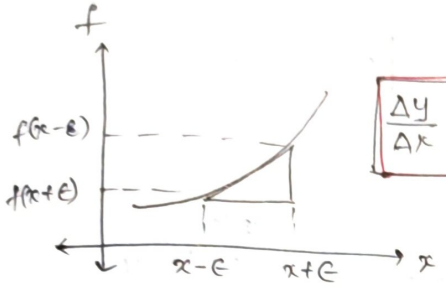
Index

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:- $W = w_1, w_2$, $X = x_1, x_2$ $\quad f(w_1, w_2, x_1, x_2) = w_1^2 x_1 + w_2^2 x_2$

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

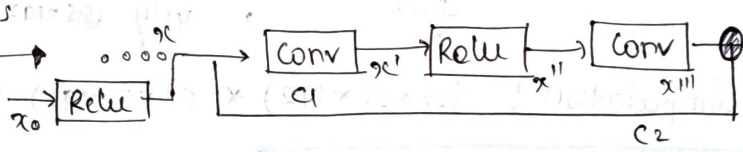
$(dw) = 2 w_1 x_1 = 6 \rightarrow (1)$ $\quad (dw_1)_{approx} = 5.9999999 \rightarrow (2)$

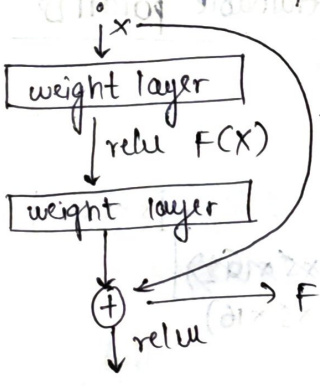
$$\text{Grad-check} = \left(\frac{6 - 5.9999\cdots}{6 + 5.9999}\right) = 4.251268e^{-13}$$

∴ 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 $x \rightarrow x$





skip conn

why it works?

$Relu(x_0) = x$, $Relu(x) = Relu(Relu(x_0)) = x_0$

$= Relu(x_0) = x$

L_1, L_2, dropout

Now consider due to strong regularization C_1 and C_2 become useless, ie w or $l = 0$;

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

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

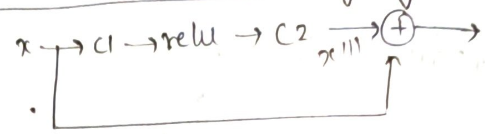
if +ve, $Relu(x) = Relu(x) = x$

−ve $= 0$

(22) Key takeaway → gauranted no decrease in perfomance, small increase

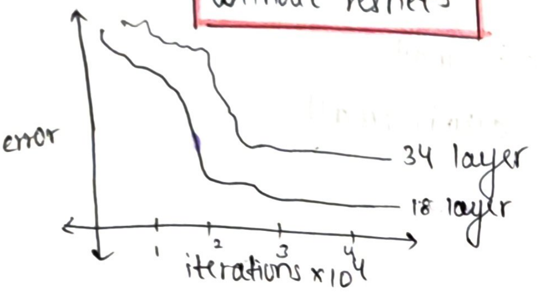
* adding additional layers would not hurt performance as they get skipped if they are found useless.
* if new layers are useful
  regt weights are non-zero.

Note :



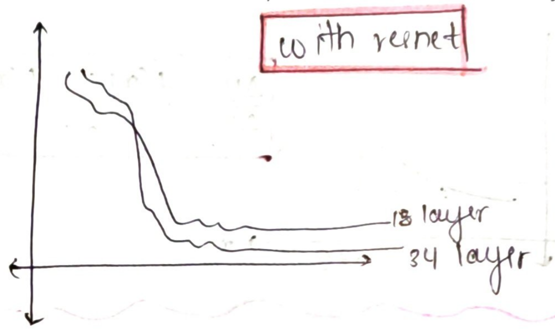
make sure (x) and (x''') have same shape - use padding

without resnets



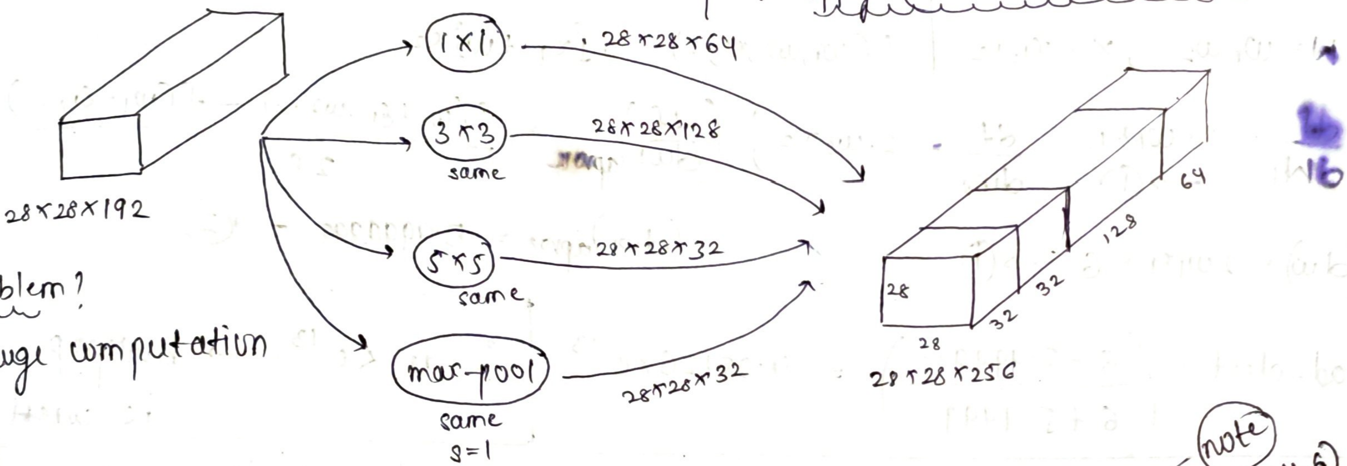
with → res-nets

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: $K = (4,6)$ ||| $(4,6,\text{prev layer d})$

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

no of trainable params

using 1x1 conv as bottleneck layer

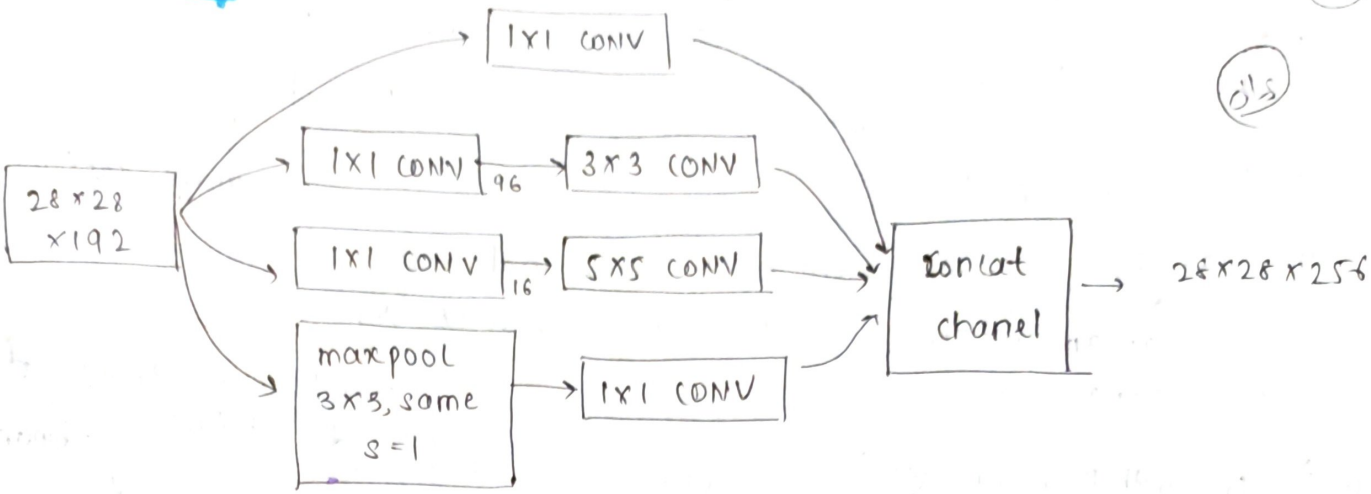
28x28x192 → CONV 1x1,16 → 28x28x16 → CONV 5x5,32 → 28x28x32

computations : [(28x28x192) * (1x1x192)] + [(28x28x16) * ~~(5x5x(64))~~ (5x5x16)] 5x5x16

= 12.4 million <<<< 120 milion.

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

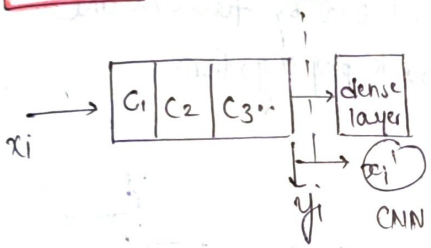
Inception module



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 VGG + 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 3 D ≈ ImgNet, |D| medium sized

→ fine tune last few layers

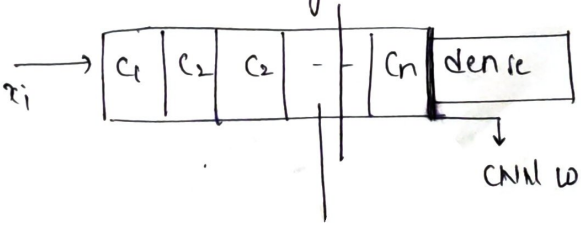
Case 5 |D| large (size of D), D ≠ ImgNet (similarity to ImgNet)

→ use pretrained model as init and train/tune whole model

Case 2 D ≈ ImgNet, |D| large

→ tune full n/w

Case 4 |D| is small, D ≠ ImgNet



more generalized as Cn will be useful if similar

→ pick from later CONV layers but not last

(24)

Recurrent Neural Networks

↳ retains and leverages sequence info

core idea : sequence of inputs

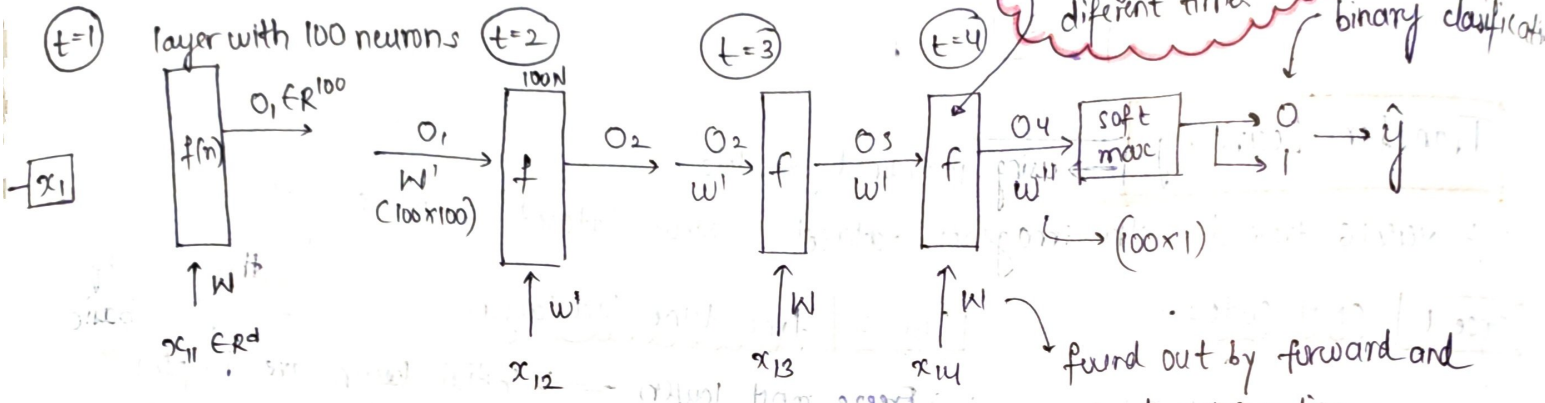
↳ app^n → machine translation, time series, speech recognition, image captioning

Ex: Amazon fine food reviews

$D = \{x_i, y_i\}$ → Each word is 'd' dimensional vector where d = size of vocabulary

$x_1 = \langle x_{11}, x_{12}, x_{13}, x_{14} \rangle$

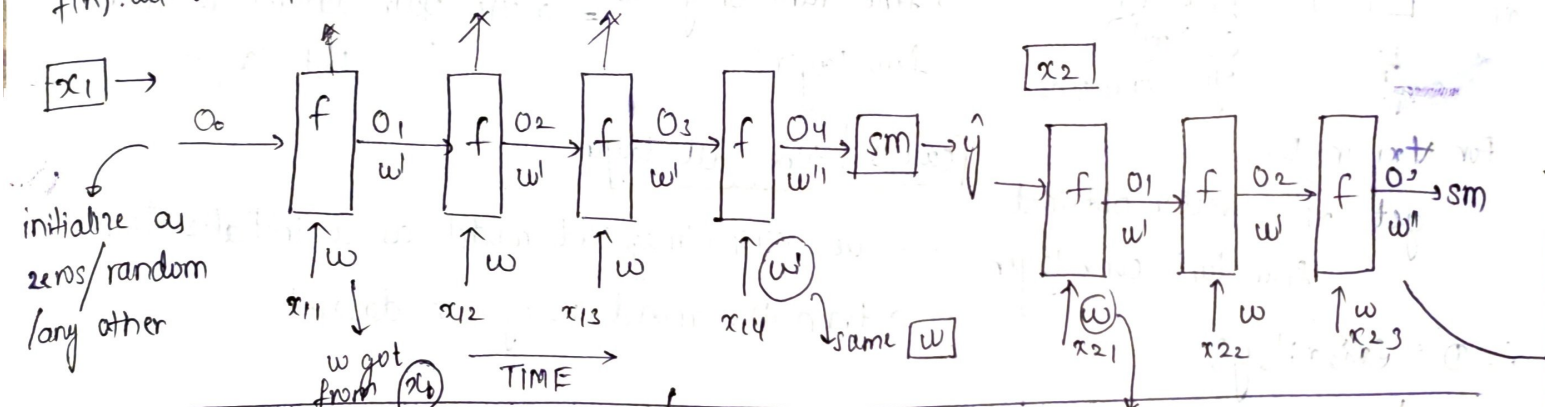
$x_2 = \langle x_{21}, x_{22}, x_{23} \rangle$



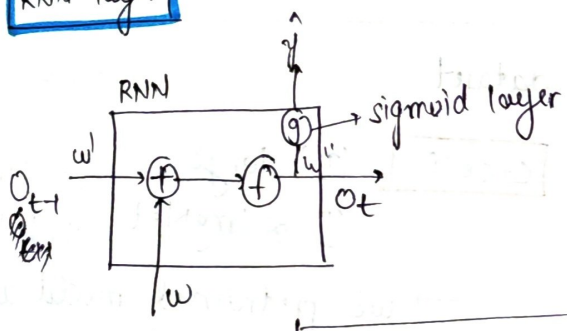
W: d×100

$f(n)$: activation fn

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



RNN layer

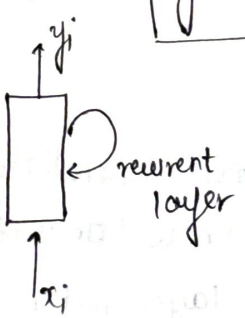


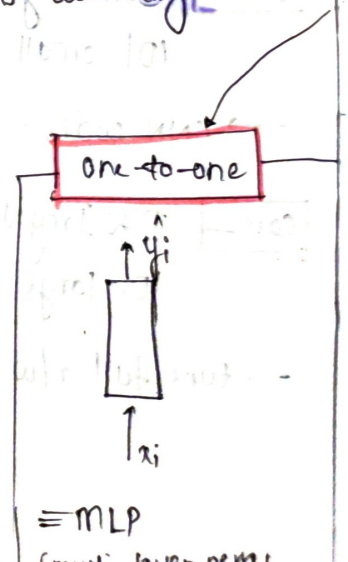
$$O_t = f(wx_{it} + O_{t-1}w')$$

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

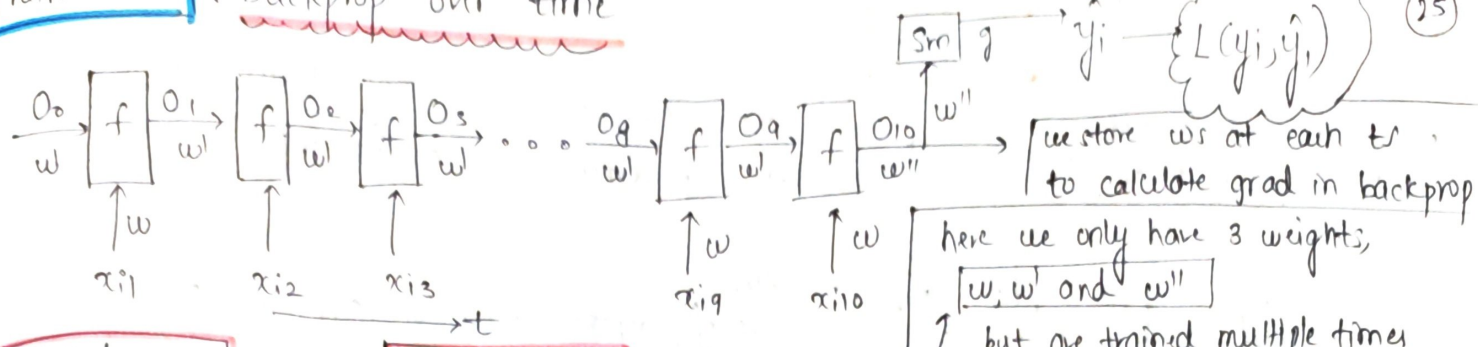
→ In the whole process we have only 3 matrices, w, w' and w''

→ they are given different inputs according to time t





Train RNN : Backprop over time



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 $\frac{\partial L}{\partial \hat{y}_i}$ → ② $\frac{\partial L}{\partial w''_{10}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_{10}}$ → ③ $\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial O_{10}} \frac{\partial O_{10}}{\partial w_{11}}$

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

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

Problems with backprop over time

$\left(\frac{\partial L}{\partial w}\right)_{t=1}$ = multiplication and addition of multiple values → vanishing gradient, exploding gradient (not bcoz of deep layers but bcoz of backprop over time ∝ depends on t's)

→ LSTM and GRU used

↳ $w_{t=9} = \frac{\partial L}{\partial w_{10}}$

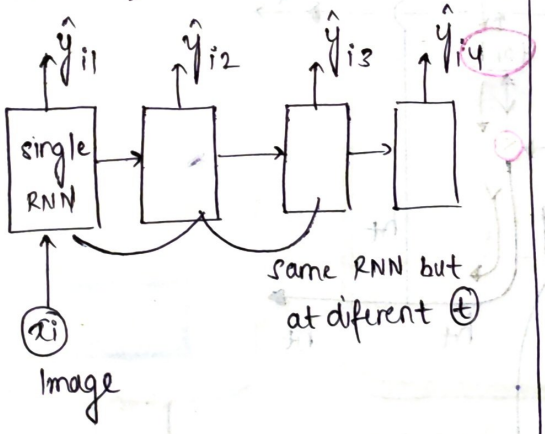
Types of RNN architectures

① Many-to-one RNN

uses: sentiment analysis, movie review rating

② One-to-many RNN

i/p : single
o/p : sequence



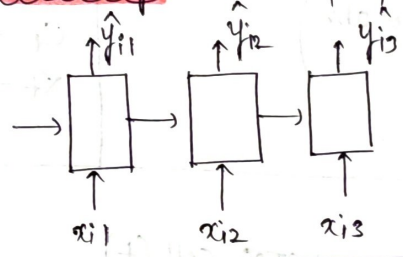
Ex: Image captioning

Many-to-many

len(i/p) = len(o/p) (seq2seq)

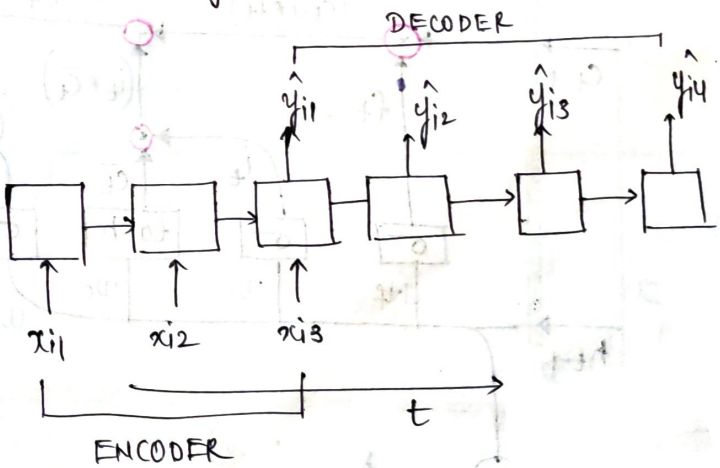
→ # word → which part of speech

① same length :- parts of speech



[parallel execution] can be done

② different length — Encoder-Decoder



use :- language translation = Eng → Spanish

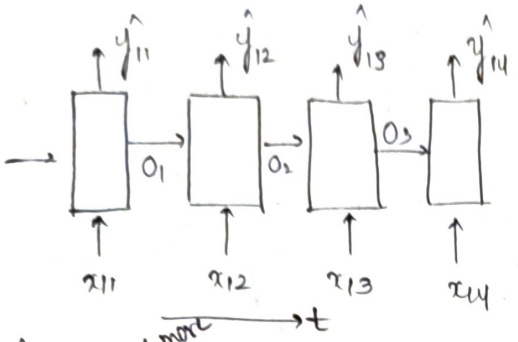
a sentence in Eng with 10 words → may have only 8 words when translated into Spanish

1997 LONG SHORT TERM MEMORY RNN

LSTM

why we need LSTM when RNN exists?

- RNN cant take care of long term dependency ie later o/p depends on earlier input



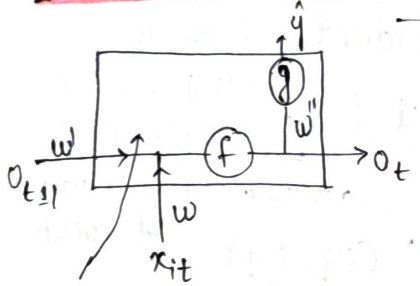
→ here,
* $\hat{y}_{14}$ depends more on x_{14} and O_3 and depends very less on O_1 and x_{11}
* lets say we want $\hat{y}_{14}$ to be more dependent on x_{11} which cant be accomplished

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

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

LSTM

Improved simple RNN



concatenation

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

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

merge vectors → [w' w]

concatenate: $[x_{it}\ O_t]$ (stack one abv other)

similarity

$C_t \equiv O_t$

$X_t \equiv X_{it}$

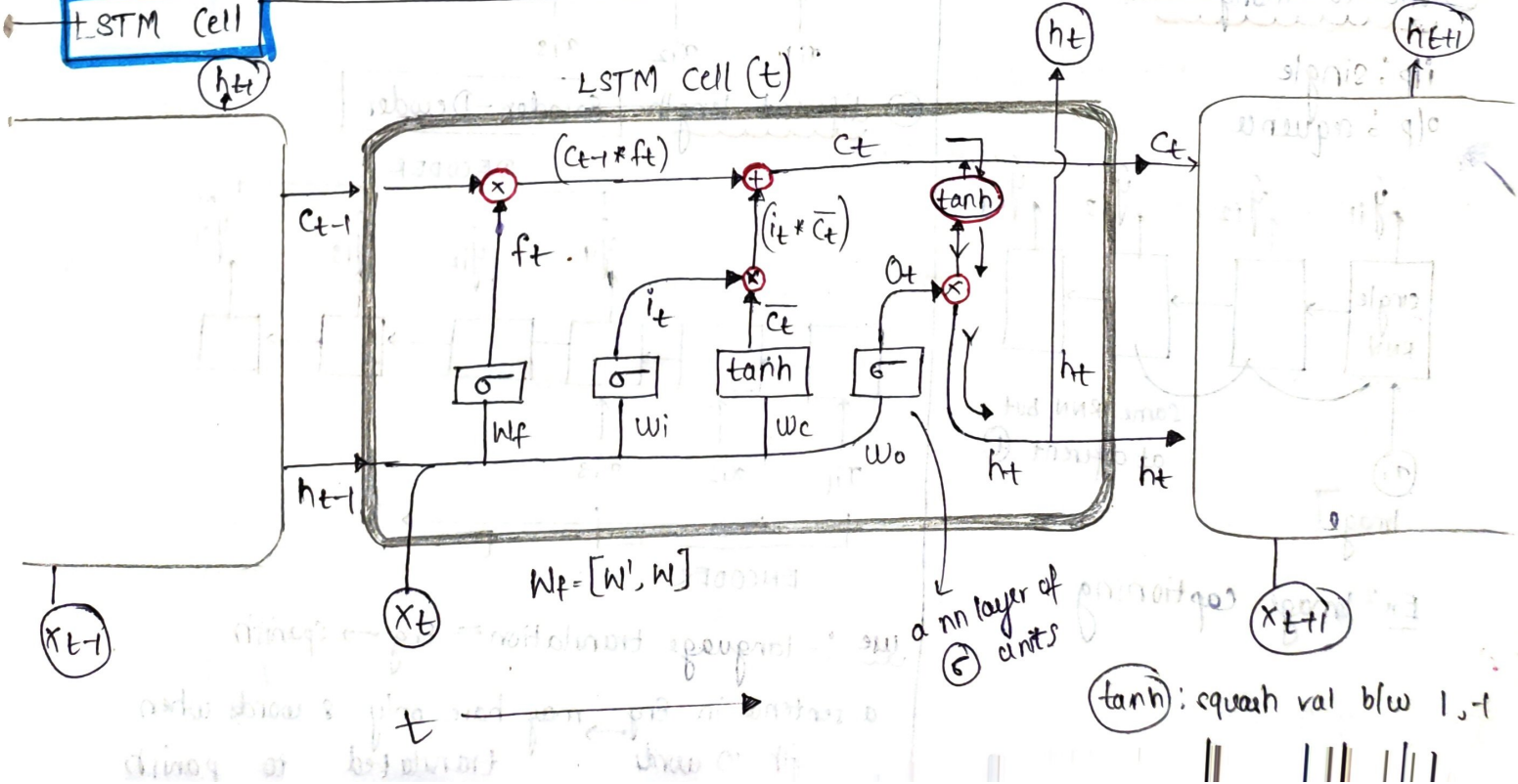
$h_t \equiv \hat{y}_{it}$

Notations

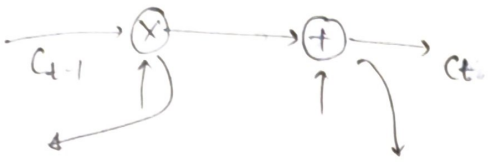
- ○ → point operation (apply 2 each)
- □ → Neural network layer
- C_t → cell state
- X_t → i/p point
- → concatenate
- → copy

Ex: [1 2 3] [1 1 1] ⊗ [1·1 2·1 3·1]

LSTM Cell



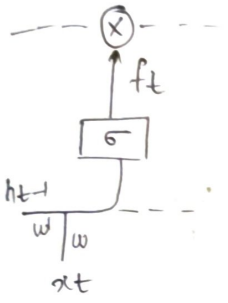
skip / identity layer → if your op needs long term memory just skip



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

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

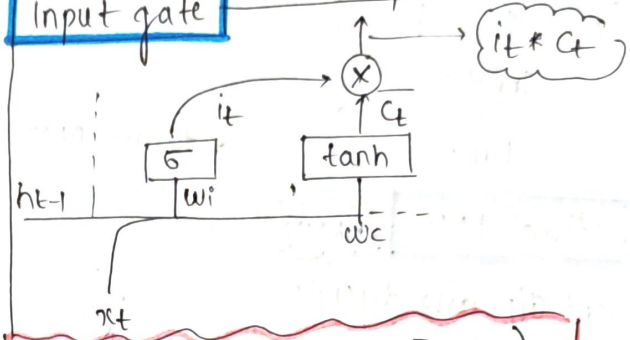
W_f : forget weights $[w, w_i]$

b_f : bias factor

f_t → how much data to forget

Input gate

add new info



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

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

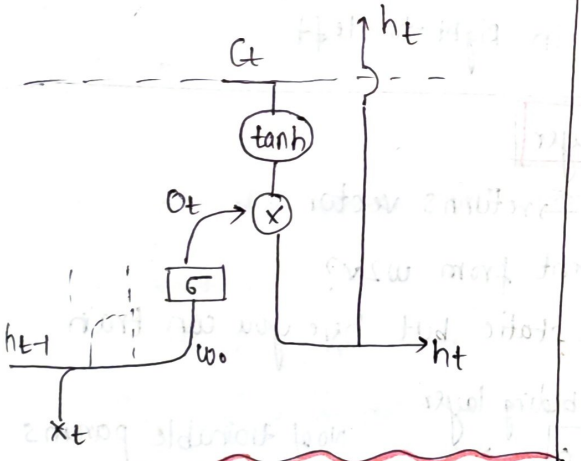
identity + forget + input gate ⟹

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

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

Output gate

how much op to send?

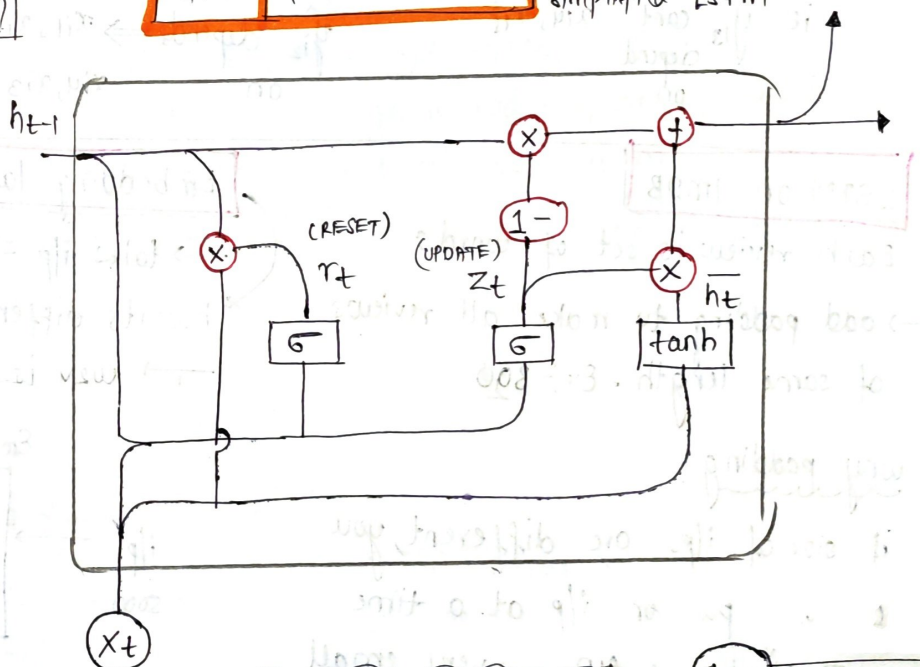


$$O_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

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

GRU | Gated RNN unit

simplified LSTM



GRU vs LSTM

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

∴ As lesser gates → lesser eqn → faster to train

o/p, forget, ip

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

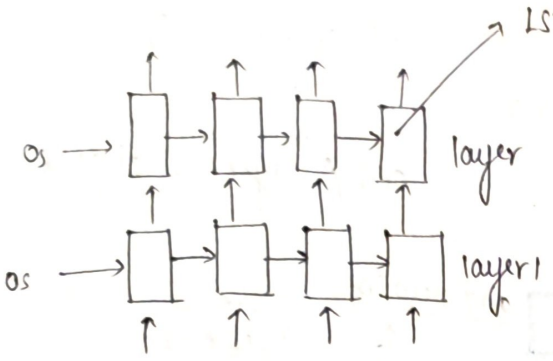
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

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

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

1 − [1,1, ...] − [ip vector]

Deep RNN



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

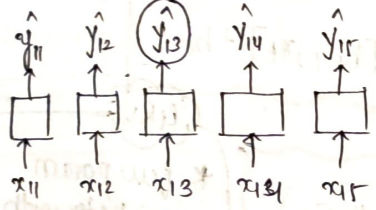
→ cant be used for live audio stream

→ to use BRNN whole i/p must be present

Bi-Directional RNN — BRNN

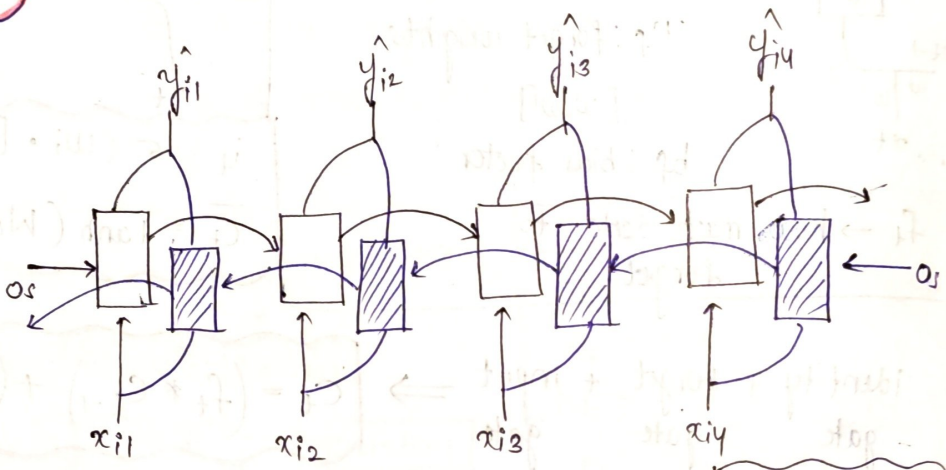
what if current O/p also depends on future i/p ?

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 ⟹ 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:- 3900

why padding ?

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

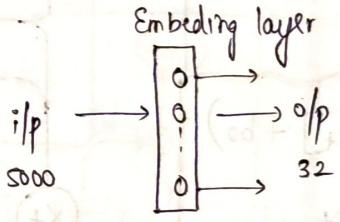
→ if i/p size is same, you can parallelly process inputs like in batch GD → faster

Embedding layer

→ takes i/p → returns vector

how its different from w2v?

→ w2v is static but here you can train



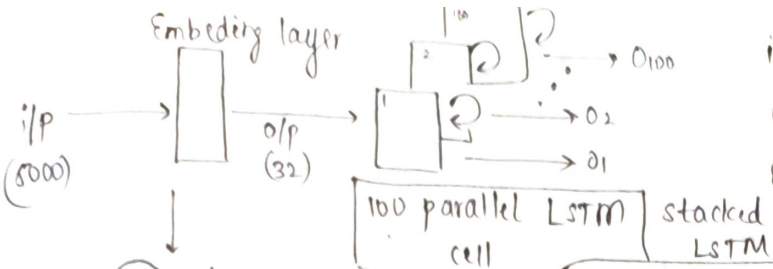
No of trainable params
= 5000 × 32
= 160K params

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) chuck dissimilars

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

Generative Adversarial Networks

simpler task : 1D scalar → (step3) check how similar D and D'

D = {set of heights}

Generate D' similar to D

(prob model)

→ (s1) generate PDF of D, P(H)

* find which distr it is

Ex :- it is normal distr

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

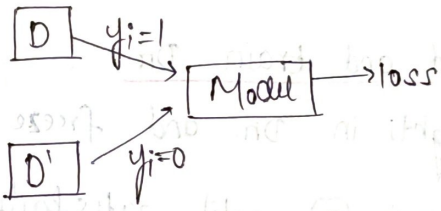
$P_\theta(H) = N(\theta = [\mu, \sigma])$

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

① use statistical tests
JS-dist
KL-div

problem : it works only with small dimension

② model aproach

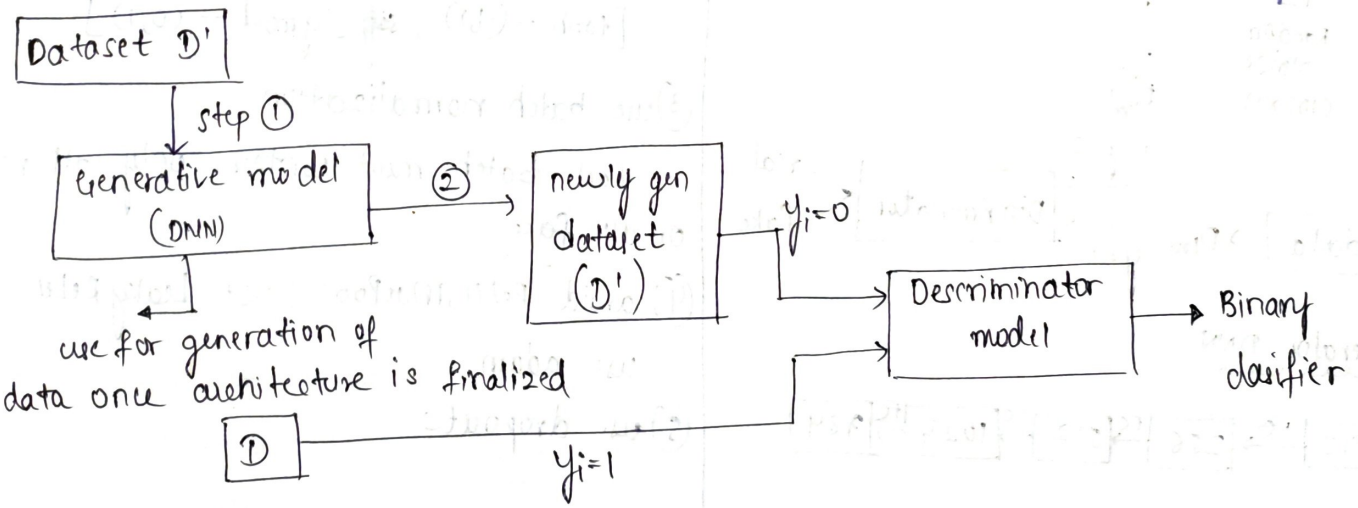


as loss ↑ → similarity ↑

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

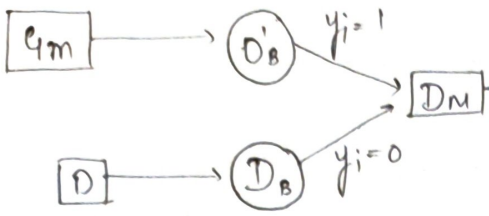
→ if loss is large ie bad performing model
→ D and D' are similar

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



(30) **Train GAN**

D'_B = batch of dataset say of 100 datapoints



(maximize loss)
↳ 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? vector
~~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

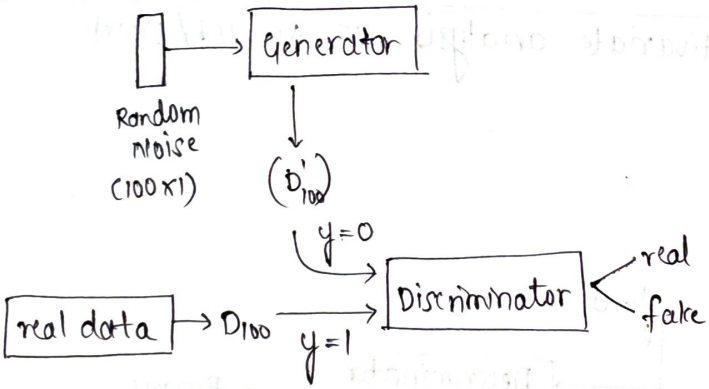
* we need to find and maximize loss using gradient ascend

step 2 Keep Gm fixed and train Dm

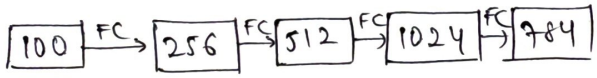
* only update weights in Dm and freeze weights in Gm

→ Alternatively do ① and ② until satisfactory loss is obtained

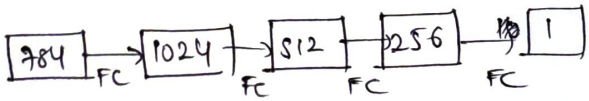
GAN for MNSIT



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

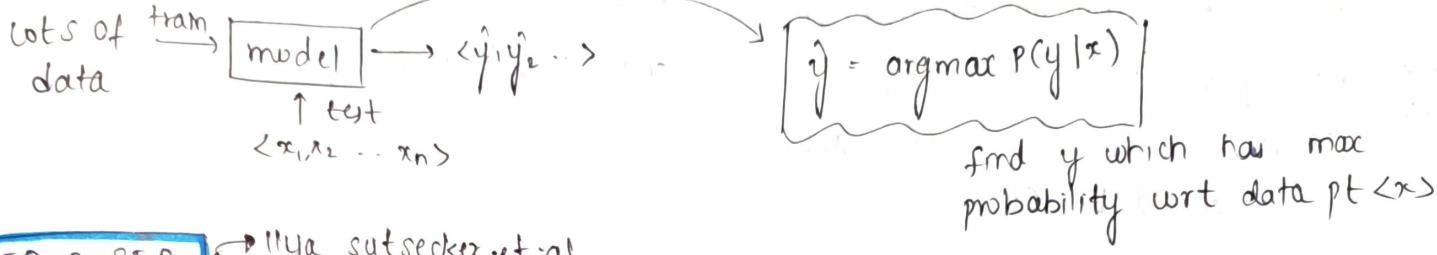
④ avoid ReLU, MaxPool ; use LeakyReLU
use adam

⑤ use dropouts

Encoder - Decoder

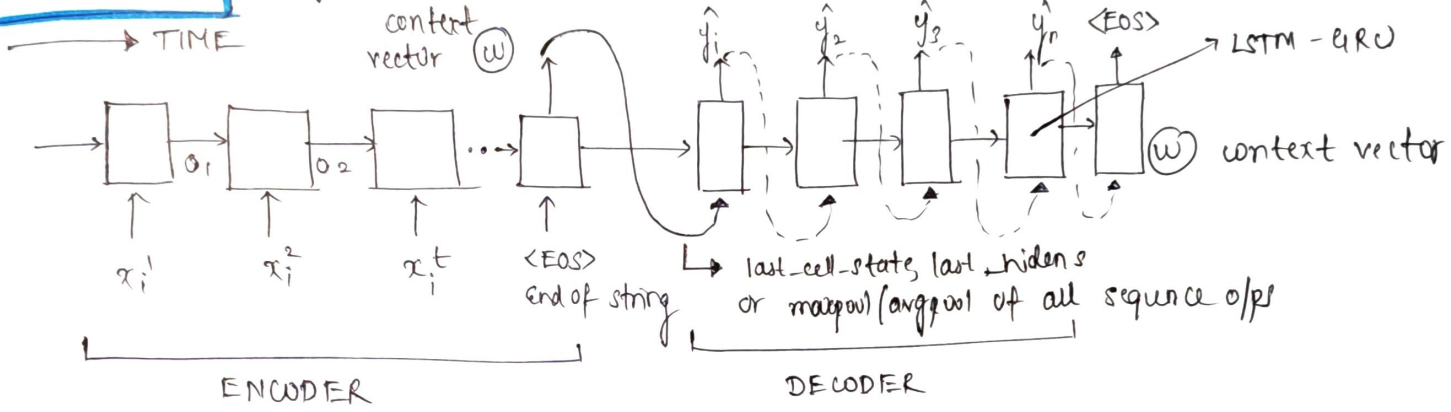
$\text{argmax}\ P(\langle y_1, y_2 \cdots y_n \rangle \mid \langle x \rangle)$

Probabilistic repr of model



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

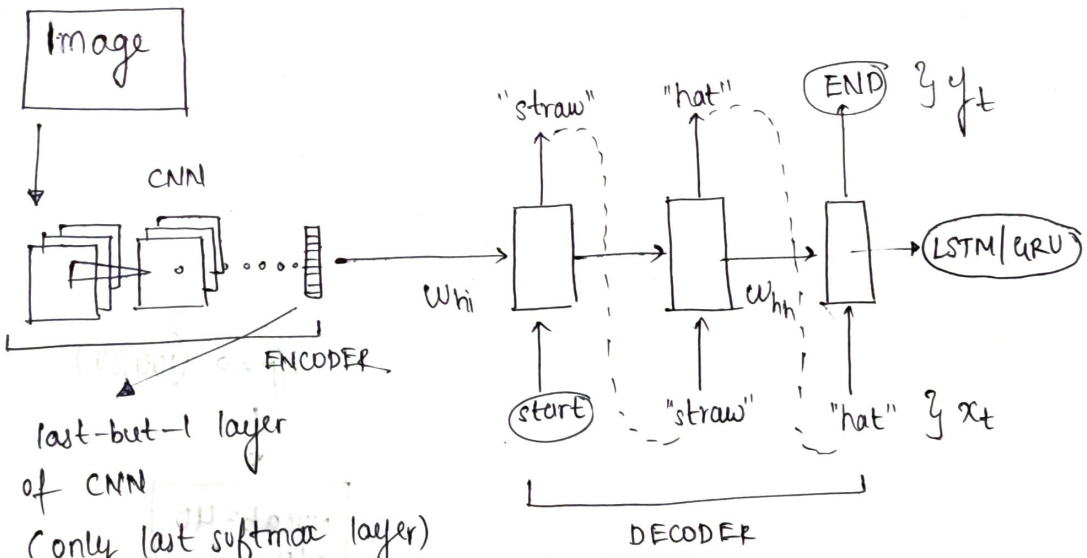


Image context vector

Applications

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

32

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 }

i/p ⇒ (100, 50)

↓

vocab=40
o/p = 8

~~Embed (8,~~

Embed (40, 8, input_len = 20)

↓

op → (None, 100, 8)

param s = vocab size * 8

Word Embedding Layer

DL + NLP

word embedding ?

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

word_embedding_layer → let nlw learn embeddings 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)

{"word": index}

→ 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

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)

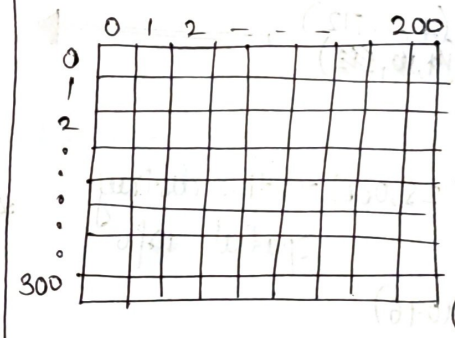
∴ we have each sentence/data point with len = maxLen

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



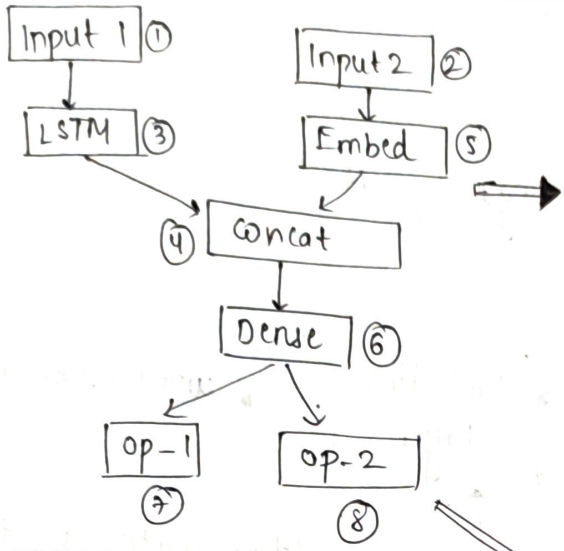
∀ each word in vocab, have a v-dim vector

```
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(em)
              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



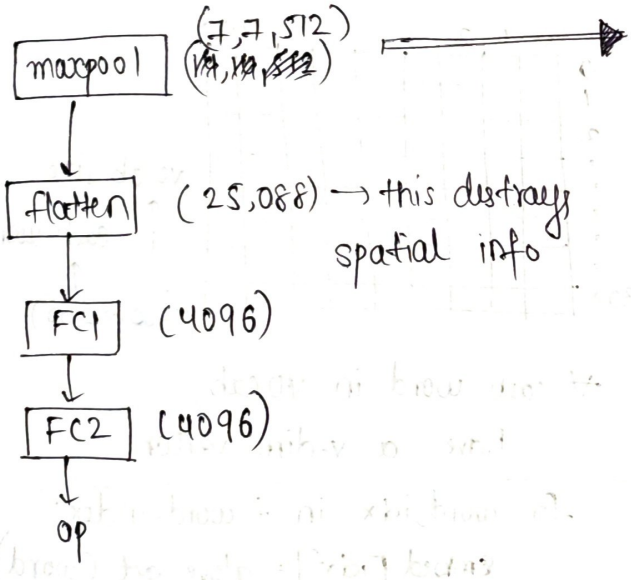
① Inp-1 = Input ((128,))
③ LSTM = LSTM( ) (Inp-1)
② Inp2 = Input ((10,))
⑤ Embed-L = Embedding( ) (Inp2)
④ concat = concatenate( )([LSTM, Embed-L])
⑥ Dense = Dense( ) (concat)
⑦ op-1 = Dense (10, softmax) (Dense)
⑧ op-2 = Dense (2, softmax) (Dense)

Train

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

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

Replace FC layer by Conv 2D



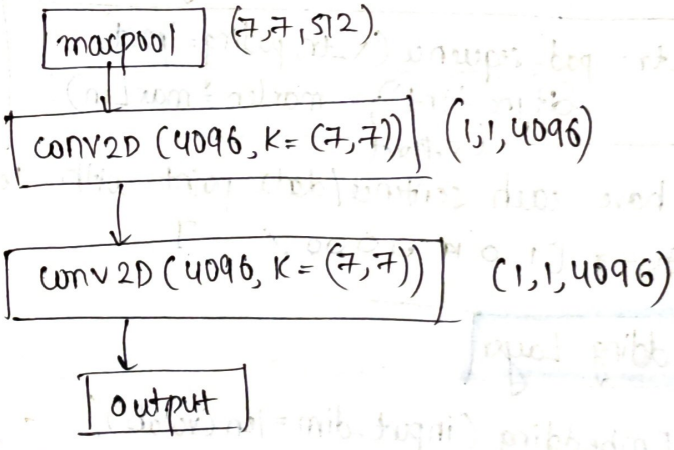
replace FC by CONV 2D

for FC1 we need 4096 ≡ (1,1,4096)

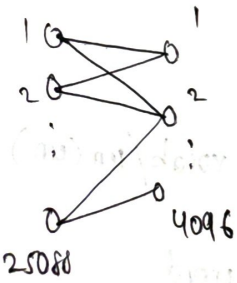
$*(7,7,512) \xrightarrow[s=1,\ fn=4096]{K=(7\times7)} (1,1,4096)$

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

$= 1,1,4096$

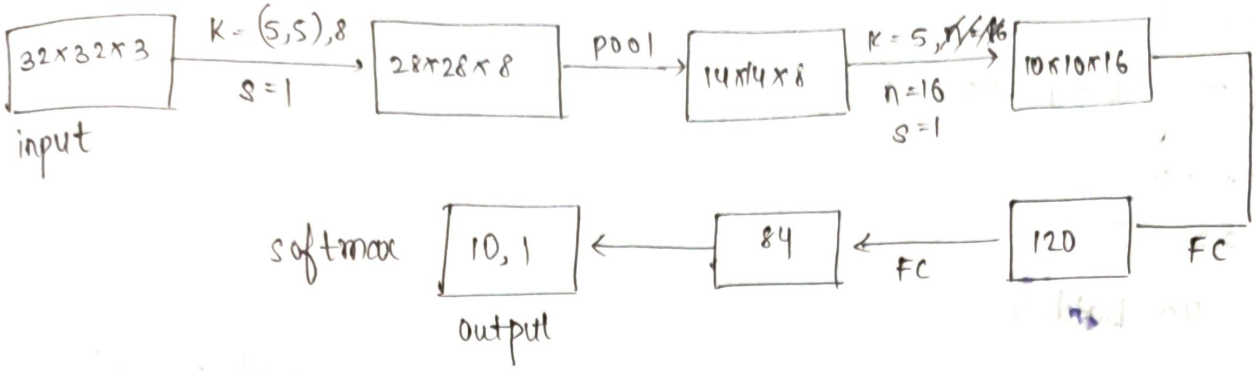


FC1



∴ 4096 * 25088 com

Calculate CNN params



* conv layers

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

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

(+1 → Bias term for each filter)

$$\#\text{params in FC} = FC(Fcn) = \#\text{neurons in last layer} * \#\text{neurons in crent FC} + Fcn$$

(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)	$\left(\left(5 \times 5 \times 3\right)+1\right) 8$ = 608	6272
POOL1	(14×14×8)	0	1568
CONV2 K=5, n=16	(10×10×16)	$\left(\left(5 \times 5 \times 8\right)+1\right) 16$ $[(5\times5\times8)+1]*16$ = 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

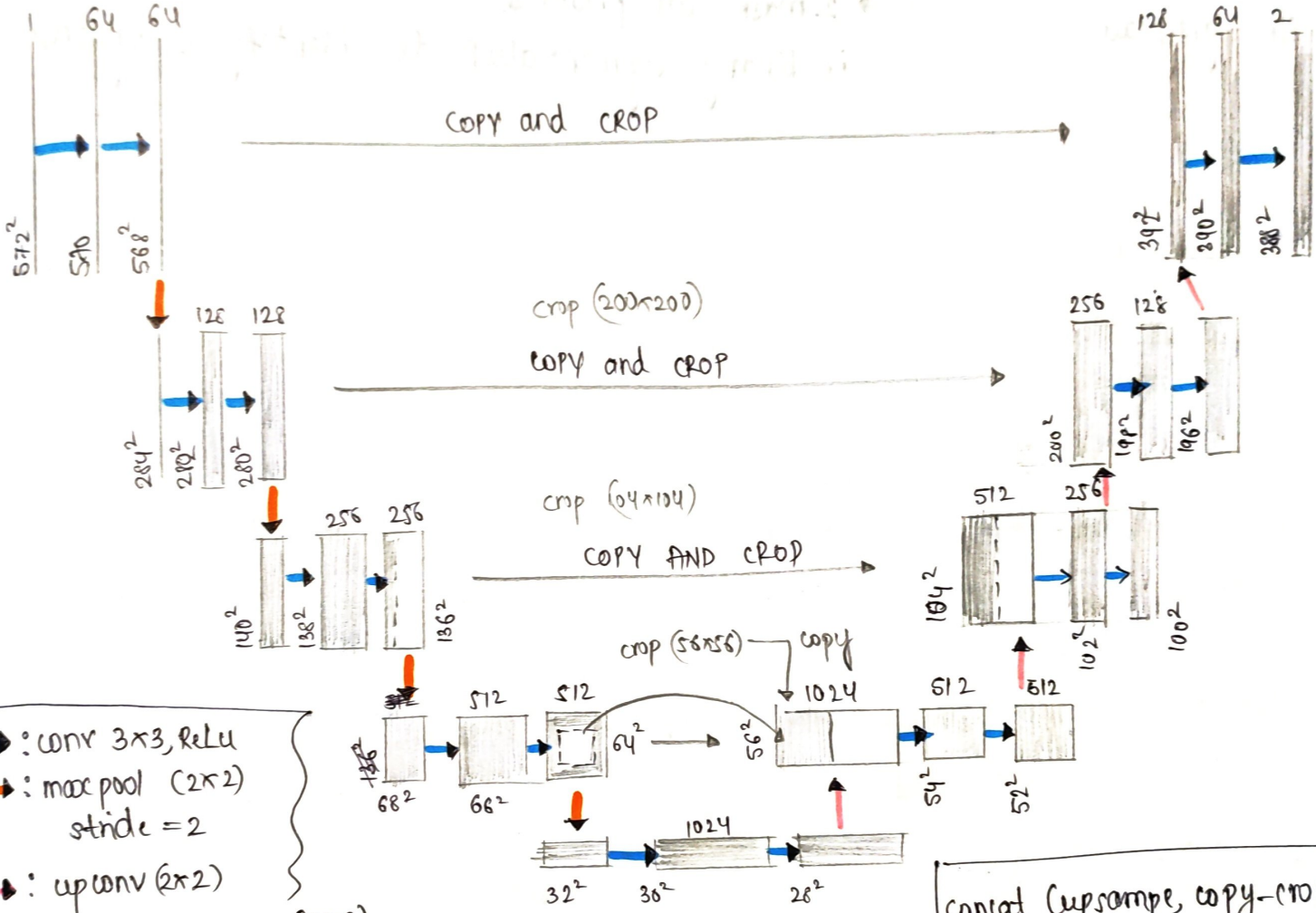
→ divide into areas / segments

→ measures :-

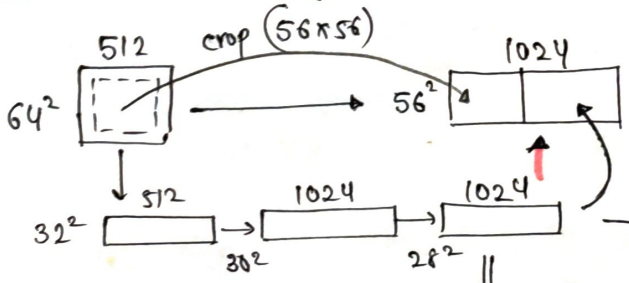
① jacards similarity = $\frac{R \cap R'}{R \cup R'}$ (IOU)

② cross entrophy = per pixel comparision

Unet



upsample base layer



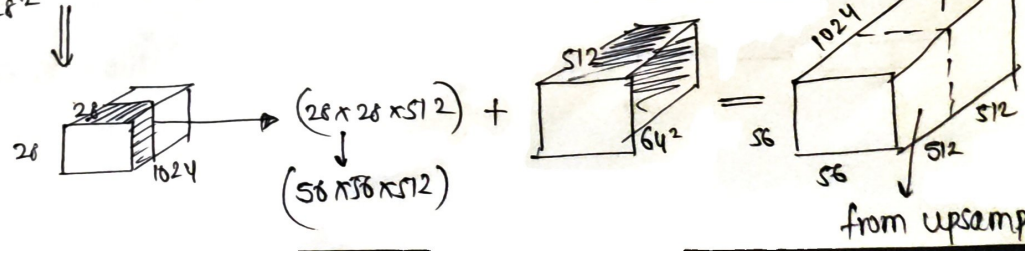
upsample

① upsample : duplicate rows and colums

28×28×1024 → 56×56×1024

② conv2D (f = 512, (2×2))

(56×56×512)



(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)
  - 2 masks are predicted
    ie binary segmentation to identify 2 regions

i/p - grayscale

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

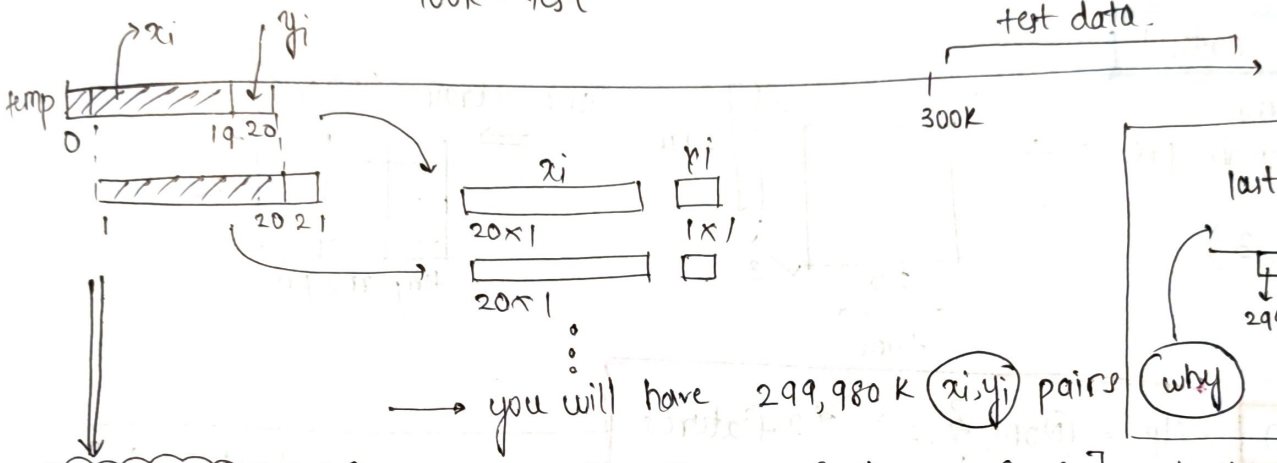
Dataset :- weather temp each 10 minutes given

Univariate prediction

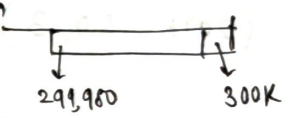
asume (K=20) we take window size = 20

* Normalize all data
  → only train data

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



last sample is



why

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

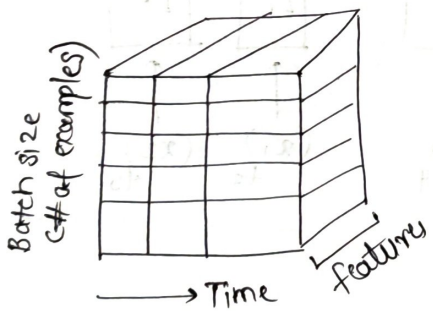
new dataset → (299,980, 20, 1) [reshape (20,) into (20,1)] → better for training model

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

LSTM training diagram

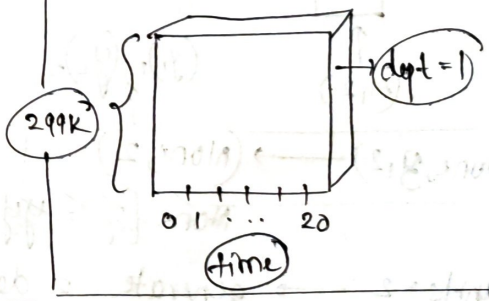
* LSTM expects data in this format → [batch, time, features]

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



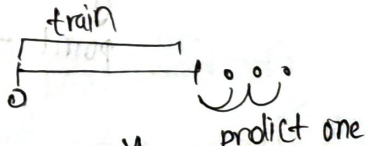
loss = mae
mean absolute error

Our data

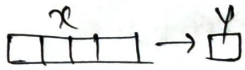


multivariate - single step

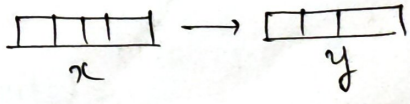
→ given a windows predict only one data at $t_1 - - t_n$



* single step → predict only one pnt at future

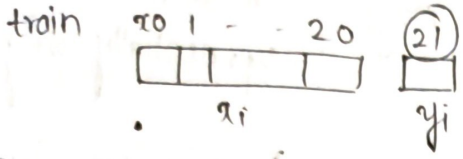


* multi step → predict over multiple points

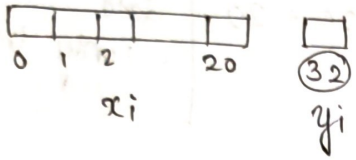


Dataset generation hacks

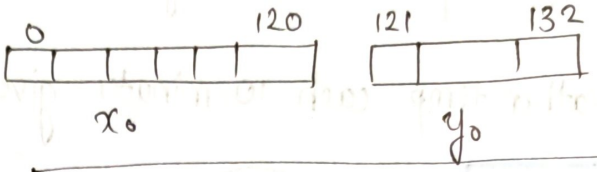
① predict next hour itself



② predict next 12 hrs



② predict next 12 hours data using past 120 hours → multi step

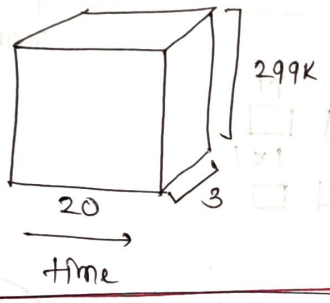


∴ xtrain = (299K, 20, 1)
ytest = (299K, 12, 1) or (299K, 12)

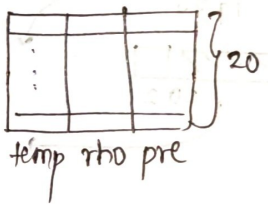
multivariate dataset

predict — temp
data — temp, rho, pre ⟹

(299K, 20, 3)



one xtrain (20×3) ⟹



Keras LSTM

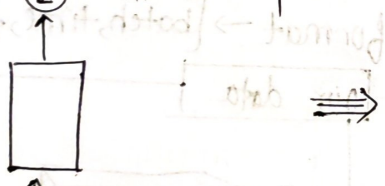
ip = (None, 4, 2) — 2 features; 4 time-steps

output → ①

LSTM (units = 2)

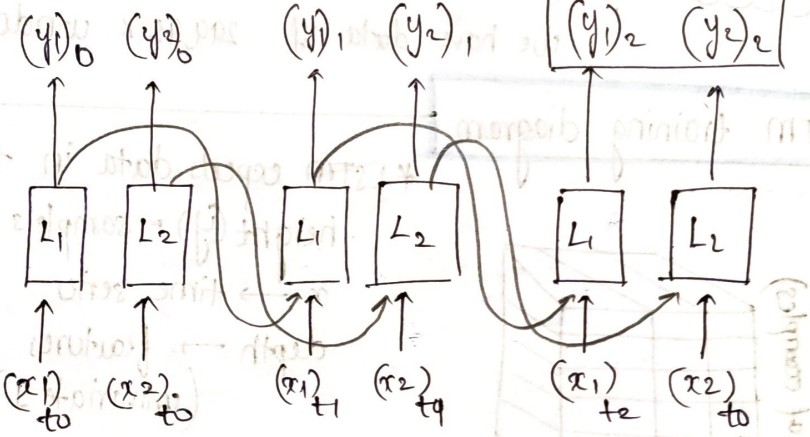
ip → (x0 x1 x2)

② → dim = no of units

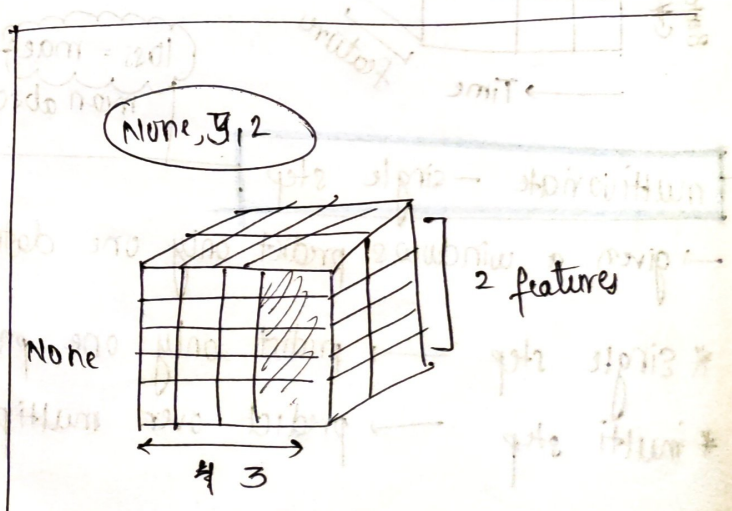


(None, 3, 2) → (None, 2)

~~None, [illegible]~~



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



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

21/3

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

(None, time_steps, units)

no of units

NO of time steps

return_states = true

shape = (3,1)

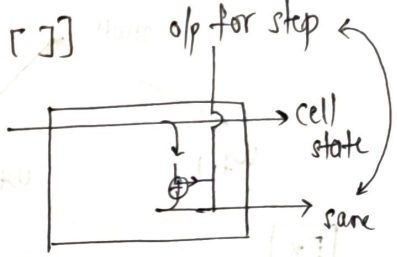
LSTM (1, ret_states) ⟹ lstm1, state_h, state_c → [[ ], [ ], [ ]] (3×5)

lstm_1 → LSTM → state_c, state_h

both

hidden state o/p for last time step

cell state for last time step



* LSTM (1, ret_states = true, ret_seq = true) → [lstm1, state_h, state_c] → [[O_1, O_2, O_3], [ ], [ ],]

i/p = (3,1)

o/p of all layers/time steps

o/p at last

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])
- 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

unit.

unit

* only ret_states → ret_states with ret_sequences

lstm_1 and state_h are same (None, units) (None, units)

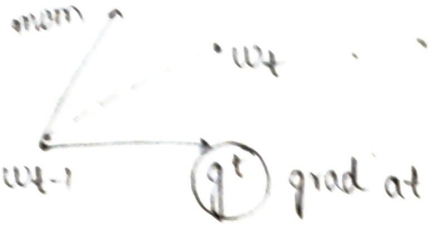
lstm_1 = (None, ts, units)

state_h = (None, units)

Nesterov Accelerated gradient

momentum = ewa of previous steps

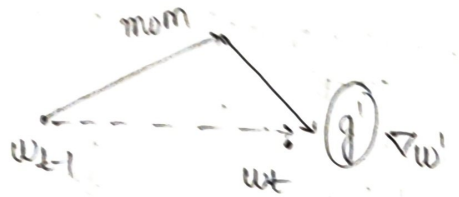
→ SGD + momentum = $w_t = w_{t-1} - [\gamma V_{t-1} + \eta g_t]$ ← gradient



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



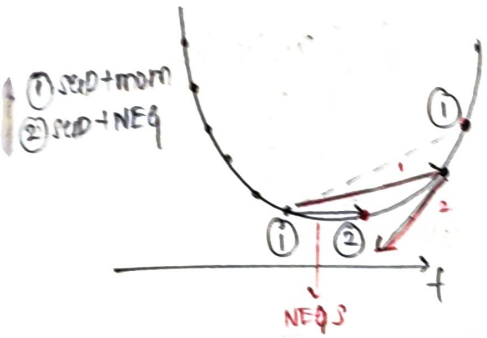
$(w_t)' = w_t - \gamma \cdot V_{t-1}$ → step1 : momentum

$V_t = \gamma V_{t-1} + \eta \cdot g'$

where,

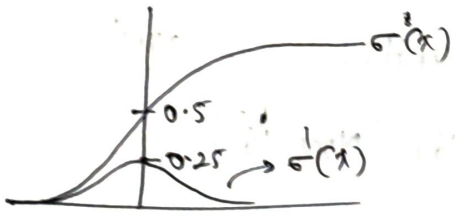
***** $g' = \nabla w_t'$ (grad at w_{t-1} + momentum

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

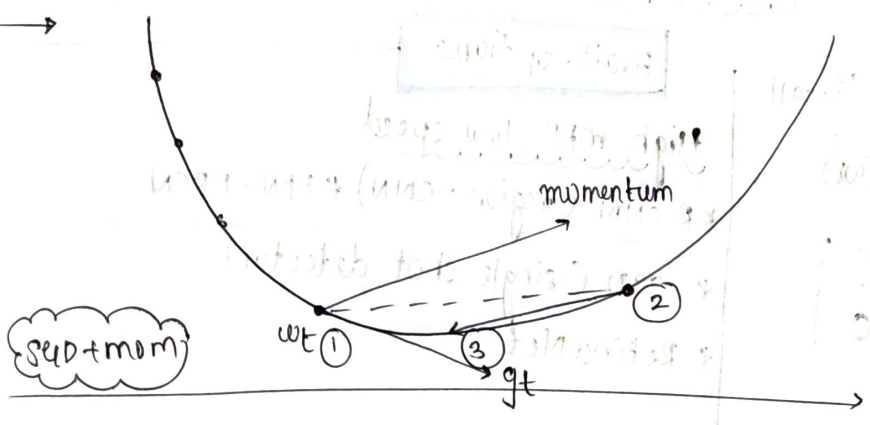
$$\frac{e^{-x}\cdot 1}{(1+e^{-x})(1+e^{-x})} \rightarrow \frac{(e^{-x}+1-1)\cdot 1}{(1+e^{-x})(1+e^{-x})} \rightarrow \left[\frac{1}{(1+e^{-x})}\right]\left[\frac{e^{-x}+1-1}{1+e^{-x}}\right]$$

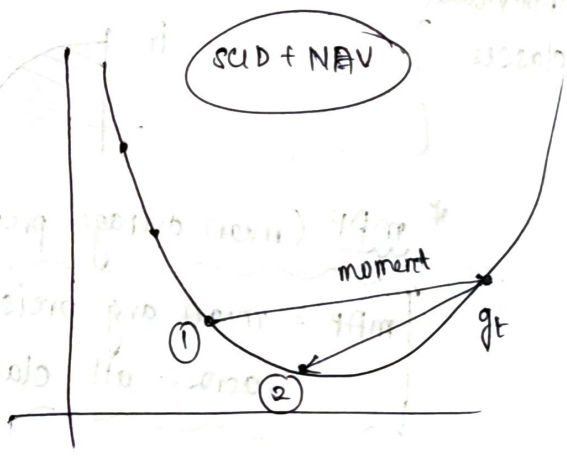
$\sigma(x)$

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

$$= \sigma(x)\left[1-\sigma(x)\right] \qquad \boxed{\therefore \sigma'(x) = \sigma(x)\cdot(1-\sigma(x))}$$

→





* at pnt w_t, v_t/momentum ↑ as we made lot of +ve increments previously

∴ due to this point overshoots to right side

↳ in case of NAG, we calculate ∇grad at momentum which is on right side ∴ it will be closer to minima and in negative direction

YOLO Algorithm

object detection

Object Detection

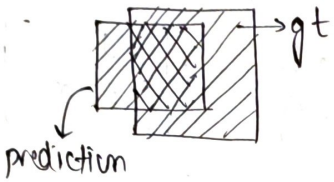
→ <x, y, w, h> → (x, y) — center, (w, h) — box width, h

* Focus on bounding boxes 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

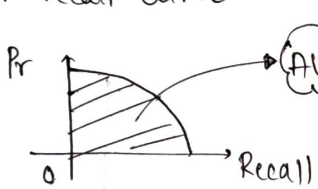
steps

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 (chair, person ...)
  * individual classes → calculate avg-precision ≡ area under pre-recall curve
  * individual classes → 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 (precision is imp) or OCR

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

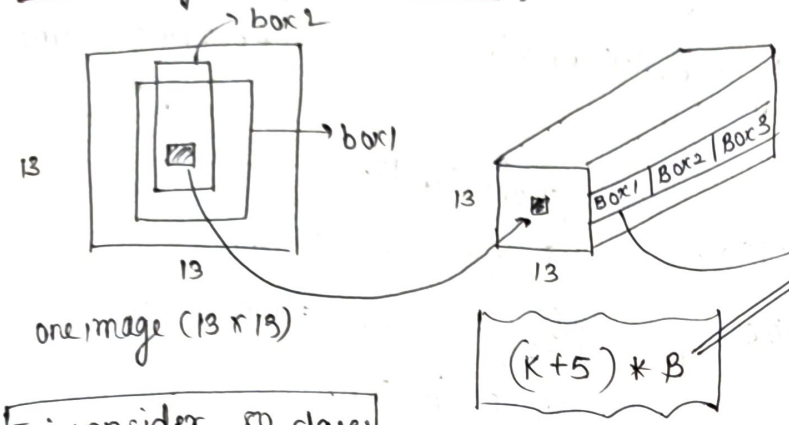
* BN ops is less (Billions of ops needed)
* BFLops/s is high (Bil of float op/sec).
* and FPS is high

② bounding boxes and output

(i/p) $416 \times 416 \times 3 \xrightarrow{\text{DNet}} 13 \times 13 \times 1024 \xrightarrow{\text{Box}} 13 \times 13 \times 425$ (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.r.t

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

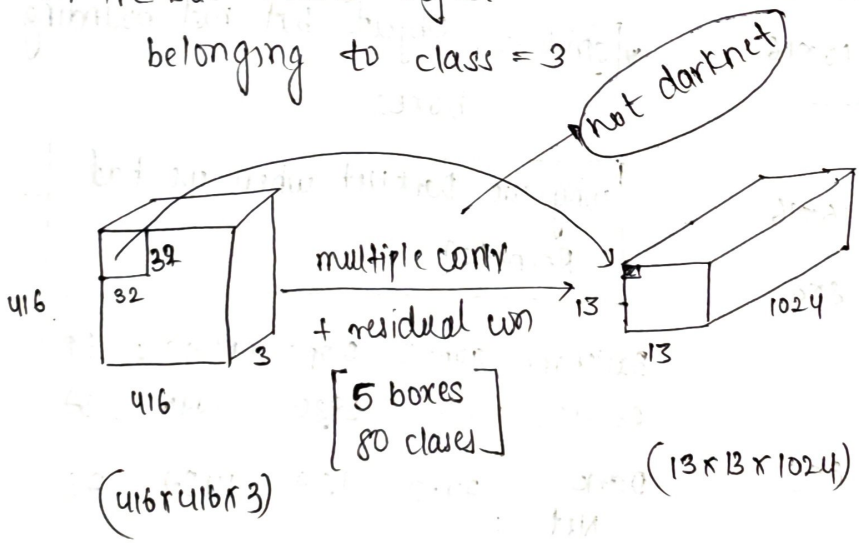
* box1 = $(P_o, x, y, w, h, C_1, \ldots C_{80})$

* 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)

[5 boxes, 80 classes]

(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

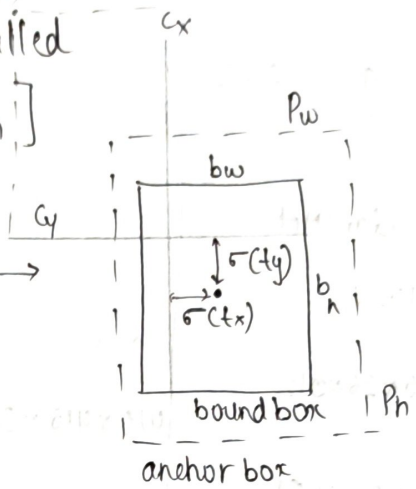
Anchor Boxes → prior boxes where you highly probable to find

* 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 — didn't work Because of heirarchy present

⇒ 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

$$\sum_{\text{all bounding boxes containing object in train data } y_{train}} \Big[\text{log-loss for } P_o + \text{log-loss for } (C_0 \cdots C_k) + \lambda_{co\text{-}ord} * \text{sq. loss for } t_x, t_y, \sqrt{t_w}, \sqrt{t_h} \Big] + \sum_{\text{all boxes in predicted which dont have obj (predicted boxes not present in } y_{train})} \lambda_{no\text{-}obj} * \text{log-loss for } (P_o)$$

4 Multiscale prediction

* 13×13 image feature can only detect large objects

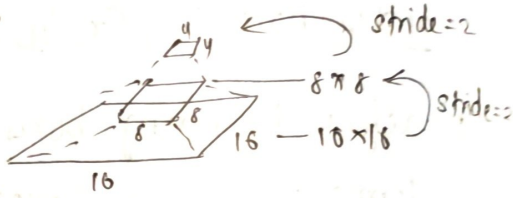
Idea

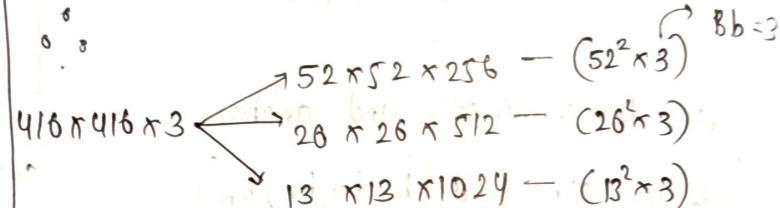
* predict with different grid sizes
  - How to get?

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



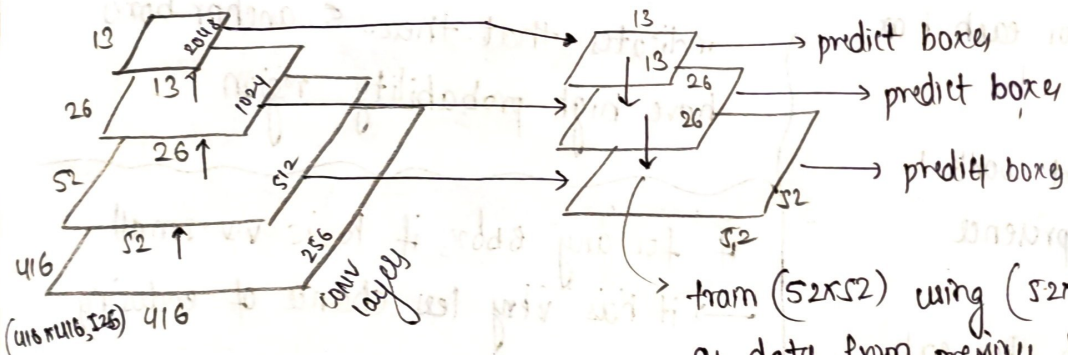


↓

10,647 Bboxes of different scales

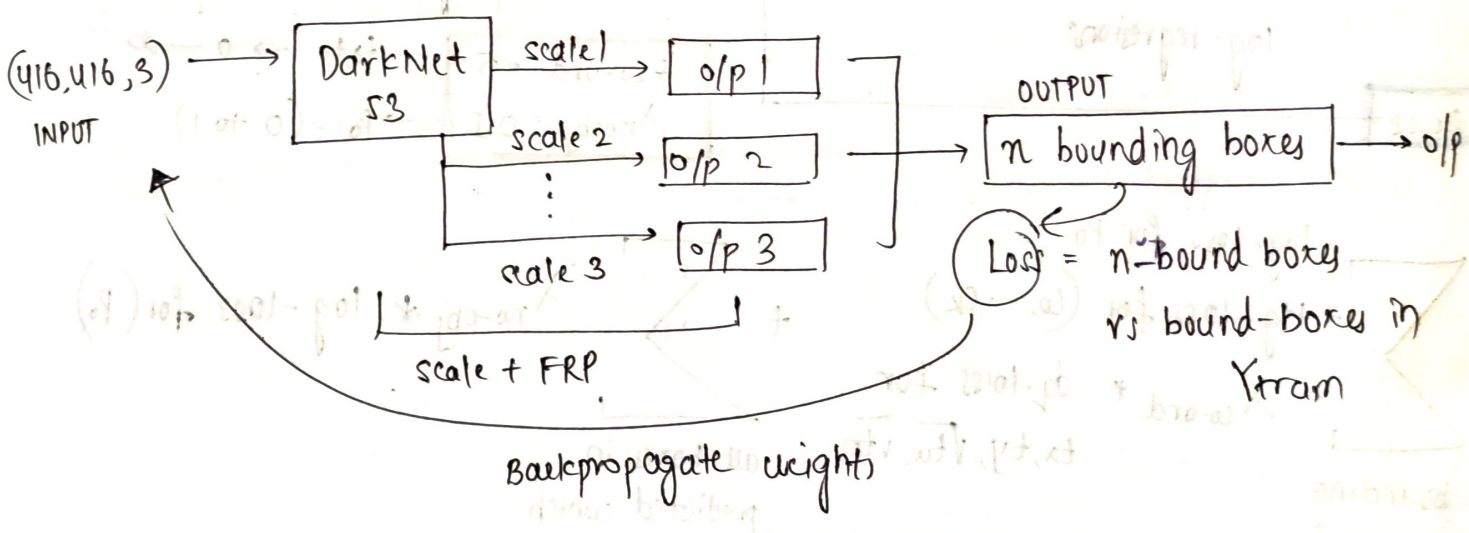
FPN — Feature pyramid networks

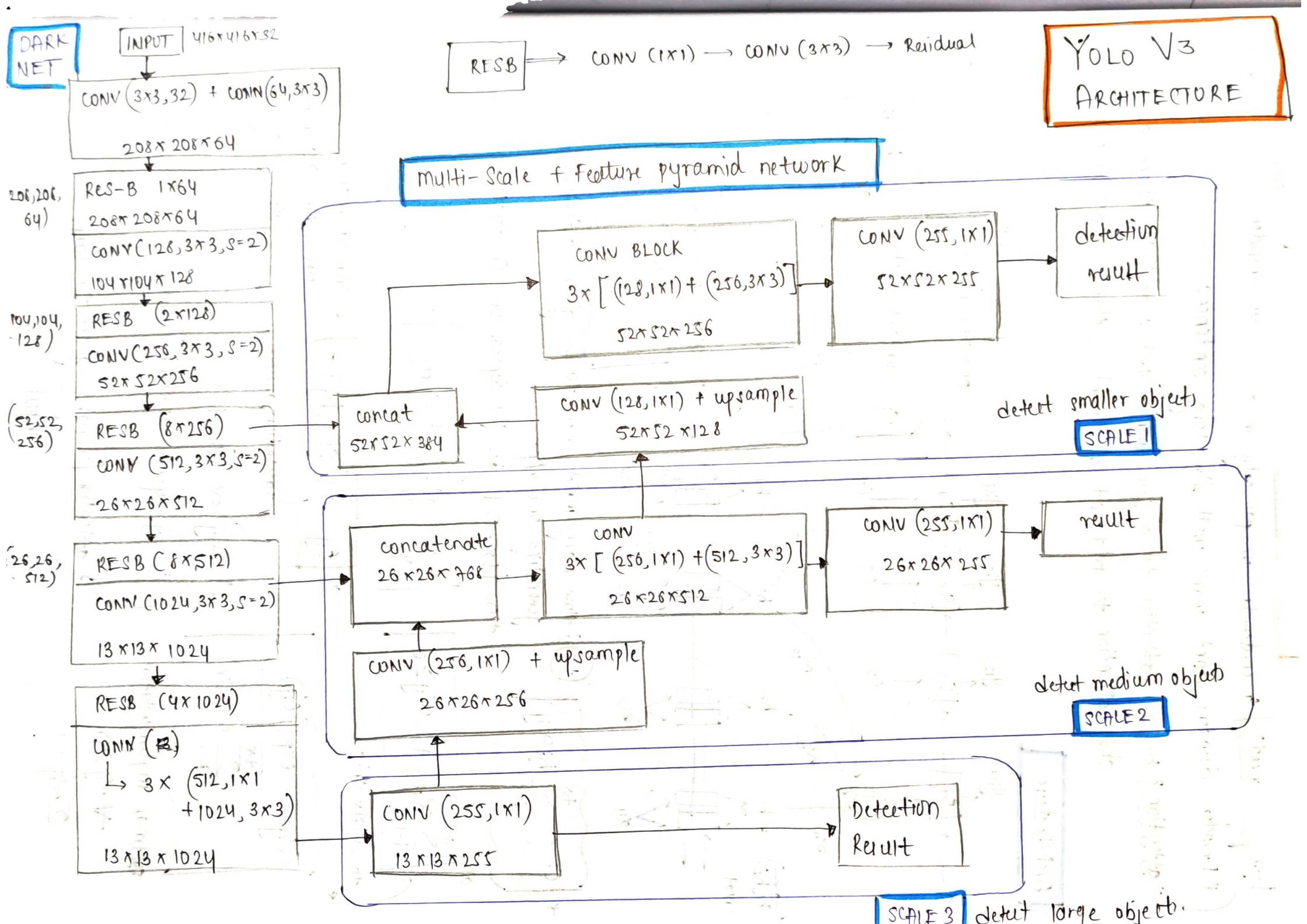
→ not only use different grid sizes to predict but also cross-grid information to use

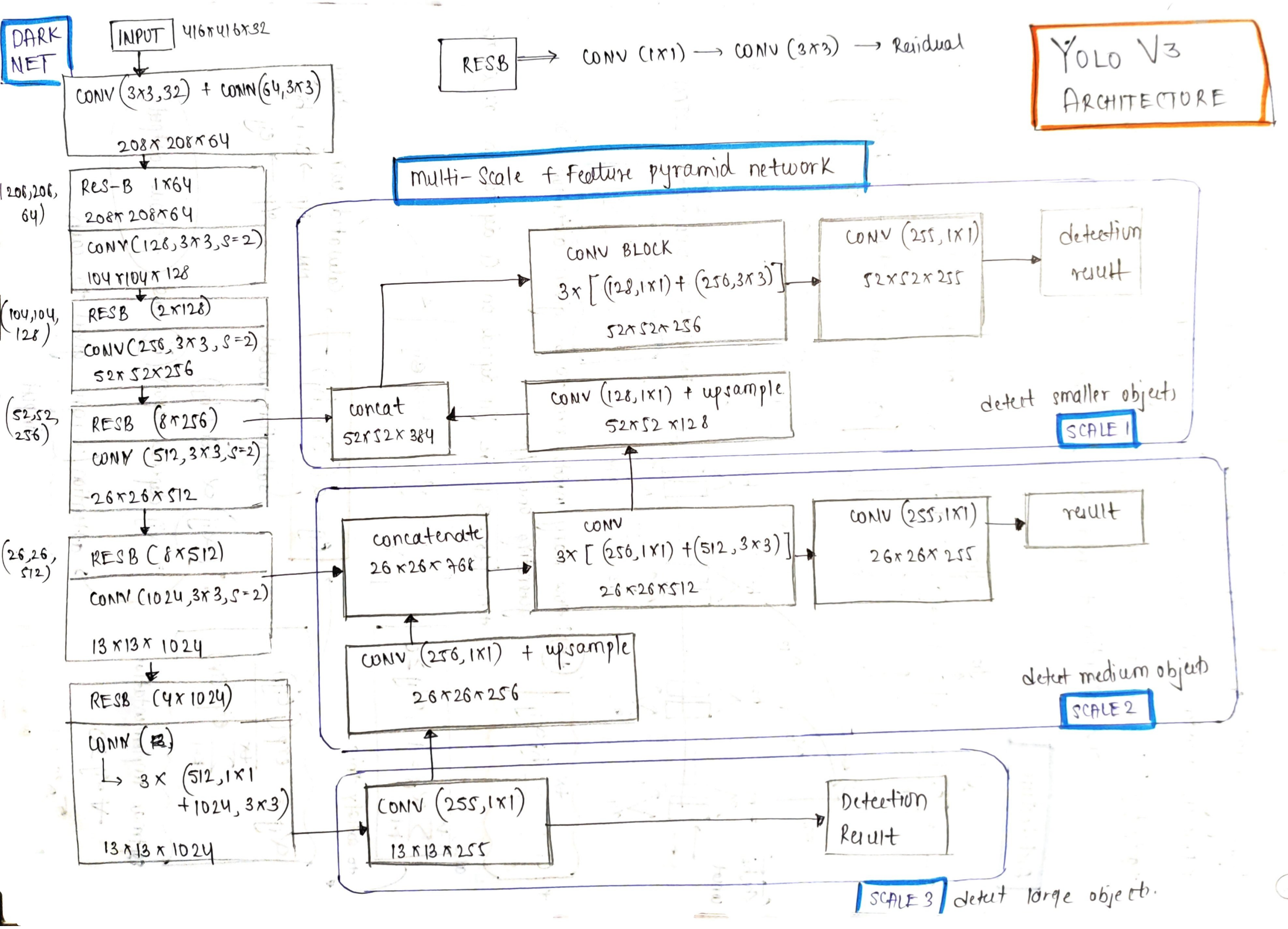


FPN + lateral connections

Workflow




DARK NET
INPUT 416x416x3
CONV (3x3, 32) + CONV (64, 3x3)
208 x 208 x 64
RESB → CONV (1x1) → CONV (3x3) → Residual
YOLO V3 ARCHITECTURE
Multi-Scale + Feature pyramid network
RES-B 1x64
208x208x64
CONV (128, 3x3, S=2)
104x104x128
(208, 208, 64)
RESB (2x128)
CONV (256, 3x3, S=2)
52x52x256
(104, 104, 128)
RESB (8x256)
CONV (512, 3x3, S=2)
26x26x512
(52, 52, 256)
RESB (8x512)
CONV (1024, 3x3, S=2)
13x13x1024
(26, 26, 512)
RESB (4x1024)
CONV (B)
3x (512, 1x1 + 1024, 3x3)
13x13x1024
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.


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

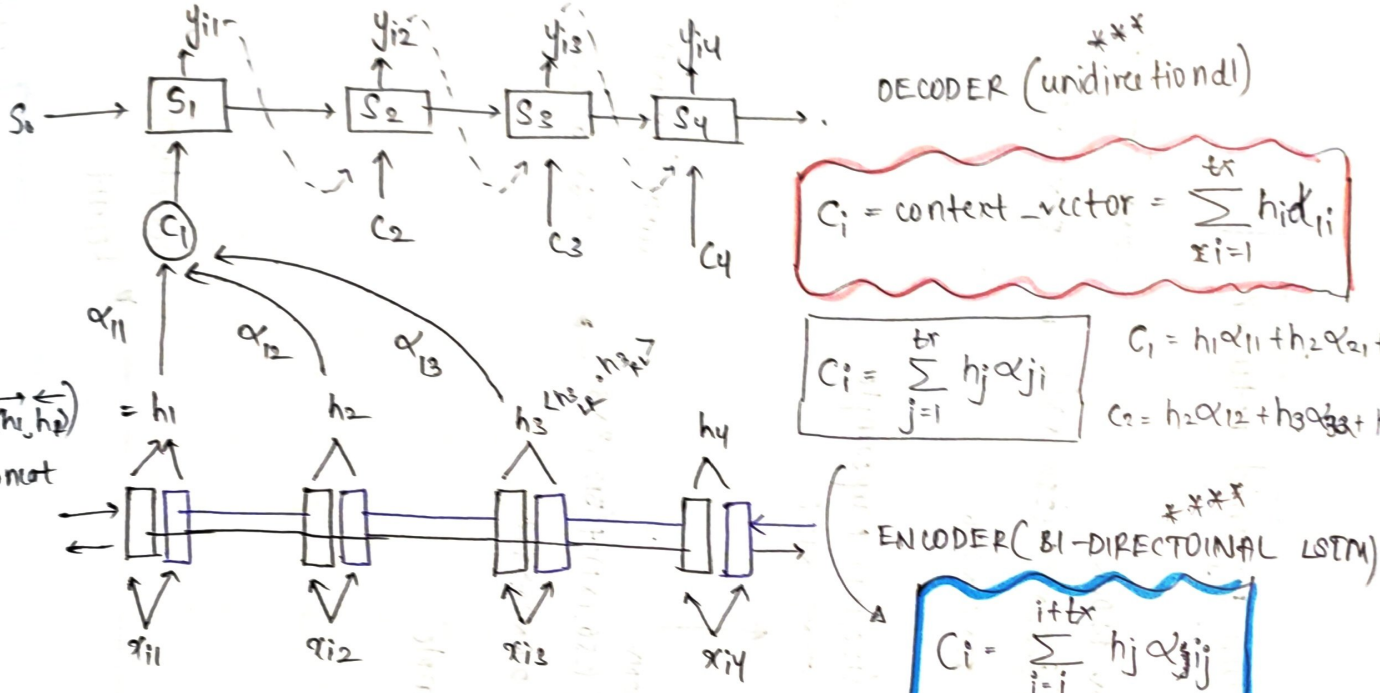
Attention Models

why you need? In en-decoders we pass only one c_i 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



$$c_i = \text{context\_vector} = \sum_{xi=1}^{tx} h_i \alpha_{ji}$$

$$c_i = \sum_{j=1}^{tx} h_j \alpha_{ji}$$

$c_1 = h_1\alpha_{11} + h_2\alpha_{21} + h_3\alpha_{31}$

$c_2 = h_2\alpha_{12} + h_3\alpha_{32} + h_4\alpha_{42}$

$$c_i = \sum_{j=i}^{i+tx} h_j \alpha_{ij}$$

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

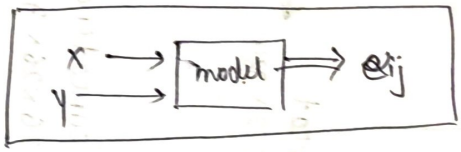
tx : how many inputs need to be given attention for generating one o/p.

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



$$e_{ij} = a(s_{i-1}, h_j)$$

attention-model

Calculated using 2-layer NN i/p → s_{i-1}, h_j o/p → e_{ij}

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

softmax implementation

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

$\sum \alpha_{ij} = 1$

$\sum \alpha_{iK} = 1$ all α corresponding to K

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

= α_{ij} depends upon h_j and previous output so

→ 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_1 is large than that particular word had more impact on prediction

Ex: french → english

	you	and	call → o/p
ti	0.9	0	1
ami	0.1	2	0
me	0	2	10

i/p

to predict w_3 = call me was most useful.

Image Captioning

i/p image → feature extraction using CNN → encoder

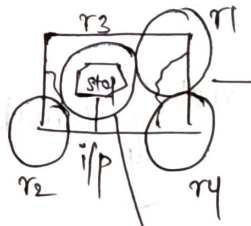
RNN with attention over image regions → decoder → LSTM → op

generate K d-dim vectors

each vector representing one region of image

(treat each vector as word i/p)

Visualize α_{ij}



A mountain road with stop sign

see highest α_{ij}

say highest is α_{23}

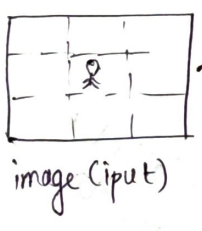
region (3) contributed most

tf.keras.Attention layer

→ inbuilt into keras

other blogs

Visualize



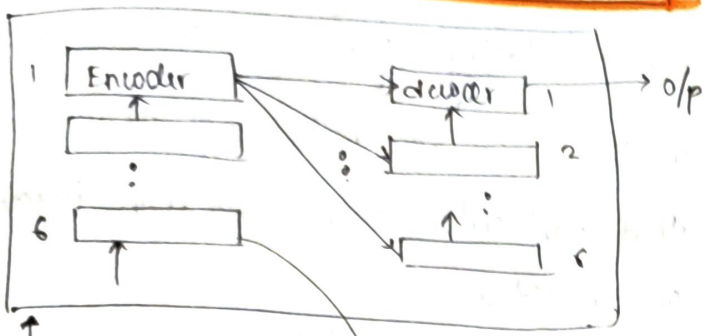
divide into 9 region vectors

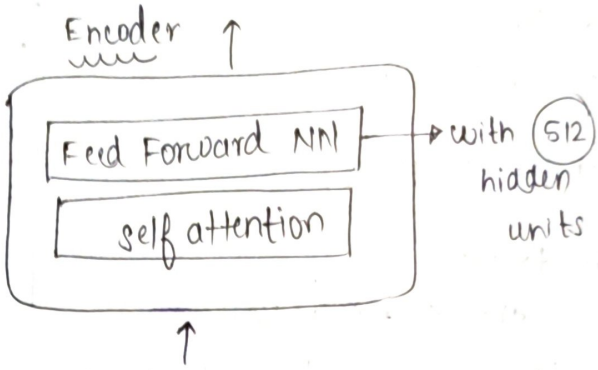
decoder with attentions → output word (man)

visualize α_{ij}s as feature importance

(region with face will have high α_{ij})

Transformers → Not RNN (! time based)





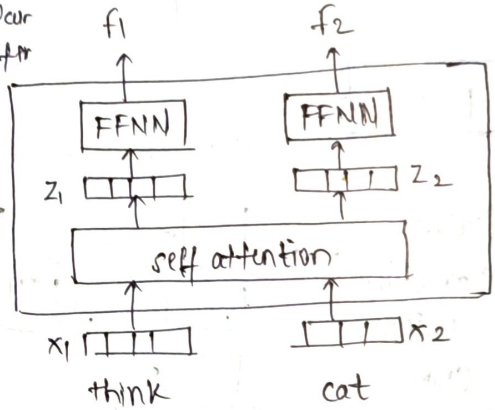
ENCODER

depends on all words

Self attention → attention :- what words determine Wcur
self-att :- what does Wcur connect/refer

the boy crossed the road as it was empty

what does it refer/connect to

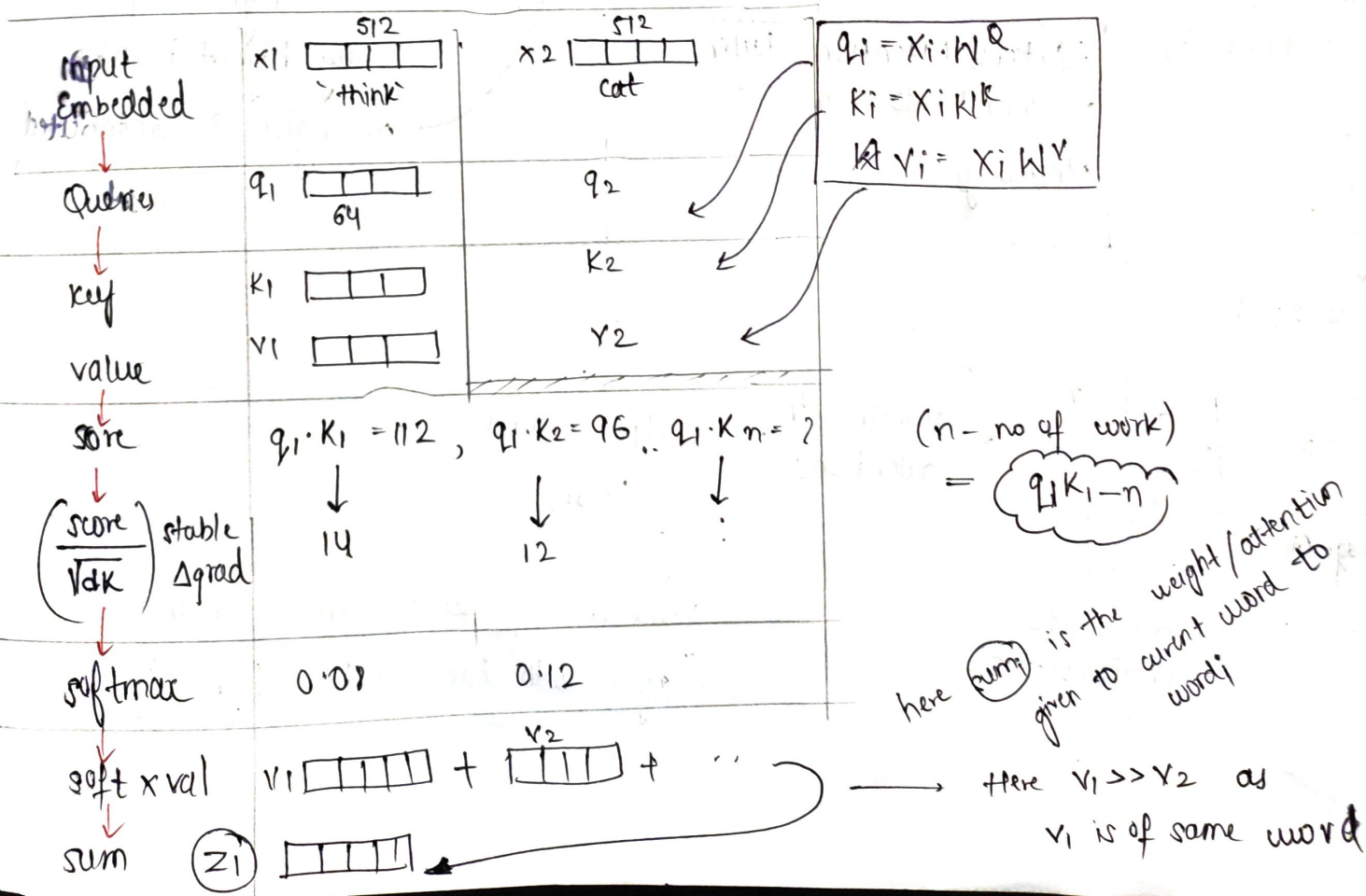


*inputs sent at once,
No time-based sending →

steps

hyper-params :

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



matrix computation

$X \times W^Q = Q$

$X \times W^K = K$

$X \times W^V = V$

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

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

One-headed :- only 1 set of matrices

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^V) = 8 (zis)

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

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

weight matrix = the dim is same as Z for one head (using W)

positional Embedding

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

input je seri would o/p

embedings + positional encoding

positional encoding: indicates position as well as distance b/w words

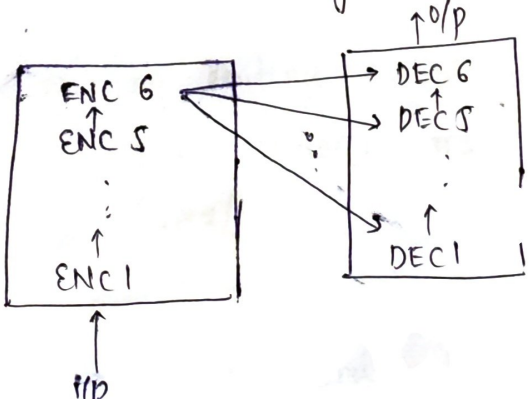
* pos vectors are static ie P_1, P_2, P_3 are precalculated

st. $P_1 > P_2 > P_3$ position

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

* original research paper

(Attention is all you need)

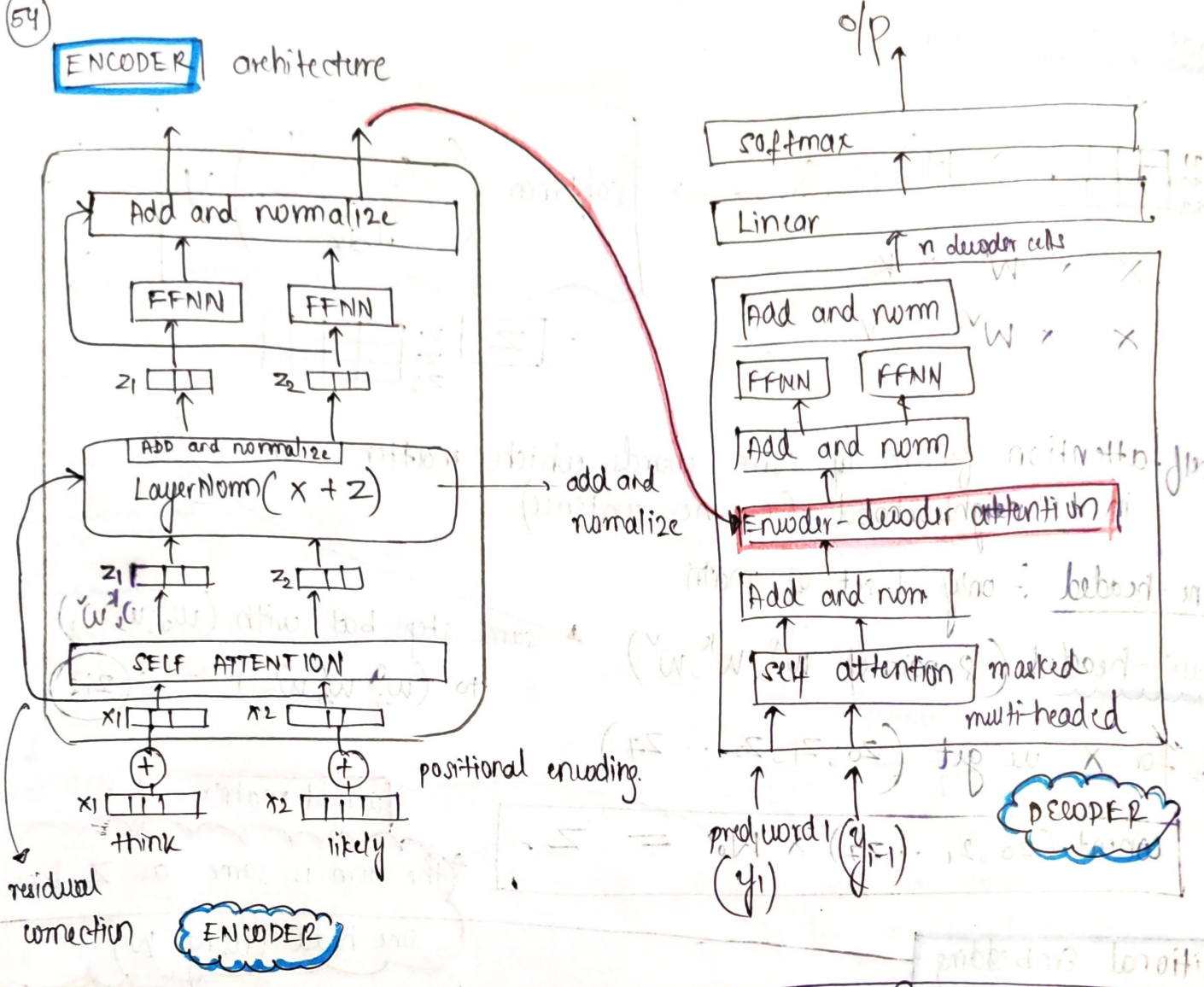


learnable matrices

① W_Q^{1-8} ② W_K^{1-8} ③ W_V^{1-8}

④ W

ENCODER architecture



Train phase

① pass whole sentence to encoders, you get a output

② pass it to decoder and predict first word

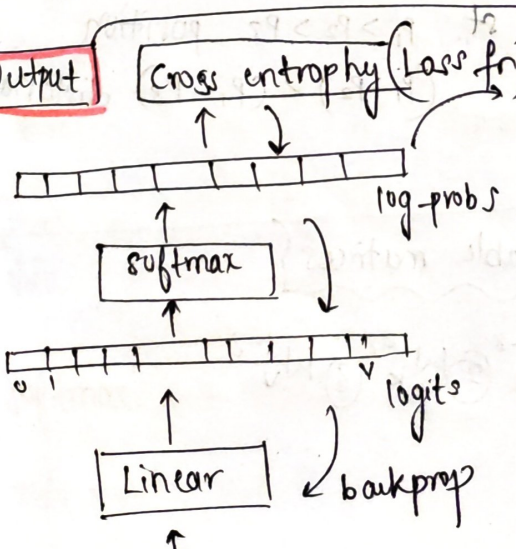
encoder → pass once
decoder → time based

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

Output



size - vocab-size

output word = vocab [argmax (op)]

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

Bidirectional encoder representation from transformer

sentence 2 vector

* BERT only has Encoder block called Transformer block.

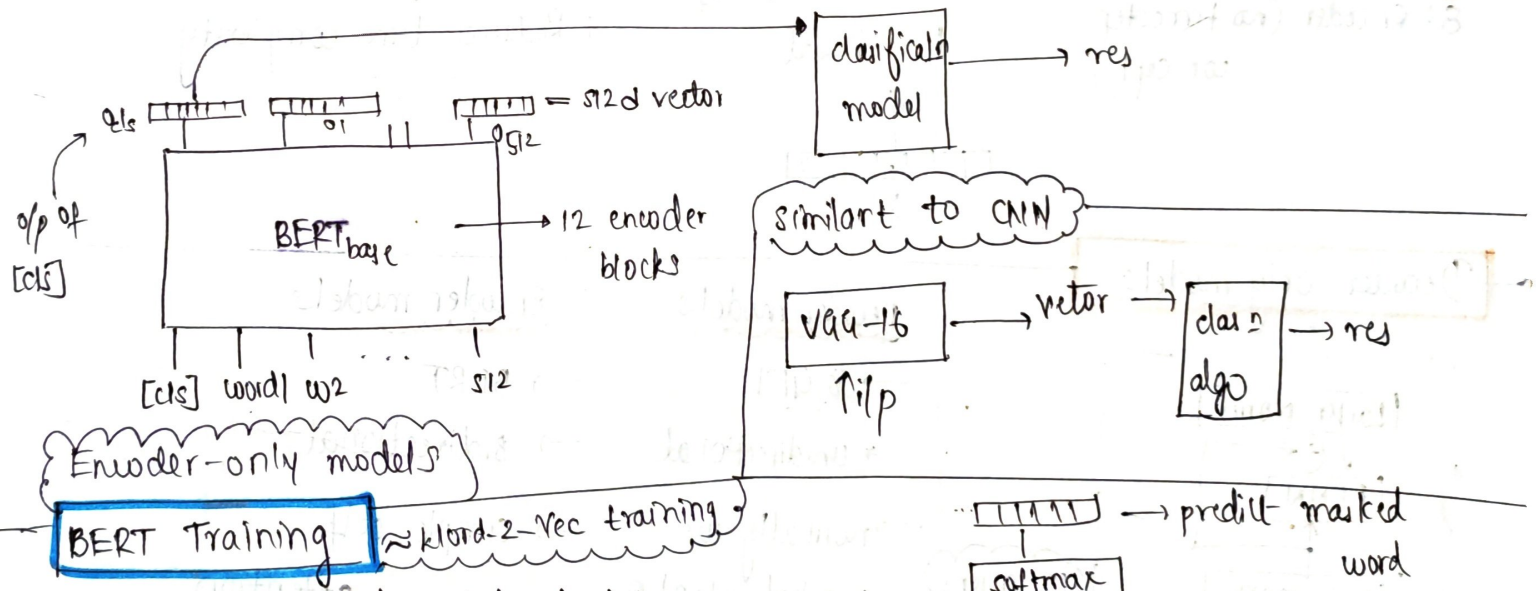
→ most useful as **Transfer Learning**

Base — FFN (768 units) + 12 attention heads

Large — FFN (1024 units) + 16 attention heads

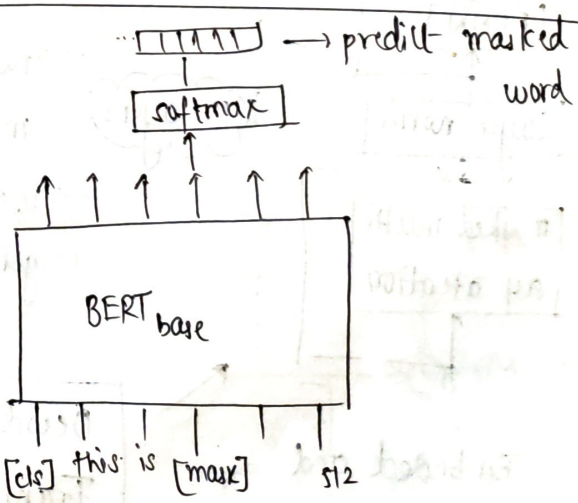
Transfer learning (spam or not)

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



Encoder-only models

BERT Training ≈ Word-2-Vec training

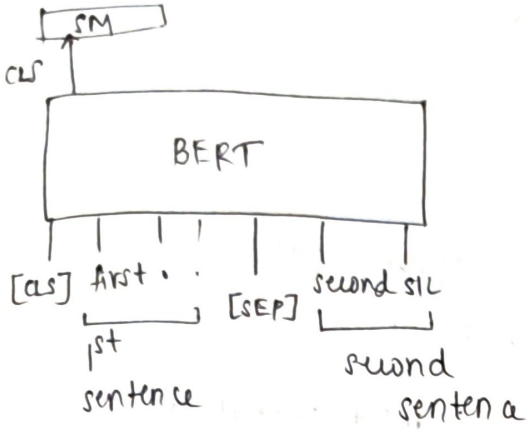


* Bidirectional models, while training look @ words before and after.

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

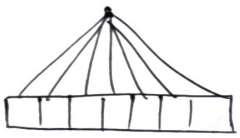


BERT training

NWP = next word prediction

Attention types

① Self attention

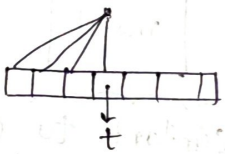


* depends on all words

Ex:- Encoder (no timestep concept)

② Masked attention

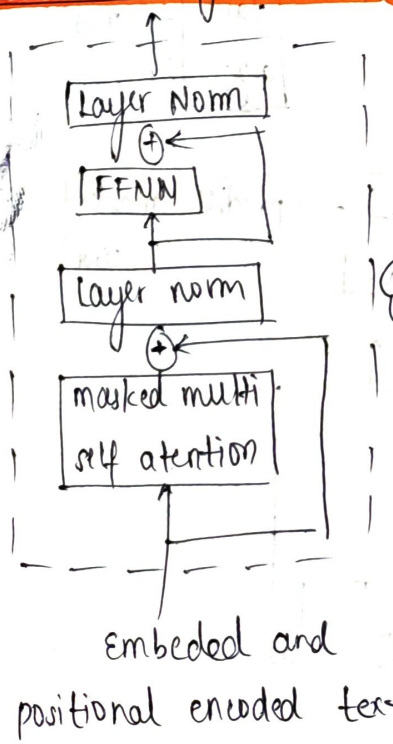
* at timestep (ti), self attention calculated on $t_0 \cdots t_{i-1}$
* used in decoders as time based



③ Sparse self attention

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

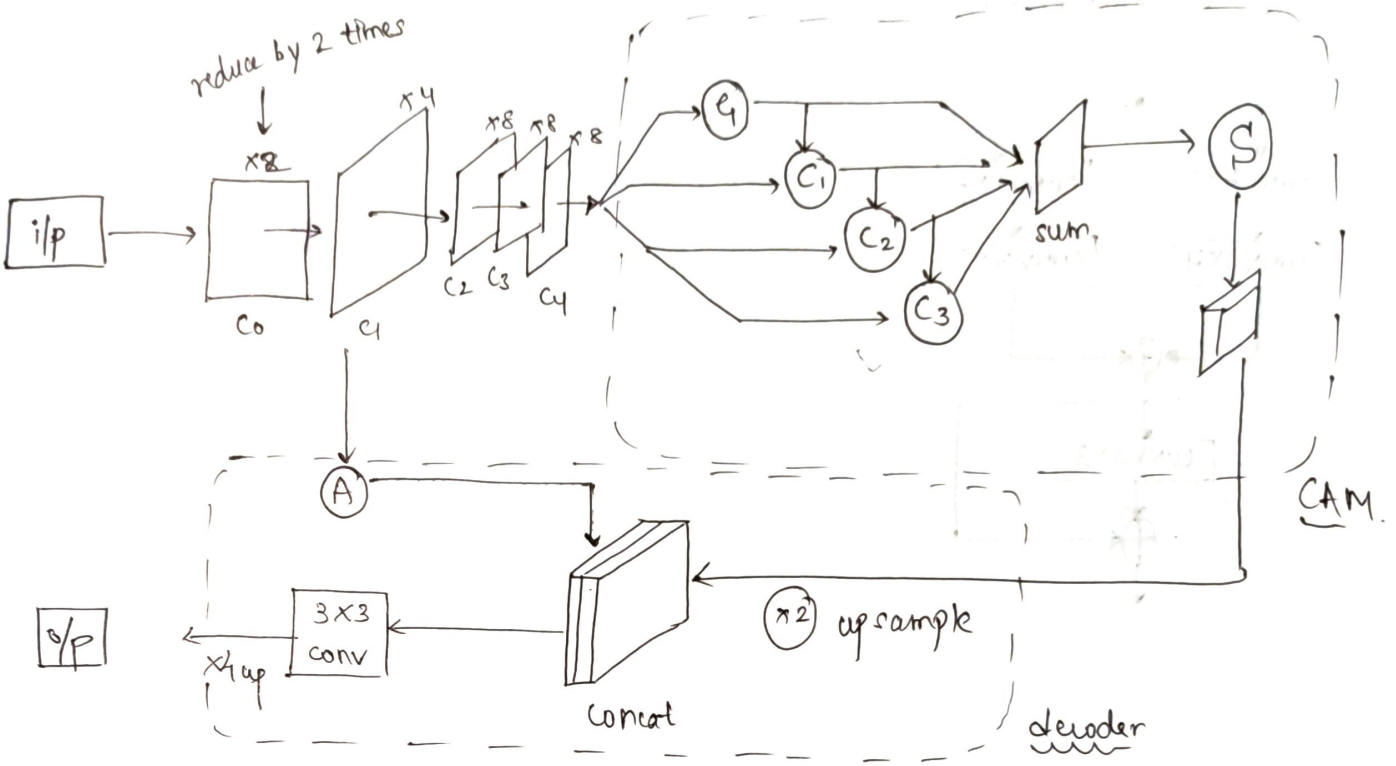
Decoder only models



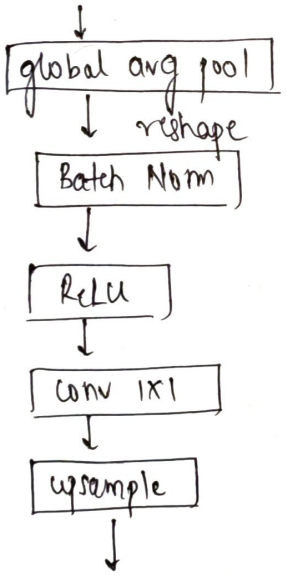
Decoder models	Encoder models
→ GPT	→ BERT
→ unidirectional	→ Bidirectional
→ normally use masked atent n	→ simple self attention
→ used more in generative tasks	→ used mostly for text representation / embedding tasks

Decoder only Transformers ≡ GPT-1

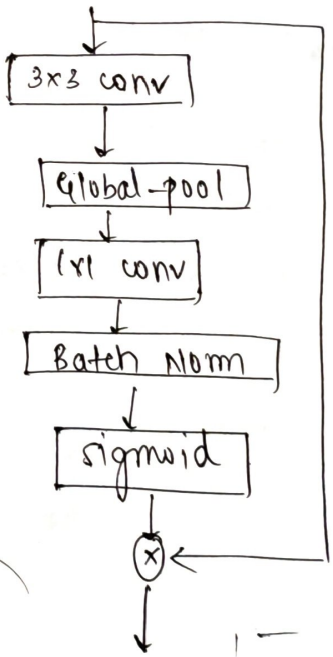
CANET — Image segmentation



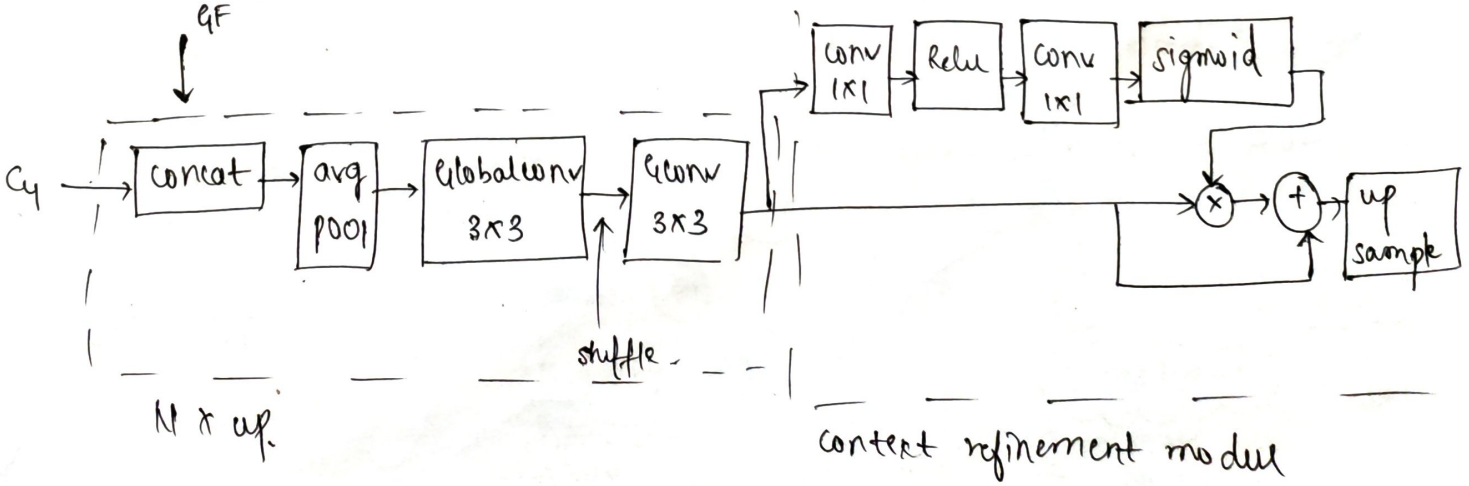
① (G) Global flow



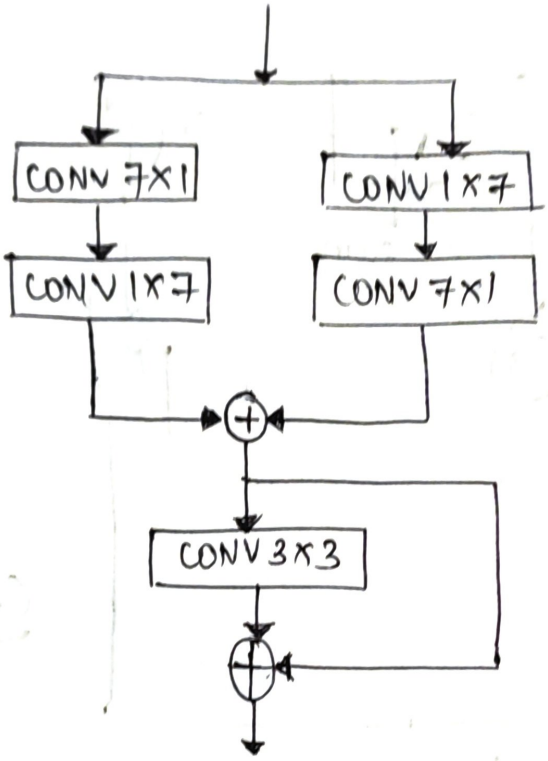
③ (S) Feature selection module



② (C) Context fusion module



AGCN Adapted global convolution n/w

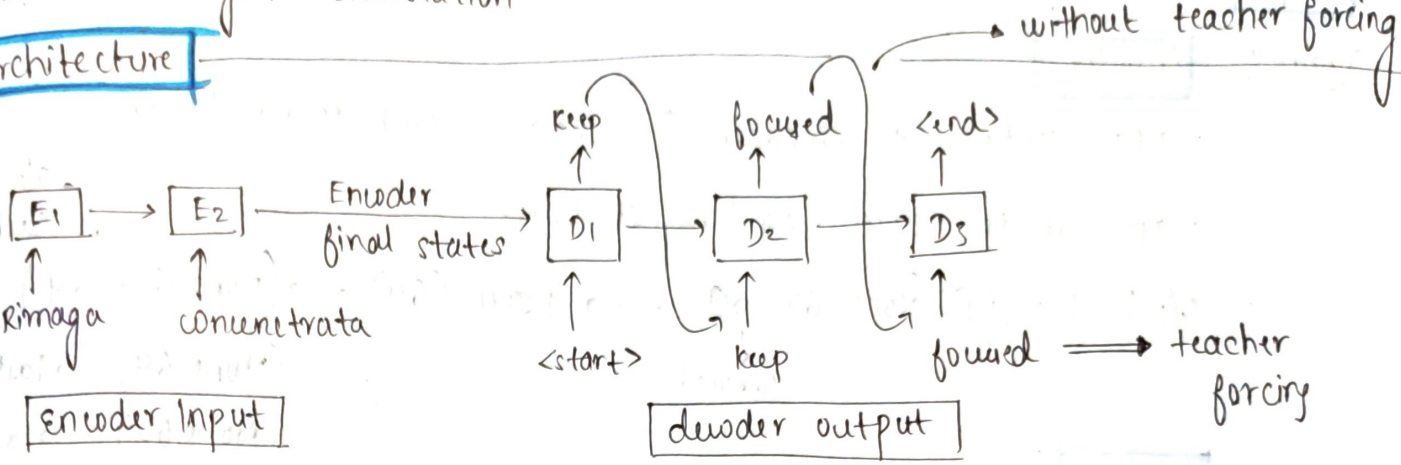


Sequence - 2 - Sequence

Use case

→ Italian to English translation

Architecture



Data

(X): rimag concentro ⟶ Encoder iput

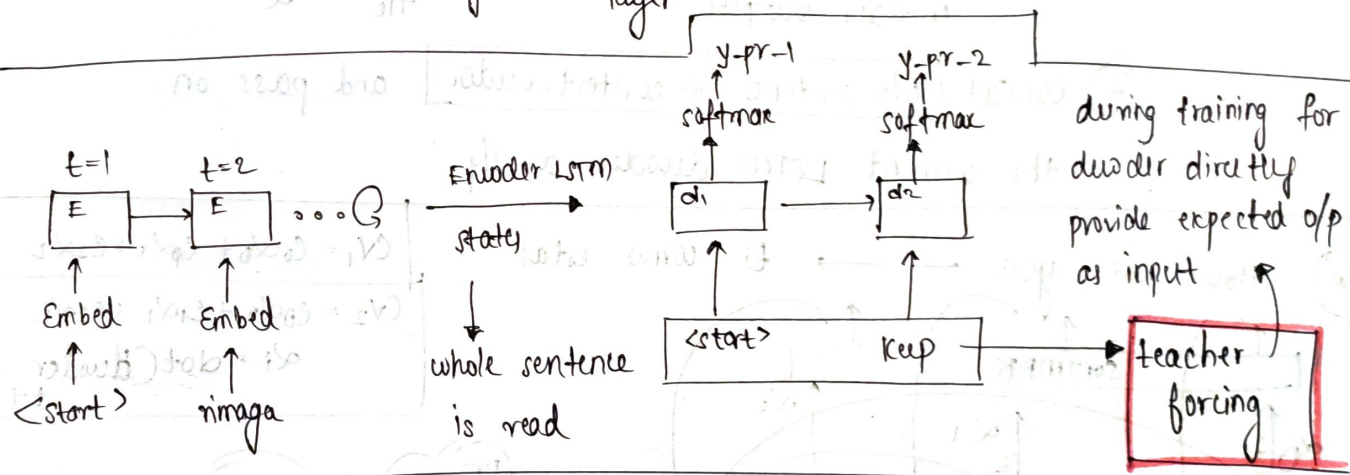
(y) keep focused ⟶ decoder i/p : <start> keep focused

decoder o/p : keep focused <end>

↳ apply tokenizer to convert text into integers.

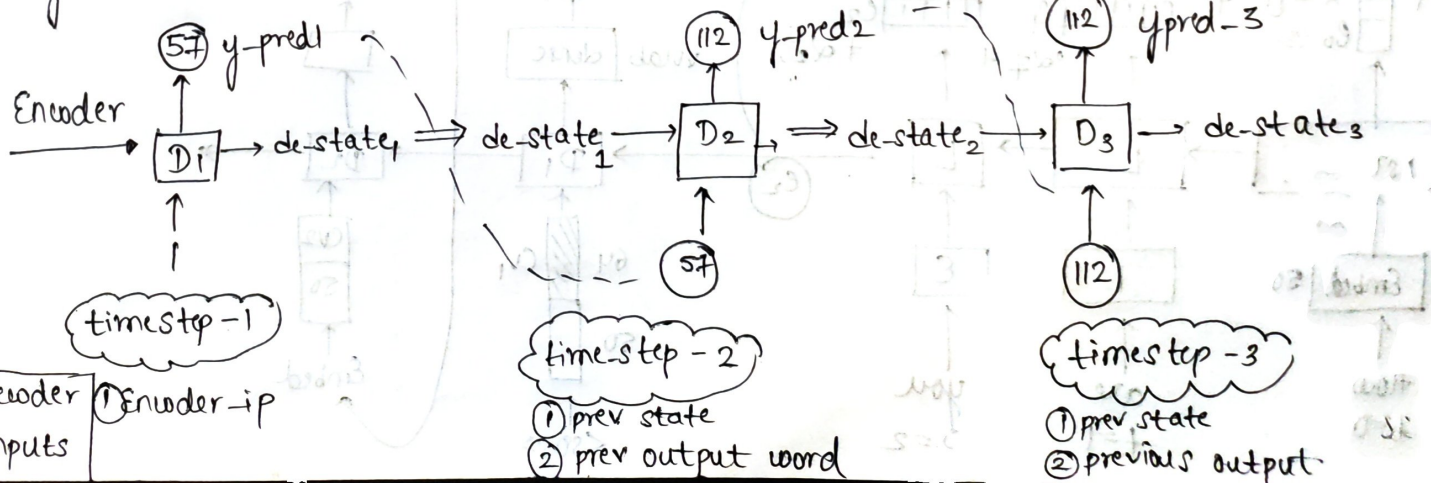
↳ (batch size, sentence_length) —Embeding layer→ (batch_size, sentence_len, output_dim)

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, 69) → 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

Here tx = input_length (all sentence)

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

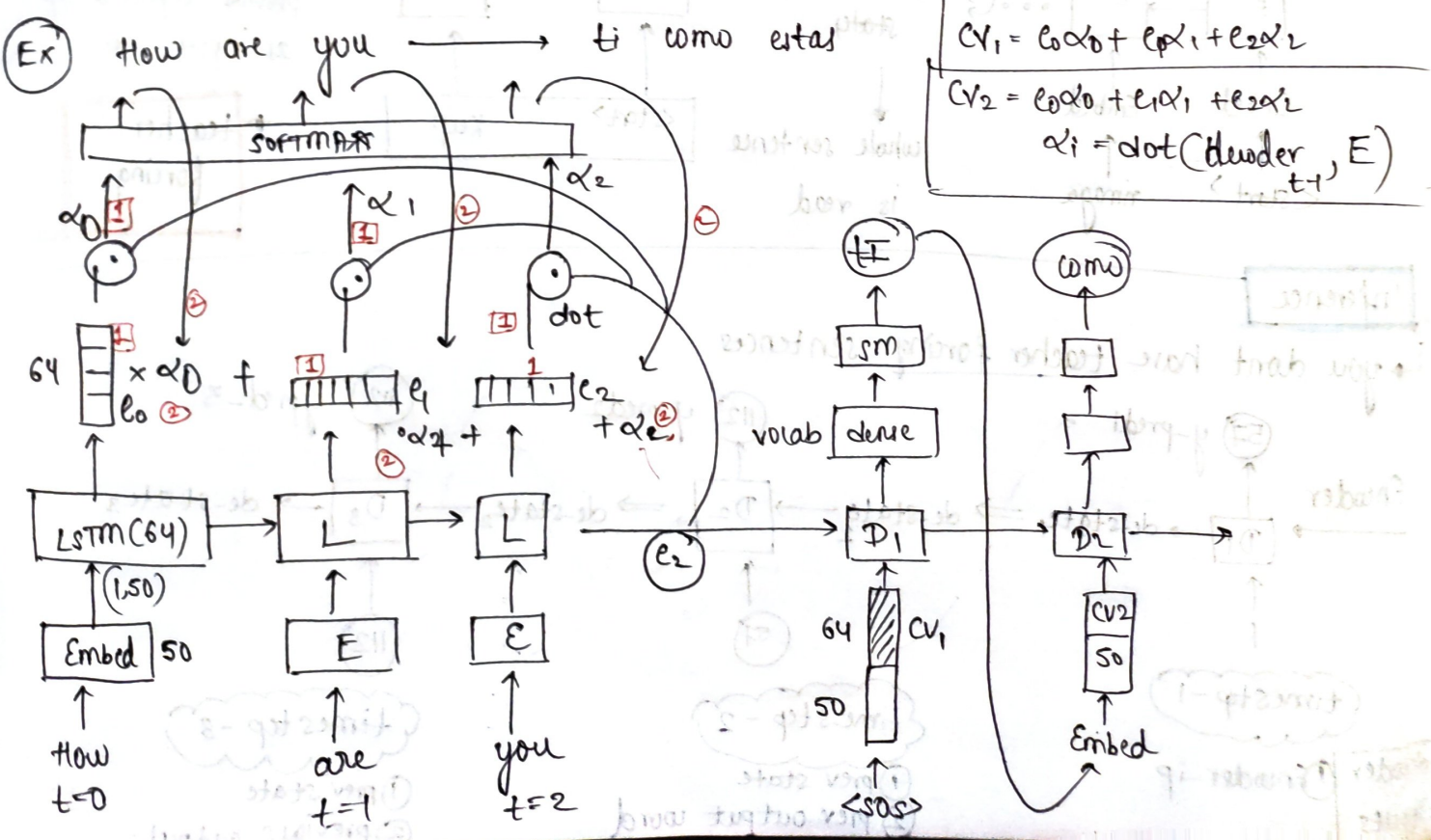
---

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 → He
(64, 1) → last h-s
(64, 1) → last c-s

decoder

LSTM = 64

at time = 1

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

Now for first we last h-s of encoder

He (h_{t-1}) = (600, 64) · (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)

He (W) · h_{t-1}
= (600, 64)(64, 128)(128, 1)
= (600, 1)

→ learned by backprop

overal

inp → encoder → (calculate α_{ij}s) → decoder → output

attention weight of each word

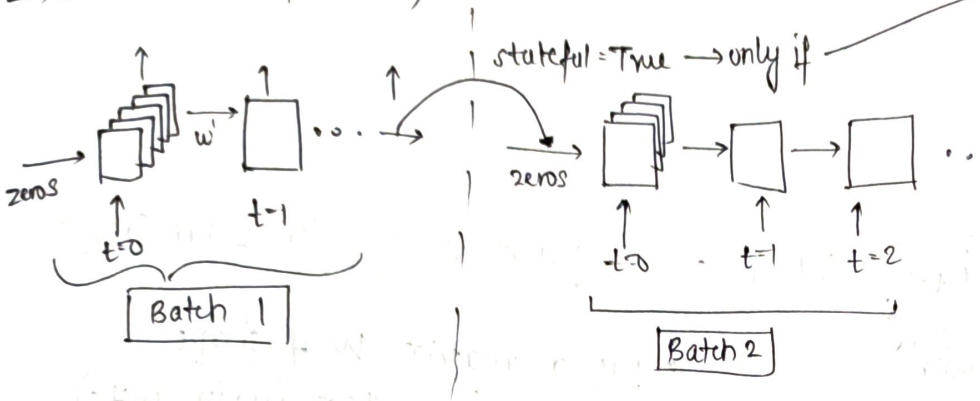
char-RNN concepts

Stateful LSTM → used when batches are sequential

row-wise or continuity

Batches must be sequential

→ LSTM(5, stateful=True)



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

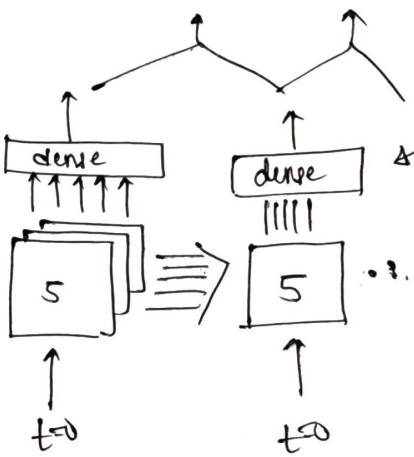
imp

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_sequence = True)



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)

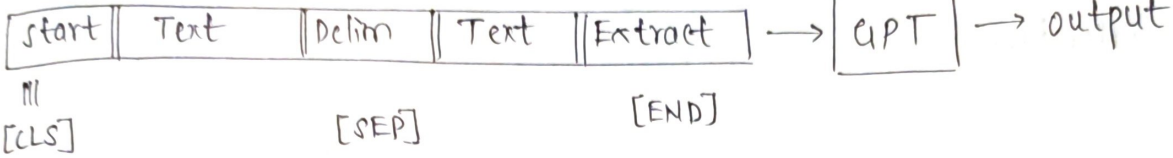
Generative Pre Training

GPT - 1

Architecture : ~~tro~~ Decoder only transfomer
↳ 12 decoder calls (pg 56)

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

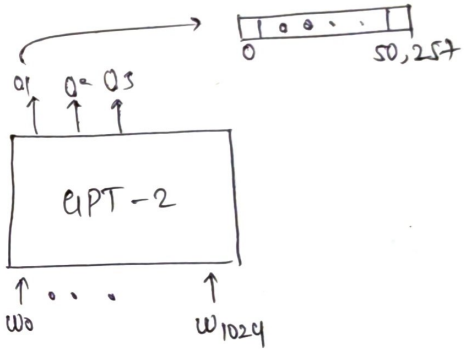
Tokens



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 modul (NWP)
  ② Fine tune model by giving your dataset
  ③ make predictions

  NO fine tune (Pretrain and directly predict)

→ few-shot learning
① → ② → ③

disadvantages

performs poorly on rare or specialized content

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

LIME

Local Interpretable model Agnostic Explainations

Intuition

* Data = {x, y}
* classifier (model) = f(x)

∴ x → f(x) → y

↓ may not be interpretable (BERT 512)

local interpretable

→ considers only a local region to give interpretability
→ not global (only focus on locality)

Algorithm

Step 1:- given x, calculate x'

$x \in R^d \rightarrow f(x) \rightarrow y$

↓

$x' \in \{0,1\}^d$

* x' is interpretable
* x' is a binary vetor (each ele is 0/1 indicating presence or absence).
* dimensions of x and x' may vary

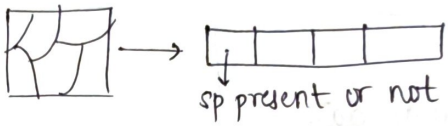
$x \rightarrow x'$

Text :- use BOW

sentence → BERT → x (512 dim)

→ BOW → x'

Images :- divide image into super pixels



Tabular data :
→ ① Categories ← OHE
② continous → binning values.

Step 2:- Surrogate model (g)

g ∈ G (models which are easily interpretable Ex:- LR, DT)

→ Ω(g) = complexity of g (depth, # non-zero weights)

(g) is trained on (x') and not x

g(x') not g(x)

Proximity

$\pi_x(z)$ → proximity b/w (x) and (z)

π_x → locality of x

local fidelity

$L(f, g, \pi_x)$ → how best g(x') approximates f(x) in locality given by π_x

↓ better, f and g are similar
↑ bad

optimization f:

$$\xi(x) = \min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

G: Linear models / DT

L: squared Loss

$$\pi_x = \exp\left(\frac{-D(x,z)^2}{\sigma^2}\right)$$ ← Exponential kernel

z | $L(f, g, \pi_x) = \sum_{z,z'} \pi_x(z)\,(f(z) - g(z'))^2$ | $z, z' \in Z$

* weighted according to proximity

1 → ∀ each p^1, z find z'

↓

find locality of x_q → (π_x)

optimisation :- $\xi(x) = \min \{ L(f, g, \pi_x) + \Omega(g) \}$ → K-Lasso

K-lasso :- L1 regularized linear regression with only top-K features

Ex: S-lasso :- L1 regularisation keeps only (3) features and makes others 0.
thus we get only 3 features used for interpretability.

Advantages

* works on image, text, tabular data
* x may not be same as x'

Drawbacks

* deciding neighborhood $-\pi_x(z)$
  Hyper param = σ
  $D(x,z)$ may be issue with high-dim data
* finding (x') from (x) may not be straight forward.
* unstable
  Similar/closer points may give different interpretability
* Lack of robustness

Research paper algo

Algo: Sparse linear explanations using LIME
Ip : $f(x)$, N, $x \rightarrow x'$, π_x (similarity kernel)
K (length of explanations)

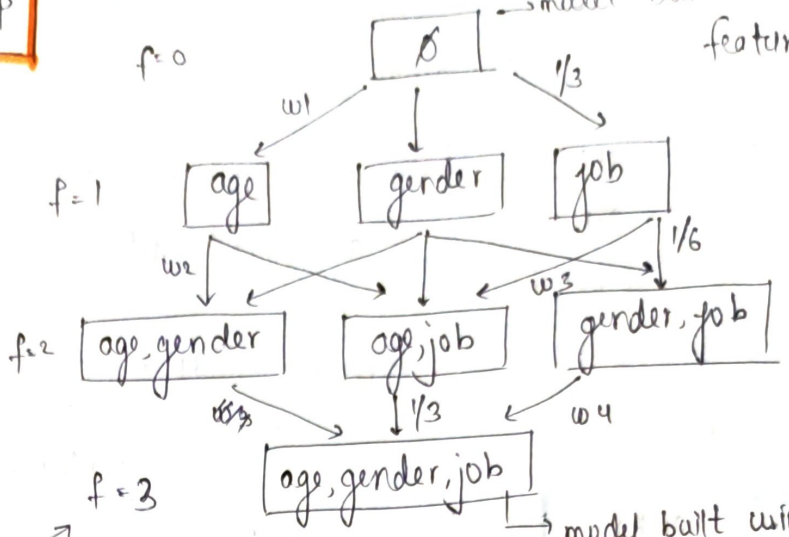
algo :- $Z \leftarrow \{\}$
for $i \in \{1, 2 \ldots N\}$ do
$z_i' \leftarrow$ sample-around (x')
$Z \leftarrow Z \cup \langle z_i', f(z_i), \pi_x(z_i) \rangle$
end for
$w \leftarrow$ K-lasso (Z, K) with z' features
$f(z)$ as target.
return w

SHAP

Shapely Additive explanations

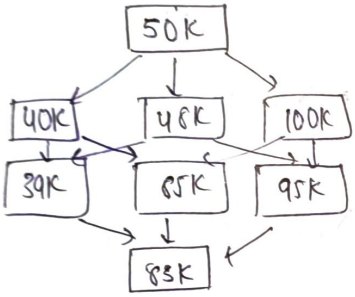
* Model : Predict salary
features : age, gender, job
output : y_i

x_0 : query point, $\bar{y} = \Sigma y_i$
$\hat{y}_0$ pred value



① for (d) feature, train 2^d models

② pass (x_0) and find predictions for each model



$\hat{y}_0 = 83K$, $\bar{y} = 50K$

Marginal Contribution

$MC_{Age\{Age\}}(x_0) = -10k$ → contribution of age when we have only age feature compared to model with {age, gender} feature

$MC_{Age\{Age,gender\}}(x_0) = -9k$ → output(age, gender) − output(age)

39 − 48 = −9K

③ Shapely values of each feature

$$SHAP(age)_{(x_0)} = w_1 \cdot MC_{age,\{age\}} + w_2 \cdot MC_{age,\{age,gender\}} + w_3 \cdot MC_{"\{age,job\}} + w_4 \cdot MC_{\{age,gender,job\}}$$

→ No nodes at level = FC_f [$F = \{f_1 \cdots f_n\}$]
No of edges = $f : FC_f$
↳ no of features selected

$$SHAP(feature)_{x_0} = \sum_{set | feature \in set} [edge\text{-}weight(pred(x)_{set} - pred(x)_{set/feat})]$$

$$edge\text{-}wt = \left(|set| \; FC_{|set|}\right)^{-1}$$

$pred(x)_{set/feature}$ ⟹ model trained using features = |set| − feature (f_i)

property

$w_1 = w_2 + w_3 = w_4$

① at a level all (w_is) are same

② all weights at level are same

→ $SHAP(age)[x_0] = -11.33K\$$

∴ In finding $x_0 \rightarrow y_0$, contribution of age is −11.33K.

Understanding

$x_0 \longrightarrow \hat{y}_0$

ϕ = 50K$

$[age, gender, job] = 83K$

SHAP(age) = S_1 = -11.33K

job = S_2 = -2.33K

gender = S_3 = 46.60K

↳ our model predicted $\hat{y}_0$ which was 33K away/different from $\phi / \bar{y}$

⇓

difference = 33K = (-11.33K) + (-2.33K) + (46.60K)

↓

contrib of age

Paper — actual SHAP

* convert x to interpretable repr

$x \longrightarrow x' \in \{0,1\}^M$ (BoW)

* train surrogate model

$g(x') \approx f(hm(x))$

① sum of shapely values

$= \hat{y}_F - \hat{y}_\phi$

$\hat{y}_F$ ↓ all feature; $\hat{y}_\phi$ ↘ no features

Additive feature attribution

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z_i'$$

M = # features

$g(z)$ = surrogate fn

ϕ_i = shapely value for feature i

$z_i' \rightarrow$ feature ① is present or not

Expectations or average value of RV

$$E(X) = \sum_{x \in X} x P(X=x) \quad \text{(PMF)} \qquad \text{or} \qquad \int_x x P(x)\,dx \quad \text{(PDF)}$$

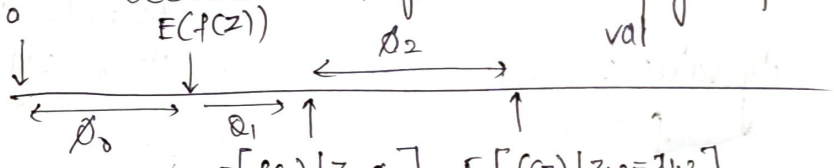
$$E(f(x)) = \sum_x f(x) \cdot P(X=x)$$

$$E(x \mid y=y_i) = \sum_x x P(x \mid y=y_i)$$

$$= \sum_{x \in X} x \cdot P(X=x \mid y=y_i)$$

rel^n with shapely values

shapely ≡ expectation val



↳ average/expectation given feature 1 is fixed

KERNEL SHAP

Kernel shap = LIME + shapely val + Shapely Kernel → and use FI

(In LIME we fit interpretable model at last, here instead we find shapely values)

$$\pi_x(z') = \frac{M-1}{{}^{M}C_{|z'|} \, |z'| \, (M - |z'|)}$$

SHAPELY KERNEL

M = # dim in x

$|z'|$ = len (z_0)

problems

computationally NP-hard

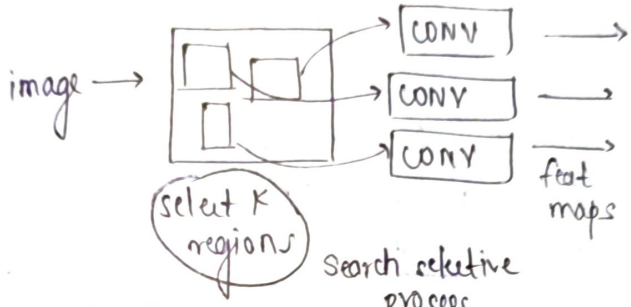
∴ use approximations

OBJECT DETECTIONS

Rigional CNN → detect regions in Imag

R-CNN

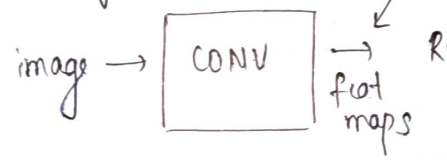
↳ slower and low FPS → CONV layer for each region



RCNN
Search selective regions → CONV on each

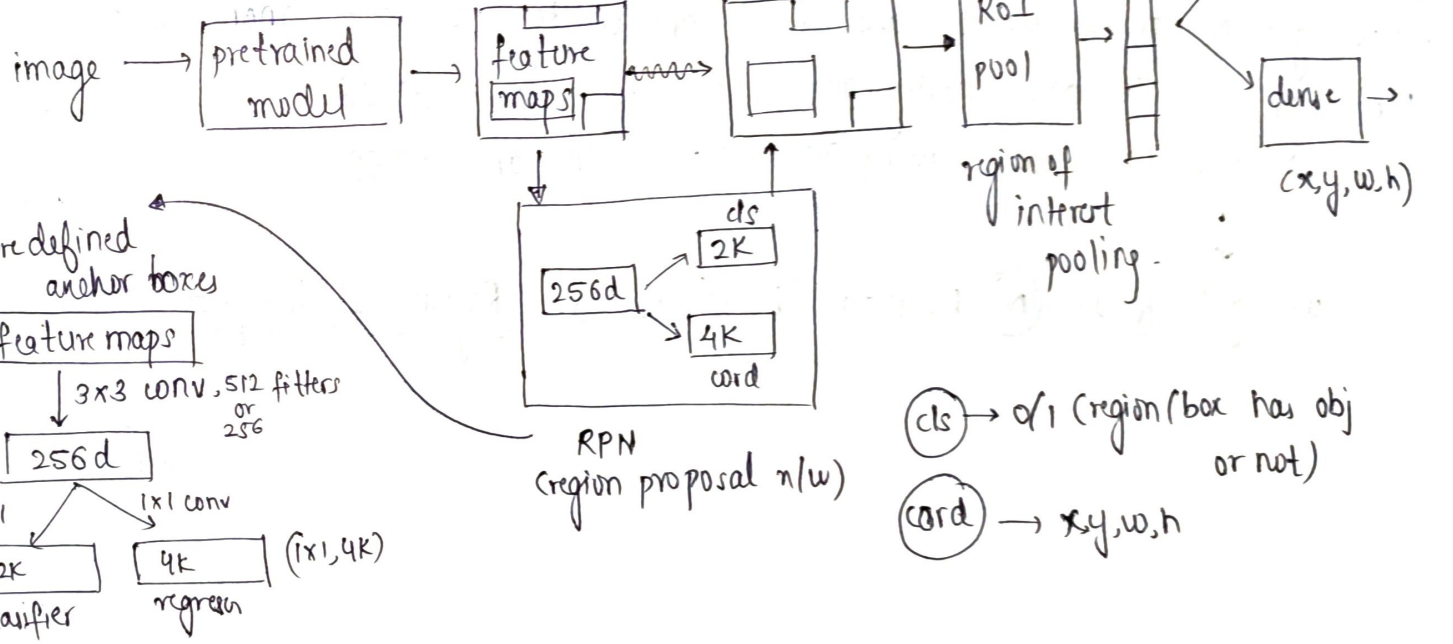
Fast-RCNN

↳ apply CONV once and then choose feature maps



Fast-RCNN
conv once → search selective regions
(returns multiple RoI's)
use maxpool to give us fixed sized regions (7×7 paper)

Faster RCNN



cls → 0/1 (region/box has obj or not)

cord → x,y,w,h

workflow

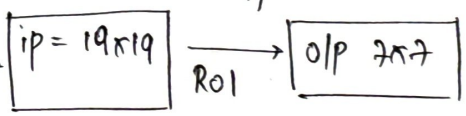
Read img + boxes → get feature maps from a CNN → RPN → ∀ each RoI do RoI pool → train for classifier / regressor models

RoI pool vs RoI algin → can deal with fractional stride by breaking each pixel into further subpixels

RoI = maxpool

↳ causes stride to be quantized

say RoIPool (7×7)



stride = $\frac{19}{7}$ = 2 ... → stride = 2

we are ignoring decimal and hence some part of image

RPN training

→ if feature map size is M×N (K: no of anchor boxes)

then clasifier wghts = M×N×(K*2) ⟶ P(forground), P(background) (object)

regresor weights = M×N×(K*4) ⟶ $t_i = \langle t_x, t_y, t_w, t_h \rangle$

RPN Loss = log loss on clasifier + distance loss fr regresor (dist b/w true and pred t_i ground t_i given by ground truth)

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{CLS}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} p_i^* (L_{reg}(t_i, t_i^*)$$

N_{cls} = no of anchors

N_{reg} = no of anchor locations

p_i^* – actual object present/not

t_i^* – actual t_i of ground truth boxes

L_{cls} – log loss

Non-max supresion

→ we get N anchor boxes and asign probability to each anchor box

We asign +ve to areas having IOU > 0.7 with any GTBB

-ve to areas having IOU < 0.3 with any GTBB

↓

we get final (N) bounding boxes called Regions of Interest

OLS for Linear Regression

$\hat{y}_i = \beta_0 + \beta_1 x_i$

Loss function $= \frac{1}{N}\sum (y_i - \hat{y}_i)^2 = \sum (y_i - (\beta_0 + \beta_1 x_i))^2$

$\beta_0 = \bar{y} - \beta_1 \bar{x}$

$\beta_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$

$\beta_0 = c$

$\beta_1 = m$

diff wrt β_0

$\frac{\partial L}{\partial \beta_0} = 2\sum (y_i - (\beta_0 + \beta_1 x_i))^{-1} = -2\sum (y_i - \hat{y}) = 0$

$\sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) = 0$

$\sum y_i - \sum \beta_0 - \sum \beta_1 x_i = n\bar{y} - n\beta_0 - \beta_1 \sum x_i$

$\rightarrow n\bar{y} - n\beta_0 - \beta_1 n\bar{x} = 0$

$n\beta_0 = n\bar{y} - \beta_1 n\bar{x}$ (divide by n)

$\beta_0 = \bar{Y} - \beta_1 \bar{x}$

$\frac{\sum y_i}{n} = \bar{y}$ $\quad \frac{\sum x_i}{n} = \bar{x}$

$\sum y_i = n\bar{y}$ $\quad \sum x_i = n\bar{x}$

diff wrt β_1

$\frac{\partial L}{\partial \beta_1} = -2\sum (y_i - \hat{y}_i) x_i = 0$

$\sum (y_i - \beta_0 - \beta_1 x_i) x_i = 0$

$\sum (y_i - \bar{y} - \beta_1 \bar{x} - \beta_1 x_i) x_i = 0$

$\sum (y_i - \bar{y} - \beta_1 (x_i - \bar{x})) x_i = 0$

$\sum x_i (y_i - \bar{y}) - \beta_1 \sum x_i (x_i - \bar{x}_i)$

$\sum x_i (y_i - \bar{y}) = \beta_1 \sum x_i (x_i - \bar{x})$

$\beta_1 = \frac{\sum (y_i - \bar{y})}{\sum (x_i - \bar{x})}$

multiply and divide by $(x_i - \bar{x})$ Nr and Dr

$\beta_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$

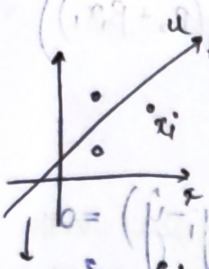
grad descent

$\frac{\partial L}{\partial \beta_0} = -\frac{2}{N}\sum (y_i - \hat{y})$

$\frac{\partial L}{\partial \beta_1} = -\frac{2}{N}\sum x_i (y_i - \hat{y})$

PCA

mathematical way



it can be proved that

$\max(x_i u^T) \equiv$ largest eigen vector

minimize fit and maximize variance

$\Rightarrow$ **steps**

↳ standardize data → covariance mat → find eigen values $\lambda_1, \lambda_2 \dots \lambda_n$ & eigen vectors $v_1, v_2 \dots v_n$

$$X_{std} = \frac{X^T X}{n-1} = S$$

$$\text{var}(x_i) = \frac{1}{n-1} \sum (x_i \cdot u^T - u^T \bar{x}_i)^2$$

$x_i \cdot u^T$ → **projection**

$u^T \bar{x}_i$ → 0 (if std)

also,

$$v_1 \perp v_2 \perp v_3$$
$$v_1 \cdot v_2 = 0$$
$$v_2 \cdot v_3 = 0$$

↳

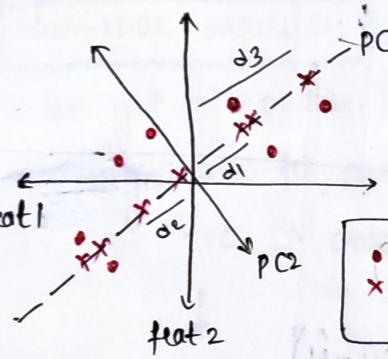
$$S v_1 = \lambda_1 v_1$$
$$S v_2 = \lambda_2 v_2$$

Loading score

- % variance explained by eigen vector v_i

$$\%\text{var} = \frac{\lambda_i}{\sum \lambda_i}$$

Graphically



standardize data ($\mu = 0$, centered at (0,0))

PC1 → **aim** minimize fit residuals ← similar to L or R

→ maximize variance of projected points

↳ in this case line and (u) on origin

∴ calculate distance of each projected point from mean ≡ origin

↳ $d_1^2 + d_2^2 \dots d_n^2 = SS(\text{distance})$ ≡ Eigen value for PC1

$\sqrt{\text{Eigval}} = \text{singular val}$

→ unit vector along PC1 = eigen vector for PC1

∴ **PC1 is a linear combination of f1 and f2**

once PC1 is set,

fit another line $\perp^r$ to PC1 $\Rightarrow$ PC2

NOW we can't fit another line $\perp^r$ to PC1, PC2 ← you can pick one line along z axis

so for 2d we have **2 PC**

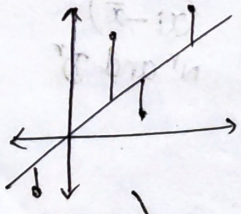
→ but if you project 2d points on it variance = 0

we can use % var / loading score to pick best one

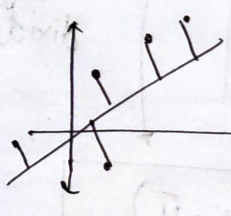
Linear reg vs PCA

Linear reg → minimize vertical dist b/w point and line

PCA → minimize orthogonal (shortest) distance

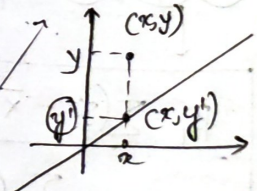


Linear reg



PCA

in LR, we have to predict y given x



∴ (y'-y) is error

XGBoost

REGRESSION

dosage	age		efectiveness	res0
5	20		13	~~0~~ 1
10	21		14	-5
12	29		12	+10
8	30		1	-8

x → y

Base model → not accurate

$\bar{y} = \frac{13+14+12+1}{4} = 20.5$ (assume)

assume $\bar{y} = 12$

$\left[\sum res_i\right]^2$ → (add)^2

Similarity score (SS) $= \frac{(\sum res_i)^2}{\#res + \lambda}$

Ig at node $= (SS_{left} + SS_{right}) - SS_{cur}$

(left, right: child nodes)

Prediction

→ you start at root until leaf node is reached

result $= \frac{(\sum res)}{\#res + \lambda}$ — No square

res_i in root node

Ex: [1, -5] is leaf → res $= \frac{-4}{2+\lambda}$

→ model 1 (BASE model)

res = [1, -5, 10, -8] SS = 10

gain $= (30+20-10) = 40$

dosage <10 / >10

you can check diff feat/value beter gain

<10: res = [1, -5] SS = 30

>10: res = [10, 8] SS = 20

gain $= 15+12-20 = 2$

d<12 / d>12

10 (SS = 15) | -8 (SS = 12)

stop = only 1 node

One model is built

prune tree using (γ) gamma

if (Ig at node) $- \gamma \geq 0$ (don't prune)

∴ if gain $- \gamma > 0$ → no prune

gain $- \gamma < 0$ → prune

gain > γ → keep

else prune

* while pruning, always start from leaf node. if you can't prune a leaf then you can't prune its parent

Ex: $\gamma = 5$

→ [10, 8] branch gets removed

d>12

model$_1$ ✓

built model 2 which fits residuals of (model$_1$)

pred res_1

$res_1 = res_1 - pred$

Regularizations

① (λ) as $\lambda\uparrow$, SS ↓

as SS ↓, gain ↓

as gain ↓, chances of pruning are here

∴ as $\lambda\uparrow$ prevents overfit

gamma (γ)

if $\gamma\uparrow$ → more branches will get pruned

as $\lambda\uparrow$ → more pruning

→ smaller output value

Classification

Base model ($p = 0.5$)

$$SS = \frac{(\sum res)^2}{\lambda + \sum (prev\_prob\,(1 - prev\_prob))}$$

probability of previous fit model (Base = 0.5)

$-\lambda$

* XGBoost has threshold for min no of leaves in tree →

$$cover = \left(\sum prev\_prob\,(1 - prev\_prob)\right)$$

if at any node if #res ≤ cover stop growth

$$cover = dr \text{ of root} - \lambda$$

min_child_split

∴ at a node, if $cover_{node} >$ min_cover

→ you can use (node) as leaf

if $cover_{node} \leq$ min_cover

→ you can't use as leaf

Ex: min_cover = 1

∀ all leaf nodes, cover (node) > 1

∴ build tree_model ① → leaf node must ② > min_cover → prune if gain < γ ③

Output

$$res(node) = \frac{\sum res}{\left(\sum pp(1-pp)\right) + \lambda}$$

→ we need probabilities ⟹ $\log\left(\frac{p}{1-p}\right)$ log (odds)

p = res

Overal

$\lambda > 0$, reduces sensitivity of tree to (outliers)

↓

prevent overfit

Pick random subset of data to (HIMT) train models

→ just like bagging xgboost also uses random rows of data

↳ increases speed & faster on large datasets

$-0.5, 0.5, 0.5, -0.5$ SS = 0

dosage > 10

-0.5 | $0.5, 0.5, -0.5$

$SS = \frac{(-0.5)^2}{0.5(1-0.5) + \lambda}$

XGBoost optimizations

Loss function

λO_{val}^2 → regularization term prevent O_{val} becoming too large

model-1

$$L = \left[\sum_{i=1}^{n} L(y_i, \hat{y}_i)\right] + \gamma T + \frac{1}{2}\lambda O_{val}^2$$

$\hat{y}_i$ → current out-put

$$= \text{Base model score} + \sum \text{residuals up to } n-1 + O_{val}$$

O_{val} = output from current model/tree for point y_i

T - no of terminal nodes (used for post pruning)

$$L = \sum_{i=1}^{n} L(y_i, (\hat{y}_i)_0 + O_{val}) + \gamma T + \frac{1}{2}\lambda O_{val}^2$$

Approximate greedy

→ we decide node split based on gain and then just pick one with best gain and split. once split we dont revisit node again which saves time.

Parallel + weighted quantile sketch

→ if feature is continous, calculating all thresholds and checking is slow

→ Instead, we pick quantiles among data and then we these quantiles

Generally, XGBoost uses around 33 Quantile (test spit 33 times)

→ **sketch algos** → they split data to various pcs and later combine to find approximate quantiles (histograms)

+ weighting (use cover and prob) ⟹ weighted Quantile sketches

└→ in this each histogram region of quantile will have same no of observations

Cache-aware access

→ try to store gradients and deri in cache as they are frequently accessed

XGBoost pros

- Improved Grad Boost
- Regularization
- aprox greedy algo
- parallel learing
- weighted quantile sketch
- *sparsity aware split find
- * Cache awareness
- * pick random rows like bagging.
- *Block out-of-core computation

→ if data is too large and cant fit in RAM compress it into chunks and save in HDD

→ when needed CPU reads this chunks parallely → uncompress and use (although zip/unzip takes time but better than slow disk read)

Sparsity aware split

assume missing features
↳ initial residue can be found (y_i - average(vals))

dosage	effe	resi
10	9	3
21	2	4
19	7	-2
?	1	-6
?	8	-1

split (?) in new table

?	1	-6
?	8	-1

10	9	3
21	2	4
19	7	-2

as data is small we use avg of 2 each non-? rows
else for large dataset, we use quantiles

initial node with non-missing residues + missing residuals

step 1 → 3, 4, -2, -6, -1

dos > 15

3 | 4, -2

mean of 2 each non-?

(don't include residuals of ? in leaves)

step 2

now add residues of (??) rows to left leaf and then right leaf and calculate gains

g_{left} = dos > 15 → 3, -6, -1 | 4, -2

g_{right} = dos > 15 → 3 | 4, -2, -1

→ pick best gain and add those leafs

step 3 pick average of next 2 non-? closes and check again (21, 19)

TSNE

t-distributed stochastic neighborhood embedding

Core idea Instead of focusing to preserve distances, focus on preserving neighbors.

① Stochastic nbrhood embedding

$$L = \sum_{ij} P_{ij} \log \frac{P_{ij}}{q_{ij}}$$ KL divergence blw similarities

P_{ij} := pairwise affinity b/w p_i and p_j in high-dim space

q_{ij} := pairwise affinities in low-dim space

* affinities tends → 1 ; if points are closer
  → 0 ; if far away
* sum of all similarities / affinities = 1
* if 2 points close in high-d are projected far away in low-dim, loss is greater

High-dim similarities =

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}$$ → used such that $\sum p_{ij} = 1$

Perplexity — how many neighbors to preserve (around 30)

$P = 2^H$; $H = -\sum_{j \neq i} P_{j|i} \log_2 P_i$

$w_{ij} = K(\|y_i - y_j\|^2)$

$$P_{ij} = \frac{P_{i|j} + P_{j|i}}{2n}$$

$$\text{loss fn} = \sum_{ij} P_{ij} \log q_{ij} + \sum P_{ij} \log \frac{w_{ij}}{Z}$$

→ low-dim similarities

$$q_{ij} = \frac{K(\|y_i - y_j\|^2)}{Z} \quad \text{where, } K = K(d) = 1/(1+d^2)$$

[t-distributed kernel for calculating similarities]

$$Z = \sum_{k \neq l} K(\|y_k - y_l\|^2)$$

decision tree with 1 root, 2 leaf

ADABoost → KEY POINTS — ① Base learners are weak (stumps)

② Each BL is dependent on information frm prev

③ Each BL may have diff weight in final voting

Steps

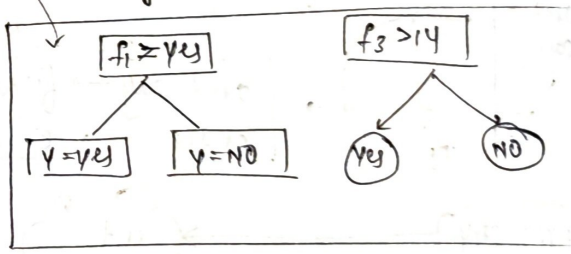
① Initially give same weight to all samples

	f_1	f_2	f_3	Y	wt
1	Yes	0		Y	1/5
2	No	1		Y	1/5
3	Yes	0		N	1/5
4	Yes	2		N	1/5
5	No	2		N	1/5

→ fit a BL (stump)

→ create stump for f_1 f_2 f_3 & check which gives more gini index

→ that stump is BL1



② update wt

→ predict (4) using BL1 — ex say (4) (5) were incorrectly ⇒ TE = $\frac{1}{8} + \frac{1}{8} = \frac{1}{4}$ classified by BL1

Total Error = ∑ weights of incorrectly classified sample

$$\text{amt\_say} = \frac{1}{2} \log\left(\frac{1 - TE}{TE}\right)$$

③ new weights (wt1)

penalize incorrect more

∀ sample,

- if sample incorrectly classified → $wt = wt \times e^{-\text{amt\_say}}$
- correctly classified → $w^t = wt \times e^{\text{amt\_say}}$

Normalize aftr calculating wt1 so that $\sum wt = 1$

Normalize entire col → $\sum x_i \rightarrow x_i = \frac{x_i}{\text{sum}(x)}$

Next BL →

→ create new dataset with same size

→ say len(dataset) = 5

* pick 5 rows from dataset randomly, such that prob of each row getting picked is proportional to wt value (proportional sampling)

* some rows may get repeated

↳ fit stumps (BL1) → TE → amt_say → wt2 → new data for BL2

majority vote

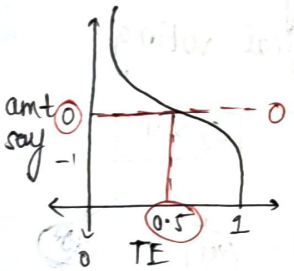
say 3 BL → yes, 3 B → No → find Σamt_say(BL1), Σamt_say(BL2) → which gives more → pick that result

amt = 3·2, 4·2 → NO ✓

Observations

$AmTE = \frac{1}{2}\log\left(\frac{1-x}{x}\right)$

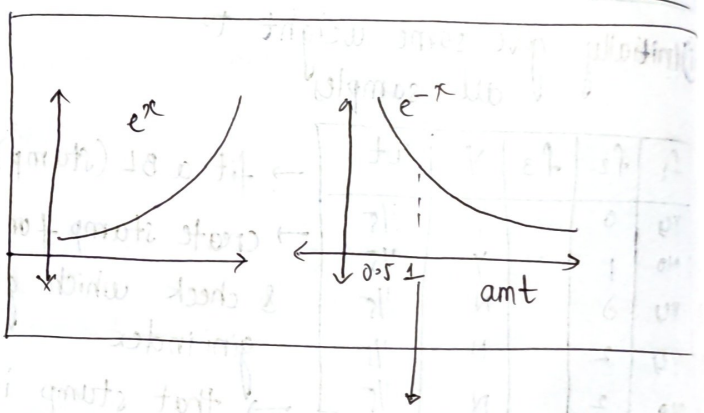
$amt\_say = \frac{1}{2}\log\left(\frac{1-TE}{TE}\right)$



error ↑ → amt_say ↓

amt_say → fixed for BL

change → e^{-x} or e^{+x}



AMT_SAY → wt updates

* If TE = 1, (worst)
  amt_say < 0
  (more error, less amt_say)

* If TE = 0 (best)
  amt_say > 0

* if TE = 0.5 (random model)
  amt_say = 0

wt updates → +ve classified (correct) → more amt say, lesser is e^{-x}

(wt, e^{-amt_say}) ∴ if amt_say >> wt update very little (it may be < wt_{i-1})

-ve classified incorrect → more amt_say, more is e^x ∴ wt update is more

wt · e^{amt_say}

Idea

TE is small (better BL)
↳ update INC ~~more~~ less
↳ reduce correct mark

* say TE is large,
  ↳ then incorrect pts wt will be increased more as we want to predict them in next stump
  ↳ correct pts wt will reduce because they have already been correctly clf by BL