Course-4

Week 1 : Computer Vision

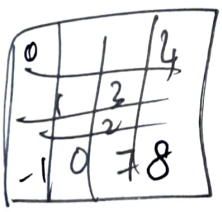
Computer Vision Problems
- → Image classification
- → Object detection
- → Neural style Transfer (Artwork)

[Merge your face with some filter]

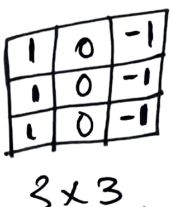
Edge detection example

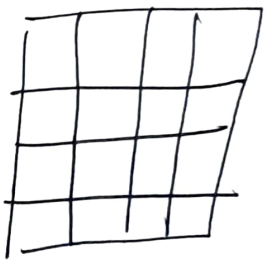
Task 1.: Given an image find its vertical and horizontal edges.

Vertical Edge detection



Convolution operator





Suppose 6×6×1 grayscale

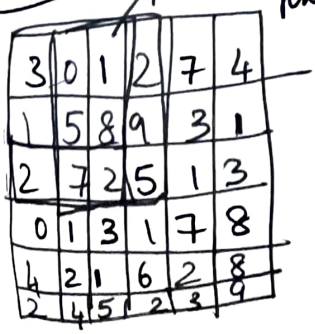
3×3

- called as filter (or) Kernel

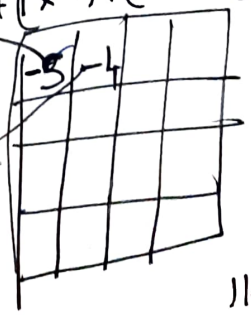
4×4

Take a 3×3 in the 6×6 matrix

element wise product



$(3\times1)+(1\times2)+0\times0+5\times0+7\times0$
$+(1\times-1)+(8\times-1)+(2\times-1) = -5$



-5 -4

Similarly do for other

for the next box shift the 3×3 by 1 to right

$(0\times1)+(5\times1)+7\times1+(-1)(2) - 9 - 5$
$+0\times1+0\times8+0\times2$

softmax.

keep sliding the 3×3 matrix over the 6×6 matrix

∴ final ans = 4×4

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

} Vertical edge detector

another image

(4×4)

python: * → conv-forward
tensflow → tf.nn.conv2d
keras → conv2D

→ what asterik means in different languages

Intuition of the filter

1	0	-1
1	0	-1
1	0	-1

Dark on the left image

Light on the right image

∴ Edge is detected

More Edge Detection

Smaller the pixel value, darker the colour

eg 1 0

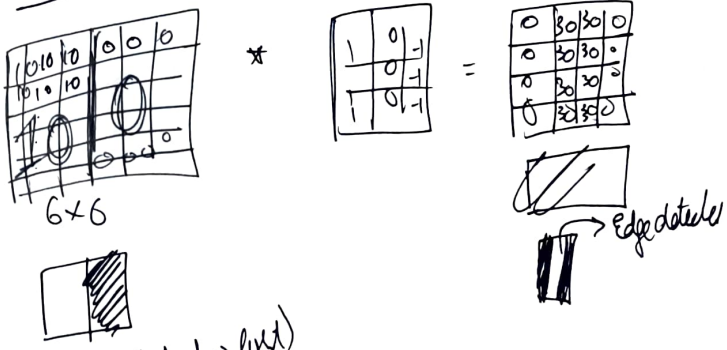
(+ve edge detection)
↓ light to dark transition

(-ve edge detection)
↓ dark to light transition

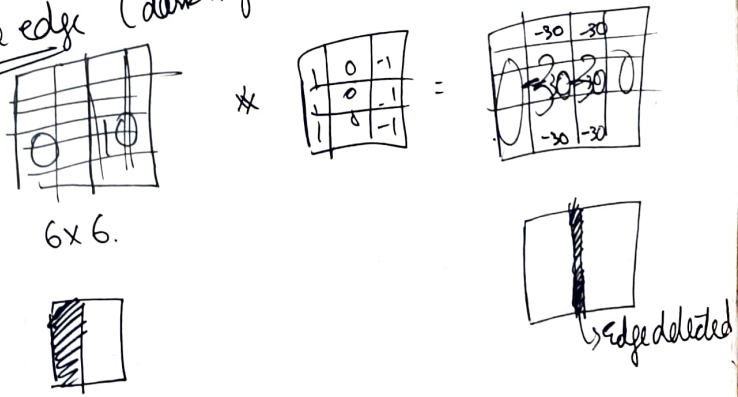
Horizontal Edge filter

1	1	1
0	0	0
-1	-1	-1

Vertical +ve edge (Light → dark)



-ve edge (dark → light)



- Again, there are other filters to!!

Eg:

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

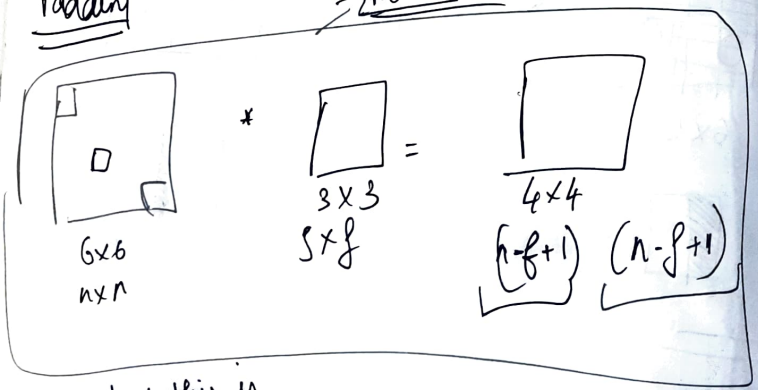
Scharr filter

- Again you can just use DL to learn the filter matrix. GG!!

The backprop will learn the 9 numbers in the filter !!

Padding

Normal



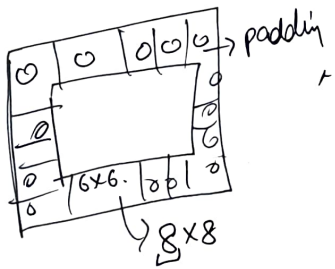
Downside of this is

⇒ You are shrinking the image ie (6×6 → 4×4)

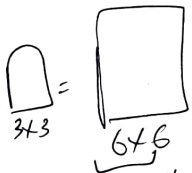
The edge pixel (1,1) and (6,6) are used only once, but the middle ones say (3,3) are used twice or thrice atleast!,

its as if you are discarding the ~~edges~~ corners of the image

(∵) you pad the image



p = padding = 1 (here)



GG!!

$(n+2p)-f+1$

$= 6+2(1)-3+1$

$= 6 \times 6$

How much to pad?

Valid and Same convolutions

Valid: NO PADDING: (p=0) $n \times n \quad * \quad f \times f \rightarrow (n-f+1) \times (n-f+1)$

Same convolution: Pad so that output size is same as input size.

called as zero padding

$(n+2p)-f+1 = n$

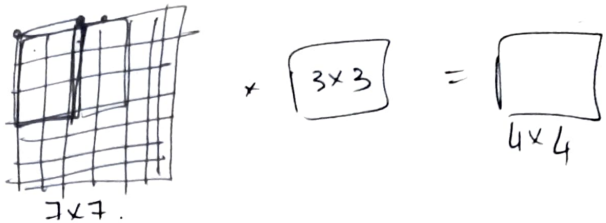
$p = \frac{f-1}{2}$

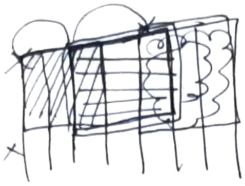
⇒ f is usually odd

↳ 3×3, 5×5, 7×7

Strided convolution

- You move the box while convolving with stride 1





If stride = 2

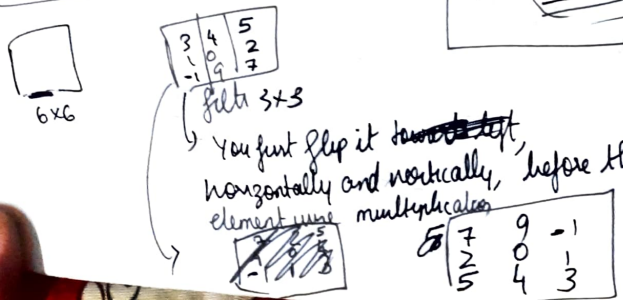
2 ~~steps~~ move by 2 rows, 2 colors while going down going sideways.

$n \times n$ padding p $*$ $f \times f$ stride s

$$\left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-f}{s}+1\right)$$

if it isn't a proper number, take FLOOR

Actual convolution in math textbook

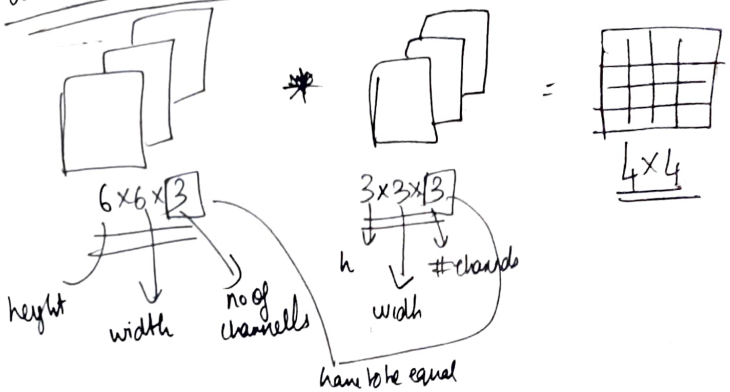


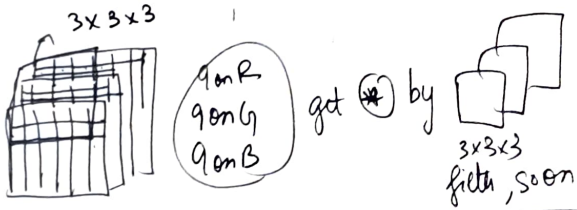
- What we have done is called cross-relation rather than convolution

This is what is done in DL!

Doesn't matter!

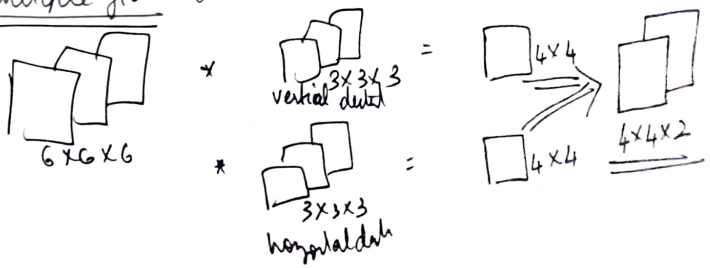
Convolution over Volumes (RGB images)





Multiple filters

(Both horizontal, vertical edge detector)

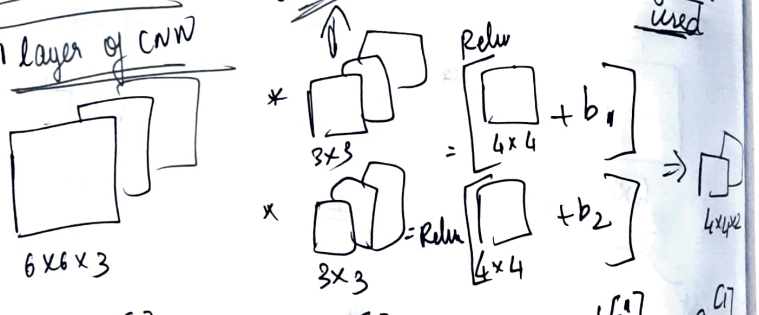


Summary

$n \times n \times \underbrace{n_c}_{\text{no of channels}} \quad * \quad f \times f \times \underbrace{n_c}_{\text{no of channels}}$

$= (n-f+1) \times (n-f+1) \times \underbrace{n_c'}_{\text{no of filters used}}$

1 layer of CNN



$a^{[0]}$ $\quad w^{[1]}$ $\quad b^{[1]}$ $\quad a^{[1]}$

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

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

1. You have 10 filters of $3 \times 3 \times 3$, how many parameters should it learn?

1: 

$3 \times 3 \times 3 + b$

$= 28$ parameters per filter

∴ 280 parameters for 10 filters

less prone to overfitting

- You have to learn 280 parameters irrespective of your input image size! (∴ No overfitting)

Notation

$f^{[l]}$ = filter size.

$p^{[l]}$ = padding

$s^{[l]}$ = stride

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

filter: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

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

o/p: $n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$n^{[l]} = \left[\frac{n + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right]$ floor.

Activation: $a^{[l]} \rightarrow n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$A^{[l]} \rightarrow m \times n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

weight $= f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times \underbrace{n_c^{[l]}}_{\text{no of filters}}$

bias : $n_c^{[l]}$ dimension

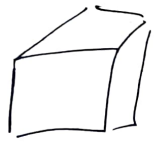
CNN eg

$39 \times 39 \times 3$

$n_H^{[0]} = n_W^{[0]} = 39.$

$n_C^{[0]} = 3$

$f^{[1]} = 3.$
$s^{[1]} = 1$
$p^{[1]} = 0$
10 filters



$a^{[1]}$

$37 \times 37 \times 10$

$\left(\frac{n+2p-f}{s} + 1\right)$

$n_C^{[1]} = 10.$

another layer.
$f^{[2]} = 5$
$s^{[2]} = 2$
$p^{[2]} = 0$
20 filters

$a^{[2]}$

$17 \times 17 \times 20.$

$n_C^{[2]} = 20$

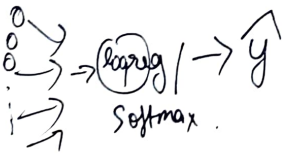
$n_H^{[2]} = n_W^{[2]} = 17$

another layer.
$f^{[3]} = 5$
$s^{[3]} = 2$
40 filters

$a^{[3]}$

$7 \times 7 \times 40.$
$= 1960.$

Unroll it

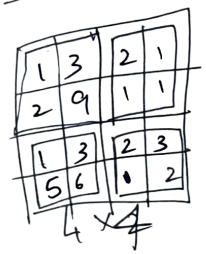


1960

Types of layer in CNN

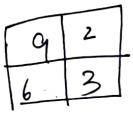
→ Convolution (CONV) ← we have only seen this
→ Pooling (POOL) ←
→ Fully connected (FC) ←

Pooling layers : Max pooling (Taking the max number from each filter)



4×4

$f = (2)$
$s = 2$



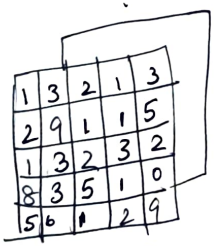
Taking max in each filter

$2 \times 2.$

Intuition : Think of 4×4 as some set of features in a NN.
If there is a large number, then it's prolly bcoz it detected something!

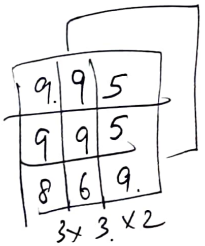
⇒ It has no parameters to learn!

Another eg:



$5 \times 5 \times 2$

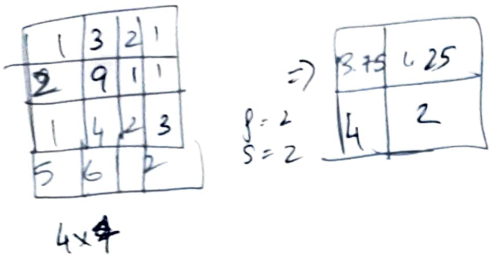
$f = 3$
$s = 1$



$3 \times 3 \times 2$

If there are multiple channels, you go channel by channel do one layer, then go to behind layer and so on

Average pooling (as the name suggests) takes avg



- max pooling is used much more frequently!

Hyperparameters

- mostly $f=2$ $S=2$ is used. [Mostly no padding]

$f=3$ $S=2$

→ max or avg pooling.

Quick handy trick

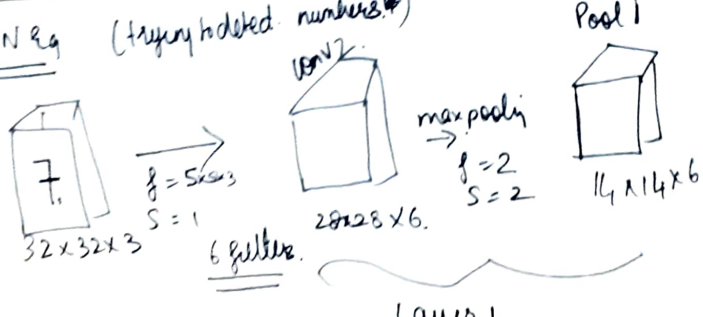
$f=2$ $S=2$ reduces it by a factor of 2

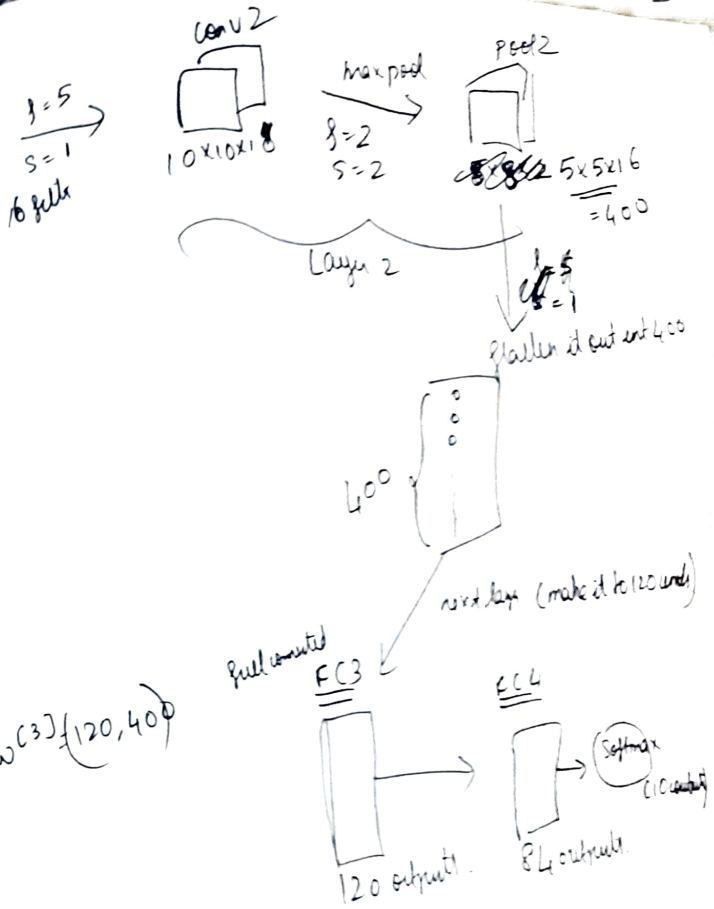
i.e. 28×28 ($f=2$, $S=2$) → 14×14

Pooling: NO PARAMETERS TO learn.

its a fixed function

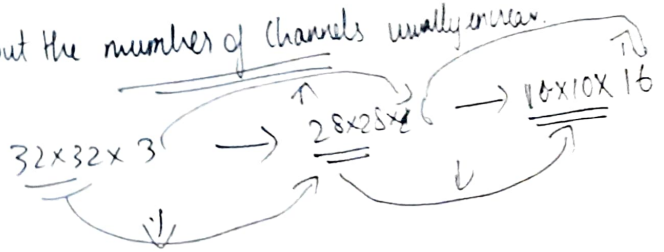
NN Eg (trying to detect numbers)





As you go deeper into the NN, the (n_H, n_W) will decrease (height, width)

but the number of channels usually increase.



NOTE:

- The activation go down as you progress deeper in the NN.

$32 \times 32 \times 3 = 3K$

$5 \times 5 \times 16 = 400$

$84 \times 1 = 84$

$10 \times 1 \rightarrow 10$

decrease

- It shouldn't drop too quickly

- As for the hyperparameters (choose from some already worked literature)

Why convolutions?

Suppose fully connected layer

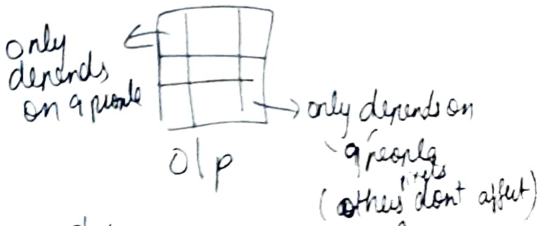
$100 \times 100 \times 3$ — You need to learn ($50 \times 50 \times 3$, 10 of $k3$) parameters

$50 \times 50 \times 3$

- There will too many parameter if you use fully connected network
- There is parameter sharing in convolution
  - ↳ a feature detector (such as a vertical edge detector) that's useful in one part of image is prolly useful in another part of image.

You can use the detectors over and over.

→ Sparsity of connections: In each layer, each o/p value depends only on a small no of inputs.

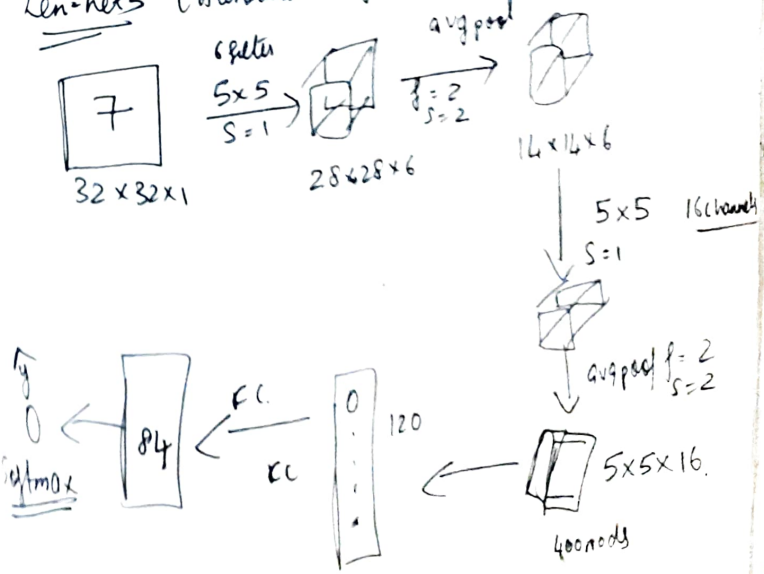


→ Less prone to overfitting

→ Translation variance (Even if a cat is shifted a few pixels to the left or right, it still a cat)
(more robust)

Week 2 : Classical Networks

Len-nets (Handwritten digit)



Sequence Models

Course 5 week 1

Ex: Speech recogn, music generation, Sentimental analysis, DNA sequence, Translation, Video activity recog, Name entity recog.

. Supervised learning

Notation (Name Entity recog)

x: H.P and Hermoine grang invented newspell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ . . . $x^{<t>}$

y: 1 1 0 1 1 0 0 0 0

$y^{<1>}$ $y^{<2>}$ . . . $y^{<9>}$

T = Temporal sequence

$T_x = 9$ $T_y = 9$.

→ t^{th} element in i^{th} ex: $x^{(i)<t>}$, $y^{(i)<t>}$

$T_x^{<i>}$ y length of i^{th} eg.

$T_y^{<i>}$ y l. of i^{th} eg in O/P.

Vocabulary (Dictionary)

a	1
aaron	2
Harry	4075
potter	6830
Zulu	10,000

one hot on x: encoding

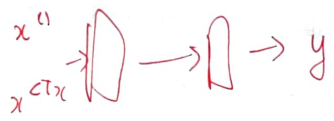
Harry: 4075 → 10,000

Potter: 6830 → 10,000.

$x^{<7>}$ a → 1 → 10,000.

$x^{<t>}$ is a one hot vector.

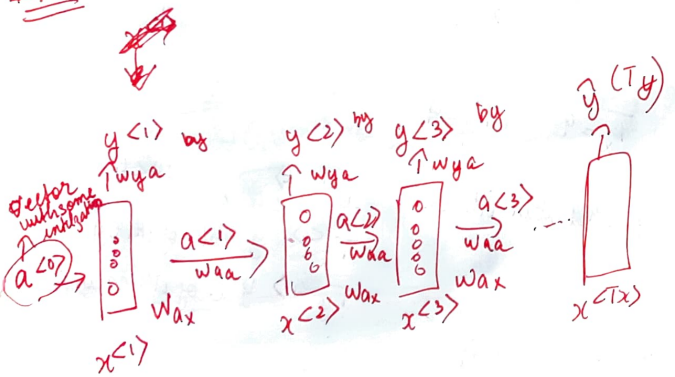
Why not Standard network.

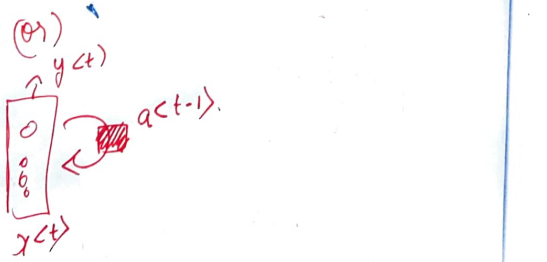


Problems: I/P O/P lengths in different eg:

Doesn't share features learned a cross different position of text.

RNN





Waa, Wax, Way are the same throughout

Backprop: You get learning info / Input from previous $x^{<t>}$ (prev words) not from the next forward set of words

$a^{<0>} = 0$.

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

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

some other activation function

Mostly it will tanh or relu

Mainly

Mostly sigmoid or softmax.

Forward prop.

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

$$\hat{y}^{<t>} = g(W_{ya} a^{<t>} + b_y)$$

Simplified version

$$a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$

$$W_a = [W_{aa} : W_{ax}]$$

$$[a^{<t-1>}, x^{<t>}] \quad [a^{<t-1>} \; x^{<t>}]$$

Eg

$W_{aa} = 100 \times 100$

$W_{ax} = 100 \times 10000$

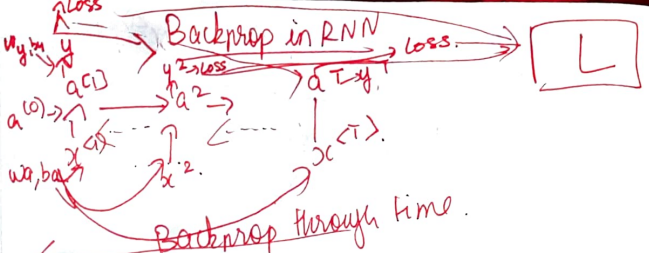
$W_a = (100, 10,000)$

$[a^{<t-1>}, x^{<t>}] = 10100$

$$[W_{aa} : W_{ax}] \begin{bmatrix} a^{(t-1)} \\ x^{(t)} \end{bmatrix} = W_{aa} a^{<t-1>} + W_{ax} x^{(t)}$$

Basically the same.

$\hat{y}^{<t>} = g(W_{ya} a^{<t>} + b_y)$



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

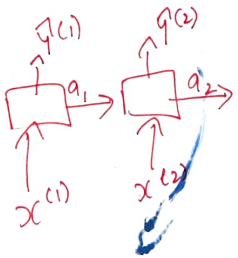
$$L = \sum_{t=1}^{T_y} L^{<t>}(\hat{y}^{(t)}, y^{<t>})$$

Sometimes in RNN length of I/p and o/p sequence aren't equal

Eg: Machine Translation

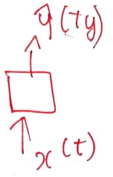
Different types of RNN

Until Now $T_x = T_y$.



Many to Many architecture (many I/ps and many o/ps)

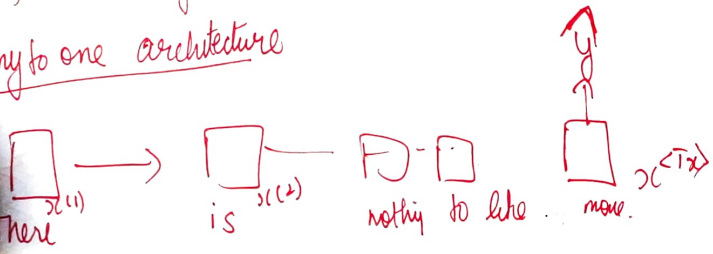
Eg: Name entity recognition



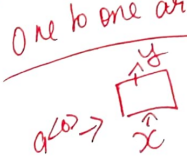
1. Sentiment

x = Text y = 0/1 or 1star 2star etc.

Many to one architecture



One to one architecture



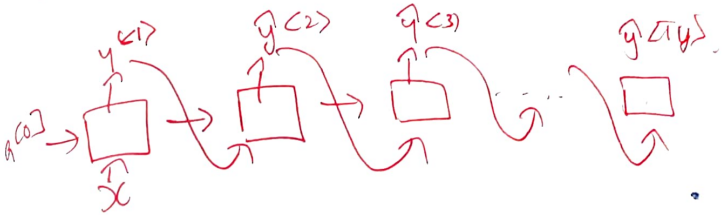
y almost like standard NN.

One to many Arch

Eg: Music generation

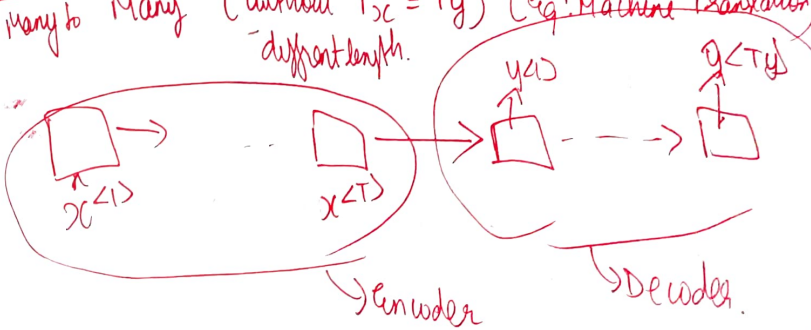
Set of Notes in music → $y^{<1>}$ $y^{<2>}$ . . $y^{<T_y>}$

(x) → genre or some integer.



Many to Many (without $T_x = T_y$) (Eg: Machine Translation)

different length.



In exploding gradient problem You get NaN as O/P

Language Model and Sequence Generation

The apple and pair Salad - } The language modelling
The " and pear salad } will help us choose this

It will o/p a probability of either of these two sentences

$P(\ldots \text{pair salad}) = 3.2 \times 10^{-13}$.

$P(\ldots \text{pear} \ldots) = 5.7 \times 10^{-10}$ } more probable

Language Model
It calculates $P(\text{sentence}) = ?$

$P(y^{(1)}, y^{(2)}, \ldots)$
Sentence 1, Sentence 2

Traenyset: large CORPUS (body) of english text

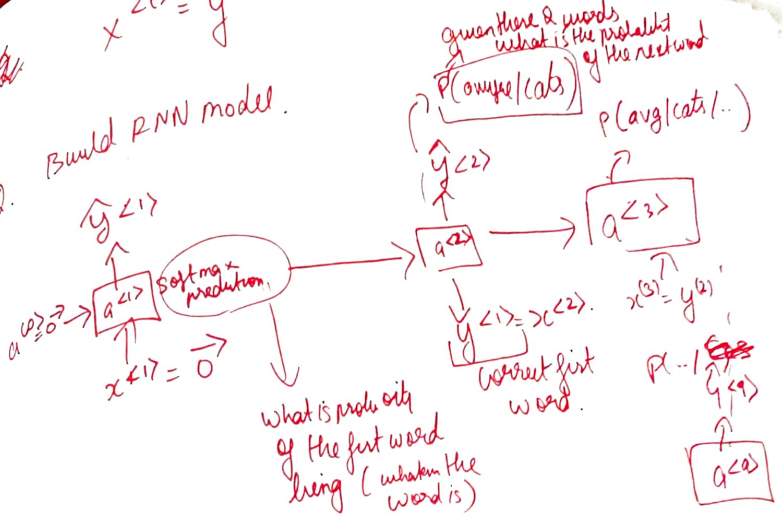
Cats average 15 hours of sleep a day. <EOS> End of sentence / Extra token

1. Tokenize. $y^{<1>}$ $y^{<2>}$ $y^{<3>}$ . . . . $y^{<9>}$.
(one hot encoding)

Suppose: Some word is not there in your dictionary (large corpus) replace it with a <UNK> Token
↳ unknown.

$x^{<t>} = y^{<t-1>}$

2. Build RNN model.



Each step will look at preceding words and calculate the probability of the next word occuring

3 Train RNN

$L(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$ Softmax Loss func.

$L = \sum L(\hat{y}^t, y)$.

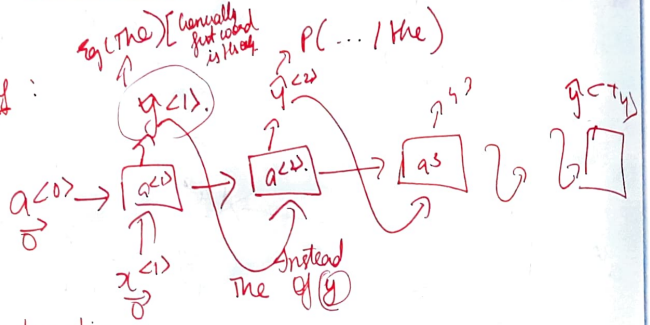
4. Tells us:-

$P(y^{(1)}, y^{(2)}, y^{(3)}) = P(y^{<1>}) \times P(y^{<2>} \mid y^{<1>}) \times P(y^3 \mid y^2, y$

Sentence
Prob of 1st word
Prob of 2nd word given first word

Sampling Novel Sequence (Random Sentence)

Sampling:



np.random.choice

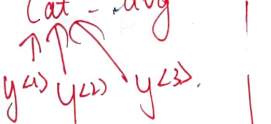
You can keep sampling till <EOS>

(Or)

Just stop when you want 20 or 30 words.

Here we are doing word modelling to build a sentence

You can also do character modelling to build a word



Adv of Char model: No need of <UNK>

Disadv: → Not that effective
→ computationally expensive

You can use this to generate a Shakespeare poem!

Vanishing gradient with RNN

The cat ..... was full

The cats ..... were full.

long statement

- It is difficult for the RNN both during forward and back prop. to remember whether it's saw

Cat or cats!

- Basic RNN, you can't remember so much!
  ∴ Highly influenced by local words
  i.e $y^{(2)}$ is highly influenced by $y^{(1)}$
  $y^{(2)}$ is hardly influenced by $y^{(170)}$!

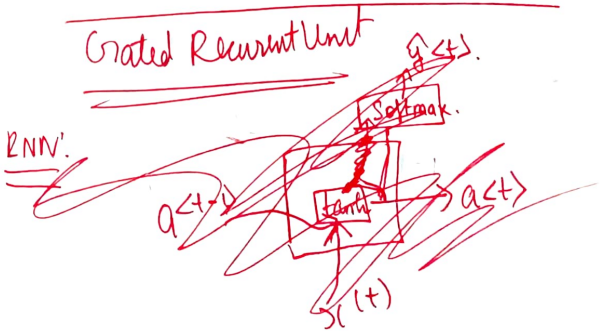
- Can't capture **Long-Range dependencies**

- Exploding gradients → Use gradient clipping
  ↳ see if it is bigger than some threshold, if so rescale the gradient vectors.

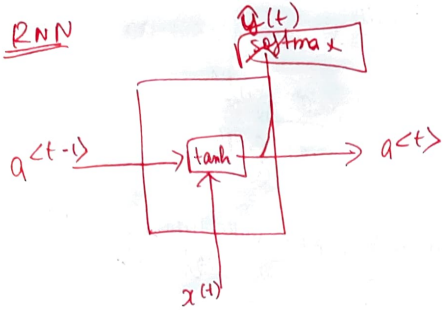
- Vanishing gradients → Use GRU
  Gated Recurrent Unit

Gated Recurrent Unit

RNN:



RNN



GRU (simplified)

C = memory cell.

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

$\tilde{c}^{<t>} = \tanh(w_c[c^{<t-1>}, x^{(t)}] + b)$

Gate $\leftarrow \Gamma_u = [0,1] = \sigma(w_u[c^{<t-1>}, x^{(t)}] + b_u)$

Γ_u ← Gamma u.

"update"

→ will decide whether to remember a word or not

→ will decide when to update.

→ The cat ..... was full.

$c^{<t>} = 1$ E1

1 is singular
0 is plural

E1 → OH! it is singular; that's why it is "WAS".

→ after this, you can forget the value.

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

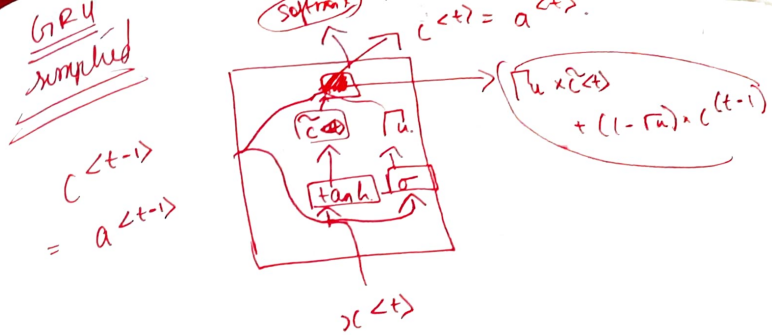
don't change it ← what we want

Just the prev value of $c^{<t>}$.

The cat was full.

$\Gamma_u = 1$ | $\Gamma_u = 0$ $\Gamma_u = 0$ $\Gamma_u = 0$ when $\Gamma_u = 0$.

GRU simplified



How does it solve vanishing gradient?

$\Gamma_u = \sigma(w_u[c^{(t-1)}, x^{(t)}] + b_u)$

↳ If it is a large negative value since it's sigmoid

$\Gamma_u = 0$

Then in the formula

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

$c^{<t>} \underset{\text{almost}}{=} c^{<t-1>}$ ∴ value of $c^{<t>}$ is maintained

∴ Solves vanishing gradient problem.

∴ Can use it over long range dependency.

Full GRU

Btw $\tilde{c}^{\langle t \rangle}$ is called Candidate Memory Value

$$\tilde{c}^{\langle t \rangle} = \tanh\left(w_c \left[\Gamma_r * c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$$

relevance gate, how relevant is $c^{\langle t-1 \rangle}$ for calculating the next $c^{\langle t \rangle}$

$$\Gamma_u = \sigma\left(w_u \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_u\right)$$

$$\Gamma_r = \sigma\left(w_r \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_r\right)$$

$$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$$

Multiple ways of obtaining (Γ_u, Γ_r, ...)
But researches said this is more robust

LSTM (Long short term memory)

Great

$$\tilde{c}^{\langle t \rangle} = \tanh\left[w_c \left[\Gamma_r, a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right]$$

$$\Gamma_u = \sigma\left(w_u\right)$$

LSTM equation

$$\tilde{c}^{\langle t \rangle} = \tanh\left(w_c \left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$$

update gate $\Gamma_u = \sigma\left(w_u \left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_u\right)$

forget gate $\Gamma_f = \sigma\left(w_f \left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_f\right)$

$$\Gamma_o = \sigma\left(w_o \left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_o\right)$$

$$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + \Gamma_f * c^{\langle t-1 \rangle}$$

$$a^{\langle t \rangle} = \Gamma_o * \tanh\left(c^{\langle t \rangle}\right)$$

$$y^{\langle t \rangle} = \text{softmax}\left(a^{\langle t \rangle}\right)$$

3 gates !!

LSTM and GRU are good at remembering values

peephole connection: just an equation change

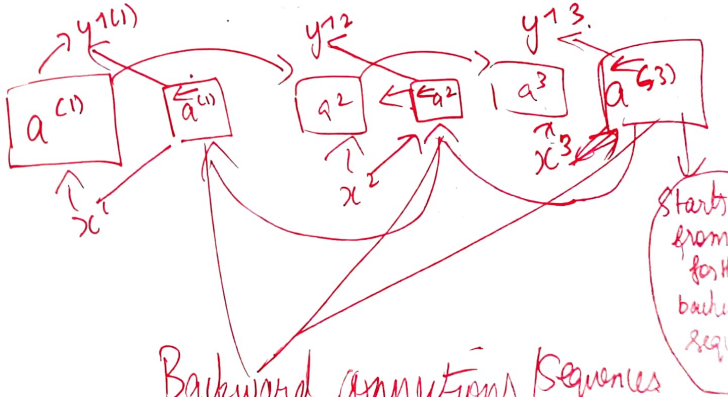
$$\Gamma_o = \sigma\left(w_o \left[a^{\langle t-1 \rangle}, c^{\langle t-1 \rangle}\right] + b_o\right)$$

Adv of GRU: Simpler, easy to compute bigger network

But LSTM is much better!

Bidirectional RNN

Motivating eg: In Name Entity — Teddy Bear / Teddy Roosevelt } You need future information here!



Backward connections Sequences