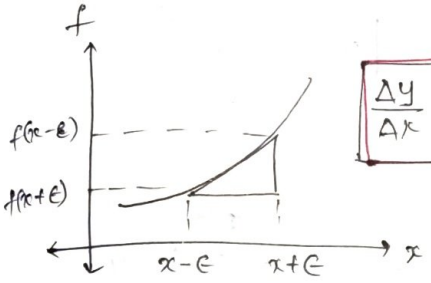
ResNets — Residual Networks

Gradient Checking

* Numerical approximation of derivative of f :

$$\lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

* Actual der → calculated by formula or backprop.



$$\frac{\Delta y}{\Delta x} \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

because ε is small, difference also is small

$$\text{Grad-check} = \frac{\|dw - dw_{approx}\|_2}{\|dw\|_2 + \|dw_{approx}\|_2}$$

→ Euclidean distance

→ vvsmall $< 10e^{-7}$ → correct

$> 10e^{-3}$ → false / not correct

Ex: $W = w_1, w_2$, $X = x_1, x_2$ | $f(w_1, w_2, x_1, x_2) = w_1^2 x_1 + w_2^2 x_2$

$$\frac{df}{dw_1} = 2 \cdot w_1 x_1 \rightarrow (1) \qquad \frac{df}{dw_2} = 2 \cdot w_2 x_2 ; \quad \left(\frac{df}{dw_1}\right)_{approx} = \frac{f(w_1+\epsilon_1, w_2 \cdots) - f(w_1-\epsilon_1, \cdots)}{2\epsilon}$$

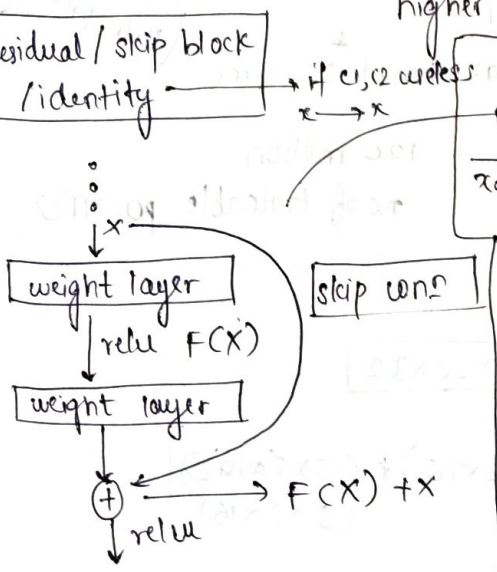
$(dw) = 2 w_1 x_1 = 6 \rightarrow (1)$ $\qquad (dw_1)_{approx} = 5.9999999 \rightarrow (2)$

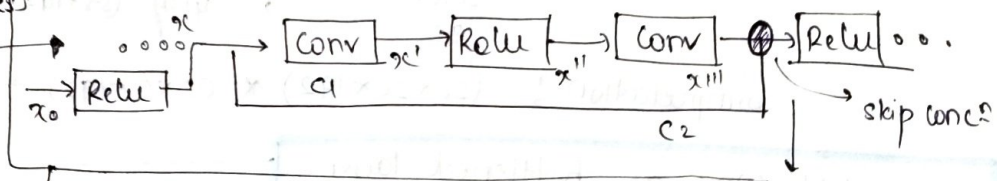
$$\text{Grad-check} = \left(\frac{6 - 5.9999\cdots}{6 + 5.9999}\right) = 4.251268e^{-13}$$

∴ as $< e^{-13}$ → backprop is correct

Resnets :- motivation ⇒ when no of layers were increased, the loss was higher than lesser layers even with huge no of iterations

Residual / skip block / identity → if c_1, c_2 useless $x \rightarrow x$





$x''' + x$ = o/p

why it works?

$\text{Relu}(x_0) = x$, $\text{Relu}(x) \equiv \text{Relu}(\text{Relu}(x_0)) = x_0$

$= \text{Relu}(x_0) = x$.

L_1, L_2, dropout

Now

consider due to strong regularization c_1 and c_2 become useless, ie w or lc=0;

so previously, → now,

o/p = $x''' + x = x$ — so called skip connection

this cont propagate further → o/p = $x''' = 0$

∴ c_1 and c_2 dont matter ie skip

Relu (Relu (x)) = Relu (x)

if +ve, Relu (x) = Relu (x) = x

-ve = 0

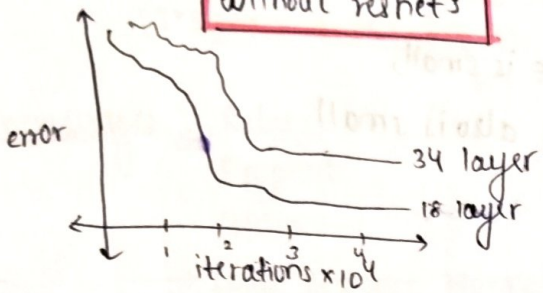
(22) Key takeaway → gaurantee no decrease in performance, small increase

* adding additional layers would not hurt performance as they get skipped if they are found useless.
* if new layers are useful
  reqt weights are non-zero.

Note: x → C1 → relu → C2 → (+ x^(111)) →

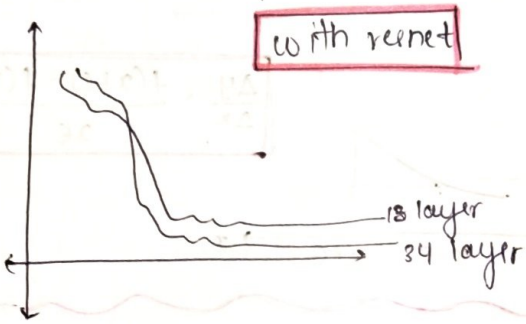
make sure (x) and (x^(111)) have same shape — use padding

without resnets



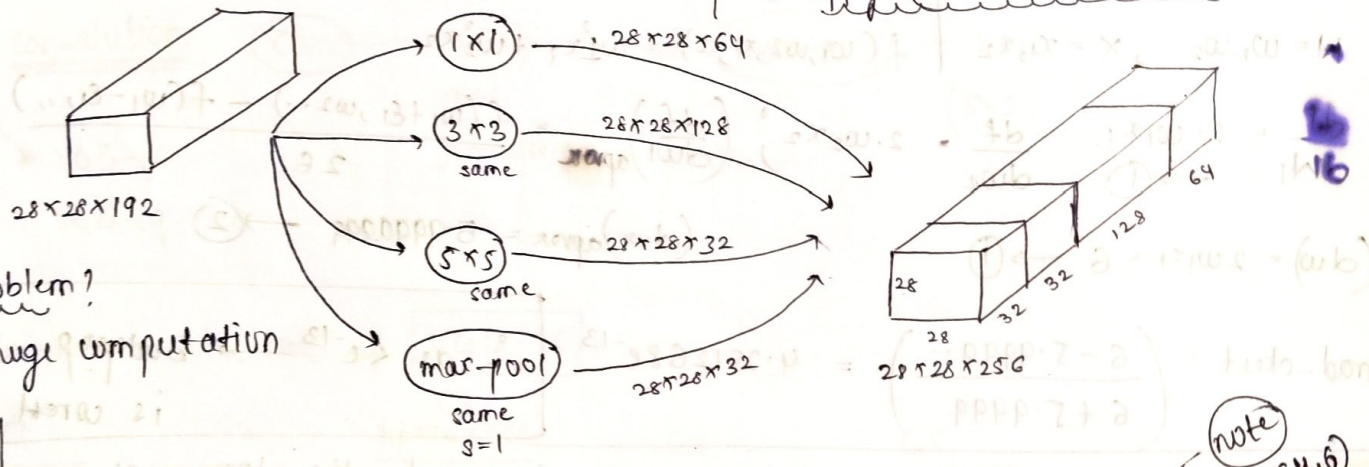
with res-nets →

with resnet



Inception Network → apply more than 1 type of kernels at once

→ apply (1x1x192) kernels 64 times



problem?

huge computation

Ex:

28x28x192 —CONV 5x5, same, 32→ 28x28x32

→ apply (5x5x192) kernel 32 times

note: K = C(4,6) = (4,6, prv_layer_d)

computation :- (28x28x192) * (5x5x192) = 120 million.

no of trainable params

using 1x1 conv as bottleneck layer

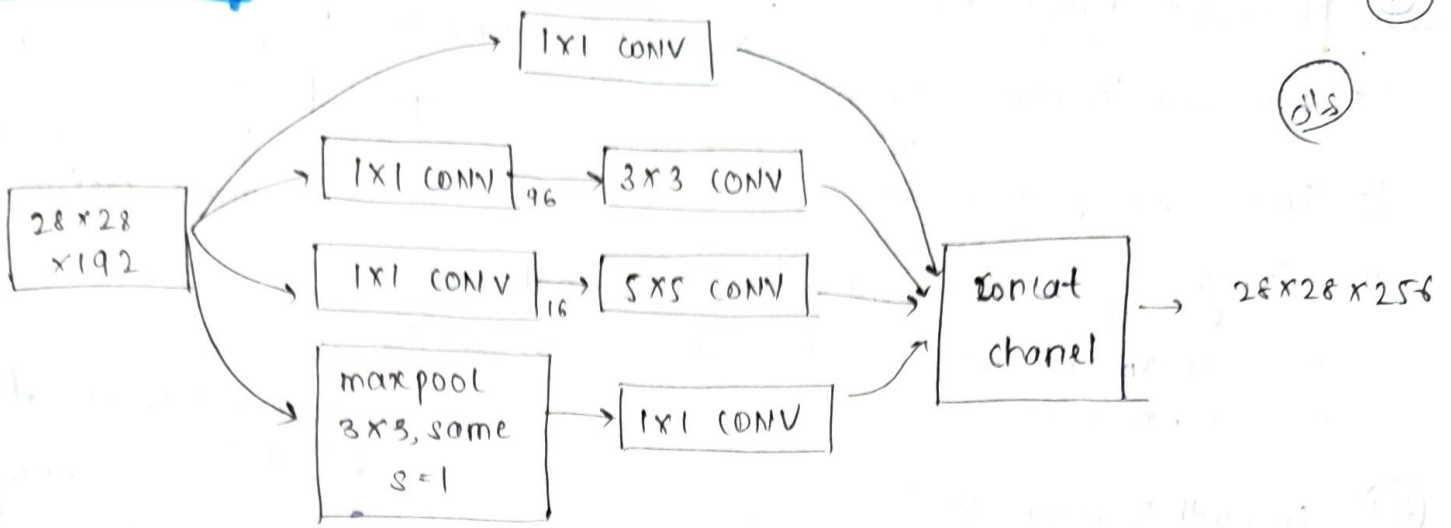
28x28x192 —CONV 1x1, 16→ 28x28x16 —CONV 5x5, 32→ 28x28x32

computations : [(28x28x192) * (1x1x192)] + [(28x28x16) * (5x5x16)]

5x5x16

= 12.4 million <<<< 120 million

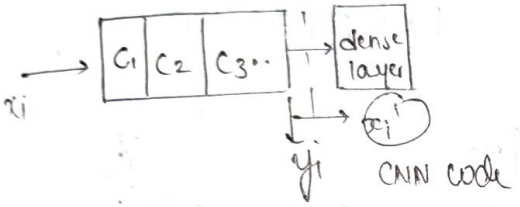
∴ (i/p → conv 1x1 → conv 5x5) is better than (i/p → conv 5x5) computationaly



Transfer Learning → using pretrained models

Ex: vgg16 trained with imagenet dataset ; New dataset = $D = \{x, y\}_i^n$

Case 1 CNN Codes



For x_i in D,
get x_i' → vector obtained from last conv layer

$\therefore D' = \{x_i', y\}$
→ train any ML model

Case 2 fine tune last layers

they identify basics

→ Freeze most layers ⟹ Initial layers are helpful
→ fine tune only last few layers ⟹ use your dataset to train last layers.

Case 3 tune all layers

→ use vgg16 + ImageNet model as a initialization
→ train the model using ur dataset

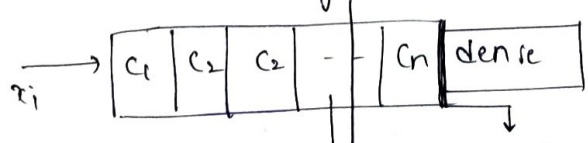
Use cases → where to use?
→ factors → similarity with Imagenet and size of dataset

Case 1 D ≈ ImageNet
|D| small
→ CNN codes.

Case 2 D ≈ ImgNet
|D| large
→ tune full n/w

Case 3 D ≈ ImgNet
|D| medium sized
→ fine tune last few layers

Case 4 |D| is small
D ≠ Img Net



more generalized as c_n will be useful iff similar
→ pick from later CONV layer but not last

Case 5 |D| large
D ≠ ImgNet
→ use pretrained model as init and train/tune whole model

Recurrent Neural Networks

- retains and leverages sequence info

core idea : sequence of inputs

→ app^n
- machine translation
- time series
- speech recognition
- image captioning

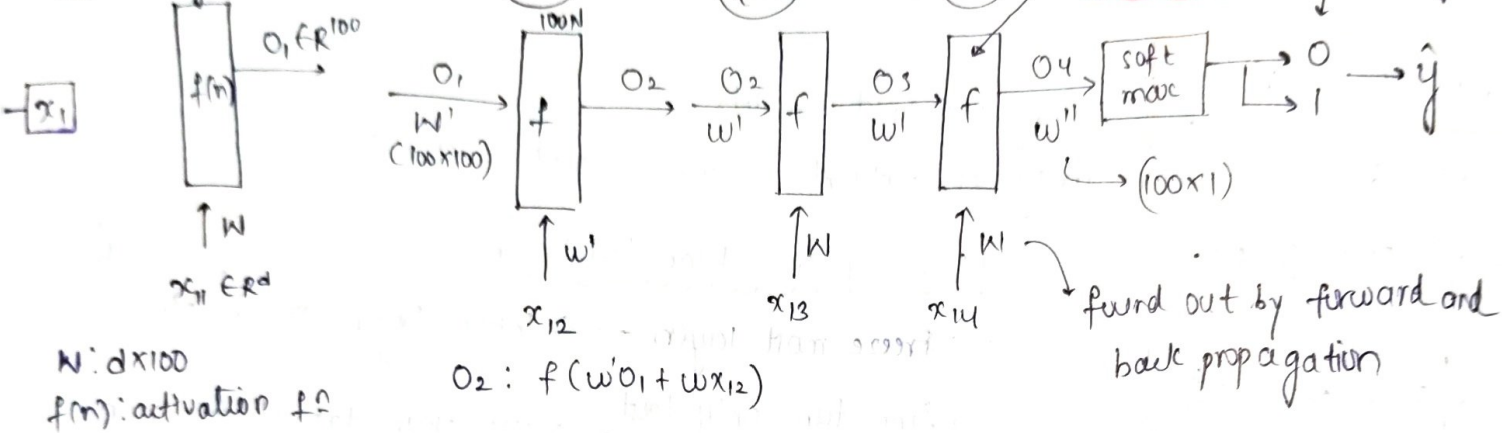
Ex: Amazon fine food reviews

$D = \{x_i, y_i\}$

$x_1 = \langle x_{11}, x_{12}, x_{13}, x_{14} \rangle$ ← each word is 'd' dimensional vector where d = size of vocabulary

$x_2 = \langle x_{21}, x_{22}, x_{23} \rangle$

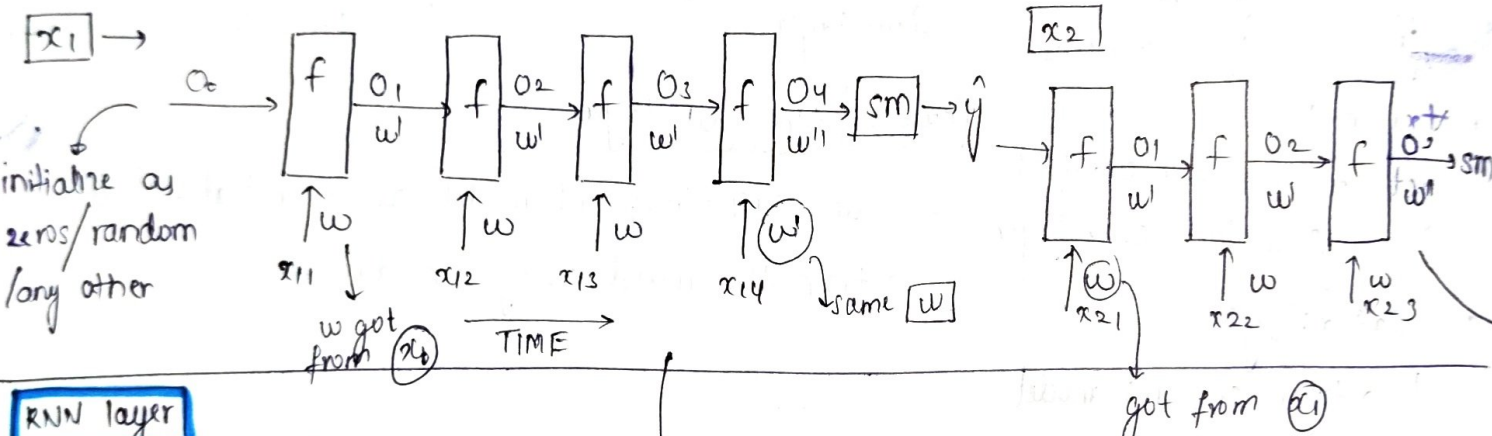
same layer but trained at different time



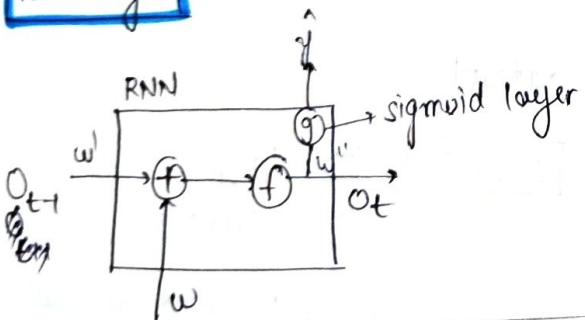
N: d×100

f(n): activation fn

$O_2 : f(w'O_1 + wx_{12})$



RNN layer

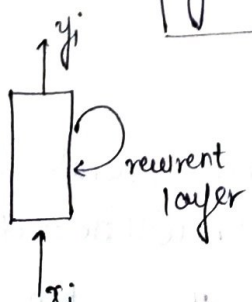


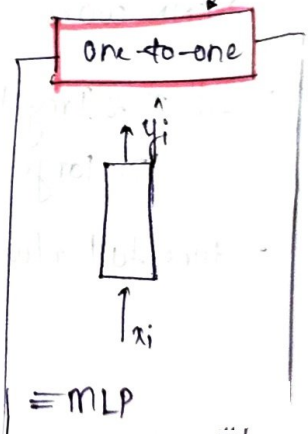
In the whole process we have only 3 matrices, w, w' and w''

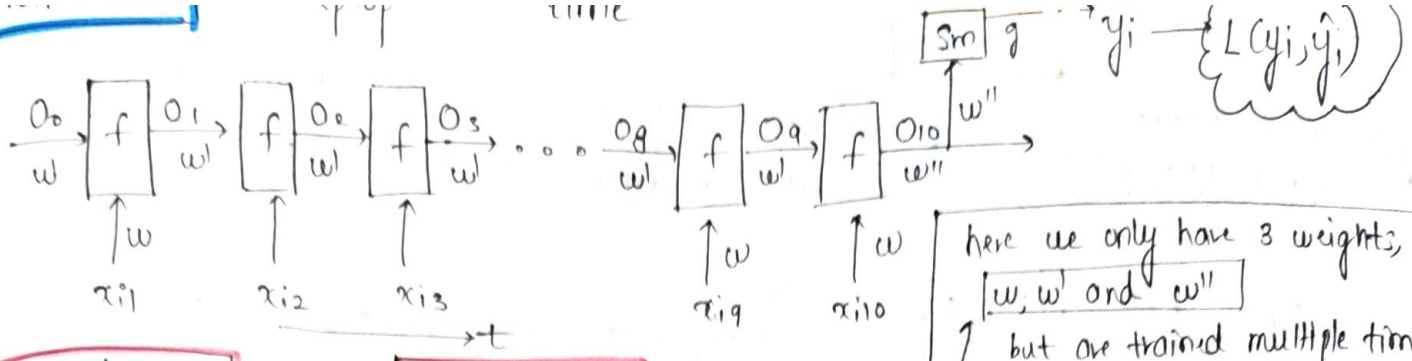
→ they are given different inputs according to time - t

$$O_t = f(wx_{it} + O_{t-1}w')$$

$$\hat{y} = g(O_t w'')$$







here we only have 3 weights, [w, w' and w''] but are trained multiple times

Forward prop

$O_1 = f(O_0 w' + x_{i1} w)$

$O_2 = f(O_1 w' + x_{i2} w)$

$\vdots$

$O_{10} = f(O_9 w' + x_{i10} w)$

$\hat{y}_i = g(w'' \cdot O_{10})$

Backward prop reverse arrows

① $\frac{\partial L}{\partial \hat{y}_i} \rightarrow$ ② $\frac{\partial L}{\partial O_{10}''} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_{10}} \rightarrow$ ③ $\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial O_{10}} \frac{\partial O_{10}}{\partial w_{11}}$

$w_{t-10} := \frac{\partial L_w}{\partial w} = \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial O_{10}} \cdot \frac{\partial O_{10}}{\partial w} \longrightarrow \left(\frac{\partial L}{\partial w}\right)_{t=1}$

$w'_{t-10} \approx \frac{\partial L}{\partial w'} = \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial O_{10}} \frac{\partial O_{10}}{\partial w'} \quad \left(\frac{\partial L}{\partial w'}\right)_{t=1}$

Problems with backprop over time

$\left(\frac{\partial L}{\partial w}\right)_{t=1}$ = multiplication and addition of multiple values $\longrightarrow$ vanishing gradient, Exploding gradient

Types of RNN architectures

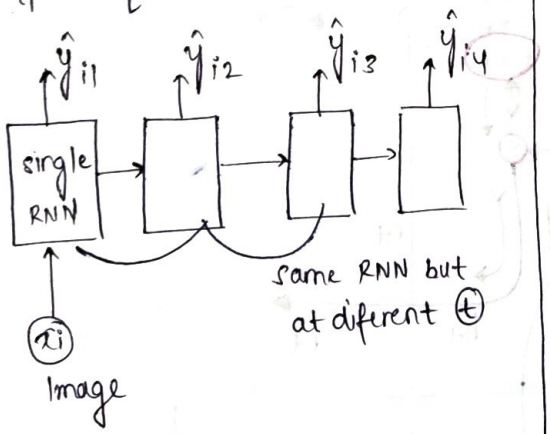
① **Many-to-one RNN**

uses: sentiment analysis, movie review rating

② **One-to-many RNN**

i/p : single

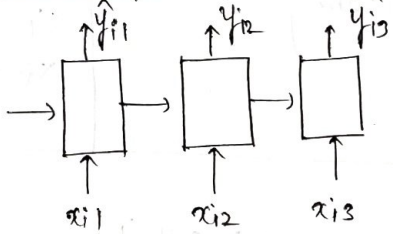
o/p : sequence



Ex: Image captioning

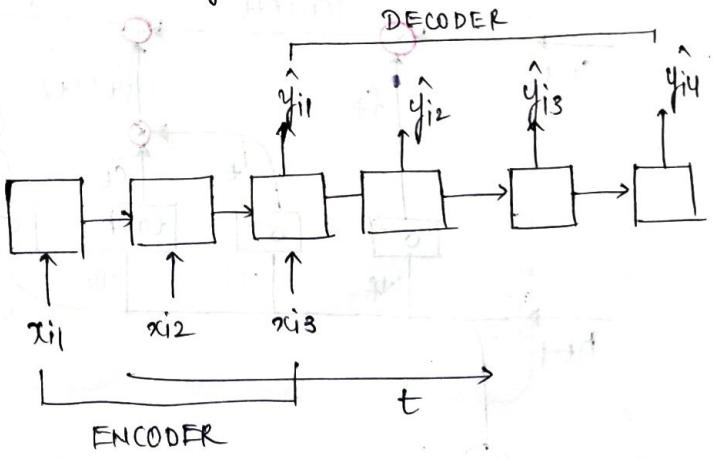
many-to-many len(i/p) = len(o/p) → # word → which part of speech (seq2seq)

① **same length** :- parts of speech



[parallel execution] can be done

② **different length** Encoder-Decoder

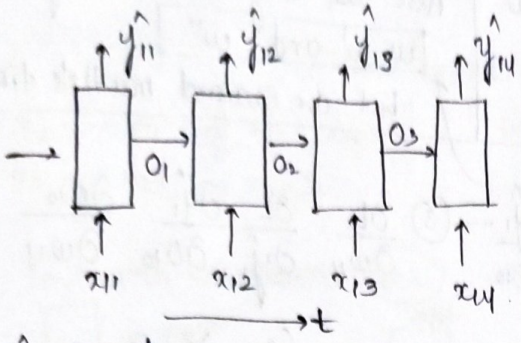


use :- language translation = Eng → spanish

a sentence in Eng with 10 words → may have only 8 words when translated into spanish

LSTM

why we need LSTM when RNN exists? → RNN cant take care of long term dependency ie later o/p depends on earlier input



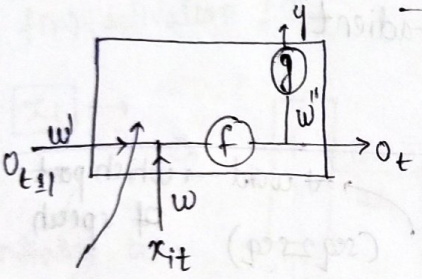
→ Here,
* $\hat{y}_{14}$ depends more on x_{14} and O_3 and depends very less on O_1 and x_{11}
* lets say we want y_{14} to be more dependent on x_{11} which cant be accomplished

Ex:- In some languages starting words might be of more importance when translating

why $\hat{y}_{14}$ depends on x_{11}?
→ there is lot of propagation & time b/w x_{11} and $\hat{y}_{14}$

LSTM

Improved simple RNN



merge vectors → $[w' w]$ (stack one abv other)
concatenate $[x_{it} O_t]$

concatenation

$O_t = f([w, w'] \cdot [x_{it}, O_{t-1}])$

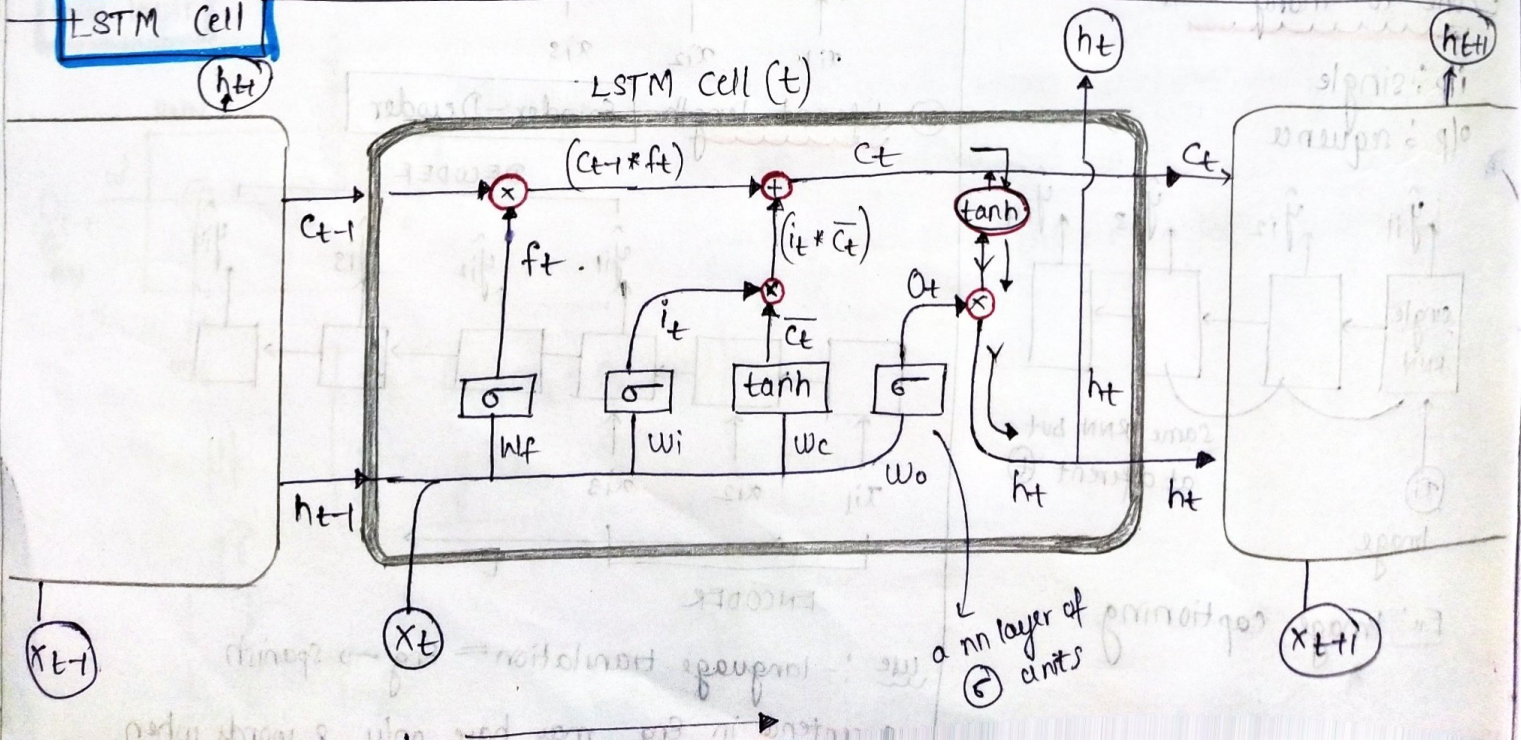
$\hat{y} = g(O_t w'')$

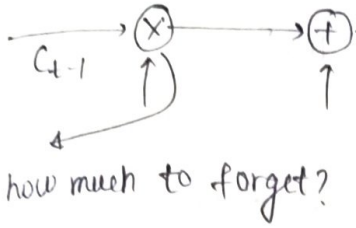
similarity
$C_t \equiv O_t$
$X_t \equiv X_{it}$
$h_t \equiv \hat{y}_{it}$

Notations
○ → point operation (apply 2 each)
□ → Neural network layer
C_t → cell state
X_t → i/p point
→ concatenate
→ copy

Ex: [123] [111] ⊗ [1·1 2·1 3·1]

LSTM Cell

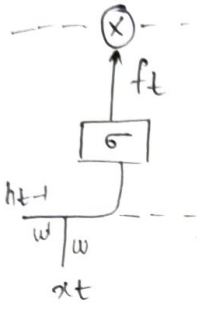




if i/p to ⊗ = [1,1, ...] and i/p to ⊕ = [0,0,0 ...] → $[C_{t-1} \longrightarrow C_t]$

cell / layer is skipped

Forget part



concat

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

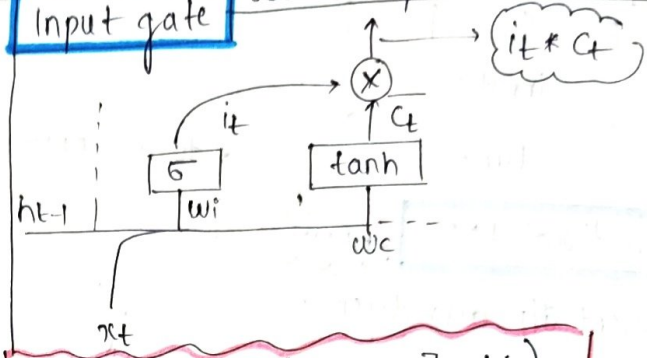
W_f : forget weights $[w, w']$

b_f : bias factor

f_t → how much data to forget

Input gate

add new info



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

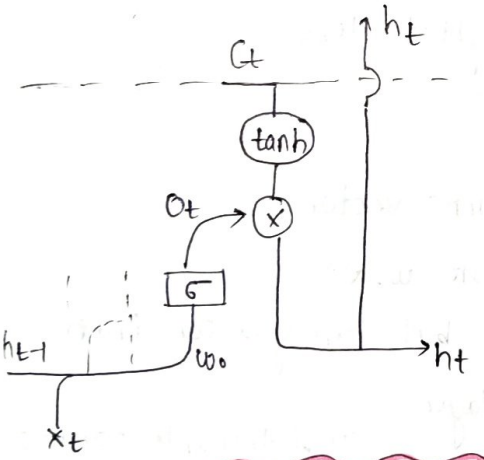
$$\bar{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

identity + forget + input gate ⟹ $C_t = (f_t * C_{t-1}) + (i_t * \bar{C}_t)$

GRU?
* few param → better with less data
* not good for long sequence

Output gate

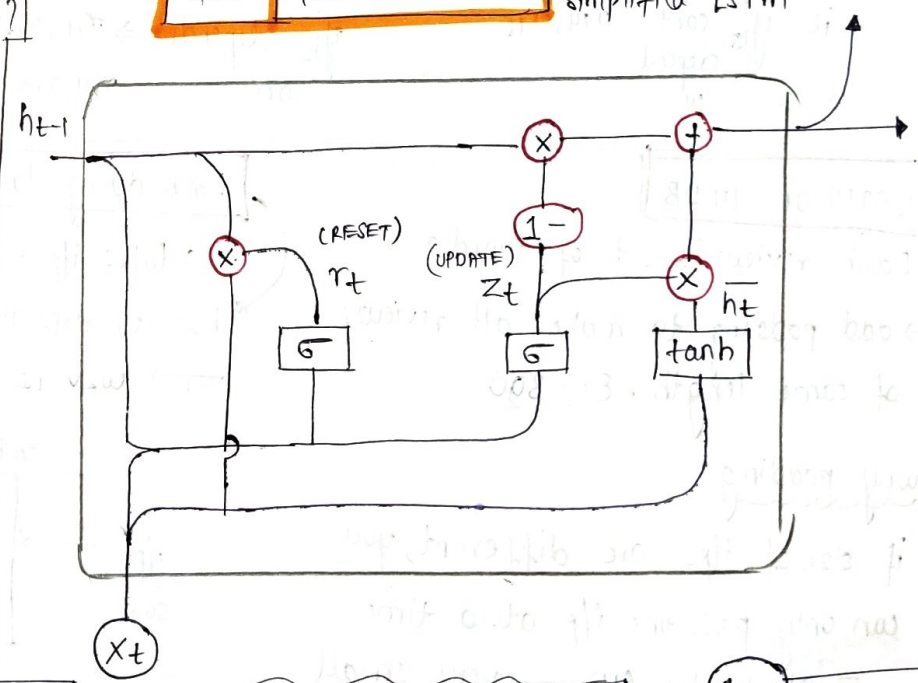
how much op to send?



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

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

GRU | Gated RNN unit | simplified LSTM



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\bar{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + (z_t * \bar{h}_t)$$

1 − [1,1, ...] − [ip vector]

GRU vs LSTM

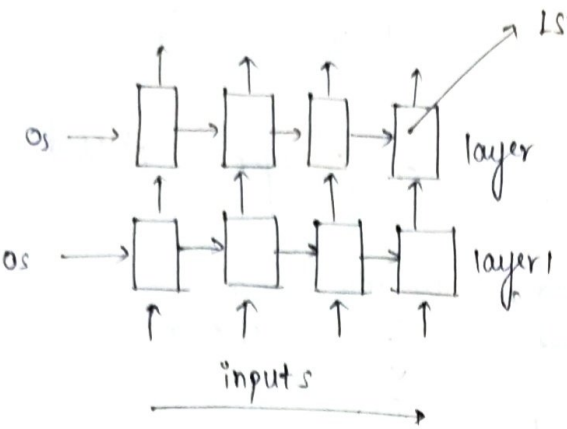
o/p, forget, ip

→ LSTM have 3 gates
GRU have 2 gates — reset, update

∴ As lesser gates → lesser eqn

→ faster to train

Deep RNN



→ Becomes same as MLP, we can use,
- → dropouts
- → Batch Normalization

→ cant be used for live audio stream

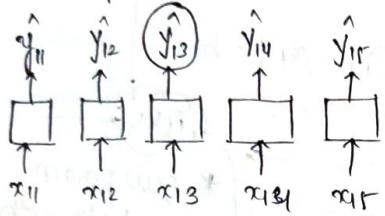
→ to use BRNN whole i/p must be present

Bi-Directional RNN

BRNN

what if current O/p also depends on future i/p ?

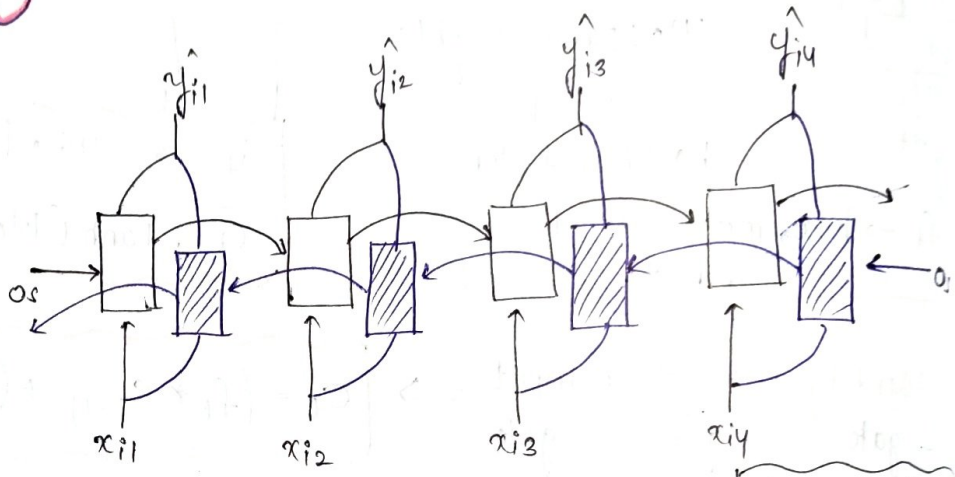
Ex:



$\hat{y}_{13}$ depends upon x_{11}, x_{12}, x_{13}

but cant depend on future i/p

ie $\hat{y}_{13}$ cant depend on x_{14}, x_{15}



Here in BRNN,

$\hat{y}_{i2}$ depends on $\Rightarrow$ x_{i1}, x_{i2} in Forward

x_{i4}, x_{i3} in Right-to-left

Blue – R→L

black – L→R

LSTM on IMDB

Each review is set of words

→ add padding to make all reviews of same length. Ex:- 300

why padding ?

→ if size of i/ps are different, you can only pass one i/p at a time ≡ stochastic GD → very small

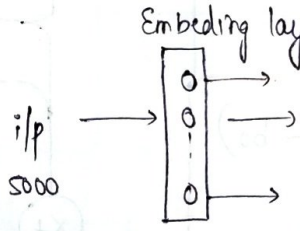
→ if i/p size is same, you can parallelly process inputs like in batch GD → faster

Embedding layer

→ takes i/p → returns vector

how its different from w2v?

→ w2v is static but here you can train



No of trainable params

= 5000 × 32

= 160K params

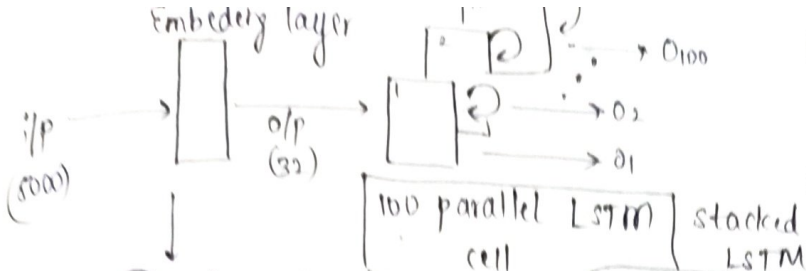
No of train params in LSTM

m = no of inputs = 32

n = no of o/ps = 100

Biases

$$\#params = 4(nm^2 + n^2 + n)$$



input : (25000×600)
output : (100) → (1)
Embeding layer → (600×32)

i/p indices : (1 - 5000)
o/p vector of size 32
#params = 5K×32

Generative : step 1 : generate samples
adversarial : step 3 : (enemy) chuk dissimilarity

core idea : generate dataset similar to D but not same/copy

Generative Adversarial Networks

simpler task : 1D scalar → step 3 check how similar D and D'

D = {set of heights}

Generate D' similar to D

s1 generate PDF of D, P(H) ← prob model

* find which distr it is

Ex:- it is normal distr

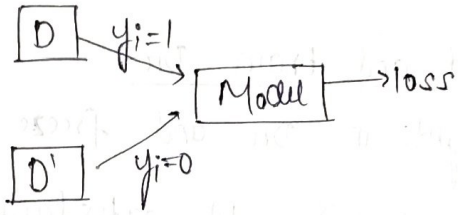
$P_\theta(H)$ → vector which is distr params μ, σ

$P_\theta(H) = N(\theta = [\mu, \sigma])$

s2 generate samples randomly from $P_\theta(H) = D'$

① use statistical tests → problem : it works only with small dimension
JS-dist
KL-div

② model approach

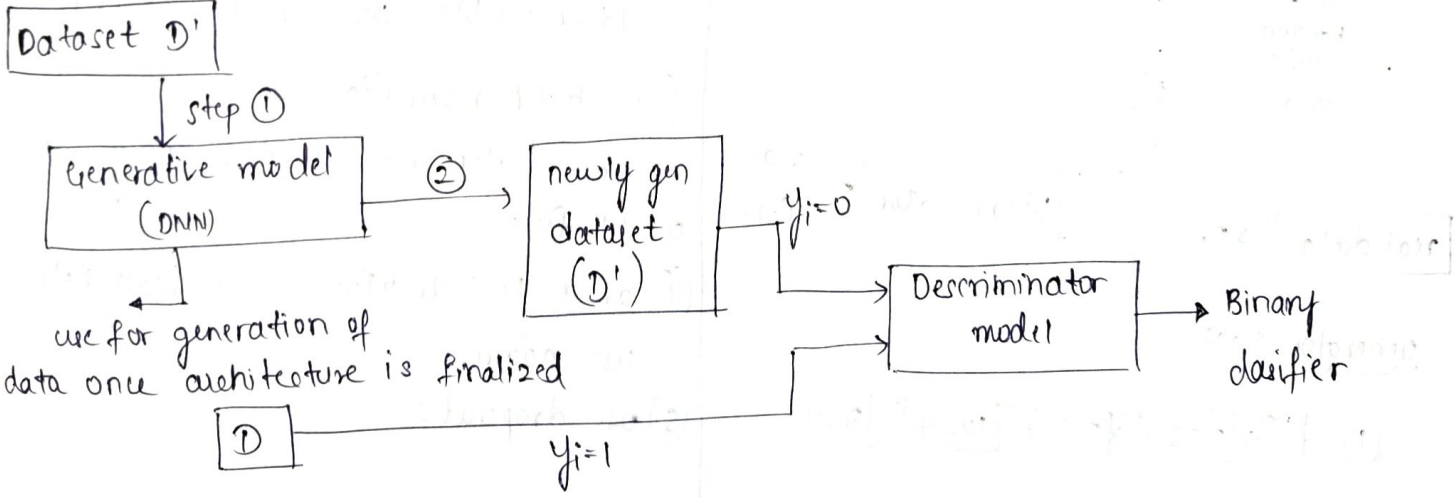


as loss ↑ → similarity ↑

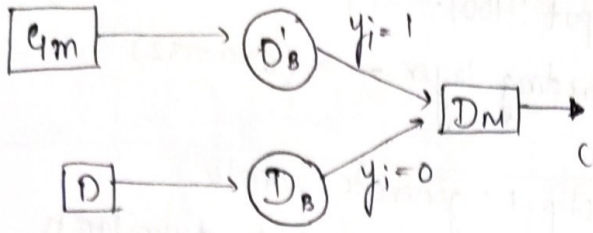
→ if loss is less, then D and D' are seperable
∵ D and D' are not similar

→ if loss is large ie bad performing model
→ D and D' are similar

Note :- probabilistic models dont work for multivariate analysis. so use ML/DNN



(30) **Train GAN**



D'_B = batch of dataset say of 100 datapoints

↳ use **gradient ascend** to maximize loss

working : **Alternate train opt**

step 1 Keep Dm fixed and train Gm

→ for Dm initialize we a random sample as train data Ex:- normal dist vecto

~~Ex:~~ train the model for it

* Here Gm does up-sampling ; i/p (100 d → op 784 d) MNSIT

(*) Initially D'_B generated would be very-very random and thus different from D. So loss will be very less which is bad for us

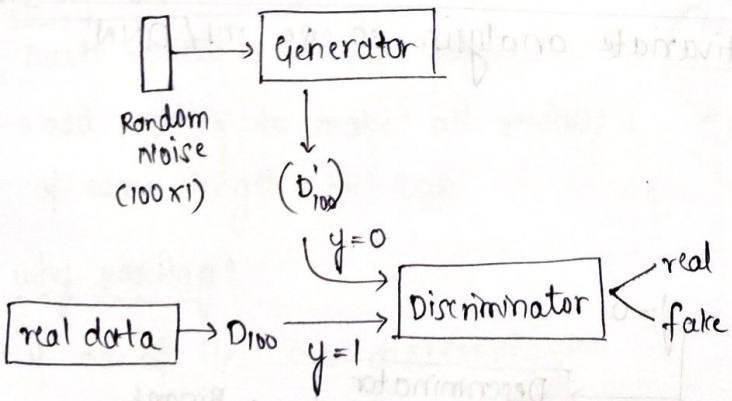
* we need to find and maximize loss using gradient ascend

step 2 Keep Gm fixed and train Dm

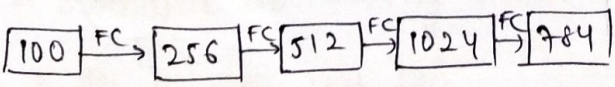
* only update weights in Dm and freeze weights in Gm

→ Alternatively do (1) and (2) until satisfactory loss is obtained

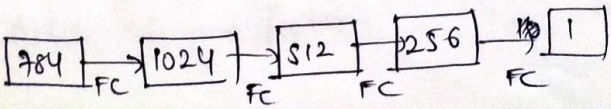
GAN for MNSIT



Generator DNN



discriminator DNN



Tips

(1) use normal distribution for sampling

(2) Normalize images b/w 1 and -1

→ use tanh as last layer of generator

[tanh - (-1,1) , ~~sig~~ sigmoid - (0,1)]

(3) use batch normalization

Each batch must contain only all real or all fake

(4) avoid ReLU, MaxPool ; use LeakyReLU

use adam

(5) use dropouts

Encoder Decoder

$\text{argmax } P(y_1, y_2 \cdots y_n \mid \langle x \rangle)$

Probabilistic repr of model

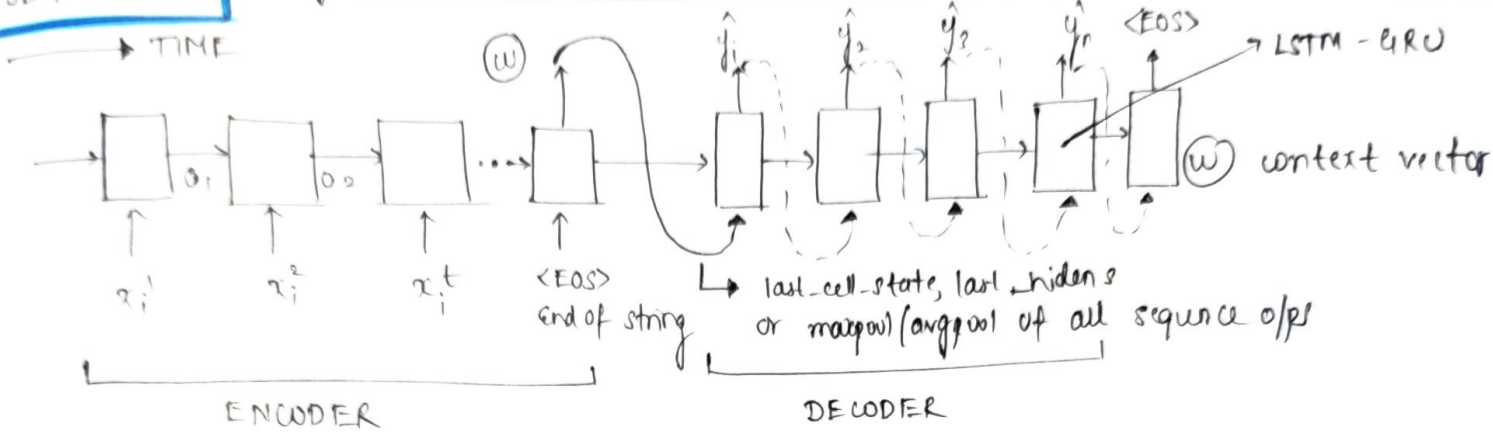
lots of data → train → model → $\langle y_1, y_2 \cdots \rangle$

test: $\langle x_1, x_2 \cdots x_n \rangle$

$\hat{y} = \text{argmax } P(y \mid x)$

find y which has max probability wrt data pt $\langle x \rangle$

SEQ-2-SEQ — Ilya sutskever et al



what is output sentence is long? [cho et al]

→ context vector is fed to all decoder cells, rather than only first decoder cell

→ but LSTM can't take 2 i/p at time stamp

↓

used modified LSTM

Image-2-caption — Karpathy et al

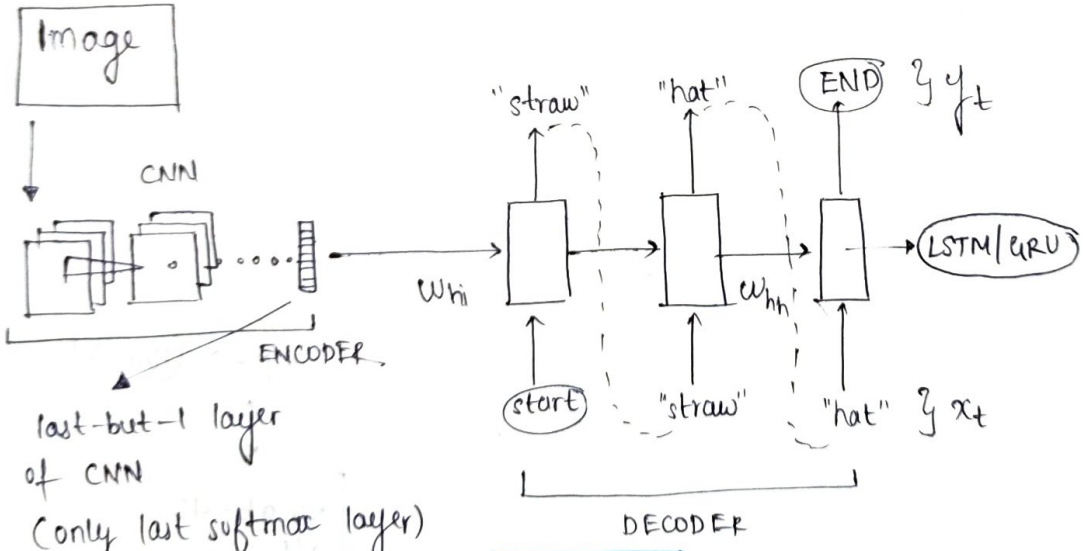


Image context vector

Applications

1. translation of text
2. Email auto complete and smart replies
3. Image captions and descriptions
4. statically finding code errors

Char RNN

* many-to-many RNN → given large corpus of sequence of data predict which char follows given char most probably
* one o/p corresponding to each i/p

(train i/p) let sequence be $c_1, c_2, c_3 .. c_n$

if i/p is c_2 } learn this
o/p should be c_3 }

Word Embedding Layer (DL+NLP)

word embedding ?

→ represent word with vector → TF-IDF, BoW, Word2Vec et

word_embedding_layer → let nlw learn embedings by itself

Process

sentence (seq of words) → Integer representation → Embeding layer (learn) → word embedings

Glove + Embedding matrix (pre-trained)

Ex:-

docs = ["sentence is one . . . ",
"sentence - 2 . . . "..]

① Tokenizer (word → int)

→ t = Tokenizer().fit_on_texts(docs)

~~X_tr = t.fit~~

→ X_tr = t.texts_to_sequences(docs)

→ X_tr is vector of integers.
Each word is represented by value indicating its index in dictionary of vocabulary.

t.word_index

{"word": index}

let len(t.word_index) = 300 (300 words)

padding make all sentences have same len

→ maxLen = for $\vec{v}$ in x_tr find largest $\vec{v}$

→ X_tr = pad_sequence(x_tr, padding='post', dtype='int' (~~float~~), maxlen=maxLen)

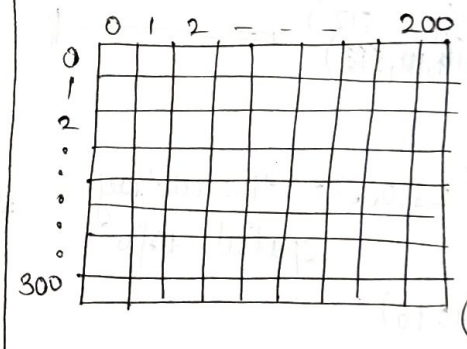
∴ we have each sentence/data point with len = maxLen

Ex → [1, 0, 14, 0, 0, 0, 0 . . .]

use glove_vector / pretrained model

→ asume ur glove vector has 200 dim-vector for each weight.

Create Embeding matrix



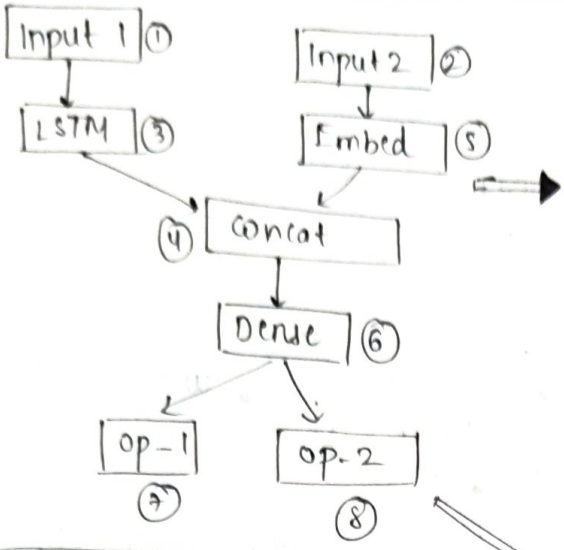
∀ each word in vocab, have a v-dim vector

```
for word,idx in t.word_index:
    embed[idx] = glove.get(word)
```

Embedding Layer

```
e = Embedding(input_dim = len(vocab),      → total no of words in vocab/len(cm)
              output_dim = 200,            → output dimension of each word (200 in our case)
              input_length = maxLen,       → length of each training data.
              trainable = false,
              weights = [Embed_matrix] )   } for pretrained glove vector
```

Functional API Keras



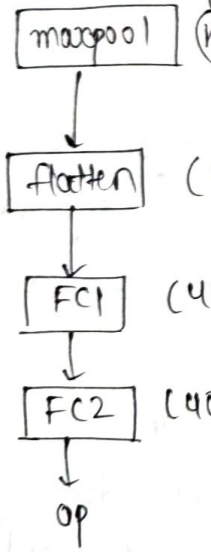
① Inp-1 = Input ((128,))
③ LSTM = LSTM( ) (Inp-1)
② Inp2 = Input ((10,))
⑤ Embed-1 = Embedding( ) (Inp2)
④ concat = concatenate( )([LSTM, Embed-1])
⑥ Dense = Dense( ) (concat)
⑦ op-1 = Dense (10, softmax) (Dense)
⑧ op-2 = Dense (2, softmax) (Dense)

Train

model.fit (~~epochs=10~~,
train ⟶ [xtrain-1, xtrain-2],
ytrain ⟶ [ytrain-1, ytrain-2],
epochs=10);

model = Keras.Model(
ip ⟶ [inp-1, inp2],
op ⟶ [op-1, op-2]);

Replace FC layer by Conv 2D



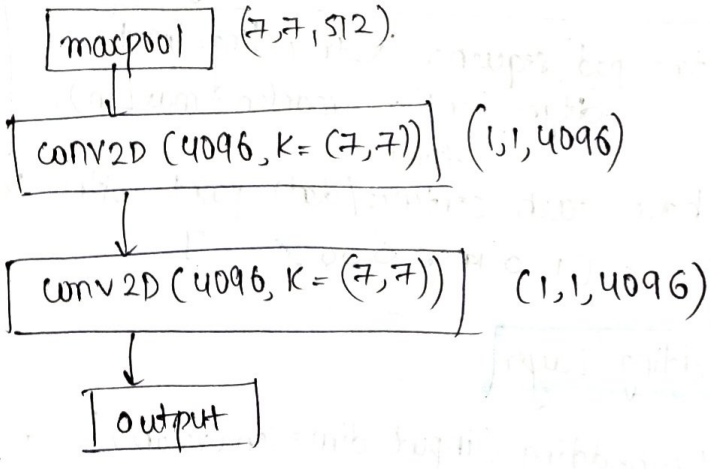
replace FC by CONV 2D

for FC1 we need 4096 = (1,1,4096)

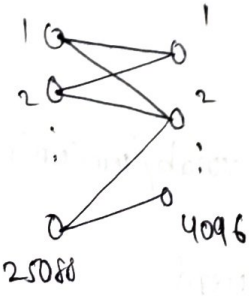
$*(7,7,512) \xrightarrow[s=1,\ fn=4096]{K=(7\times7)} (1,1,4096)$

$\left(\frac{(7-7+1)}{1}, \frac{(7-7+1)}{1}, 4096\right)$

$= 1, 1, 4096$

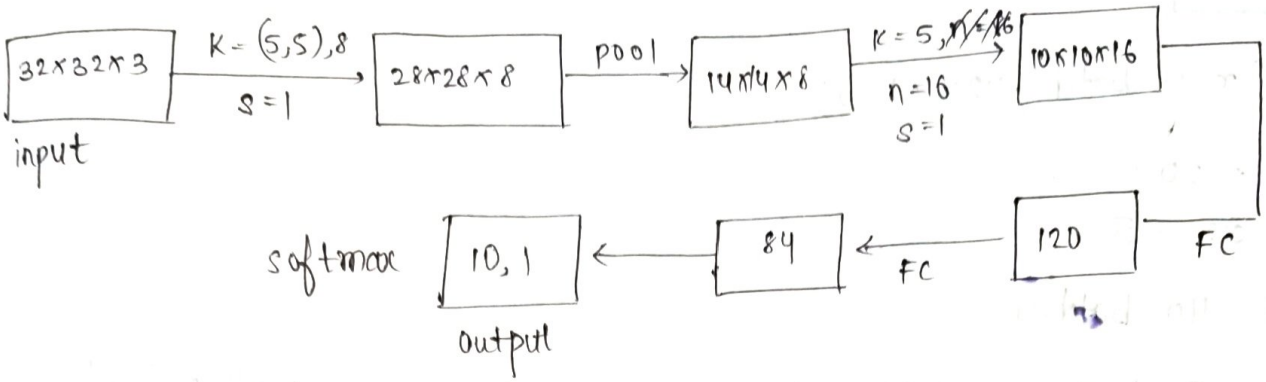


FC1



∴ 4096 * 25088 conn

Calculate CNN params



* conv layers

$(n,n) \longrightarrow \left(\frac{n-k+2p}{s}+1, \frac{n-k+2p}{s}+1, \text{no filters}\right)$

params in conv layer $= \left[\left(k \times k \times \text{no of filter in previous layer}\right) + 1\right] * \text{no of filters in current layer}$

(+1 → Bias term for each filter)

params in FC = FC(Fcn) = # neurons in last layer * # neurons in crent FC + Fcn

(Fcn → Bias term)

count

layer	shape	# params	activation size
Input	(32×32×3)	0	32×32×3 = 3072
CONV1 K=5, n=8	(28×28×8)	$((5\times5\times3)+1)\,8$ = 608	~~608~~ 6272
POOL1	(14×14×8)	0	1568
CONV2 K=5, n=16	(10×10×16)	$((5\times5\times8)+1)\,16$ $[(5\times5\times8)+1]*16$ = 3216	1600
FC1 (120)	(120,1)	(10×10×16 ×120) +120 = 192120	120

prev. kernel

$[(5\times5\times3)+1] * 8$

bias, cur kernel

* Embeding layer (vocab size, output-dim)

o/p → (None, i/p-size, output)

→ (vocab-size * output-dim)

36

Batch size

→ model.fit(x, y, batch_size = b);

let len(x) = 200

batch_size = 5

∴ we get 40 batches

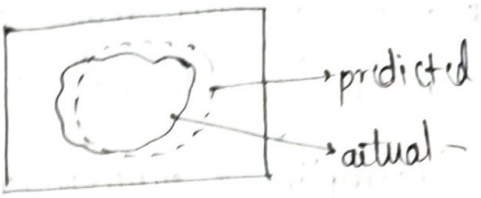
training ⟶ Epoch 1 ⟶ divide data into 40 batches ⟶ pass batch 1 & update weight
pass batch 2 & update weight.

∴ In one epoch, weights are update $\left(\frac{\text{size of dataset}}{\text{batch\_size}}\right)$ no of time ie 40 in our case

Image Segmentation

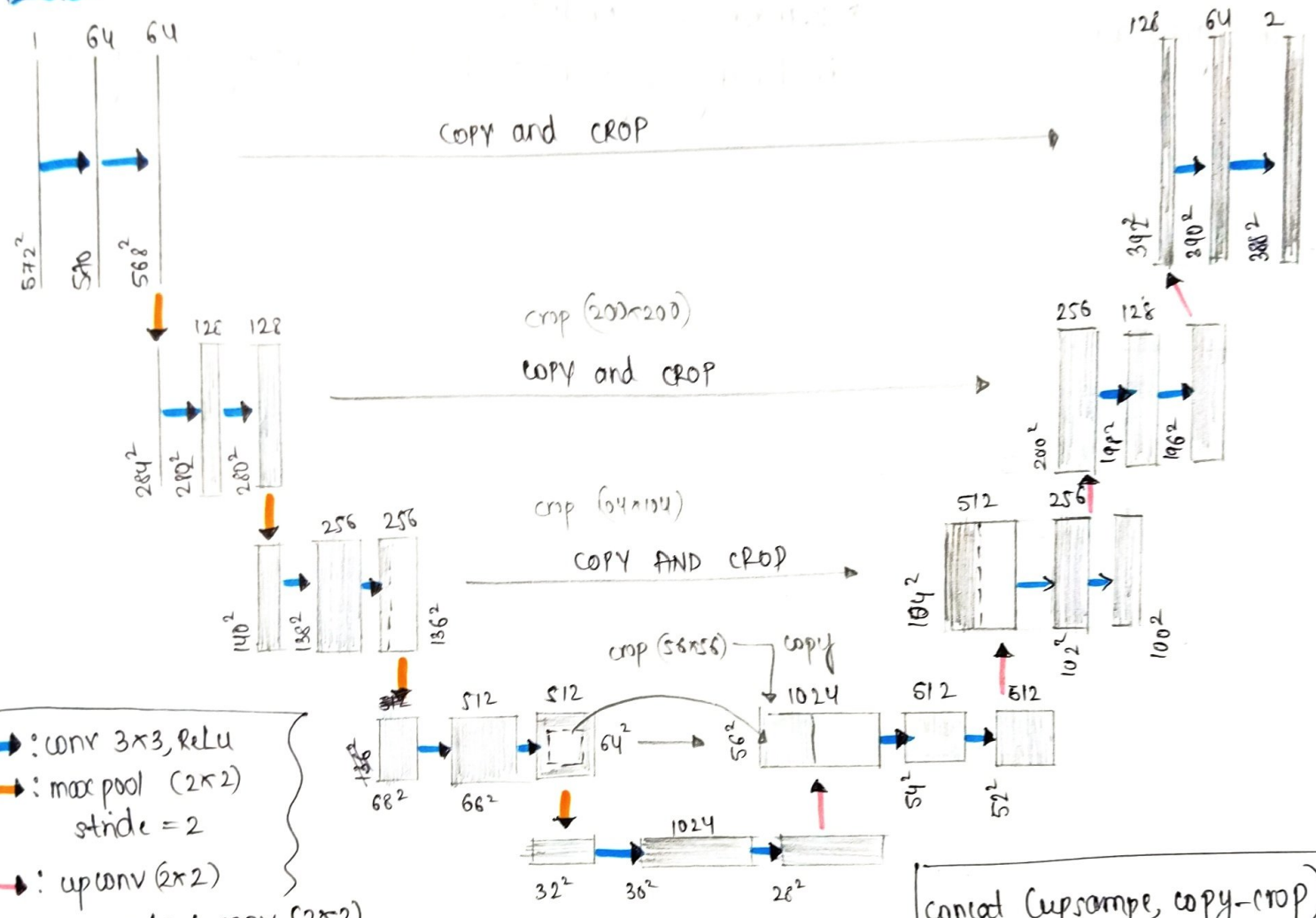
→ divide into areas / segments

→ measures :- ① jacards similarity = $\frac{R \cap R'}{R \cup R'}$ (IOU)

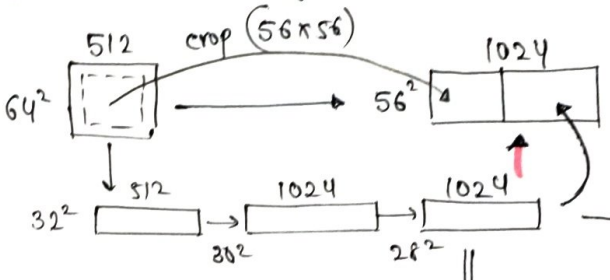


② cross entrophy = per pixel comparision

Unet



upsample base layer

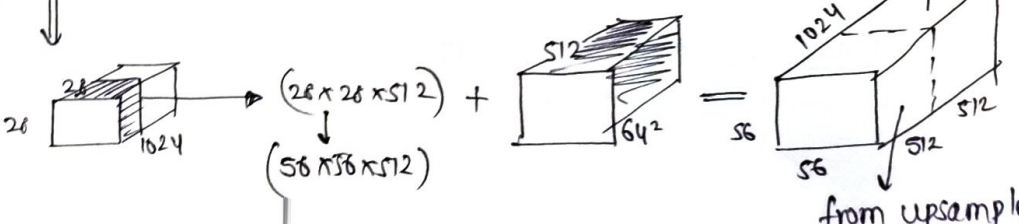


concat (upsample, copy-crop)

upsample

① upsample : duplicate rows and colums
$28 \times 28 \times 1024 \longrightarrow 56 \times 56 \times 1024$

② conv2D (f = 512, (2×2))
$(56 \times 56 \times 512)$



(38) General idea

* as we go down, image size halves and no. of filters doubles
  ie we go into pixel level as we go down
* (upsampling + copy-N-crop) acts like a residual unit ie get information from lower level as well as from copy-crop unit
* Here o/p = (388×388×2)
  - i/p - grayscale
  - 2 masks are predicted
    ie binary segmentation to identify 2 regions

moving window average

consider you have data for (t) days
future data (t+1) depends on average of previous (k) days

other ways
- mean avg
- median
- exponentially weighted

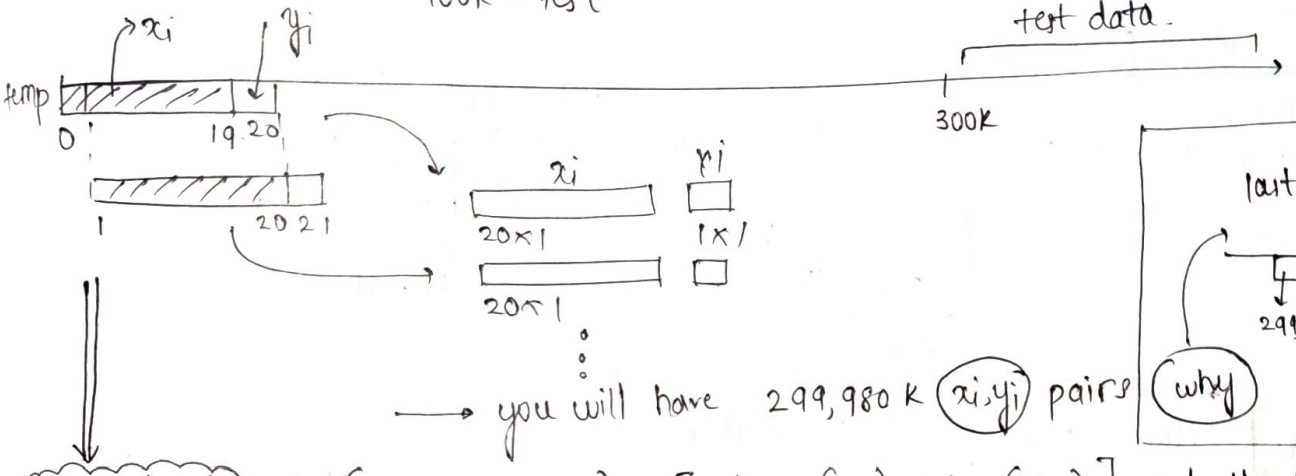
Dataset :- weather temp each 10 minutes given

Univariate prediction

asume (K=20) we take window size = 20

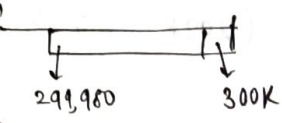
* Normalize all data
  → only train data

dataset creation :- 300K - train
100K - test



→ you will have 299,980 K (xi, yi) pairs

last sample is



why

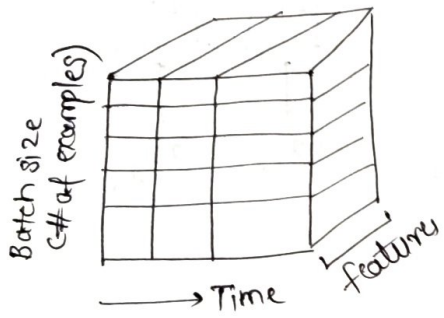
new dataset → (299,980, 20, 1) [reshape (20,) into (20,1)] → better for training model

we have data of 299,980K windows/time series

LSTM training diagram

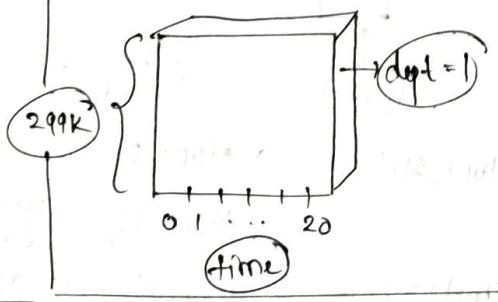
* LSTM expects data in this format → [batch, time, features]

height (y) - samples
x → time series
depth → features (univariate 1)



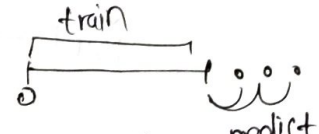
loss = mae
mean absolute error

Our data

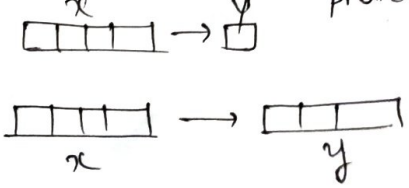


multivariate - single step

→ given a windows' predict only one data at $t_1 - t_n$



* single step → predict only one pnt at future
* multi step → predict over multiple points



Dataset generation hacks

① predict next hour help

train: x_0 1 ... 20 → x_i ; (21) → y_i

② predict next 12 hrs

0 1 2 ... 20 → x_i ; (32) → y_i

③ predict next 12 hours data using past 120 hours → multi step

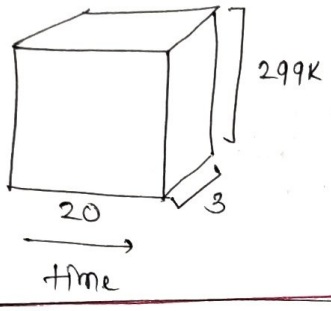
0 ... 120 → x_0 ; 121 ... 132 → y_0

$\therefore$ xtrain = (299K, 20, 1)
ytest = (299K, 12, 1) or (299K, 12)

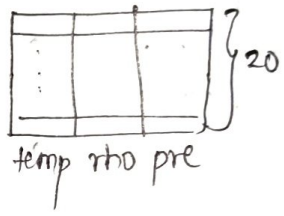
multivariate dataset

predict — temp
data — temp, rho, pre ⟹

(299K, 20, 3)



one xtrain (20×3) ⟹



Keras LSTM

ilp = (None, 4, 2) — 2 features; 4 time-steps

LSTM(units = 2)

ip → $(x_0\ x_1\ x_2)$

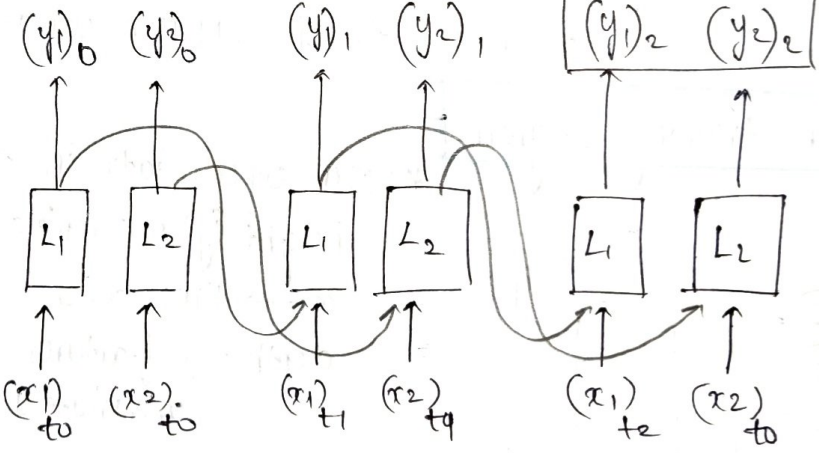
② → dim = no of units

$(H, 2)$ $(y_1)_2, (y_2)_2$

(None, 3, 2) → (None, 2)

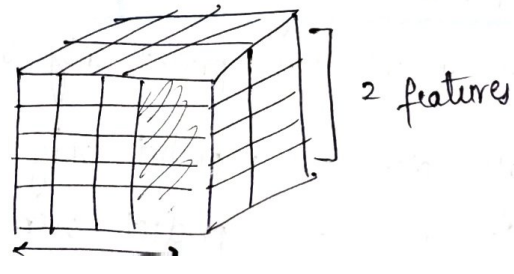
~~None $(y_1)_2, (y_2)_2$~~

output → ①



units = 2 → generate 2 data for each point → one o/p from last unit

(None, 3, 2)



return_sequences = true

* dont only return last_state o/p → $[[y_{10}, y_{20}], [y_{11}, y_{21}], [y_{12}, y_{22}]]$
but o/p for each state and time step

(None, 3, 2)

(None, 3, 2) → LSTM (units = 5, + ret_sequences) → (None, 5)

→ (None, 3, 5) — no of units

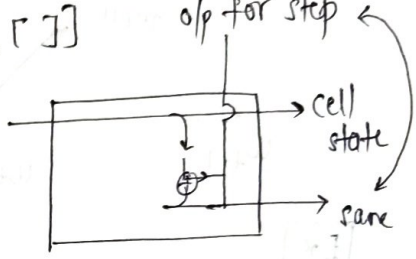
no of time steps

return_states = true

shape = (3,1)

LSTM (1, ret_states) ⟹ [lstm, stat_h, state_c] → [[ ], [ ], [ ]] (3×1)

- lstm: hidden state o/p for last time step
- stat_h: hidden state o/p for last time step
- state_c: cell state for last time step



both

* LSTM (1, ret_states = true, ret_seq = true) → [lstm, state_h, state_c]

i/p = (3,1)

→ [[o_1, o_2, o_3], [ ], [ ]]

- lstm: o/p of all layers/time steps
- state_h: o/p of last
- state_c: o/p cell state of last cell

Standard

i/p → (None, time_steps, feat)

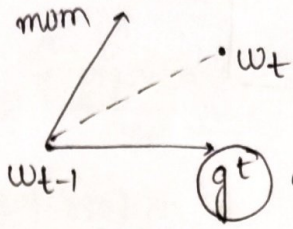
- → LSTM (units) → (None, units)
- → LSTM (units, ret_sequences) → (None, time_steps, units)
- → LSTM (units, ret_states) → ([None, 16], [None, 16], [None, 16]) — 16 → units.
- → LSTM (units + ret_states + ret_sequences) → ([None, time_steps, units], [None, units], [None, units])

return hidden state o/p for each time step
→ one timestep will have (units) outputs ∴ ts × units

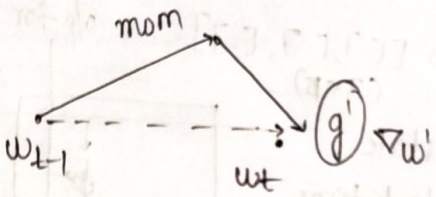
Nesterov Accelerated gradient

momentum = ewa of previous steps

→ sgd + momentum = $w_t = w_{t-1} - [\gamma v_{t-1} + \eta g_t]$ → gradient



NAG → it says ① calculate momentum
② calculate gradient at momentum pt $(w_{t-1} - \gamma v_{t-1})$
③ move in direction of grad

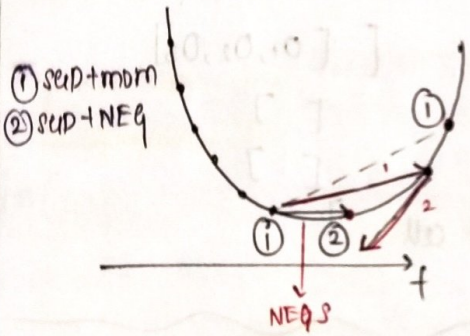


$(w_t)' = w_t - \gamma \cdot v_{t-1}$ → step1 : momentum

$v_t = \gamma v_{t-1} + \eta \cdot g'$

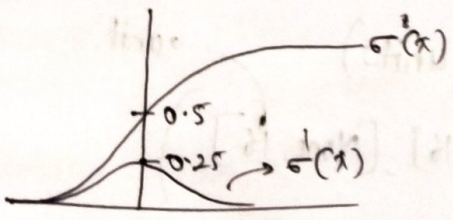
where, $g' = \nabla w_t'$ (grad at w_t - it moment)

Ex



* at point ①, we previously had many +ve update ∴ velocity / momentum will be high and it may overshoot minima
* NEG :- it move first to momentum point which is on other side, and then gradient is calculated which is -ve as slope is -ve ∴ it overshoots less than sgd + momentum

derivative of sigmoid



wkt $\sigma'(x) = \sigma(x)(1-\sigma(x))$

max = 0.25 at $x=0$

min = 0

differentiation rules

① Reciprocal rule

$\left[\frac{1}{g(x)}\right]'$ or $\left[g(x)^{-1}\right]' \xrightarrow{d/dx} \frac{-g'(x)}{g(x)^2}$

$\frac{d}{dx}\left(\frac{1}{g(x)}\right) = \frac{-\frac{d}{dx}(g(x))}{(g(x))^2}$

② Exponential rule

$\frac{d}{dx}\left(e^{f(x)}\right) = e^{f(x)} \cdot f'(x)$

$$\Rightarrow \sigma'(x) = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \boxed{\frac{-\frac{d}{dx}(1+e^{-x})}{(1+e^{-x})^2}} \rightarrow (1)$$ using reciprocal rule

→ solving Numerator of (1)

$$-\frac{d}{dx}(1+e^{-x}) = -\left[\frac{d}{dx}(1) + e^{-x}\cdot\frac{d}{dx}(-x)\right] = -\left[0 + (e^{-x})(-1)\right]$$

$$= e^{-x}$$

$\therefore$ (1) becomes $\Rightarrow \boxed{\frac{e^{-x}}{(1+e^{-x})^2}} \Rightarrow (2)$ $\therefore \sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$

Simplifying (2) further

$$\frac{e^{-x}\cdot 1}{(1+e^{-x})(1+e^{-x})} \rightarrow \frac{(e^{-x}+1-1)\cdot 1}{(1+e^{-x})(1+e^{-x})} \rightarrow \left[\frac{1}{(1+e^{-x})}\right]\left[\frac{e^{-x}+1-1}{1+e^{-x}}\right]$$

$\sigma(x)$

$$\rightarrow \sigma(x)\left[\frac{e^{-x}+1-1}{(1+e^{-x})}\right] \rightarrow \sigma\cdot\left[\frac{e^{-x}+1}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right]$$ {split into fractions}

$$= \sigma(x)\left[1-\sigma(x)\right]$$

$$\boxed{\therefore \sigma'(x) = \sigma(x)\cdot(1-\sigma(x))}$$

YOLO Algorithm

object detection

Object Detection

* Focus on bounding boxes → <x, y, w, h>

(x, y) — center
(w, h) — box width, height

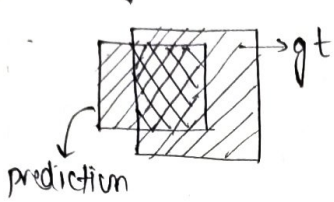
and not [on pixels] → use image segmentation (slower and not suitable for real-time apps)

* Ip → algo → o/p image with bounding boxes with confidence level (0-1)

Performance Metrics

IoU = overlap area / union area

GT (ground truth) — human labelled data



* prediction is correct if IoU ≥ 0.5

Steps

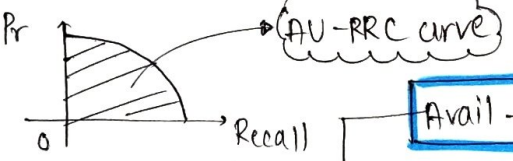
① convert o/p to binary ie True or false using IoU. ∀ bounding boxes

② Image may have multiple bounding boxes,

So,

* for each class (chairs, person ...)

∀ individual classes →
- Calculate avg-precision ≡ area under pre-recall curve
- plot Pr-recall curve



* mAP (mean average precision)

mAP = mean avg precision across all classes

Trade off

→ speed vs mAP

- speed → video based (self-dr cars, real time face)
- mAP → medical diagnosis or OCR (precision is imp)

Avail-options

High mAP, low speed

* R-CNN (region-CNN) * FPN-FRCN

* SSD (single shot detector)
* RetinaNet

YOLO → faster and good mAP

- YOLOV3-320 → input image size
- YOLOV3-416
- YOLOV3-608

Note: as image size ↑ time ↑

YOLO V3

Architecture → only used fr initialize

YOLOV2 → DarkNet-19
V3 → DarkNet-53

1) Feature Extractor / Backbone (Pre-Training)

DarkNet-53 — 53 conv-layer (fully convolution n/w)

padding = same

	Layer	Filters	Size	o/p
	conv	32	(3×3)	256×256
	conv	64	(3×3) stride=2	128×128
1×	conv	32	(1×1)	
	conv	64	(3×3)	
	residual			128×128
	conv	128	3×3/2	64×64
2×	conv	64	(1×1)	
	conv	128	(3×3)	
	residual			64×64
	conv	256	3×3/2	32×32
8×	conv	128	1×1	
	conv	256	3×3	
	residual			32×32
	conv	512	3×3/2	16×16
8×	conv	256	1×1	
	conv	512	3×3	
	residual			16×16
	conv	1024	3×3/2	8×8
4×	conv	512	1×1	
	conv	1024	3×3	
	residual			8×8
	Avgpool	global		
	connected	1000		
	Softmax			

416×416×3 → 13×13×1024
(416/32) — last filter size

Characteristics
① 53 conv layers
② fully conv neural n/w
③ Each conv
≡ CONV → Batch Norm → Leaky ReLu
④ conv with stride-2 → acts as max-pool
⑤ no maxpooling

* whole model is trained on ImageNet and save weights
→ this only detects image objects but not bounding boxes

stride=2

why we DarkNet when we had Resnets?

Backbone	Top-5	Bops	BFlops/s	FPS
Res-152	77	2930	1090	37
Dark Net	77.2	18.7	1457	78

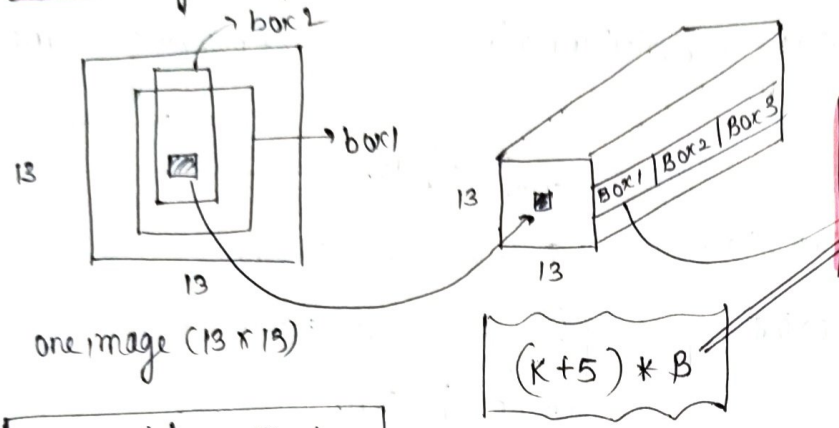
→ darknet almost gives same or little less accuracy comp to resNet
* BNops is less (Billions of ops needed)
* BFlops/s is high (Bil of float op/sec).
* and FPS is high

(46) (2) Bounding boxes and output

(ip) $416 \times 416 \times 3 \xrightarrow{DNet} 13 \times 13 \times 1024 \xrightarrow{Box} 13 \times 13 \times 425$ (o/p)

→ assume K = 80 and B = 5

How do you represent boxes?



one image (13 × 13)

(K+5) * B

(Each cell may belong to many boxes) → # no of boxes

Each bounding box

t_x	t_y	t_w	t_h	P_o	C_o	...	C_K	* B

(x, y) → centre of box with dimensions w × h

P_o → objective score — (0/1) whether it contains an obj / not

$C_o \rightarrow C_K$: class scores for K no of

Ex: consider 80 classes

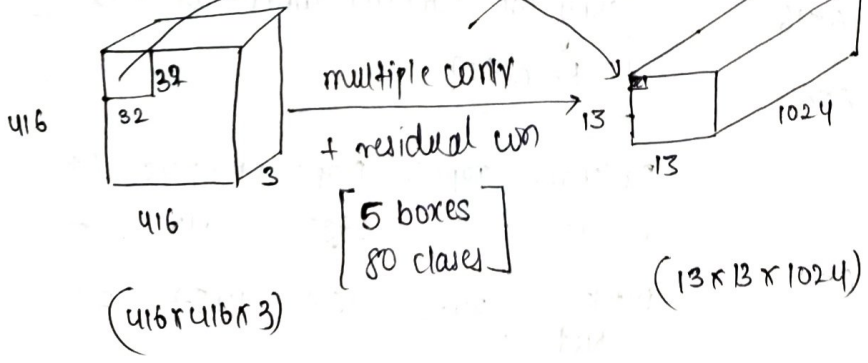
* box1 = $(P_o, x, y, w, h, C_1, \ldots C_{80})$

* score = $P_o * C = \begin{bmatrix} P_o C_1 \\ P_o C_2 \\ P_o C_3 \\ \vdots \\ P_o C_{80} \end{bmatrix} = \begin{bmatrix} 0.12 \\ 0.21 \\ 0.80 \\ \vdots \\ 0.10 \end{bmatrix}$ → probability that object bounded ∈ class 2

$P(C=K) = P_o * P_K$ or $P_o * C_K$

* assume we get $P_o C_3 = 0.80$ as max

∴ the box bounds object belonging to class = 3



(416 × 416 × 3)

(13 × 13 × 1024)

$\frac{416}{13} = 32$

∴ Each pixel in o/p is of size (1 × 1 × 1024) is captured from (32 × 32 × 3) grid input.

Effective Receptive Field

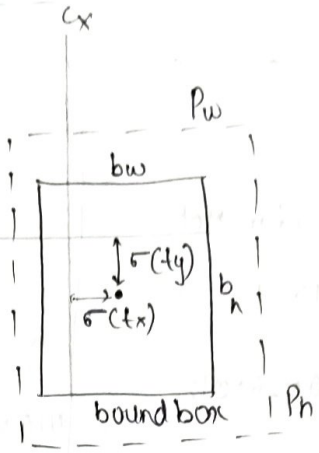
Anchor Boxes → prior boxes where you highly probable to find

* Tried to predict (t_x, t_y, t_w, t_h) directly but couldn't get good results

So,

① Fix some predefined boxes called anchor-boxes : $[C_x, C_y, P_w, P_h]$

② define bounding boxes in terms of anchor boxes →



$C_x, C_y \equiv$ priors

$$b_x = \sigma(t_x) + C_x$$
$$b_y = \sigma(t_y) + C_y$$
$$b_w = P_w \cdot e^{t_w}$$
$$b_h = P_h \cdot e^{t_h}$$

$\sigma = 0$-1

(b_x, b_y) should be atmax 1 pixel away from (C_x, C_y)

∴ given box cordinates and anchor boxes you can find $[b_x, b_y, b_w, b_h]$

learned box + anchor box → actual bounding box

How to find how many anchor boxes and location?

→ apply kmeans clustering on COCO dataset which is set of all bounding boxes

→ $(K=5)$ was found to be best which indicates that these 5 anchor boxes have high probability region

Objectness score — (P_o)

Is there any object or not? 0/1

↳ have a log-regresion for each box

* $[C_0 - - C_K]$

↳ tried softmax — didnt work Because of heirarchy presence

⇒ Soln

Build one logistic regresion for each class ie u need $P_o + (C_0 \cdots C_K)$ log-regresions

if for any Bbox, if P_o is v.v.small

→ it has very less chance of enclosing any object

∴ can be ignored

$\lambda_{co\text{-}ord} = 5$ ← $t_x, t_y \to 0 - \infty$

$\lambda_{noobj} = 0.5$ → $P_o = (0 \text{ to } 1)$

Loss

$$\sum_{\text{all bounding boxes containing object in train data } Y_{train}} \left[\text{log-loss for } P_o + \text{log-loss for } (C_0 \cdots C_K) + \lambda_{co\text{-}ord} * \text{sq.loss for } t_x, t_y, \sqrt{t_w}, \sqrt{t_h} \right]$$

$$+ \sum_{\text{all boxes in predicted which dont have obj (predicted boxes not present in } Y_{train})} \lambda_{no\text{-}obj} * \text{log-loss for } (P_o)$$

42 (4) multiscale prediction

* 13×13 image feature can only detect large objects

Idea

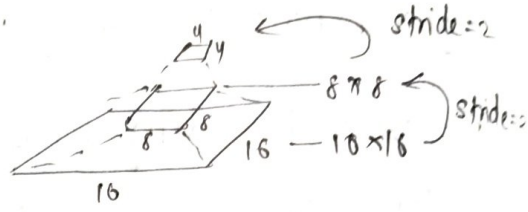
* predict with different grid sizes
  ↳ How to get?
  Extract from layers in darkNet

① last layer → 13×13×1024

② last but-1 → 26×26×512
conv layer with s=2

③ 2nd last → 52×52×256
conv layer
s=2

Image pyramid



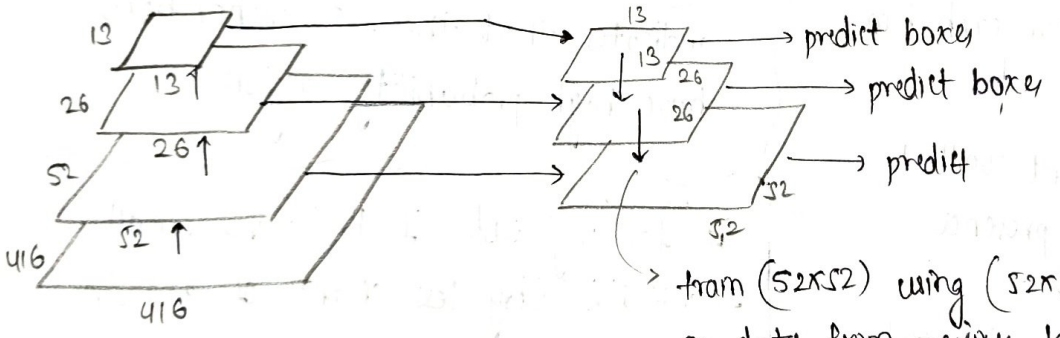
416×416×3 →
52×52×256 — $(52^2 \times 3)$
26×26×512 — $(26^2 \times 3)$
13×13×1024 — $(13^2 \times 3)$
Bb=3

↓
10,647 Bboxes
of different scales

Feature pyramid networks

FPN

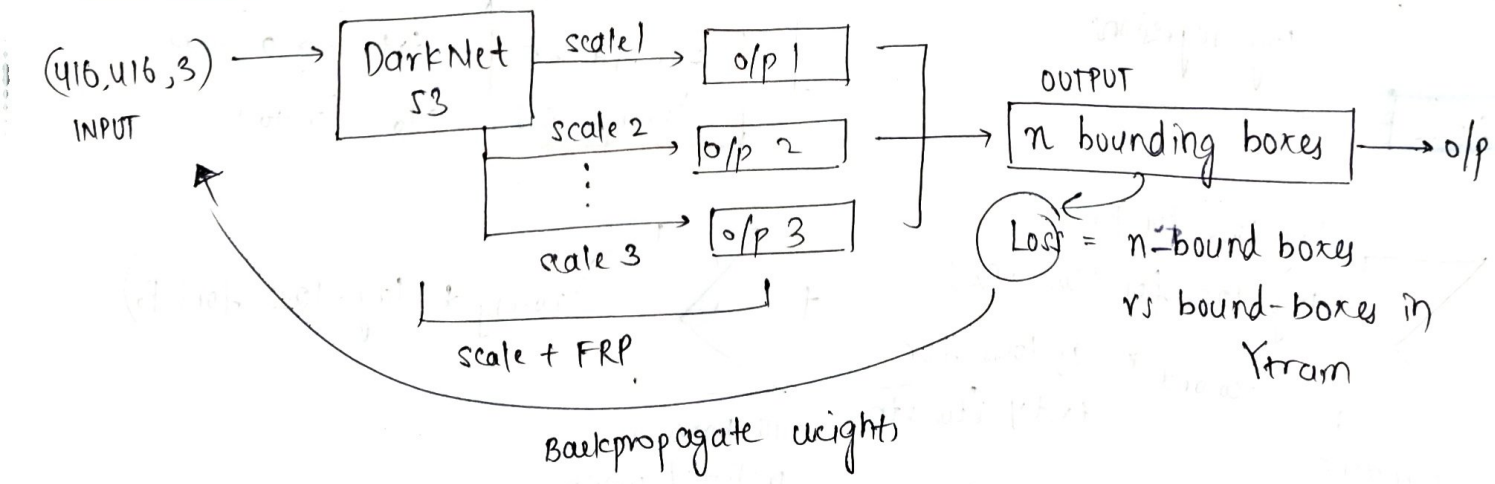
→ not only use different grid sizes to predict but also cross-grid information to use

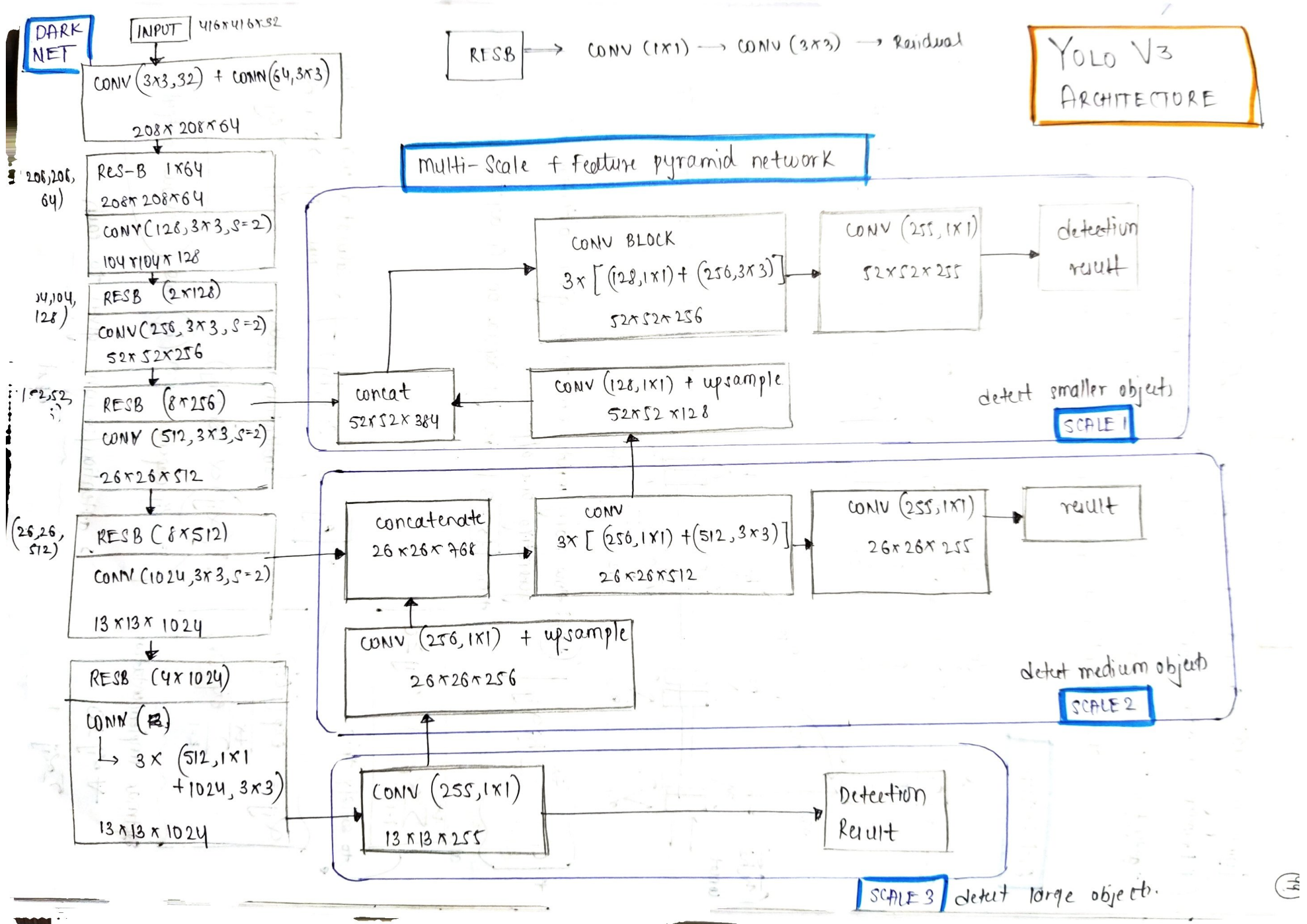


FPN + lateral connections

→ from (52×52) using (52×52×256) layer as well as data from previous layers.

Workflow




DARK NET
INPUT
416x416x32
CONV (3x3, 32) + CONV (64, 3x3)
208x208x64
RES-B 1x64
208x208x64
CONV (128, 3x3, S=2)
104x104x128
RESB (2x128)
CONV (256, 3x3, S=2)
52x52x256
RESB (8x256)
CONV (512, 3x3, S=2)
26x26x512
RESB (8x512)
CONV (1024, 3x3, S=2)
13x13x1024
RESB (4x1024)
CONV (
3x (512, 1x1
+1024, 3x3)
13x13x1024
RESB → CONV (1x1) → CONV (3x3) → Residual
YOLO V3 ARCHITECTURE
Multi-Scale + Feature pyramid network
CONV BLOCK
3x [(128, 1x1) + (256, 3x3)]
52x52x256
CONV (255, 1x1)
52x52x255
detection result
concat
52x52x384
CONV (128, 1x1) + upsample
52x52x128
detect smaller objects
SCALE 1
concatenate
26x26x768
CONV
3x [(256, 1x1) + (512, 3x3)]
26x26x512
CONV (255, 1x1)
26x26x255
result
CONV (256, 1x1) + upsample
26x26x256
detect medium objects
SCALE 2
CONV (255, 1x1)
13x13x255
Detection Result
SCALE 3
detect large objects.

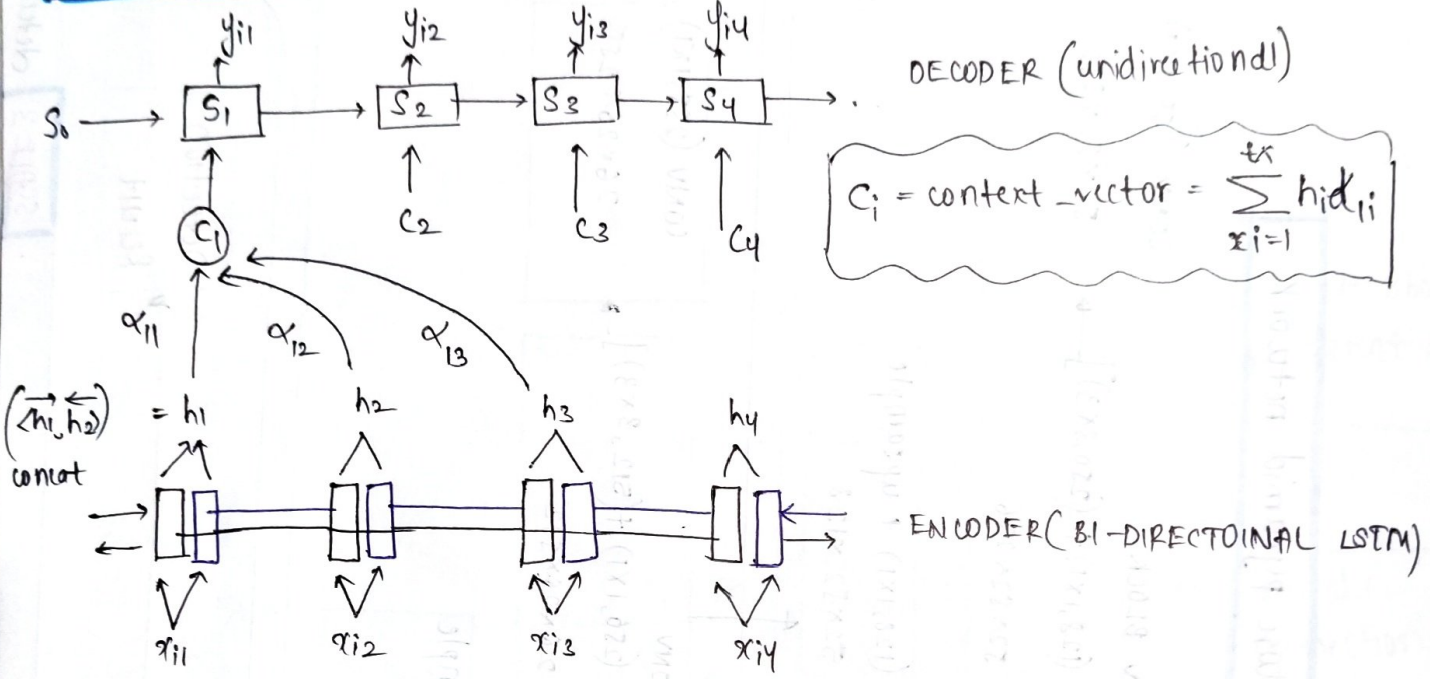
Attention Models

why you need? In en-decoders we pass only one o/p (w) from encoder → this may not capture the essence of whole sentence

* Encoders decoders dont work well for long sentences

attention? We give attention to a few words (tx) translate them and shift attention to next set of words (Human characteristic)

Architecture



* $C_i = \sum_{j=1}^{tx} h_j \alpha_{ij}$

(tx): How many inputs need to be given attention for generating one o/p.

Here $tx = 3$ ie y_{i1} depends on C_1 depends on x_{i1}, x_{i2}, x_{i3}

also, $\sum_{i=1}^{tx} \alpha_{1i} = 1$, $\alpha_{ij} \geq 0$

to satisfy this

$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{tx} \exp(e_{ik})}$

softmax implementation

$\forall \alpha_{ij} \geq 0$

$\Sigma \alpha_{ij} = 1$

$e_{ij} = a(S_{i-1}, h_j)$ → calculated using 2-layer NN i/p → S_{i-1}, h_j o/p → e_{ij}

attention-model

$\therefore \alpha_{ij} \propto (S_{i-1}, h_j)$

= α_{ij} depends upon h_j and previous output

→ attention/weight given to each input

Time complexity

K_1 = len (input)

K_2 = len (output)

$TC = O(K_1 \cdot K_2)$

if $tx = K_2$,

then $TC = O(K_1, K_2)$

ie consider whole sentence as attention to generate o/p.

α_{ij} importance

Ex: for calculating y_{i2} you use $\alpha_1, \alpha_2, \alpha_3$ in this case if α_i is large than that particular word had more impact on prediction

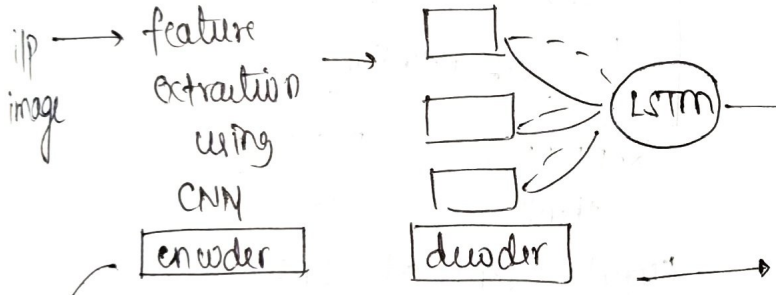
Ex: french → english

	you	and	call	→ o/p
ti	0.9	0	1	
ame	0.1	2	0	
mè	0	2	10	

i/p

to predict w_3 = call mè was most useful.

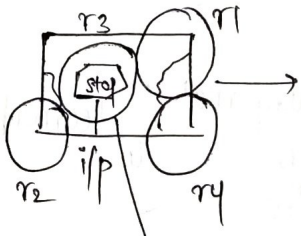
Image Captioning



generate K d-dim vectors

each vector representing one region of image

Visualize α_{ij}



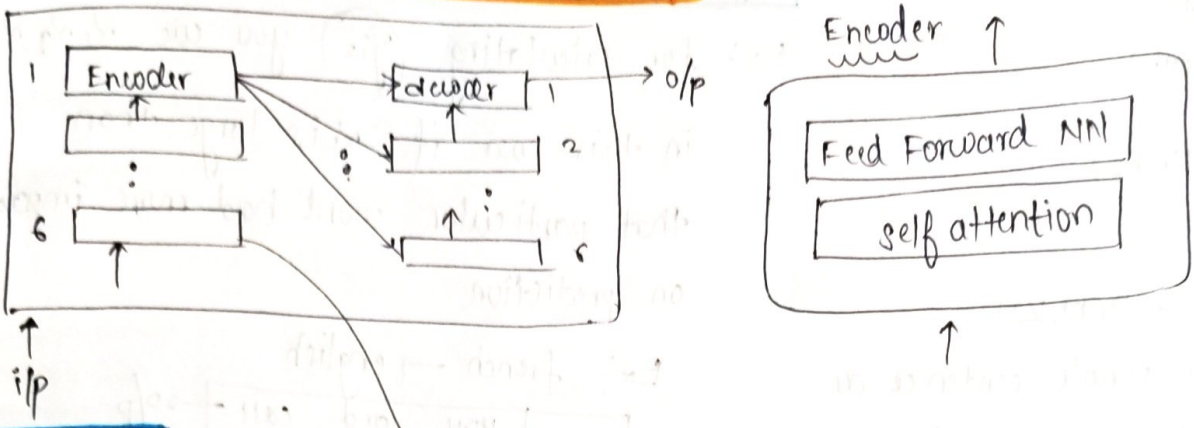
→ A mountain road with stop sign

see highest α_{ij}

say highest is α_{23}

region (3) contributed most

Transformers

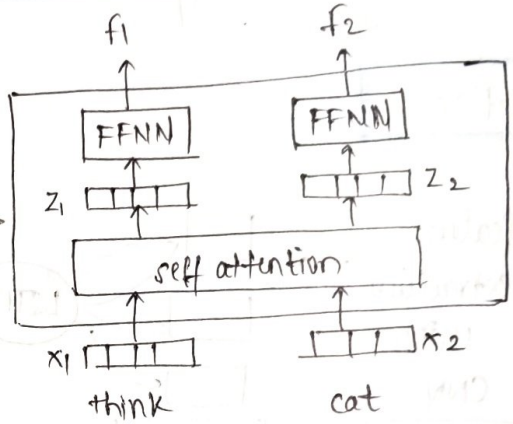


ENCODER

Self attention →

the boy crossed the road as it was empty

what does it refer/connect to



*inputs sent at once
No time-based sending

steps

hyper-params :

$d_k = 64$, W^Q – Query mat, W^K – keys, W^V – value } learnable

Input Embedded	x1 (512) think	x2 (512) cat
Queries	q_1 (64)	q_2
key	K_1	K_2
value	V_1	V_2
score	$q_1 \cdot K_1 = 112$	$q_1 \cdot K_2 = 96$ .. $q_1 \cdot K_n = ?$
$\left(\frac{score}{\sqrt{d_K}}\right)$ stable Δgrad	14	12
softmax	0.01	0.12
soft x val sum	V_1 + V_2 + ..	→ z_1

$q_i = x_i \cdot W^Q$
$K_i = x_i W^K$
$V_i = x_i W^V$

(n – no of work) = $q_1 K_{1-n}$

→ Here $V_1 >> V_2$ as V_1 is of same word

matrix computation

$X \times W^Q = Q$

$X \times W^K = K$

$X \times W^V = V$

$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

Z = z_1, z_2, z_3

∴ self-attention focuses on some words which matter more in its neighborhood (ie same sentence)

*One-headed :- only 1 set of matri

multi-headed (8-pairs of W^Q, W^K, W^V)

same steps but with (W_0^Q, W_0^K, W_0^V) to $(W_7^Q, W_7^K, W_7^T) = 8$ (zis)

∴ for X we get $(Z_0, Z_1, Z_2 \cdots Z_7)$

concat $(Z_0, Z_1 \cdots Z_7) \times W_0 = Z$

positional Embedding

we pass all words at once, how to mantain time/position

input je seri would

x_1 x_2 x_3 + embedings

p_1 p_2 p_3

x_1' x_2' x_3'

positional encoding

indicates position as well as distance b/w words

* pos vectors are static

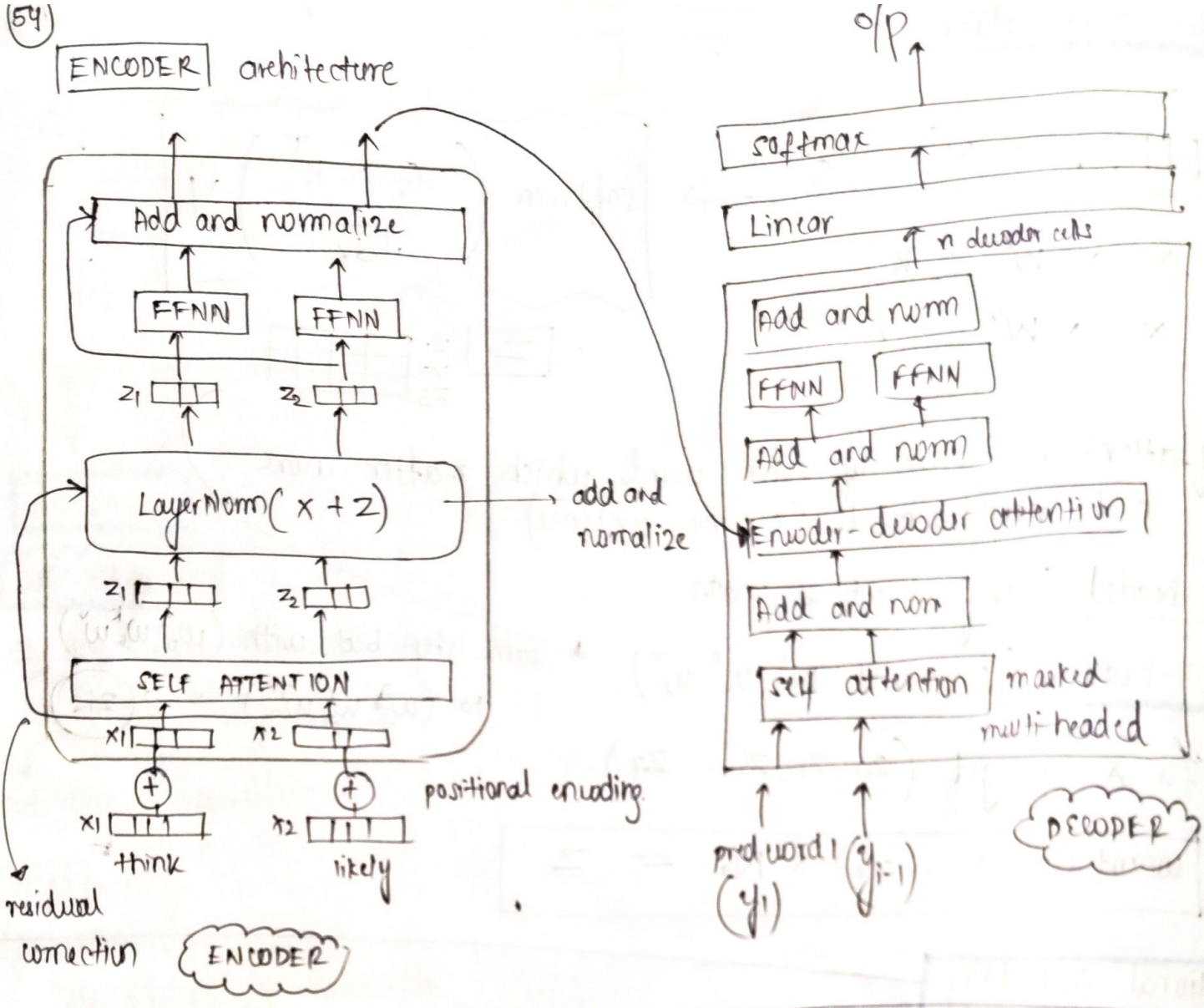
ie P_1, P_2, P_3 are precalculated

st $P_1 > P_2 > P_3$ position

$(P_1 - P_2) < (P_1 - P_3)$ distance

(54)

ENCODER architecture



Train phase

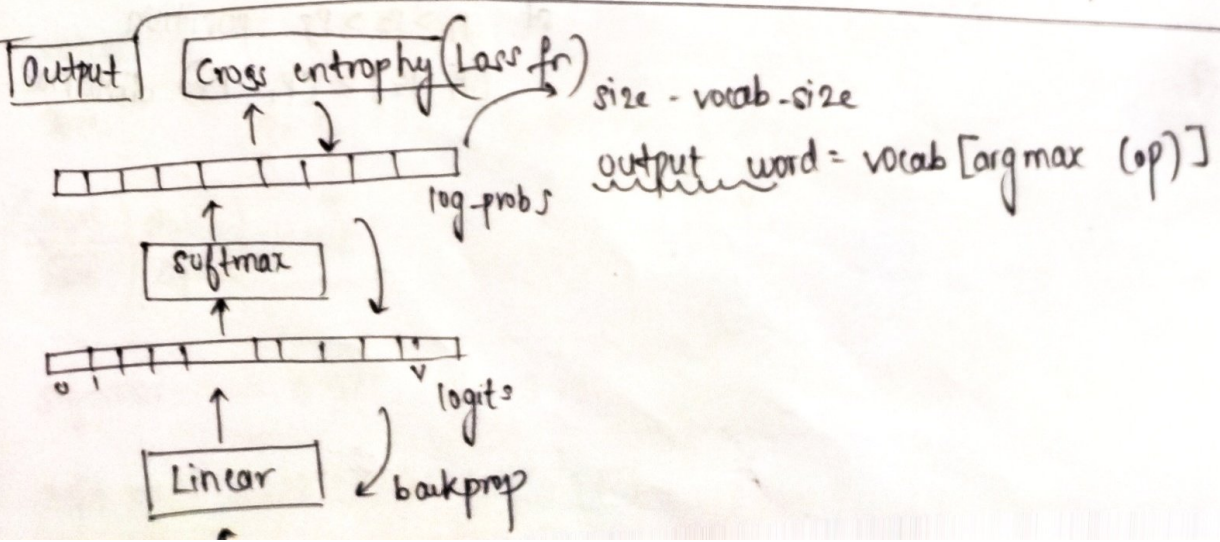
① pass whole sentence to encoders, you get a output

② pass it to decoder and predict first word

second word :- don't touch encoder

→ pass first word to encoder → o/p second word

third word :- pass 1st and 2nd o/p word → third o/p word



Advantages of Transformers

* Faster than RNN models as all data is ingested once
* state-of-the-art (2018)
* Inspired from CNN (self-attention / neighborhood ≈ convolutions in CNN)

BERT

sentence 2 vector

BERT — Bidirectional encoder representation from transformer

* BERT only has Encoder block called Transformer block

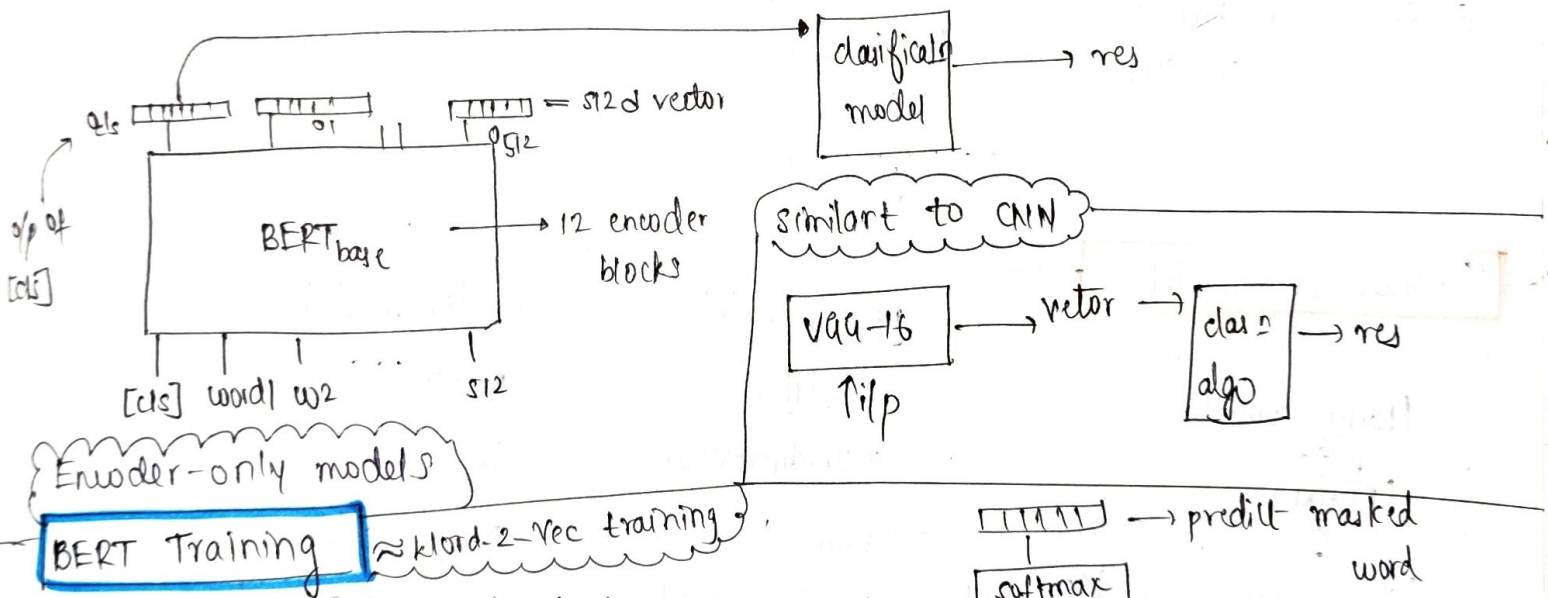
→ most useful as **Transfer Learning**

Base — FFN (768 units) + 12 attention heads

Large — FFN (1024 units) + 16 attention heads

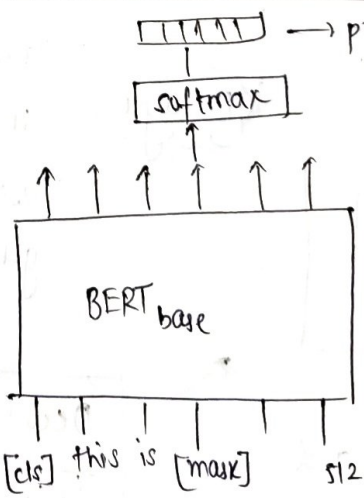
Transfer learning (spam or not)

Base: Base Bert :- can take upto 512 worded sentency and produces 512 o/ps each of dimension (768)



BERT Training ≈ klord-2-vec training

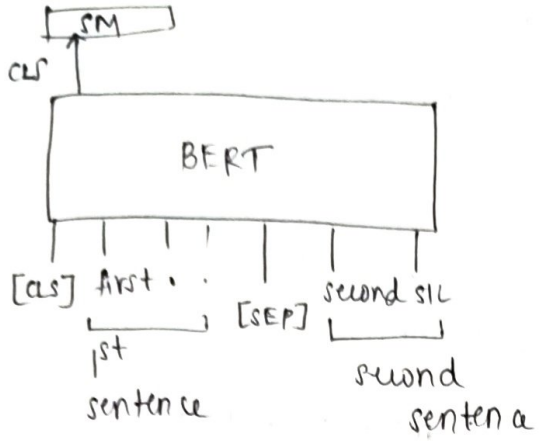
* Bidirectional models, while training look @ words before and after



steps

* provide any sentence as i/p and while doing so randomly mask 15% of words/token
* try to predict masked tokens given other words and optimize

BERT with 2 sentences

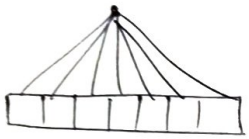


BERT training

NWP = next word prediction

Attention types

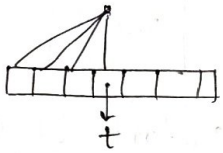
① Self attention



* depends on all words
& Encoder (no timestep concept)

② masked attention

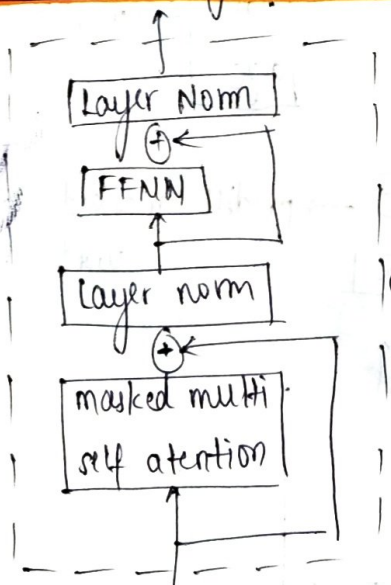
* at timestep (ti), self attention calculated on $t_0 \cdots t_{i-1}$
* used in decoders as time based



③ sparse self attention

* if len(sentence) = N
choose only $\sqrt{N}$ words for calculating self attention
* Reduces time complexity.

Decoder only models

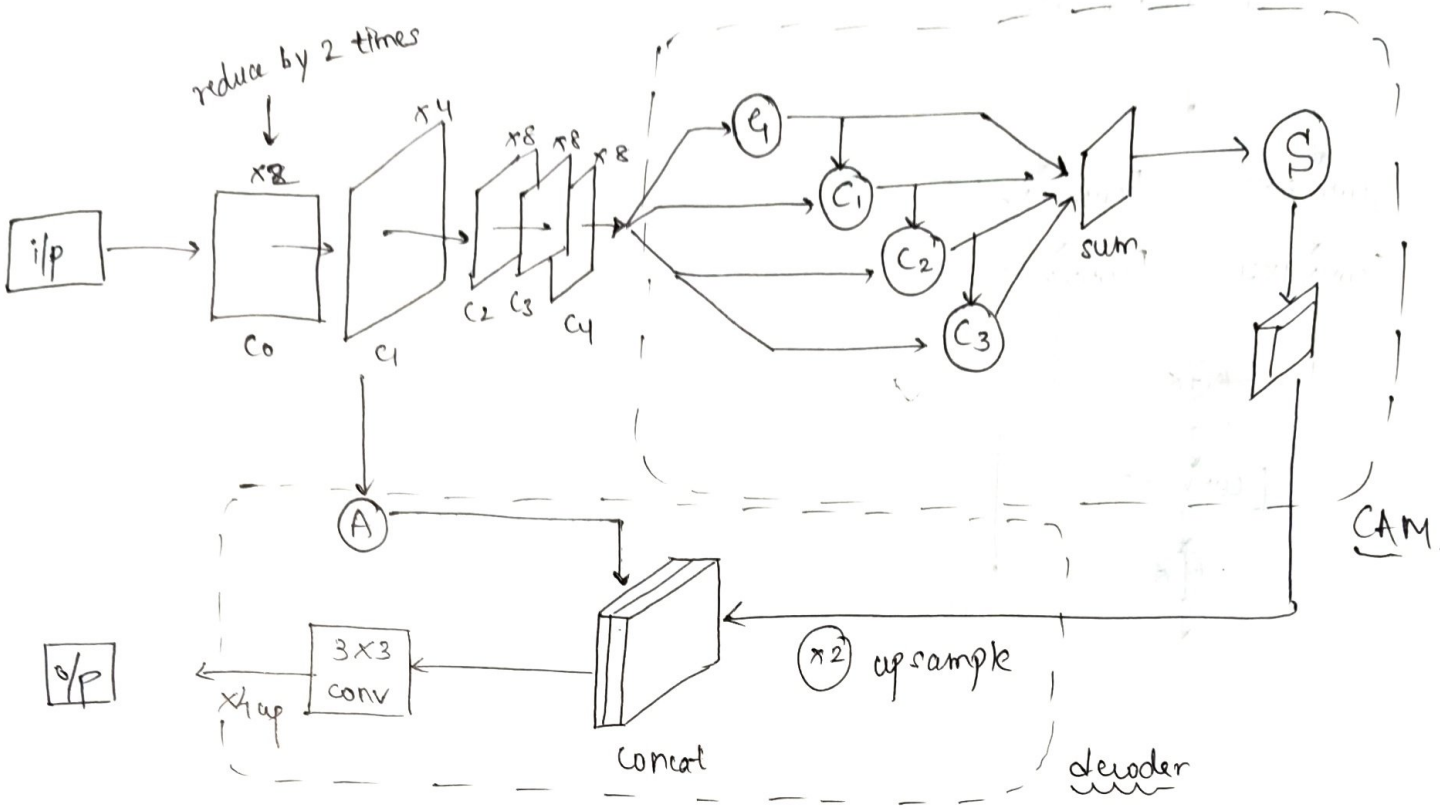


Embeded and positional encoded text

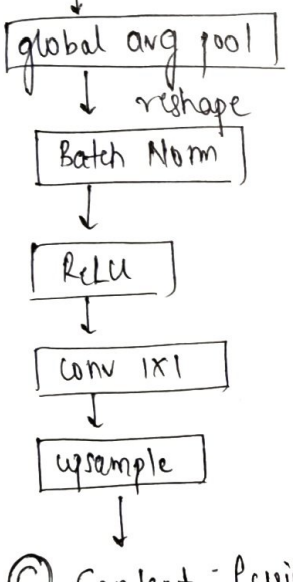
Decoder only Transformers ≡ GPT-1

Decoder models	Encoder models
→ GPT	→ BERT
→ unidirectional	→ Bidirectional
→ normally use masked atent n	→ simple self attention
→ used more in generative tasks	→ used mostly for text representation / embedding tasks

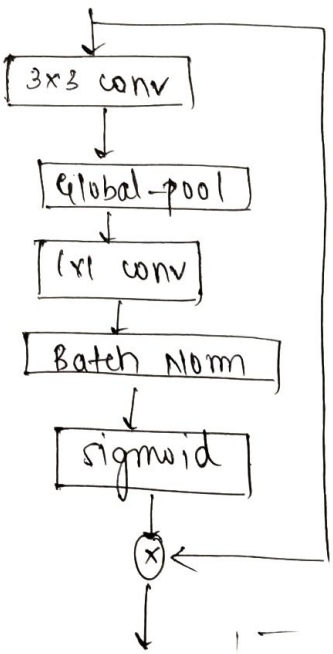
CANET — Image segmentation



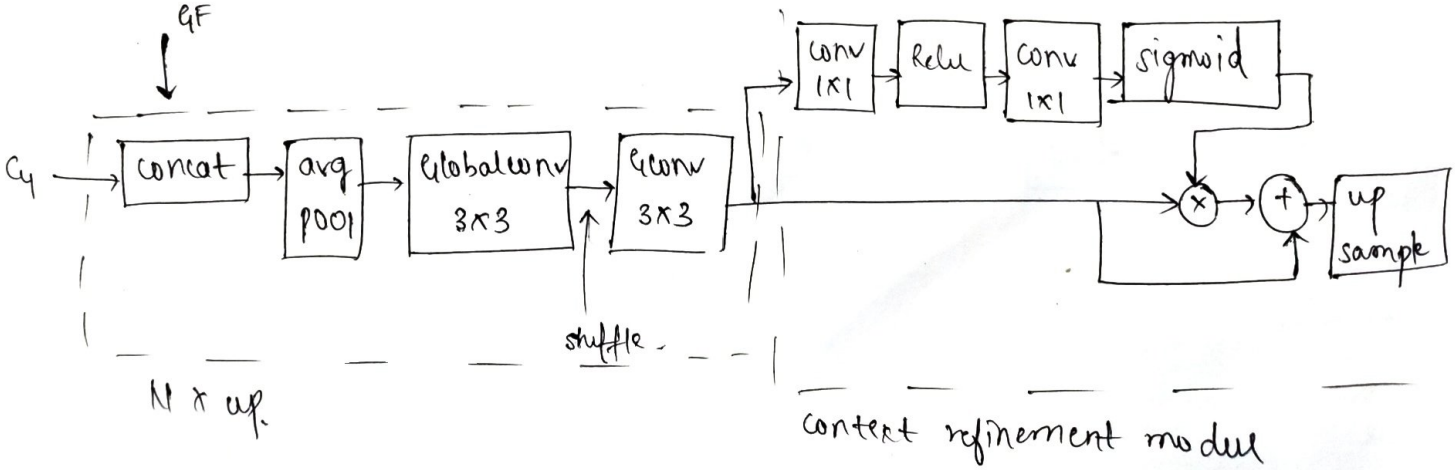
① (G) Global flow



③ (S) Feature selection module

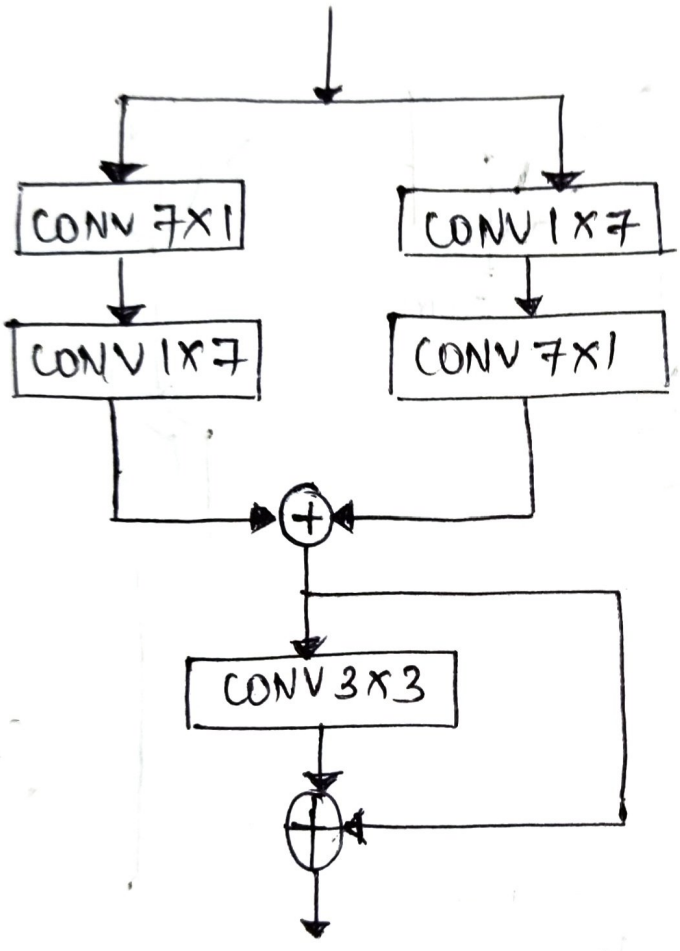


② (C) Context fusion model



58

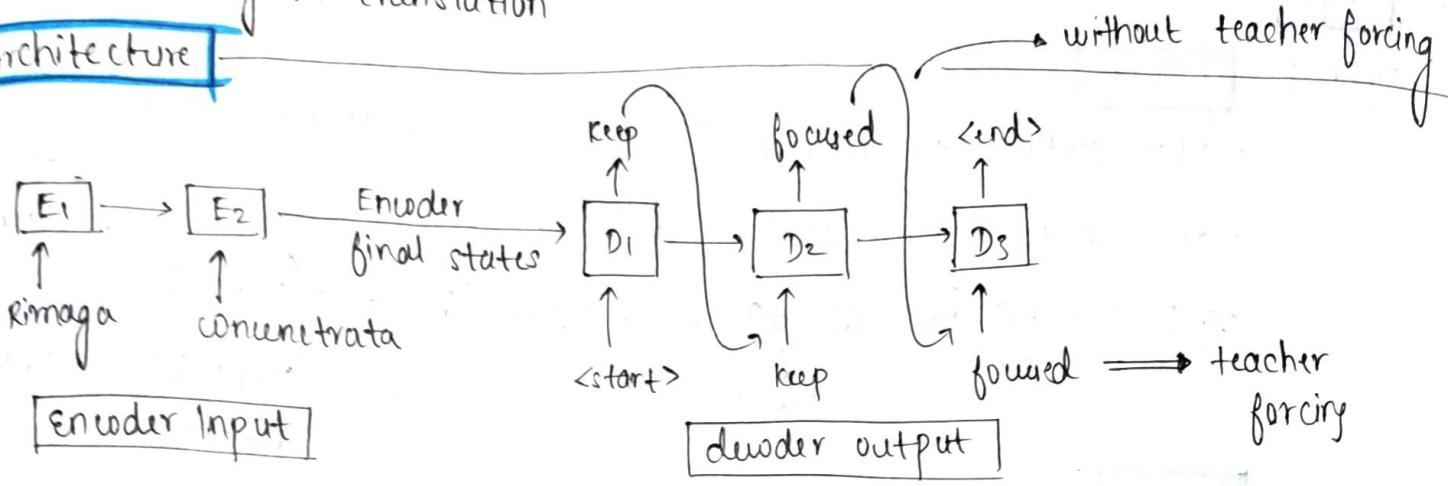
AGCN — Adapted global convolution n/w



Use case

→ Italian to English translation

Architecture



Data

(x): rimag concentro → Encoder iput

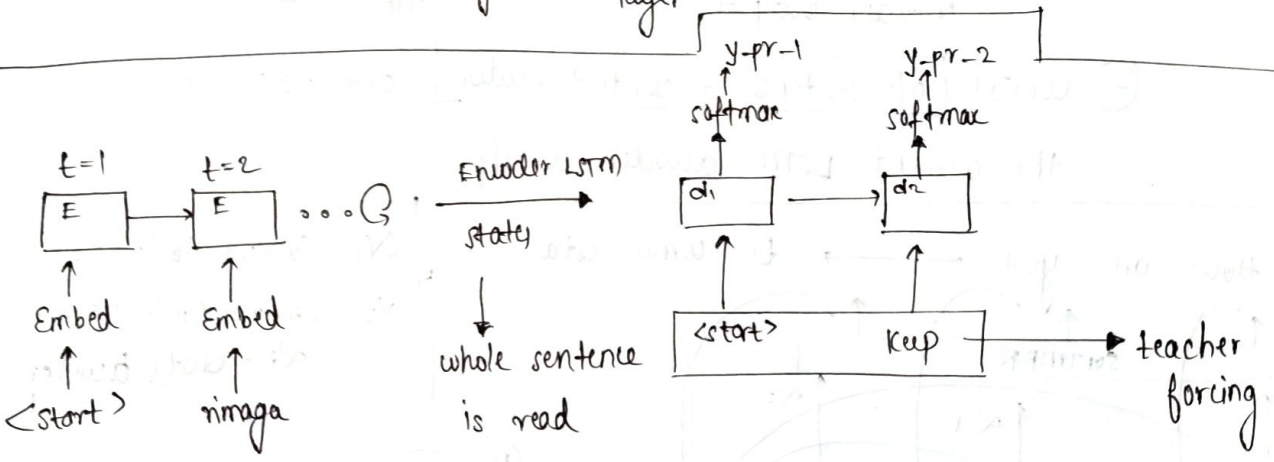
(y) keep focused → decoder i/p : <start> keep focused

decoder o/p : keep focused <end>

↳ apply tokenizer to convert text into integers.

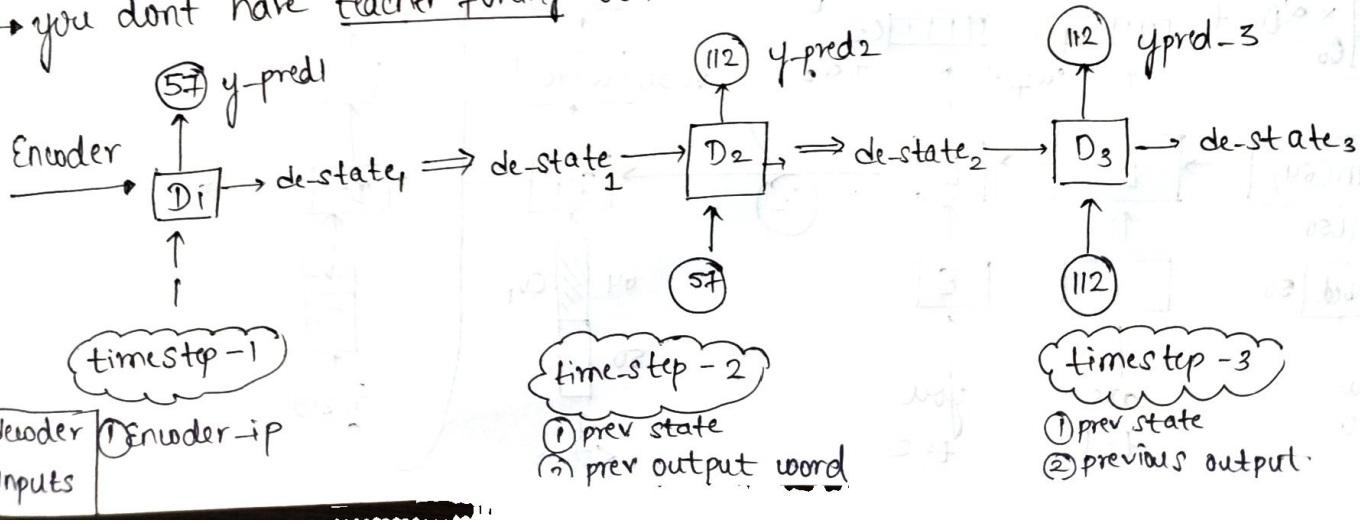
↳ (batch size, sentence_length) → Embeding layer → (batch_size, sentence_len, output_dim)

Ex



Inference

↳ you dont have teacher forcing sentences



Seq-2-Seq Attention

Steps

Train :- **Encoder**

→ pass sentence to encoder : words → embedding layer

↳ pass it to LSTM (seq=True, st=True)

o/p : Ex (i/p) (None, 600, 64) → LSTM (64) → (None, 600, 64)

(o/p of each ts)

embeded

(None, 64) → last h_s

(None, 64) → last c_s

Decoder

→ pass desired o/p sentence (teacher forcing)

at ts (t)

① using h_{t-1} and (He) find **context vector**

h_{t-1} : decoder hiden_state at prev ts

He : $(h_1, h_2 \dots h_n)$ encoder outputs

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \cdot \begin{bmatrix} \alpha_{t1} \\ \alpha_{t2} \\ \alpha_{t3} \end{bmatrix} = h_1\alpha_{1t} + h_2\alpha_{2t} + \dots$$

He α

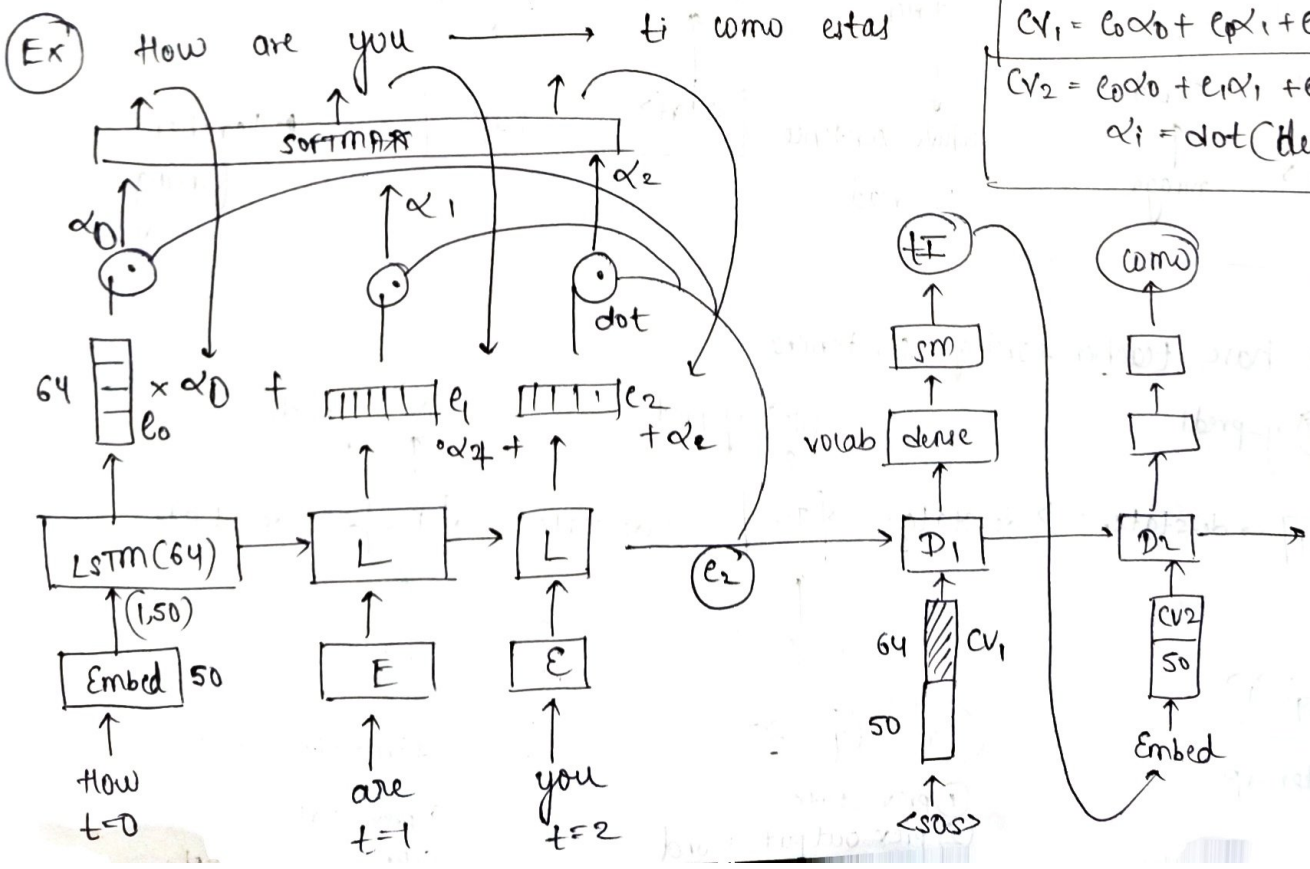
② concat [o/p sentence + context_vector] and pass on the current LSTM decoder as i/p.

Ex How are you → ti como estas

$CV_1 = e_0\alpha_0 + e_1\alpha_1 + e_2\alpha_2$

$CV_2 = e_0\alpha_0 + e_1\alpha_1 + e_2\alpha_2$

$\alpha_i = dot(Decoder_{t-1}, E)$



How to find α_{ij}s

Encoder

i/p → (None, 600, 50) → (None, 600, 64)

(600, 64) → all hidden states → h_e
(64, 1) → last h-s
(64, 1) → last c-s

Decoder

LSTM = 64

at time = t

i/p from prev decoder step = (h_{t-1}) (64, 1) → what if decoder LSTM (128)?

Now for first we last h-s of encoder

$h_e \cdot (h_{t-1}) = (600, 64) \cdot (64, 1)$

$= (600, 1)$

∴ you get 600 (α) = $\alpha_0 - \alpha_{600}$

→ pass it to softmax so that $\sum \alpha_i = 1$

* use a matrix W of shape (enc_units × decoder_units)

$h_e \cdot (W) \cdot h_{t-1}$

$= (600, 64)(64, 128)(128, 1)$

$= 600, 1$

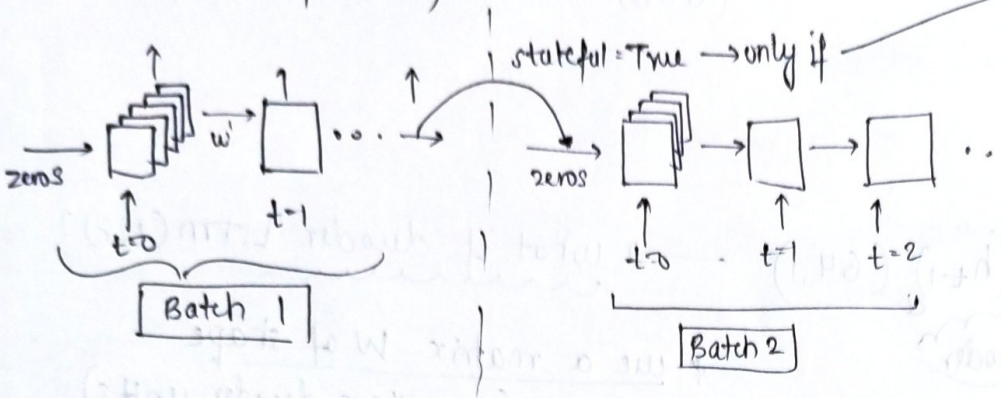
→ learned by backprop

Char-RNN concepts

Stateful LSTM → used when batches are sequential

(row-wise or continuity)

→ LSTM(5, stateful=True)



Batches must be sequential

0 1 2 3 4	5 6 7 8
9 10 11 12 13	14 15 16 17
⋮	⋮
Batch 1	Batch 2

Here stateful can be used

* while training batch 1, initialy weights are zeros and are gradually updated They are discarded at last step
* when batch 2 is trained again initial weights are zeros
  → Instead we can use the weights which are obtained from last unit of batch 1 ← stateful

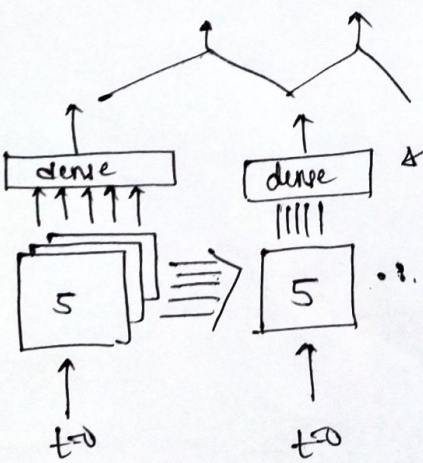
(Imp)

Use if dataset is proper, we can learn longer RNN sequences

Ex: given ex dataset we would learn sentences/sequences of len=4 (0-4, 5-8) → * with stateful we can learn longer ie (0,8) (9,17)

TimeDistributed Layer → apply a dense layer on top of output of each LSTM timestep

* LSTM(5, ret_sequences=True)



time distributed dense layer

Code → TimeDistributed(Dense(vocab_size))

Ex: LSTM(64, ret_seq=True)

(Batch, 18) → (batch, time_steps, 64)

(Batch, 18) —LSTM→ (bat, ts, 64) —TimeDist(Dense(16))→ (batch, ts, 16)

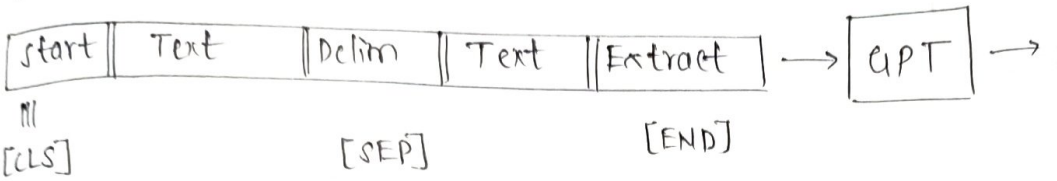
Generative Pre Training

GPT - 1

Architecture : ~~tra~~ Decoder only transfomer
↳ 12 decoder cells (pg 56)

Training : Next word prediction
→ give word (w_i) predict (w_{i+1})

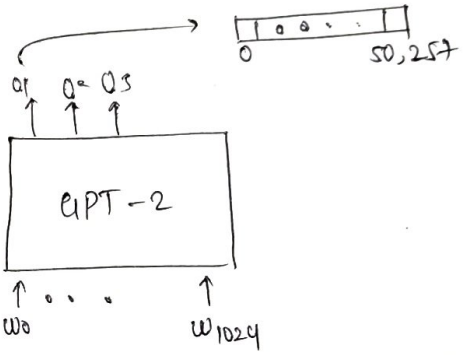
Tokens



GPT-2

Improvements on GPT-1

* max input length = 1024 (GPT1 - 512)
* vocab size = 50,257 (GPT1 - 40K)
* Layer Norm is used @ input of each sub block
* larger batch size : 512 (GPT1 - 64)
* token embeding size = $d_{model} = 768$
* used 10x more data to train
  +
  Had 10x more parameters



Params	layers	d model	
117M	12	768	→ GPT2 small
345M	24	1024	
762M	36	1280	
1.542B	48	1600	→ GPT2 large

* Zero-shot learning
  ① Pretrain model (NWP)
  ② fine tune model by giving your dataset → NO fine tune (Pretrain and directly predict)
  ③ make predictions

→ few-shot learning
① → ② → ③

disadvantages

performs poorly on rare or specialized content

(64)

GPT-3

→ 100x more params (175B)

→ trained on

[300B text tokens + common crawl data (2016–2019)]

→ uses sparse attention

disadvantages

→ Cost and compute power

→ Interpretability

→ Biases in predictions