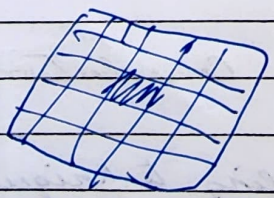
$$2P = 4.$$

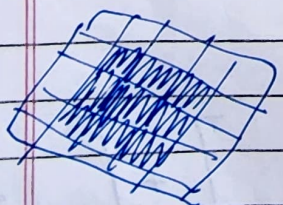
$$P = 2$$

Dilated Convolution

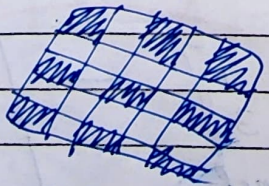
Do padding in btw the pixels also (& not just the boundary).







3×3 padding
dilation = 1
(leave 1-1=0 gaps btw cells)



3×3 padding
dilation = 2
(leave 2-1=1 gap

Convolution Algorithms

Stride Conv
Padding Conv
Transpose Conv
Dilated Conv

Parameters of Conv layer

✓ How many filters
✓ footprint (size of the filter - 3x3, 4x4, ...)
✓ striding (Reqd or not, if yes how much)
✓ activation fn
✓ Padding

Op of conv layer → feature map

✓ learning rate
✓ loss fn
✓ batch size
✓ initial weights
✓ loss fn

ImageNet → dataset!

Initializing filter weights

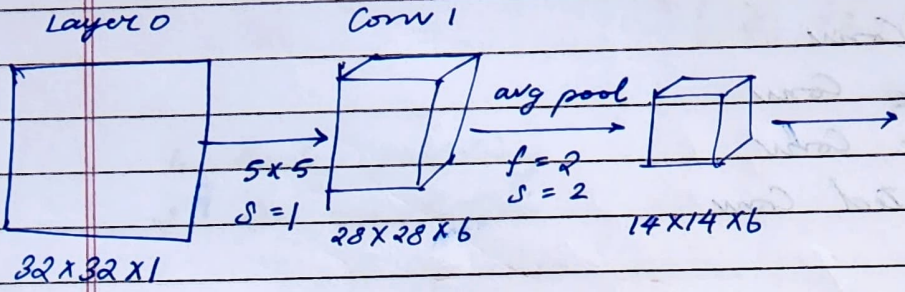
Symmetrical filters ⇒ do the same job
⇒ Redundant

Approaches:
1. Random initialization
2. Glorot initialization
3. He initialization

? x?

Date
Page

LeNet 5



Pooling

- ✓ reduces size of ip to speed up processing
- ✓ gives some robustness to feature detections
  512×512 ⇒ lot of pixels (use pooling)

Conv layer

- ✓ convolve using filters
- ✓ Add bias
- ✓ Apply activation fn

Transposed Convolution :

~~was~~ like conv. with fractional striding
upsampling by just padding

Stride :

fast way to ↓ img size to speed up later blocks in the network

Why padding ?

- ✓ Conv. reduces img size (use padding to make op size same as ip or even ↑)
- ✓ pixels at edge are not used only in fewer convolutions than central pixels ⇒ throws away a lot of information at the edges

classmate
Date
Page

Types of layers in CNN

- ✓ Conv
- ✓ Pooling
- ✓ FC

downsampling using strided conv better than conv + pooling

Deeper in the network

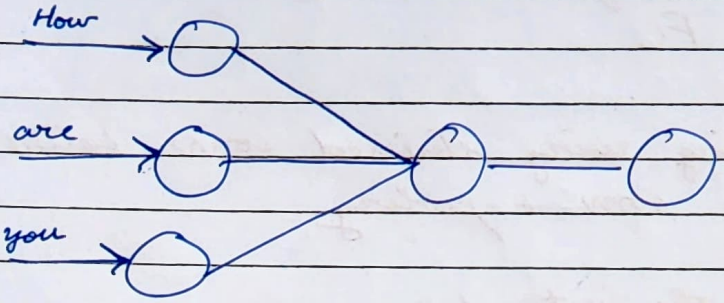
size of img ↓
#channels ↑
} we use lower resolution to work with larger collection of features without requiring large filters

Always
#channels in filter = #channels in input

23/09/22

RNN

~~Sequence data~~ Why ~~RNN~~ not ANN ?

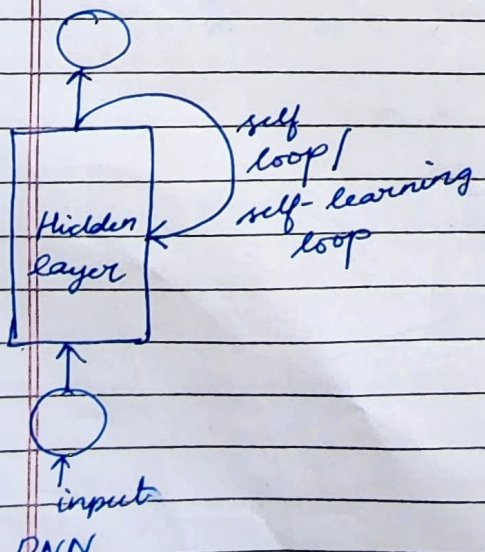


① Semantics are not preserved in ANN (when using text data).
('How' processed separately, 'are' processed separately and so on).

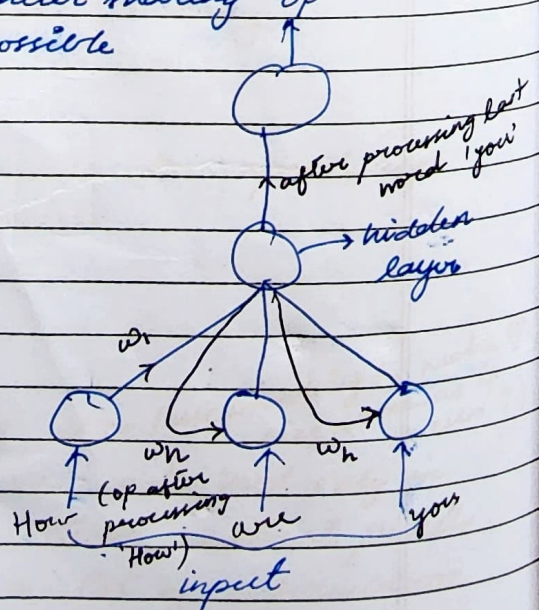
② Computationally expensive to use ANN (sequence processing involves too many computations)

Encode → How are you
[1 0 0] [0 1 0] [0 0 1]. (One-hot encoding).

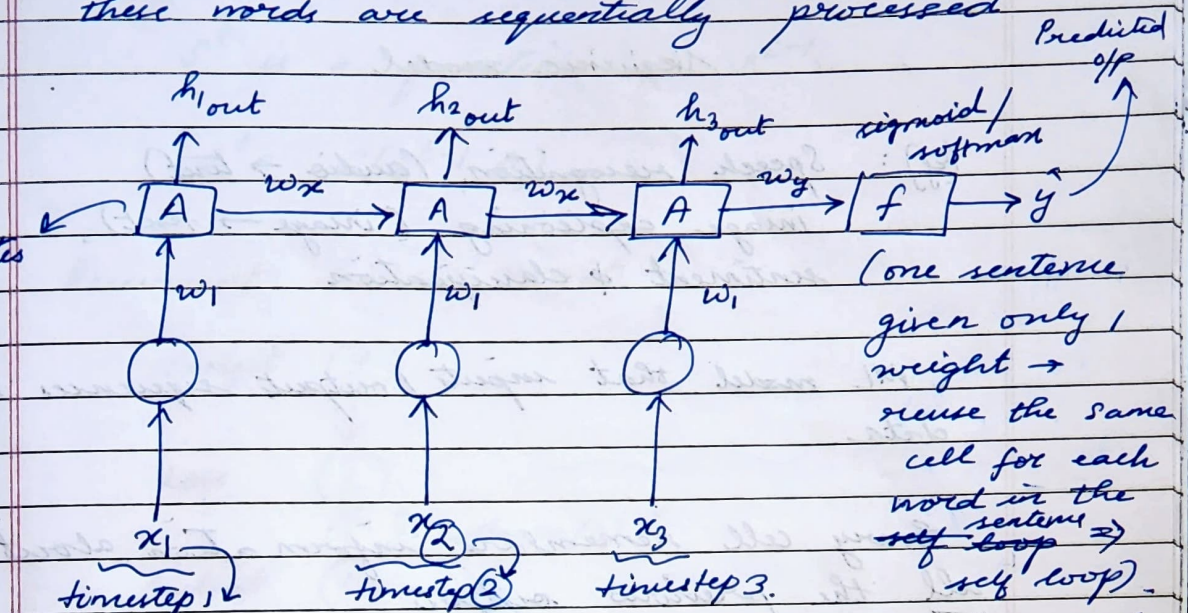
③ No memory



④ Parameter sharing op is not possible



RNN → Sentence is split into individual words and these words are sequentially processed



(one sentence given only 1 weight → reuse the same cell for each word in the ~~self loop~~ sentence ⇒ self loop).

(previous output and current input are used to make current prediction)

$h_{1out} = A(w_1 x_1 + b)$. A is activation fn

$h_{2out} = A(w_1 x_2 + h_{1out} w_x + b)$.

$h_{3out} = A(w_1 x_3 + h_{2out} w_x + b)$

$\hat{y} = f(h_{3out} w_y + b)$ f can be sigmoid / softmax

weights and activation fns are same because we are processing the words in the same sentence and it performs the same task

Sequential data:

Data that has to be processed one after another

(Eg): videos, documents

x_i → word input at i^{th} timestep

Sequence model

(Eg): Speech recognition (audio → text)
image captioning (image → text)
sentiment classification

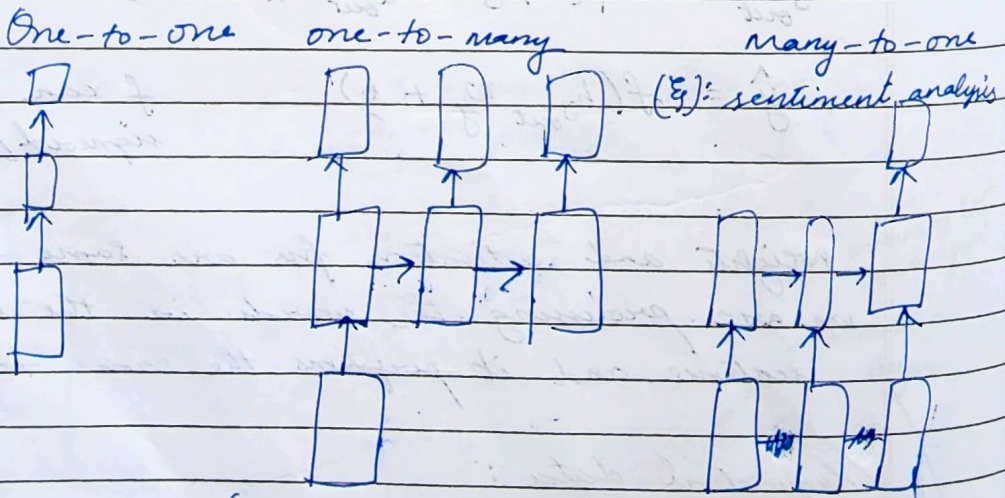
ML model that inputs / outputs sequences of data

Every cell remembers information about all the previous outputs

Types of RNN

Language Translation → many-to-many
Movie rating → many-to-one
generate poem from one word → one-to-many

Product review (good or bad) → many-to-one



(Eg): Music generation
~~sentiment Analysis~~
~~language generation~~

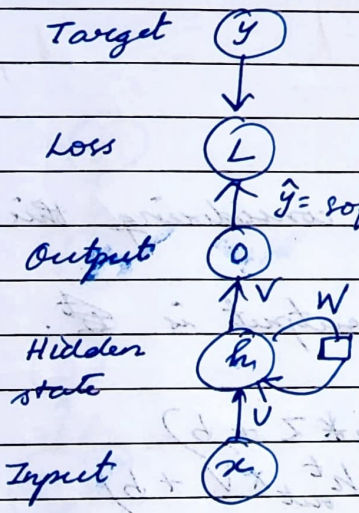
sequence input single output (Movie rating)

A system is recurrent if

s^t is a function of s^{t-1}

$s^t = f(s^{t-1}, \theta)$

Recurrent hidden units



for every time step

Same content ⇒ Same weights, activation fns.

RNN loss

1- compute at individual time steps

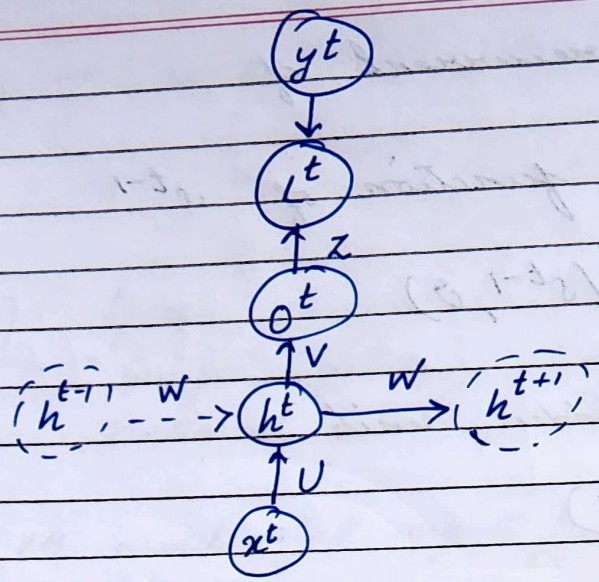
2. Overall loss after all time steps

Backpropagation happens through time

⇒ BPTT

⇒ Backprop applied to RNN unfolded in time

*



find V_{new} only considering this layer

For that layer, output is ~~$\hat{y}^t$~~.

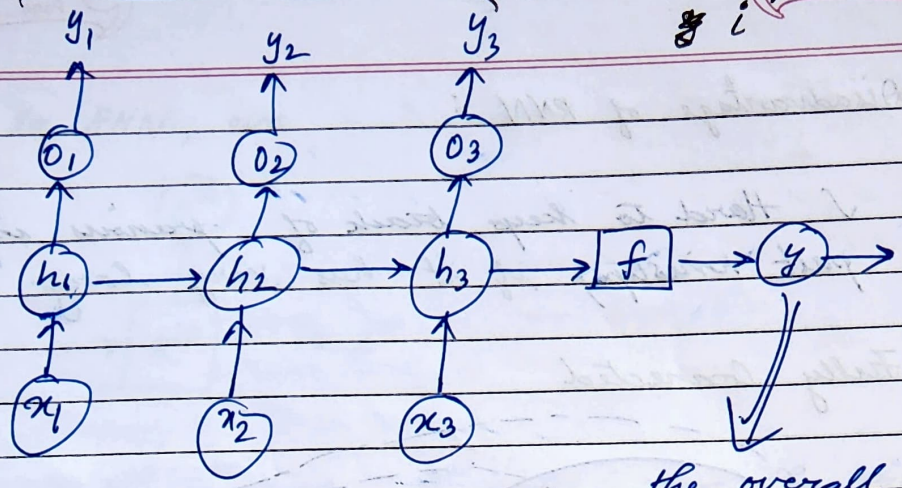
$$\hat{y}(t) = A(O_t * z + b)$$
$$O_t(t) = f(h^t_{out} \times V + b)$$
$$h^t_{out} = f(h^t_{in})$$
$$h^t_{in} = (x^t \times U + b).$$

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

$$\frac{\partial L}{\partial U} = \frac{\partial L}{\partial \hat{y}^t} \times \frac{\partial \hat{y}^t}{\partial o^t} \times \frac{\partial o^t}{\partial h^t_{out}} \times \frac{\partial h^t_{out}}{\partial h^t_{in}} \times \frac{\partial h^t_{in}}{\partial U}$$

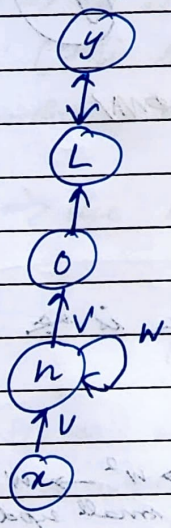
Individual loss - ~~on~~ useful to fill blanks in sentences (language generation?)
(Use output from layer 1, predict word for layer 2 and so on..)

each y_i is used to compute individual loss at timestep i



the overall output (used to compute overall loss)

(Eg):



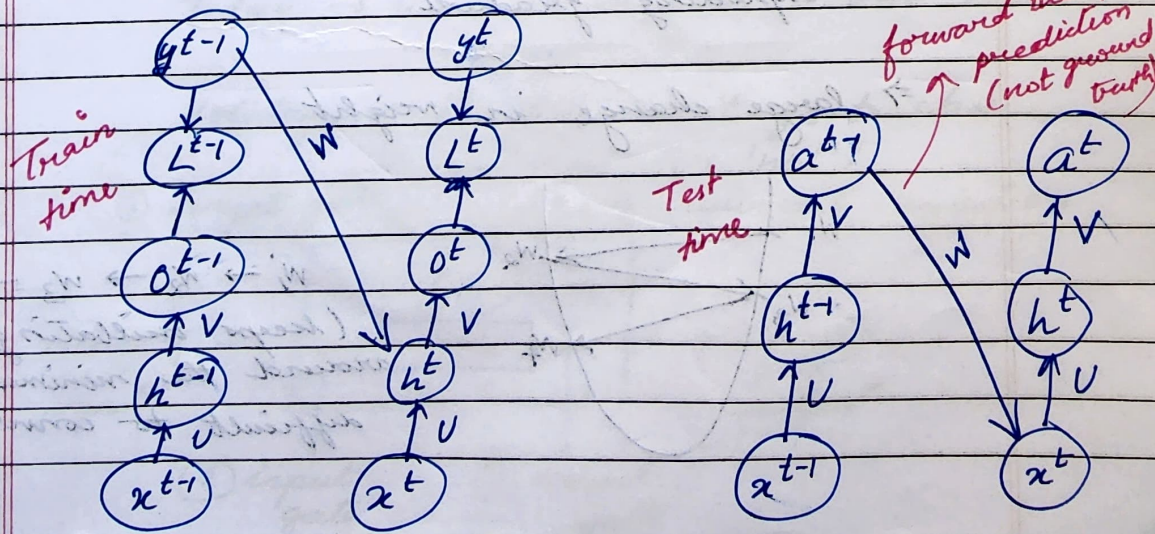
$$g^{(t)} = \text{softmax}(o^{(t)})$$
$$o^{(t)} = c + Vh^{(t)}$$
$$h^{(t)} = \dots$$

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = \tanh(a^{(t)})$$
$$o^{(t)} = c + Vh^{(t)}$$
$$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$$

$a^{(t)}$ is $h^{(t)}$ in

$b, c \rightarrow$ bias.

Teacher forcing

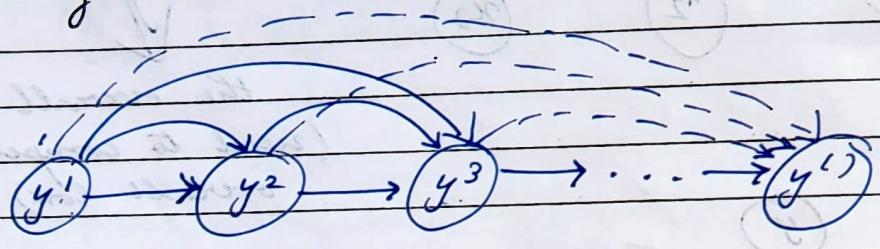


Date
Page

Disadvantage of RNN :

✓ Hard to keep track of previous info (at past timesteps) if it has very large inputs

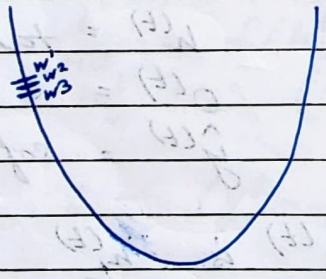
Fully Connected



✓ main issues with standard RNN

✓ Vanishing gradient

⇒ change in weight value is very less

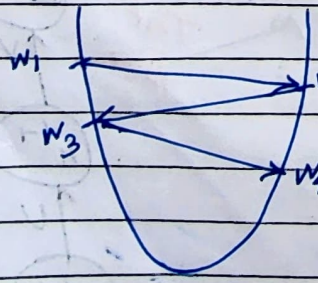


$w^1 \rightarrow w^2 \rightarrow w^3$
(very small updation)

→ model does not learn even after many epochs

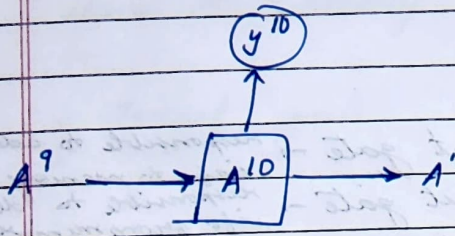
✓ Exploding gradient

⇒ large change in weight



$w_1 \rightarrow w_2 \rightarrow w_3 \rightarrow w_4$
(keeps oscillating around the minima difficult to converge).

In RNN, we



y^{10} resembles A^{10} more than the previous information (upto A^9).

⇒ so it does not remember past info well.
(This happens more & more as we go ~~further~~ through more & more timestamps).

⇒ Use LSTM

27/09/22

LSTM
(Long-short term memory).

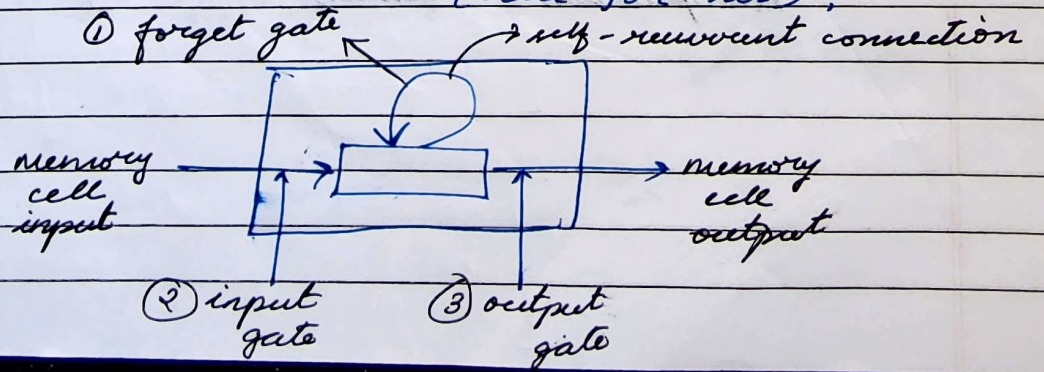
⇒ Stores whatever is important in a sentence in a memory and use it wherever needed

(Eg): Today is holiday, so students ___ ___ in class
are not

Keywords : Today, Holiday
(both are stored in memory)

Today → used to indicate present tense (are)

Holiday → so they won't go to class (clue for 'not').



LSTM - 4 interacting layers

↓

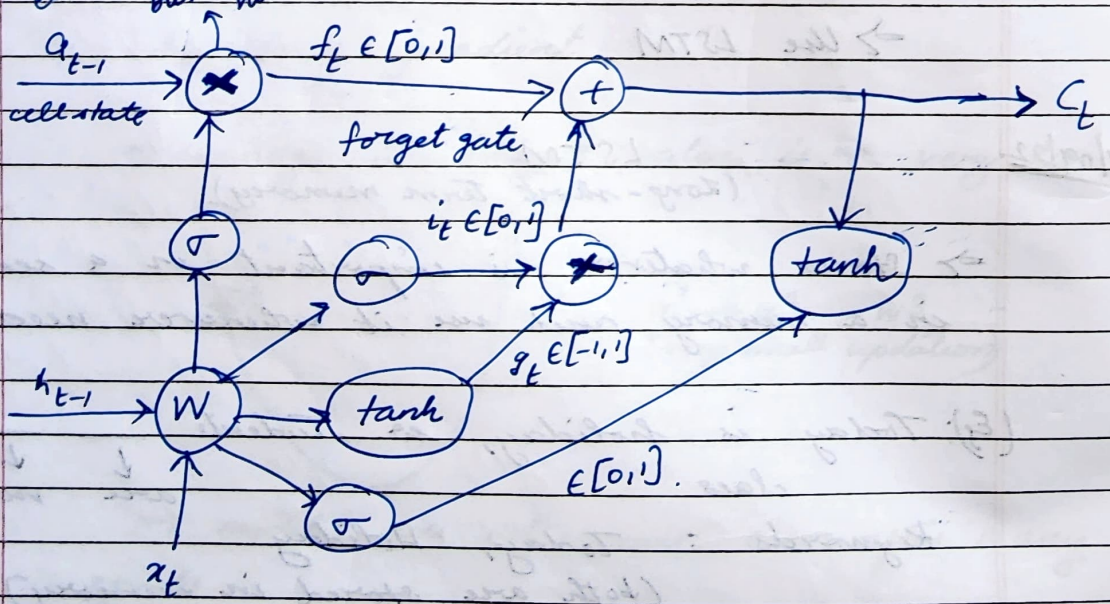
discovered in 1976

uses gating mechanism

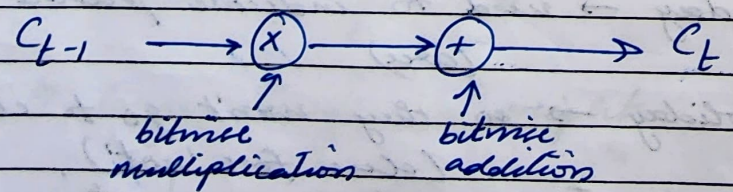
[3 gates → input gate → responsible to add new info to memory
output gate → responsible to decide what info from memory goes to next input
forget gate - remember info if important else forget

tanh - selects inputs

output from LSTM - 4x larger than that of RNN cell



cell state - represents memory of LSTM

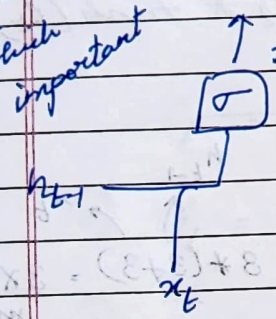


Gates

Have the ability to add or remove

1. Forget gate - controls what info to store throw ~~v~~ memory away from

remove info which is not important from the memory



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

dimension is dimension of h_{t-1} + dimension of x_t

$f_t = 0 \Rightarrow$ forget

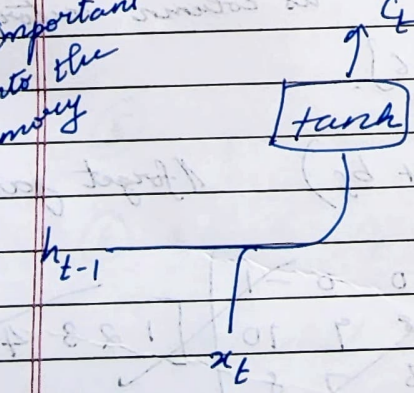
$f_t = 1 \Rightarrow$ remember completely

f_t passed to calculate cell state

→ Remembering gate

2. Input gate → decides what info to put inside the memory

puts important info into the memory



(forget gate o/p + input → input gate output)

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

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

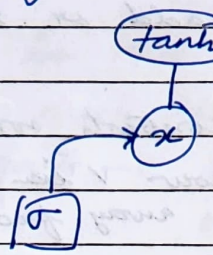
3. Memory update

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

$C_t * f_t + i_t *$

→ selection gate

Output gate - decides what to output from the memory



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

$h_t = o_t * \tanh(c_t)$

1. $h_{t-1} = [1, 2, 3]$

$x_t = [4, 5, 6]$.

h_{t-1} ↑ , x_t ↗

we have 3 units say and → so $W = 3*(3+3) = 3 \times 6$ matrix

$$W_f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$

$b_f = [1, 2, 3]$. (Consider all 1D vectors as column vectors)

$[h_{t-1}, x_t] = [1, 2, 3, 4, 5, 6]$.

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

$= \sigma\left(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} [1\ 2\ 3\ 4\ 5\ 6] + [1\ 2\ 3]\right)$

$$W_f \cdot [h_{t-1}, x_t] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

$$= \begin{bmatrix} -6 \\ 175 \\ 133 \end{bmatrix}$$

$$W_f \cdot [h_{t-1}, x_t] + b_f = \begin{bmatrix} -6+1 \\ 175+2 \\ 133+3 \end{bmatrix} = \begin{bmatrix} -5 \\ 177 \\ 136 \end{bmatrix}$$

$$f_t = \sigma\left(\begin{bmatrix} -5 \\ 177 \\ 136 \end{bmatrix}\right)$$

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

② $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$. // input gate

③ $\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$.

④ $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$ // memory updation

$*$ ↓ elementwise multiplication

$$\left(\begin{bmatrix} a \\ b \end{bmatrix} * \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} ac \\ bd \end{bmatrix}\right).$$

⑤ $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$ // output gate

⑥ $h_t = o_t * \tanh(C_t)$

For next time step, pass C_t and h_t

For NN architecture, write the o/p layer as

(1)

(2)

⋮

(25)

o/p explicitly

$16 \times 16 \times 3$ $\quad$ $3 \times 3 \times 3$ $\quad$ 5

5

5 → # filters

$16 \times 16 \times 3$ $\quad$ $3 \times 3 \times 3$

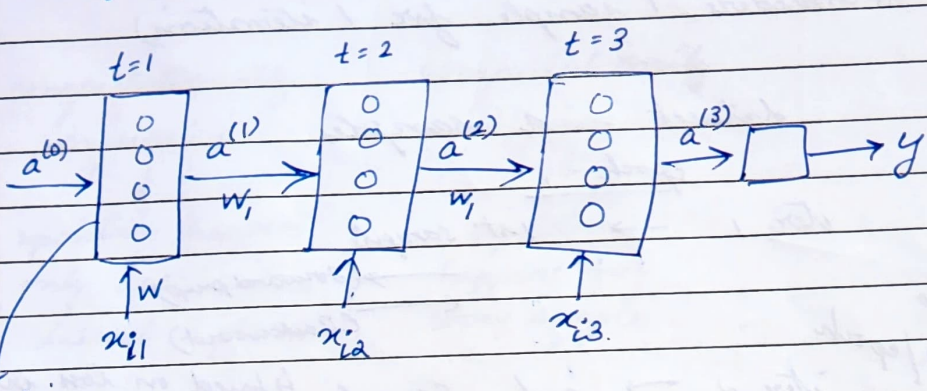
consider as filter (3rd dimension is 3 implicitly)

so answer is $n \times n \times 3$

07/10/22

Timeseries data input to RNN

$X_i = \langle X_{i1}, X_{i2}, X_{i3} \rangle$



10/10/22

Optimization in DL

1. Gradient Descent

✓ Calculate derivative (gradient)
✓ Update weights using gradients during backpropagation.

1 epoch - forward + backward prop through entire training set.

✓ considers whole dataset during 1 iteration (processes entire dataset as a whole)

dataset - n samples

epoch 1

iter 1 = n samples → (forward prop on all n samples together)
← (Backward prop for entire dataset at once)
+ loss)

2. Stochastic gradient descent (SGD):

GD but for every individual sample
(considers 1 sample for 1 iteration)

dataset - n samples.

epoch - 1

iter 1 → 1st sample
→ (Forward prop)
← (Backward)

n samples
⇒ n iters/epoch

iter 2 → 2nd sample
→
←
4 based on loss incurred for 1st sample alone

⋮

iter n → nth sample
→
←

processes every individual sample and updates weight

3. Mini batch SGD:

Divide dataset into batches and process 1 batch during every iteration

Dataset - 100 samples
batch size = 10
⇒ #iters = $\frac{100}{10}$ = 10

1 epoch → 10 iters

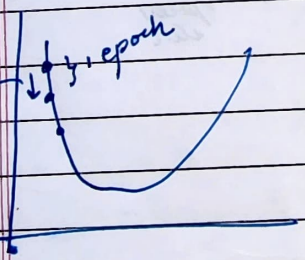
epoch - 1
iter 1 → batch 1
(1 to 10 samples)
iter 2 → batch 2
(11 to 20 samples)

GD | SGD | mini-batch

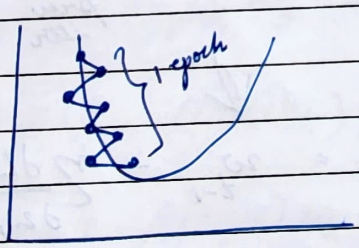
More RAM | less RAM | less RAM (> SGD, < GD ??)

computationally expensive → updation happens only after entire dataset is processed

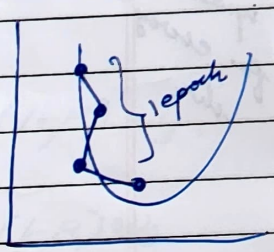
computationally expensive → updation happens for every sample



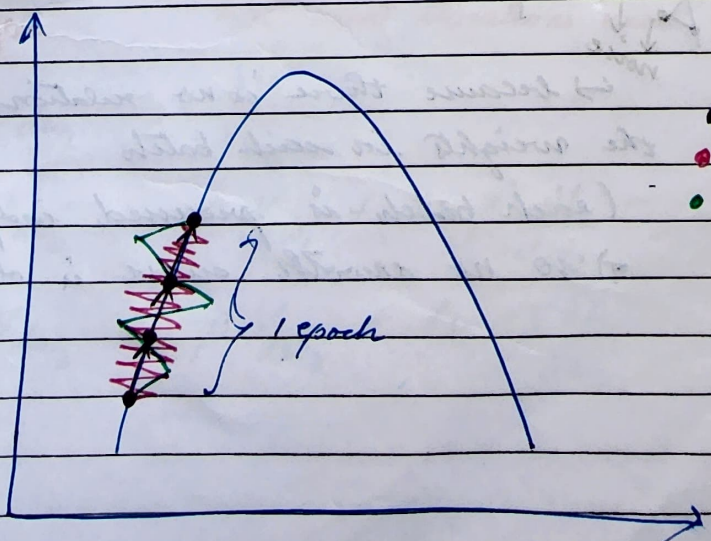
this decrease requires processing entire dataset (longer time to converge ⇒ too many epochs).



more zig-zag line (100 samples ⇒ 100 updations so it won't be uniform as updation depends only on 1 sample).



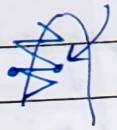
100 samples ⇒ only 10 updations



• - GD.
• - SGD
• - minibatch SGD

Drawback:

All the above methods, do not yield an optimum path to reach the peak (the path followed is not smooth and keeps oscillating between some left and right extremes)



SGD updation

1 updation for every iter

$$(w_{new})_{iter} = (w_{old})_{\substack{iter \\ prev}} - \eta \frac{dL}{d(w_{old})_{\substack{iter \\ prev}}}$$

$$w_t = w_{t-1} - \eta \frac{dL}{dw_{t-1}}$$

minibatch SGD updation

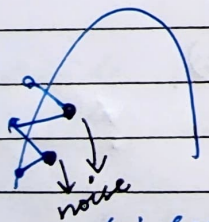
1 updation for every batch

$$(w_{batch})_t = (w_{batch})_{t-1} - \eta \frac{dL}{d(w_{batch})_{t-1}}$$

(No concept of previous iteration used in current iteration

w_t → current iter.'s new value

w_{t-1} → cur. iter's old value which is the same as prev. iter's ~~old~~ new value?



↳ because there is no relationship between the weights in each batch (each batch is processed independently).

⇒ so no smooth curve is observed

4. SGD with momentum :

✓ Gives a smooth curve
✓ Considers previous weights
✓ Exponential weighted average is used

$V_{t_1} = w_1$

$V_{t_2} = \beta V_{t_1} + (1-\beta) w_2 = \beta w_1 + (1-\beta) w_2$

V_{t_1} → velocity
w_i → acceleration

importance given to previous weight (probability).

$V_{t_3} = \beta V_{t_2} + (1-\beta) w_3$

$= \beta(\beta V_{t_1} + (1-\beta) w_2) + (1-\beta) w_3$

$= \beta^2 V_{t_1} + \beta(1-\beta) w_2 + (1-\beta) w_3$

$= \beta^2 w_1 + \beta(1-\beta) w_2 + (1-\beta) w_3$

$V_{t_1}, V_{t_2}, V_{t_3}$ → exponentially weighted avgs.

SGD : $w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$

SGD with momentum : $w_t = w_{t-1} - \eta V_{dw_t}$

V_{dw_t} → derivative wrt current iterations old value

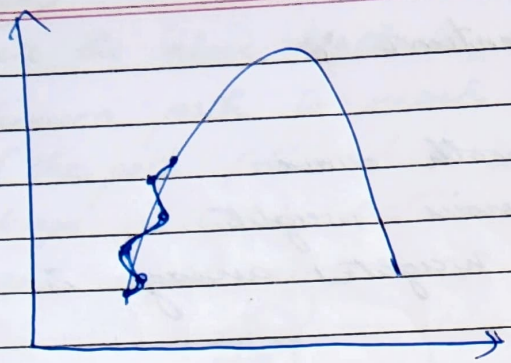
w_t → current iterations new value

$V_{dw_1} = \frac{\partial V_{t_1}}{\partial w_t}$

$V_{dw_2} = \beta V_{dw_1} + (1-\beta) \frac{\partial L}{\partial w_2}$

$V_{dw_3} = \beta V_{dw_2} + (1-\beta) \frac{\partial L}{\partial w_3}$

the same as prev iter's new value ??



(smooth curve).

curve depends on β value

11/10/22

5. Adagrad :

→ learning rate (η) is ~~up~~ dynamic (Adaptive gradient descent).

→ gives better results than SGD with momentum

$$w_t = w_{t-1} - \eta'_t \left(\frac{\partial L}{\partial w_{t-1}}\right)$$

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_t + \epsilon}} \quad \text{where} \quad \alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w}\right)_i^2$$

(cumulative sum)

$$\underline{t=3}$$

$$\alpha_3 = \left(\frac{\partial L}{\partial w_1}\right)^2 + \left(\frac{\partial L}{\partial w_2}\right)^2 + \left(\frac{\partial L}{\partial w_3}\right)^2$$

η → ~~is~~ initial value we assign for the learning rate

Disadvantage :

May face vanishing gradient prob (η'_t → might be very small if α_t is large

→ very small updation, so model does not learn much).

→ provides better convergence than SGD with momentum

6. RMSProp :

→ Root mean square propagation

Change α_t and use ~~root mean~~ exponential averaging to compute it

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{V_t + \epsilon}} \left(\frac{\partial L}{\partial w_{t-1}} \right)$$

$$V_t = \beta V_{t-1} + (1-\beta) \left(\frac{\partial L}{\partial w_{t-1}} \right)^2$$

→ Improvement to Adagrad

→ Instead of computing cumulative sum, it considers exponential weighted average

7. Adadelta :

→ Adaptive delta optimization

→ Completely removes η terms and added delta term

$$w_t = w_{t-1} - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{V_t + \epsilon}} \times (\Delta w_t)^2$$

where $\Delta w_t = w_t - w_{t-1}$

$$D_t = \beta_1 D_{t-1} + (1-\beta_1)(\Delta w_t)^2$$

$$V_t = \beta_2 V_{t-1} + (1-\beta_2) \left(\frac{\partial L}{\partial w_t} \right)^2$$

→ improvement to Adagrad (just like RMSProp)

8. Adam

→ Adaptive moment estimation.

→ Combines momentum and RMSProp.

$$w_t = w_{t-1} - \frac{\eta * V_{dw_t}}{\sqrt{S_{dw_t} + \epsilon}}$$

(V_{dw_t}: momentum; S_{dw_t}: RMSProp)

$$V_{dw_t} = \beta_1 V_{dw_{t-1}} + (1-\beta_1)\frac{\partial L}{\partial w_t} \quad \text{Momentum}$$

$$S_{dw_t} = \beta_2 S_{dw_{t-1}} + (1-\beta_2)\left(\frac{\partial L}{\partial w_t}\right)^2 \quad \text{RMSProp}$$

After bias correction

Adam formula

$$\Rightarrow w_t = w_{t-1} - \frac{\eta * \frac{V_{dw_t}}{(1-\beta_1)^t}}{\sqrt{\frac{S_{dw_t}}{(1-\beta_2)^t} + \epsilon}}$$

→ this is done so that we consider the entire gradient $\frac{\partial L}{\partial w_t}$ ~~(without weighted)~~ ??

without any exponential averaging

$$w_t = w_{t-1} - \frac{\eta * V'_{dw_t}}{\sqrt{S'_{dw_t} + \epsilon}}$$

$$V'_{dw_t} = \frac{V_{dw_t}}{(1-\beta_1)^t}$$

$$S'_{dw_t} = \frac{S_{dw_t}}{(1-\beta_2)^t}$$

Epoch - 1

$w_{t\,new} = w_{t-1}$

$w_{t\,new} = w_{t-1} - \eta \cdot \nabla dw_{t-}$

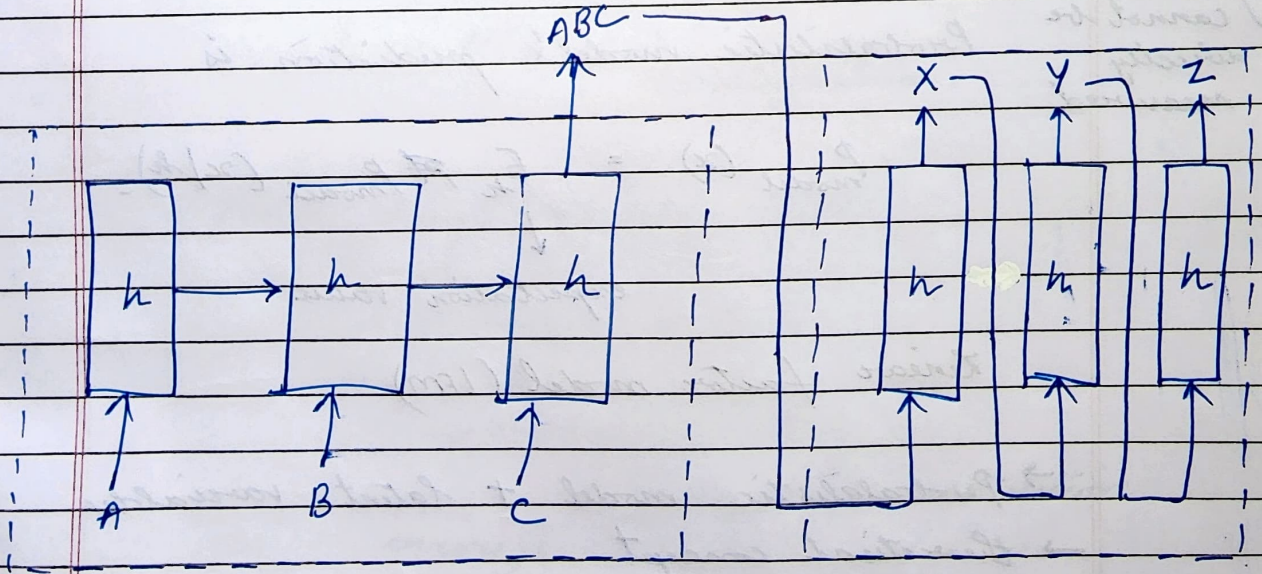
18/10/22

Encoder Decoder RNN

Text data

i/p → strings of lowercase, uppercase letters, digits, punctuations, whitespaces, ...

Input - ABC
output - XYZ
} translation of a sentence



Encoder (processes the input) → comprising

Decoder (does the translation)

RNN - 1 activation fn only

LSTM - has 4 gates, > 1 activation fns (sigmoid, tanh, ...)

Unit 3

Qns asked in mess feedback form :

6 features

- ✓ ambience
- ✓ Arrangement
- ✓ Cleanliness

} Service

only 2 features

- ✓ Quality of food
- ✓ Quantity of food
- ✓ Variety in menu

} Food

⇓

can be grouped as above

⇒ Feature reduction

Techniques - PCA, LDA, ICA

Latent variables → new features obtained by say reducing or feature engineering input features.

- ✓ inferred from the observed variables
- ✓ cannot be directly measured

Probabilistic model's prediction is

$$p_{model}(x) = E_h \, p_{model}(x/h)$$

↓

expectation value

Linear factor model (LFM)

→ Probabilistic model + latent variables
→ theoretical concept
→ linear decoder that generates x by adding some noise to a linear transformation of h

Data generation process

1. sample h from a distribution

$$h \sim p(h)$$

$$p(h) = \prod_i p(h_i)$$ (processing each letter of a word as in encoder-decoder).

$$\Downarrow$$

$$p(ABC) = p(A) \cdot p(B) \cdot p(C)$$

2. sample

$$x = Wh + b + \text{noise}$$

Factor analysis

$$h \sim N(h; 0, I)$$

↳ identity matrix.

Read Deep Learning book

- ✓ Factor analysis
- ✓ PCA
  - → probabilistic PCA (to remove reconstruction errors) ↳ same as PCA as $\sigma \to 0$.
- ✓ ICA
  - → prior $p(h)$ is fixed
  - → deterministically generates $x = Wh$

applications