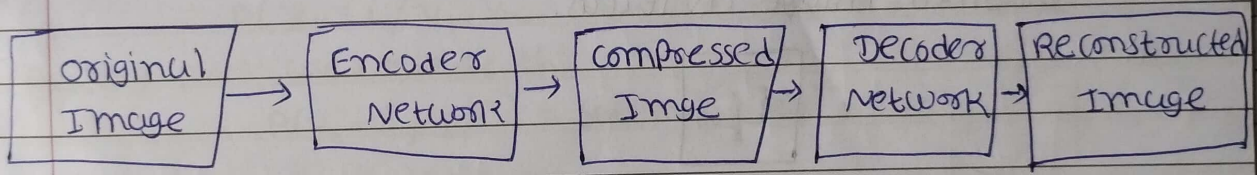
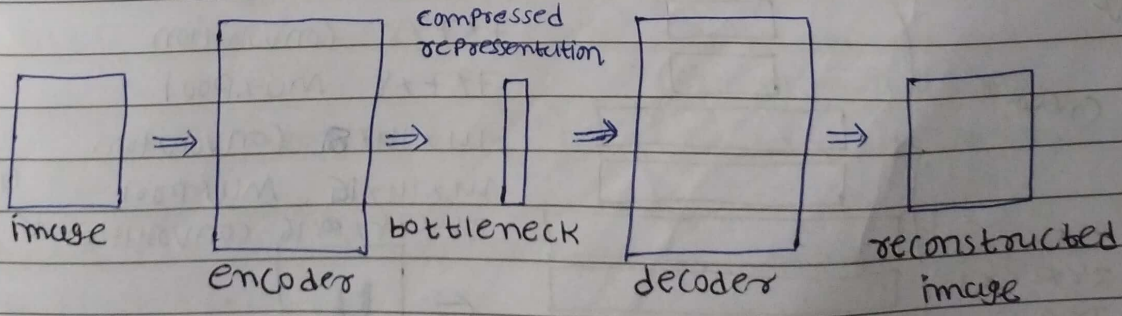
convolution Autoencoder in CNN

⇒ For anomaly detection train data on non anomaly data and then anything fall out of the norm that automatically labeled as an anomaly
⇒ Reconstruction error is high for anomaly data

⇒ A convolution ~~auto en~~ autoencoder is a neural network (a special case of unsupervised learning model) that is train to reproduce its input image in output layer.
⇒ Image is pass through an encoder which is a convolution network that produce the low dimentional representation of Image.
⇒ The decoder which is another sample of convolution network (Convnet) takes this compressed image and ~~re con~~ reconstructs the original image
⇒ Encoder is use to compress the data and decoder is used to reproduce the original image
⇒ compression logic is data-specific meaning it is learned from data rather than predefined compression algorithems such as JPEG, MP3. and so on.



⇒ autoencoder ~~is~~ are deep learning architecture capable of reconstructing data instance from their feature vector.
⇒ They work on all sort of data but Here we see ~~application~~ their application on image data
⇒ Autoencoder is made with 3 main components
  i) encoder
  ii) bottleneck
  iii) decoder



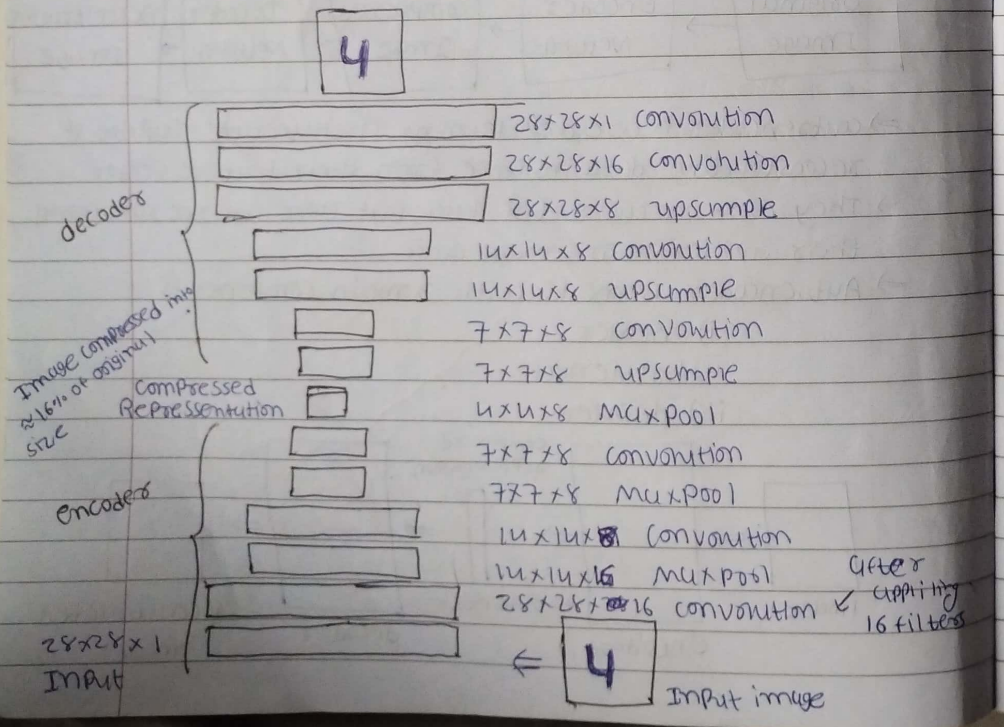
⇒ In autoencoder model will be train in a such a way so that we can get original data from compressed data ~~that~~ that means compressed data contain all important information of input data

⇒ Auto encoder has analogy with PCA

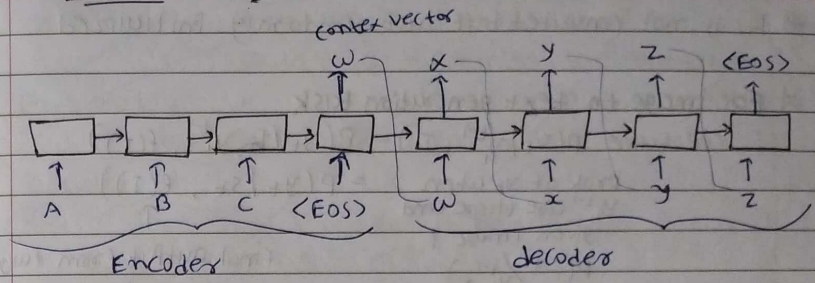
i) Encoder : The first section of auto encoder. encoder extract the most salient(main) features from images and return them as a vector.

ii) Bottleneck: Bottleneck also called a code layer, serves as an extra layer which help to compress the extracted feature into a small vector representation. This is done in a bit to make it more difficult for the decoder to make sense of the features and force it to learn more complex mappings

iii) Decoder: decoder attempts to make sense of the features coming from the encoder, which have been subsequently compressed in the bottleneck so as to reconstruct the original image as it was.



Encoder-decoder NPTEL-NOC-IITM



⇒ For decoder model we give context vector and and predict next step output

⇒ given t-1 word we want to predict t^{th} word

h_{t-1} contain all y_1^{t-1}

$$y^* = \text{argmax}\ P(y_t / y_1, y_2 \dots y_{t-1})$$

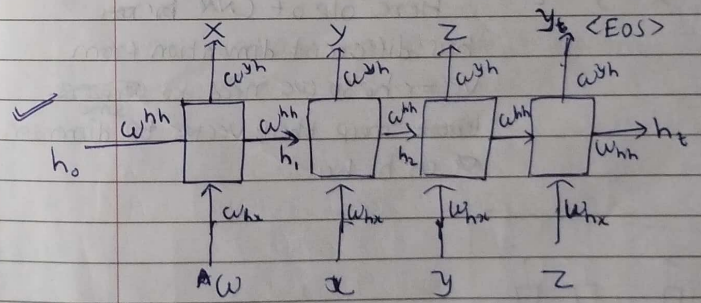
$$y^* = \text{argmax}\ P(y_t / y^{t-1}) = \text{argmax}\ P(y_t / h_{t-1})$$

so we interested in

$$P(y_t = j / y_1, y_2 \dots y_{t-1}) \quad \text{where } j \in V$$ (all vocabulary word)

⇒ using RNN we compute this

$$P(y_t = j / y_1^{t-1}) = \text{softmax}(w^{yh} h_t + c)_j$$



$$h_t = \sigma(w^{hx} x_t + b + w^{hh} h_{t-1})$$

$$y_t = w^{yh} h_t$$

Loss: $L(\theta) = \sum_{t=1}^{T} L_t(\theta)$

$$L_t(\theta) = -\log P(y_t = l_t / y_1^{t-1})$$

l_t = true word at time t

⇒ and we do BACK Propogation further and adjust weight so that $L(\theta)$ is minimize

⇒ h_0 is not computed but just randomly initialized

★ For image to text generation task

we take $P(y_t | y_1^{t-1}, I) = P(y_t | y_1^{t-1}, f(I))$
$= P(y_t | s_t, f(I))$

Prob of y_t when y_1^{t-1} are these and given image I
$P(y_t / y_1^{t-1}, I)$

Final output from fully connected layer from CNN

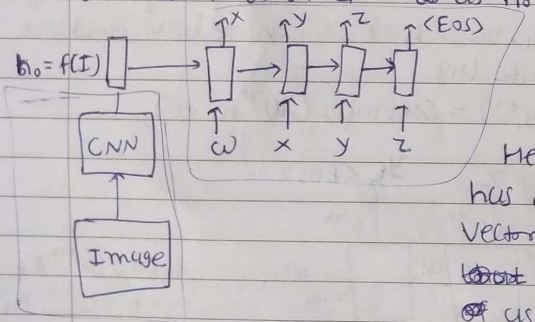
⇒ Here we pass a image through CNN and get output as f(I) which contain information about image that we given.

⇒ we can do

i) connect op of CNN to cell as h_0

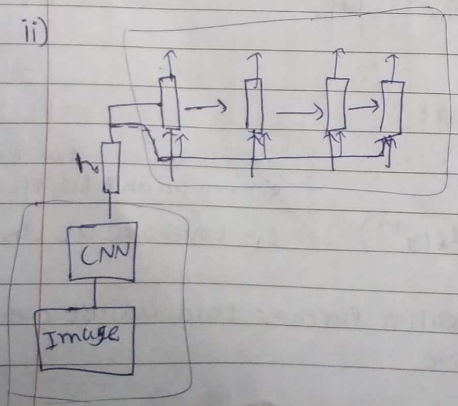
OR ii) connect op of CNN to cell for every time ~~step~~ stemp

i) connect op ~~to~~ of CNN ~~to~~ as h_0



Here o/p of CNN ~~is~~ has different dimention from vector h_0 so we need to ~~apply~~ same ~~that~~ map that vector to ~~dimention~~ as h_0 has.

ii)



Code for i) Task : Textual entailment

Data: $\{x_i$ = premise, y_i = hypothesis$_i\}_{i=1}^{N}$

Encoder:

$h_t = RNN(h_{t-1}, x_{it})$

Decoder:

$h_0 = h_T$ (T is length of input)

$h_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$

e means embedding

★ Task: Image captioning

Data: $\{x_i$ = image, y_i = caption$_i\}_{i=1}^{N}$

image & corresponding caption is given for traing

Model:

Encoder: $h_0 = CNN(x_i)$

Decoder: $h_t = RNN(h_{t-1}, \bar{e}(\hat{y}_{t-1}))$

embedding of previous output It can be any one hot, word2vec

$P(y_t / y_1^{t-1}, I) = \text{Softmax}(w^{yh} h_t + b)$

Parameters: w^{yh}, w^{hh}, w^{xh}, w_{combine} biases

Loss: $L(\theta) = \sum_{i=1}^{T} L(\theta) = -\sum_{i=1}^{T} \log P(y_t = \underset{\text{actual op}}{l_t} | y_1^{t-1}, I)$

Back Propagation gradient descent

★ Textual entailment

Task: Textual entailment

data: $\{x_i = \text{Premise}, y_i = \text{hypothesis}\}_{i=1}^{N}$

Model:

(Option 1) Encoder: $h_t^{\text{Encoder}} = RNN(h_{t-1}, x_{it})$

Decoder: $h_0 = h_T$ (T is length of input)

$h_t = RNN(h_{t-1}, e(\hat{y}_{t-1}))$

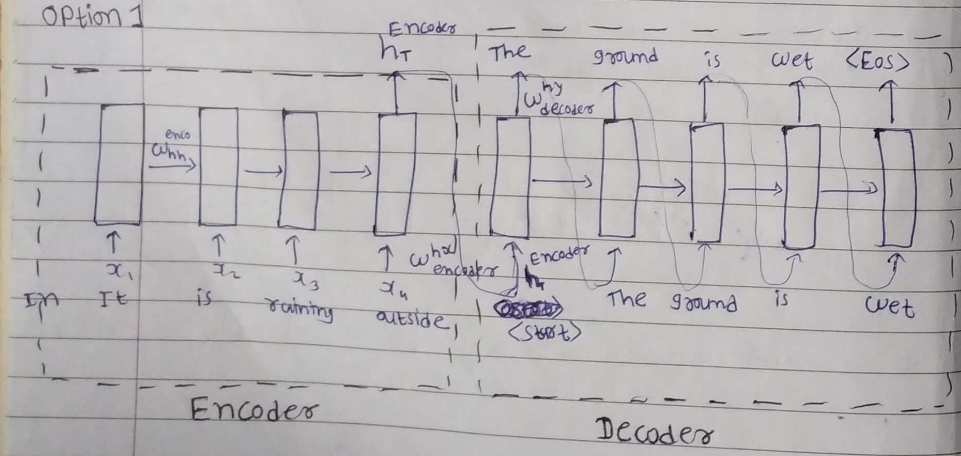
$P(y_t | y_1^{t-1}, x) = \text{softmax}(W^{yh} h_t + b)$

Parameters: $W^{hh}_{\text{decoder}}, W^{hx}_{\text{decoder}}, W^{hy}_{\text{decoder}}, W^{hh}_{\text{encoder}}, W^{hx}_{\text{encoder}}$, biases

Loss: $L(\theta) = \sum_{i=t}^{T} L(\theta) = \sum_{t=1}^{T} \log P(y_t = l_t | y_1^{t-1}, x)$

Algo: gradient decent with back Propagation

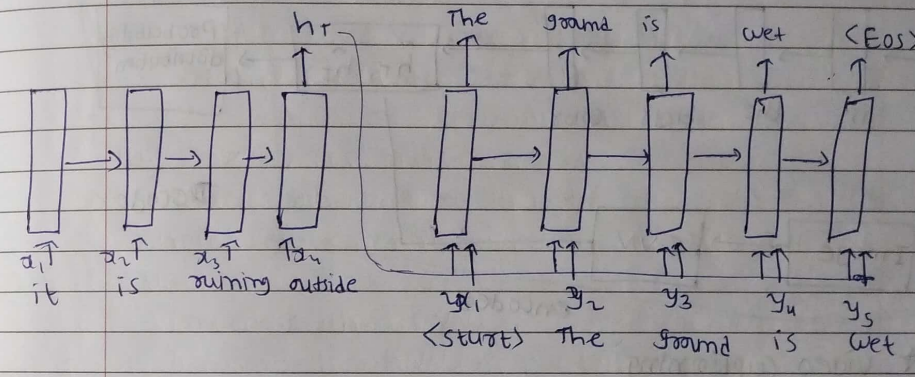
Option 1



Model2: Decoder: $h_0 = h_T$

$h_t = RNN(h_{t-1}, [h_T, e(\hat{y}_{t-1})])$

$P(y_t | y_1^{t-1}, x) = \text{softmax}(W^{yh} h_t + b)$



★ Machine translation

I/P: I am going home

O/P: Mein ghar ja raha hoon

Task: Machine translation

Data: $\{x_i = \text{source}, y_i = \text{target}\}_{i=1}^{N}$

other same as above There is two model

★ Transliteration

I/P: ई ण् डि या — O/P: India

same as above

same for Document Summarization

★ Image Question Answering

Data: $\{x_i = \{I, q\}_i, y_i = \text{Answer}_i\}_{i=1}^{N}$

I: Image, q: Question (Text)

Model: Encoder: $\hat{h}_I = CNN(I)$, $\tilde{h}_t = RNN(\tilde{h}_{t-1}, q_{it})$

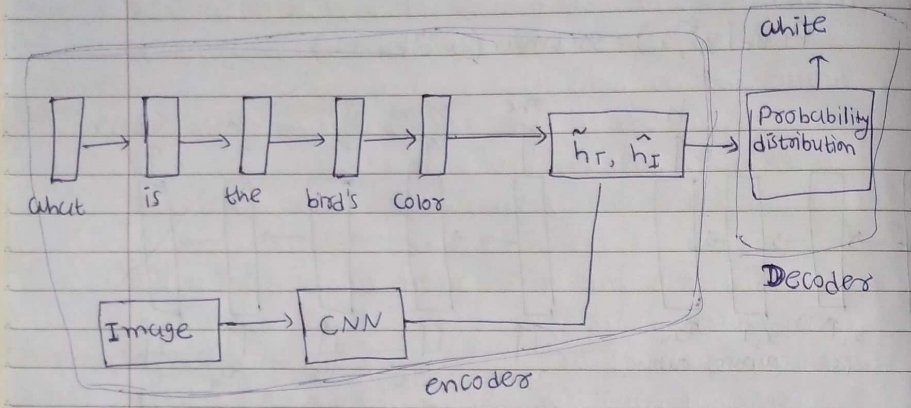
$S = [\tilde{h}_T ; \hat{h}_I]$ (concatinating)

Decoder: $p(y | q, I) = \text{softmax}(W^{yh} S_0 + b)$

Prob distribution

$(W^{yh} S_0 + b)$

Parameters: w^{yh} $w_q^{hh}, w_q^{xh}, b_{ias}$ (for question), $w_{convolution}, b_{ias}$ (convolution network), $w_{convolution}, b_{ias}$ (convolution network)



★ Video captioning

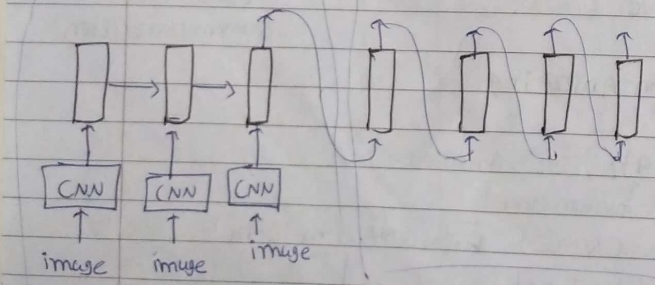
Data: $\{x_i = \text{video}, y_i = \text{desc}_i\}_{i=1}^{N}$ (desc: description)

Model: Encoder: $h_0 = h_T$

$h_t = RNN(h_{t-1}, CNN(x_{it}))$

$P(y_t | y_1^{t-1}, x) = \text{softmax}(w^{yh} h_t + b)$

All other same



★ Video classification

Encoder: same as Video Captioning

Decoder: Probability distribution

⇒ Attention model is better due to better modling choice
⇒ have linear tranformation follow by non linearity ($\because f_{ATT}$)

Attention model

Attention Mechanism

NPTEL-NOC-IITM

⇒ Encoder reads the sentences only once and encode it
⇒ At each timestep the decoder uses this embedding to produce a new word.
⇒ Human try to produce each word in the output by focusing only on certain word in the input
⇒ Essentially at each time step we come up with a distribution on the input words.

I/P : I am going home
t_1: [1 0 0 0 0]
t_2: [0 0 0 0 1]
t_3: [0 0 0.5 0.5 0]
t_4: [0 1 0 0 0]

⇒ This distribution tell us how much attention pay to each input word at each time step.

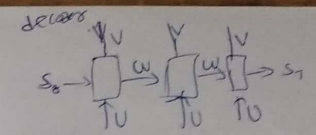
I/P Main ghar ja rahi hoon

⇒ we could just take a weighted average of the corresponding word representations and feed it to decoder.
⇒ For Ex time step 3 we can just take weighted avg of 'ja' and 'rahi'
⇒ This help to ignore irrelevant information (which is not important at that perticular time step)
⇒ To do that we define function

$$e_{jt} = f_{ATT}(h_{t-1}, h_j^{Enco})$$

e_{jt}: t^{th} time step how important is j^{th} term
h_{t-1}: old from decoder
h_j^{Enco}: input j^{th} word is how important for t^{th} timestep; states of all the word from encoder

⇒ so e_{jt} is depend on what is happend in decoder so fur h_{t-1} and what is current word actually look like.
⇒ e_{jt} is captures the importance of the j^{th} input word for decoding t^{th} output word



⇒ we can normalize those weights by using softmax

$$\alpha_{jt} = \frac{exp(e_{jt})}{\sum_{j=1}^{M} exp(e_{jt})}$$

⇒ α_{jt} is Prob of focusing on the j^{th} word to produce the t^{th} output word
⇒ h_t = decoder RNN's state at t^{th} timestep
h_j^{Enco} = Encoder RNN's state at j^{th} time step
⇒ one Possible choise for function f_{ATT} is (There are many Possi)

$$e_{jt} = V_{att}^T \tanh(U_{att} h_{t-1} + W_{att} h_j^{Enco})$$

scalar; $\mathbb{R}^{d_1}$; $\mathbb{R}^{d_1 \times d}$, $\mathbb{R}^d$, $\mathbb{R}^{d_1 \times d}$, $\mathbb{R}^d$; R^{d_1}

⇒ $V_{att} \in \mathbb{R}^d$, $U_{att} \in \mathbb{R}^{d \times d}$, $W_{att} \in R^{d \times d}$ are additional parameters of the model
⇒ These parameters will be learned along with the other parameters of the encoder and decoder
⇒ This model would make a lot of sense if were given the true α's at training time
let $\alpha_{tj}^{true} = [0, 0, 0.5, 0.5, 0]$
$\alpha_{tj}^{Pred} = [0.1, 0.1, 0.35, 0.35, 0.1]$
by minimizing loss by adjusting weights gives us appropriate values of weight
But, in practice it is very hard to get α^{true}
⇒ Then how would this model work in the absence of such data?
⇒ It's work because of better modeling choice (model in which we give attention on only relevent word)
⇒ This is a more informed model
⇒ we are essentially asking the model to approach the problem in a better way (natural way)
(Ex of ⇒ like human can ride cycle if helher put hand on handle then ....)

PAGE NO.:

⇒ given enough data it should be able to learn these attention weight just as human do (Ex of cycle)

⇒ and in practice indeed this models work better than the vanilla encoder decoder models

Task: Machine Translation

Data: $\{x_i = \text{source}, y_i = \text{target}\}_{i=1}^{N}$

only input and output sentence are given. and which word use attention to is every time is not given

Encoder:

input sentence

$h_t^{Enco} = RNN(h_{t-1}, x_t)$

$h_0^{Enco} = h_T = h_0$

O/P from Encoder and input for decoder

Decoder $e_{jt} = V_{attn}^T \tanh(U_{attn} h_j^{Enco} + W_{attn} h_t)$

from decoder

Prob distri $\alpha_{jt} = \text{softmax}(e_{jt})$

$$c_t = \sum_{j=1}^{T} \alpha_{jt} h_j^{Enco}$$

o/p of pre time

for decoder $\rightarrow h_t = RNN(h_{t-1}, [e(\hat{y}_{t-1}), c_t])$

$\hat{y}_t = \text{Softmax}(W^{yh} h_t + b)$

deco deco

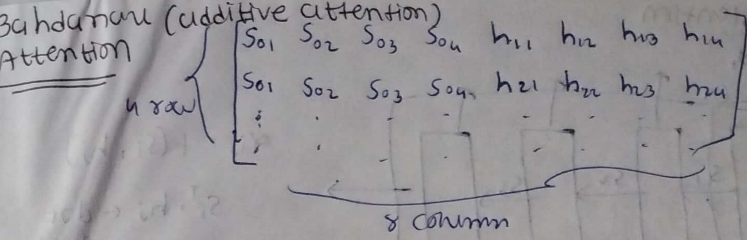
Parameters: $W^{1h}_{deco}, W^{yh}_{deco}, W^{hh}_{deco}, b$ (decoder), $W^{xh}_{enco}, W^{hh}_{enco}, b$ (encoder)

U_{attn}, V_{attn} (attention)

Loss and algorithum: same as ~~before~~ others

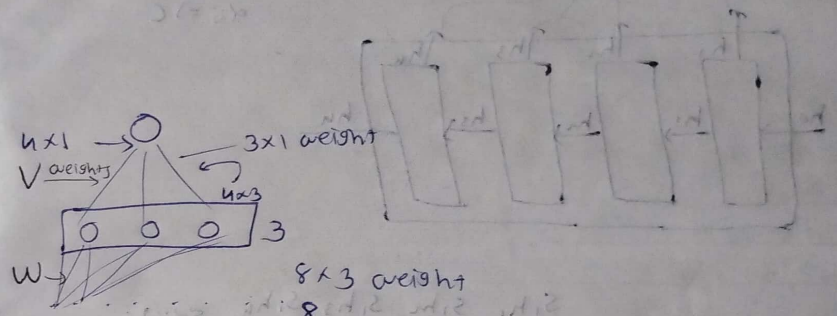
entropy, SD with Back propagation

Bahdanau (additive attention) Attention



$[s_{i-1} : h_j]$.

let h_0 4D vector $[a\ b\ c\ d]$.



$4\times8 \quad 8\times3 \Rightarrow 4\times3 \Rightarrow$ activation $\Rightarrow 4\times3 \Rightarrow 3\times1$

$[s_{i-1}, h_j]$ weight

softmax 4×1

$c_1 \Leftarrow [\alpha_{11}\ \alpha_{12}\ \alpha_{13}\ \alpha_{14}] \Leftarrow [e_{11}\ e_{12}\ e_{13}\ e_{14}]$

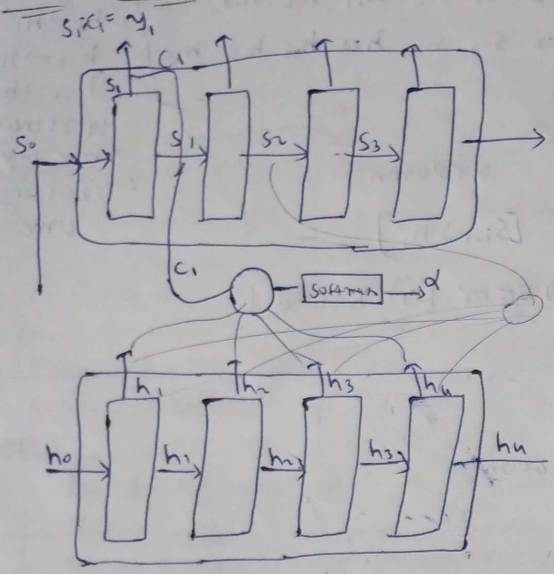
from y_{t-1} & c_1 ⇒ lstm give out of first step "I"

$c_{ij} = \Sigma \alpha_{ij} h_j$

$\alpha_{ij} = \dfrac{\exp(e_{ij}^i)}{\sum_{i=1}^{4} \exp(e_{ij})}$

$e_{ij} = V \tanh(W[s_{i-1}; h_j] + b)$

Luong Attention



$\alpha_{ij} = f(s_i, h_j)$

$s_i^T \cdot h_j \leftarrow$ dot Product

$e_{ij} = s_i^T h_j$

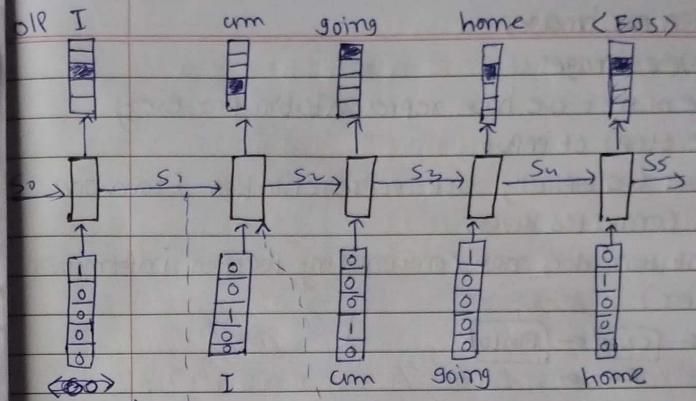
$\alpha_{ij} = \text{softmax}(e_{ij})$

$\alpha_{ij} \Rightarrow C$

$s_1 h_1 \quad s_1 h_2 \quad s_1 h_3 \quad s_1 h_4 \quad \Sigma \alpha_{ij} h_j$

$[e_{11} \quad e_{12} \quad e_{13} \quad e_{14}] \longrightarrow c_1$

⇒ Here we use s_i so we get recent information and get more accurate c value.



α : alignment score

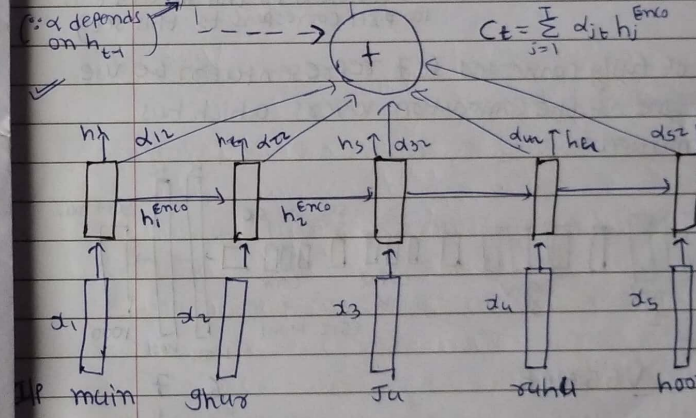
α_{12} depends on $h_1 \& s_1$

$\alpha_{22} \rightarrow h_2 \& s_1$

$\alpha_{jt} = f(s_{t-1}, h_j)$

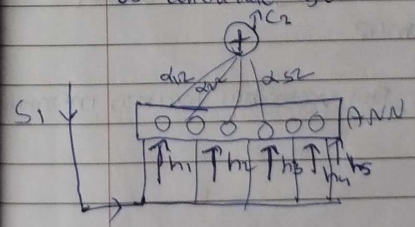
$\because \alpha$ depends on h_{t-1}

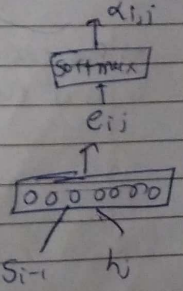
$C_t = \sum_{j=1}^{I} \alpha_{jt} h_j^{Enco}$



to calculate α_{jt}

$\alpha_{jt} = f(s_{t-1}, h_j)$





Tus
Du
Enc
Decoc
Prok
Ann
Loss
entropy

PAGE NO.:

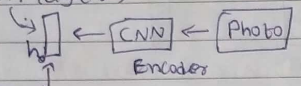
Attention on images

★ Attention over images

⇒ In the case of text we have representation for every location (time step) of input

⇒ But for images we typically use representation from one of the fully connected layers

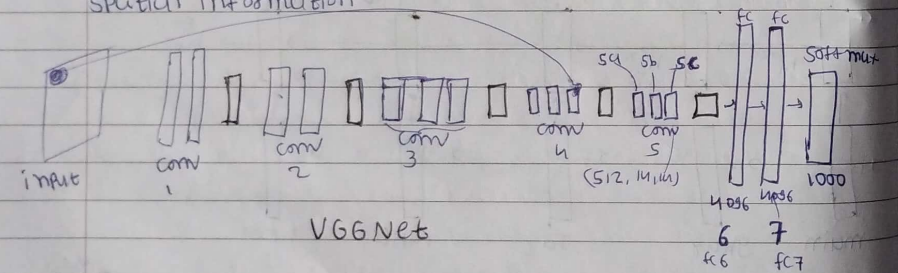
⇒ The representation does not contain any location information (∵ Flatten layer)



This vector is flat and does not have any location base encoding that contain every thing in image

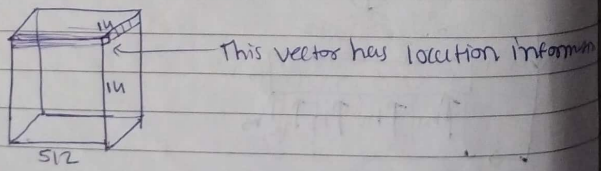
(For Ex vector size 512 we can't ~~say about the~~ that this 5 pixel is correspond to this set and this 100 pixel correspond to this set)

⇒ well instead of fully connected fc7 representation we use the output of one of the convolution layers which has spatial information



VGGNet

⇒ For example the o/p of 5th convolution layer of VGGNet is a 14×14×512 size feature map



This vector has location information

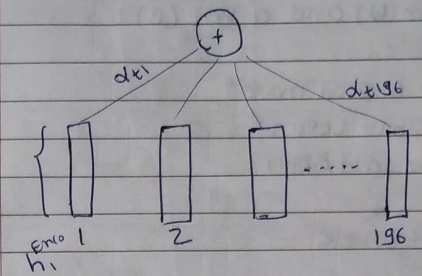
⇒ we could think of this as 196 locations (each having 512 dimention representation)

PAGE NO.:

$\alpha_{ti} = f_{ATT}(h_{t-1}, h_j^{Enc_0})$

decoder

other calculation same as previous



⇒ model will learn attention over those ~~total~~ locations (which in turn correspond to actual locations in image)

Hierarchical Attention

⇒ Consider dialog bet user (U) and a bot (B)

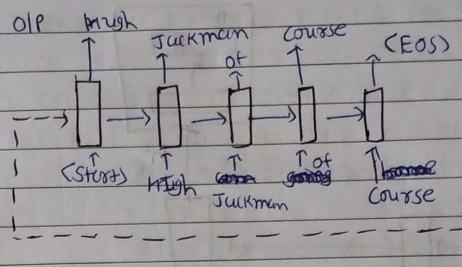
context U: Can you suggest a good movie?
B: yes, sure How about Logan?
U: okay, who is the lead actor?

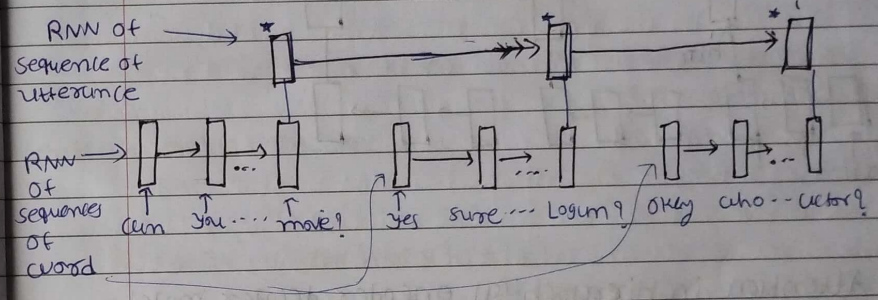
Response: Hugh Jackman, Of Course

⇒ The dialog contains a sequence of utterance bet^n the user and bot
⇒ Each utterance in ~~term~~ turn ~~of~~ is a sequence of words
⇒ Thuse what we have here is a "sequence of sequence" as input
⇒ Can you think of an encoder for such a sequence of sequences? ⇒ RNN of RNN
~~the~~ two level hierarchical RNN encoder

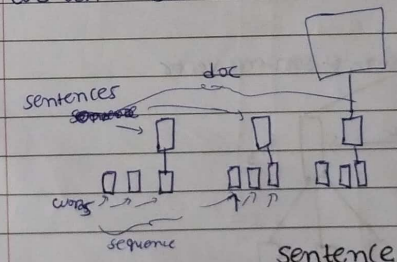
⇒ The first level RNN operates on the sequence of words in each utterance and give us a representation
⇒ we have know sequence of utterance representations
⇒ so another RNN encodes this sequence ~~of~~ and gives a single representation for sequences of utterances
⇒ decoder can then produce an output sequence conditioned on this utterance

2 types of model of prev





⇒ Another example ~~to~~ document classification or summarization
⇒ A doc is seq of ~~words~~ sentences
each sentence in turn is a sequence of words
⇒ we can again we use hierarchical RNN to model this.



Data: $\{DOC_i, class_i\}_{i=1}^{N}$

word level (1) encoder:

$h'_{ij} = RNN(h'_{ij-1}, x_{ij})$

j^{th} word of i^{th} sentence; pre output; input

$h_i = h'_{iT_i}$ [T_i is length of sentence i]

sentence level (2) encoder:

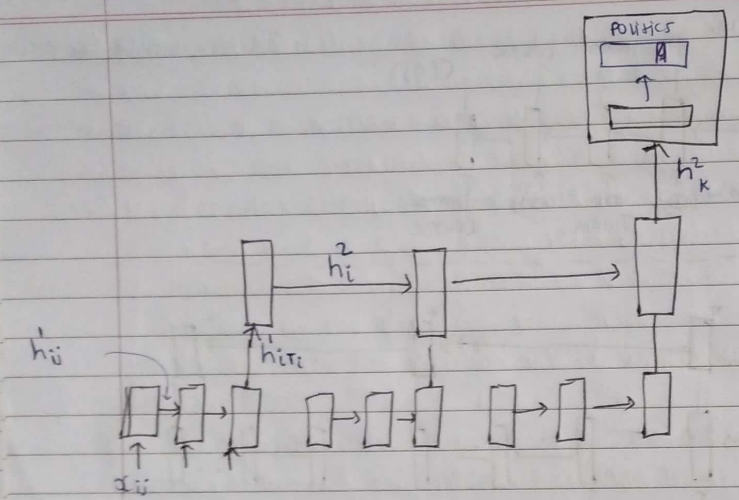
$h^2_i = RNN(h^2_{i-1}, h_i)$

$h = h^2_K$ [K is number of sentences]

Decoder: $p(y|doc) = softmax(w^{yh} h + b)$ - (class (writers use) $Vh + b$)

Params: $w'_{enc}, U'_{enc}, w^2_{enc}, U^2_{enc}, V, b$

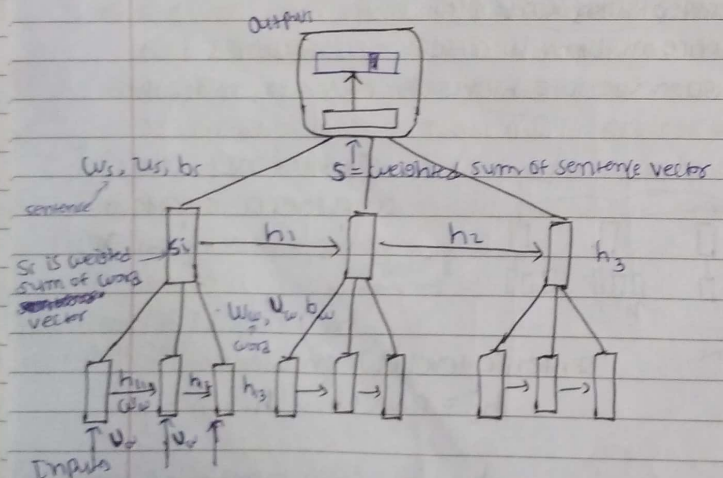
others same as previous



☆ Attention in Hierarchical encoder decoder model

⇒ First we need to attend to important words in sentence

⇒ Then we need to attend to important sentences in doc.



Data: $\{Doc_i, Class_i\}_{i=1}^{N}$

word level encoder: $h_{ij} = RNN(h_{ij-1}, x_{ij})$

$e_{ij} = V_w^T \tanh(W_w h_{ij} + b_w) + U(w_{ij})$

(∵ OP is one only no time step)

decoder state word has no previous time step

$\alpha_{ij} = softmax(e_{ij})$

$S_i = \sum_j \alpha_{ij} h_{ij}$ — S_i is representation of sentence i in term of vector

sentence level (s) encoder:

$h_i = RNN(h_{i-1}, S_i)$

$e_i = V_s^T \tanh(W_s h_i + b_s)$

$\alpha_i = softmax(e_i)$

$S = \sum_i \alpha_i h_i$

NTPEL-NOC-IITM

LSTM & Gated Recurrent Units (GRUs)

⇒ RNN has vanishing gradient problem so at t time step information store in time step t-k get completely morphed If k is large enough

⇒ and similar problem occures during back propagation

⇒ let consider the task of predicting the sentiment of a review

→ RNN reads the doc from left to right and after each word updates the state

→ By the time we reach the end of document RNN loss the information which obtain by first few words

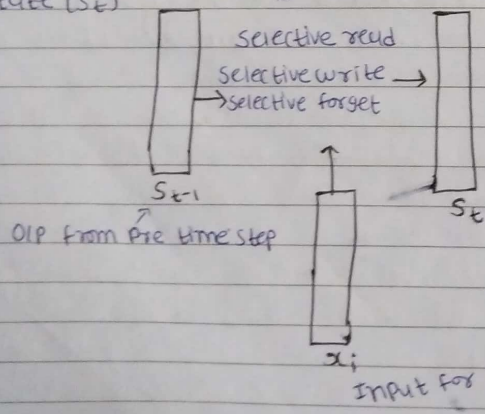
⇒ so we want to do

Forget: Forget the information add by stop words (a, the, ...)

selectively read: The information added by previous sentiment bearing words (awesome, amazing, etc.)

selectively write: selectively write new information from current word of the state

⇒ so what we want to do here is we have computed a state s_{t-1} at time step t-1 and now we want to overload it with new information (x_t) and compute a new state (s_t)



⇒ In ML when ever we want to learnt something always introduce Parametric form of these quntity and the learnt the parameter of that function

⇒ Remember that there is no explicit supervision here it is just that we have a better modling choise.

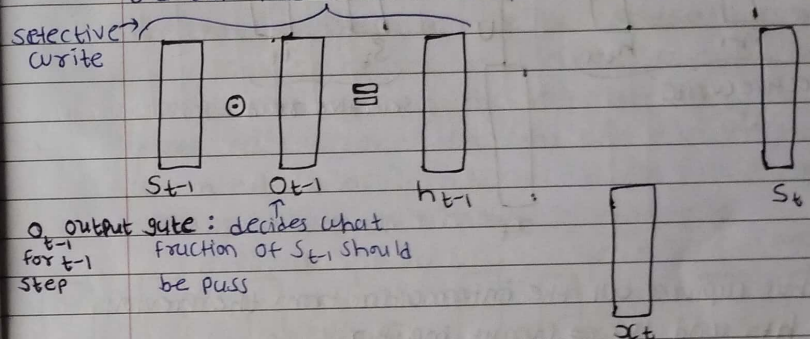
selective write

⇒ Recall that RNNs we used s_{t-1} to compute s_t

$$s_t = \sigma(W s_{t-1} + U x_t)$$ ignoring bias

⇒ but now we pass (write) only some important portion to next in this case strictest decision is binary (will given information pass 100% or not? pass 100%) 0 or 1

⇒ but more sensible way of doing this is insted of 0 or 1 we pass some fraction (selective portion) to next state



o_{t-1} for t-1 step

output gate: decides what fraction of s_{t-1} should be pass

⇒ each element of o_{t-1} mutiplied with the corresponding element of s_{t-1} and each ele of o_{t-1} is or <1

⇒ Here problem is how we compute o_{t-1}? does RNN knows what fraction of the state to pass on?

so we introduce parametric form for o_{t-1}

we compute o_{t-1} and h_{t-1} as

$$o_{t-1} = \sigma(W_o h_{t-2} + U_o x_{t-1} + b_o)$$ σ is used because we want o_{t-1} bet^n 0 to 1

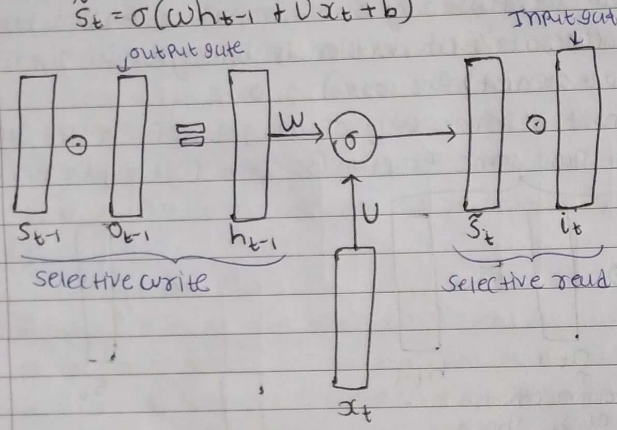
$$h_{t-1} = o_{t-1} \odot s_{t-1}$$

Selective read

⇒ we use h_{t-1} to compute new state at the next time step t

⇒ we will also use x_t which is input at time step t

$$\tilde{S}_t = \sigma(W h_{t-1} + U x_t + b)$$



⇒ $\tilde{S}_t$ thus captures all the information from the previous state h_{t-1} and the current input x_t

⇒ However, we may not want to use all info and selectively read from it before constructing the new cell state S_t

⇒ to do that we introduce new gate called **input gate**

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

⇒ So far we have

Previous state: S_{t-1}

Output gate: $o_{t-1} = \sigma(W_o h_{t-2} + U_o x_{t-1} + b_o)$

Selective write: $h_{t-1} = o_{t-1} \odot S_{t-1}$

Current state: $\tilde{S}_t = \sigma(W h_{t-1} + U x_t + b)$ (temporary)

input gate: $i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$

Selectively read: $i_t \odot \tilde{S}_t$

In papers $S_t \rightarrow C_t$

Selective forget

⇒ How do we combine S_{t-1} and $\tilde{S}_t$ to get the new state

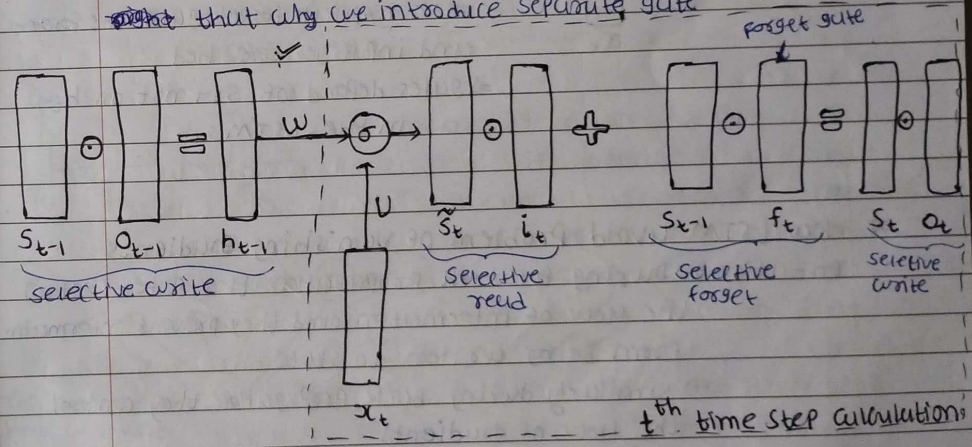
⇒ Here simple way to doing this

$$S_t = S_{t-1} \overset{\odot f_t}{+} i_t \odot \tilde{S}_t$$

⇒ But we may not want to use the whole of S_{t-1} but forget some parts of it

★ ⇒ Here you should have some confusion why we use S_{t-1} why not use h_{t-1} and why should we do selectively forget for this that we have already done in selective write

so the answer is selective write is used to compute how to read the information but now once you have read the new info you want to see how to assimilate (adopt) it back with old info that you had that why we introduce separate gate



⇒ we have full set of eqn for LSTM

Gates:

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

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

$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$

States:

$\tilde{S}_t = \sigma(W h_{t-1} + U x_t + b)$

$S_t = f_t \odot S_{t-1} + i_t \odot \tilde{S}_t$

$h_t = o_t \odot \sigma S_t$

GRU

⇒ The full set of eqⁿ for GRUs

Gates: $o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$

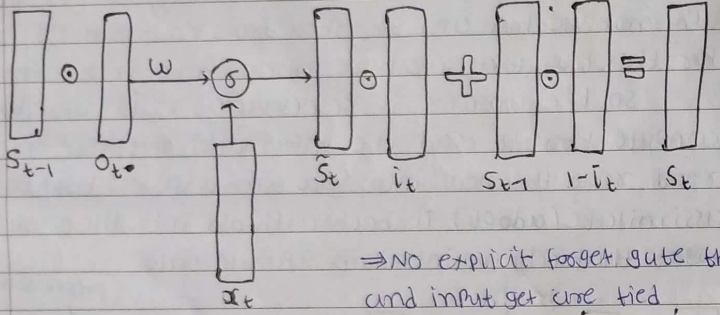
$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$

States:

$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$

$s_t = (1 - i_t) \odot s_{t-1} + i_t \odot \tilde{s}_t$

⇒ GRU is type of RNN that in certain cases, has advantages over LSTM it use less memory and faster than LSTM, however LSTM better accurate when using dataset with longer sequences



⇒ No explicit forget gate the forget and input get are tied,

⇒ gates depend of s_{t-1} not on h_{t-1} as in the LSTMs

★ How LSTM avoid Problem of Vanishing gradients

Intuition:→ During forward propagation the gates control the flow of information and they prevent information from being written to state

→ similarly during back propagation they control the flow of gradients

$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$

Here f_t decides how much goes on forward direction and for backward propa $\frac{\partial s_t}{\partial s_{t-1}} = f_t$ (let neglect 2nd Term this is cause assumption) so f_t also decides how much goes on backward direction.

⇒ Here vanishing occures in forward as well as back ward propa so this types of vanishing is OK. If in forward pass $s_{0,1}$ is not contribut to s_t then for backward pass also s_i not ⇒ In over fitting

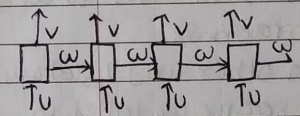
⇒ If state at time t-1 did not contribute much to the state at time t (i.e. If $\|f_t\| \to 0$ and $\|o_{t-1}\| \to 0$) then during backprop the gradient flowing into s_{t-1} will vanish

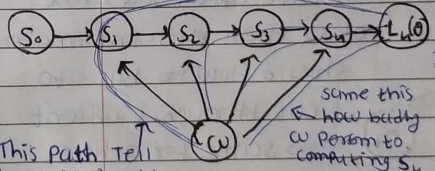
⇒ This kind of vanishing is ok because if s_{t-1} did not contribute to s_t we don't want to hold it responsible for crimes of s_t

⇒ so in LSTM key difference from vanilla RNN is that flow of info and gradient controlled by the gates which ensure that gradients vanish only when they should. (i.e. when s_{t-1} didn't contribute much to s_t)

⇒ Recall RNNs had multiplication terms which caused the gradient to vanish

$$\frac{\partial L_t(\theta)}{\partial W} = \frac{\partial L_t(\theta)}{\partial s_t} \sum_{k=1}^{t} \prod_{j=k}^{t-1} \frac{\partial s_{j+1}}{\partial s_j} \frac{\partial^+ s_k}{\partial W}$$





This path tell how W should be adjusted inorder to reduce loss because of s_1 (Loss occures because W is not good enough to compute good value for s_1)

$\prod_{j=k}^{t-1} \frac{\partial s_{j+1}}{\partial s_j}$ Term represent no number of term involves This path is vanishes with

⇒ In particular, if the loss at $L_4(\theta)$ was high because W was not good enough to compute s_1 correctly then info will not be propagate back to W as the gradient $\frac{\partial L_4(\theta)}{\partial W}$ along this long path will vanish in RNN

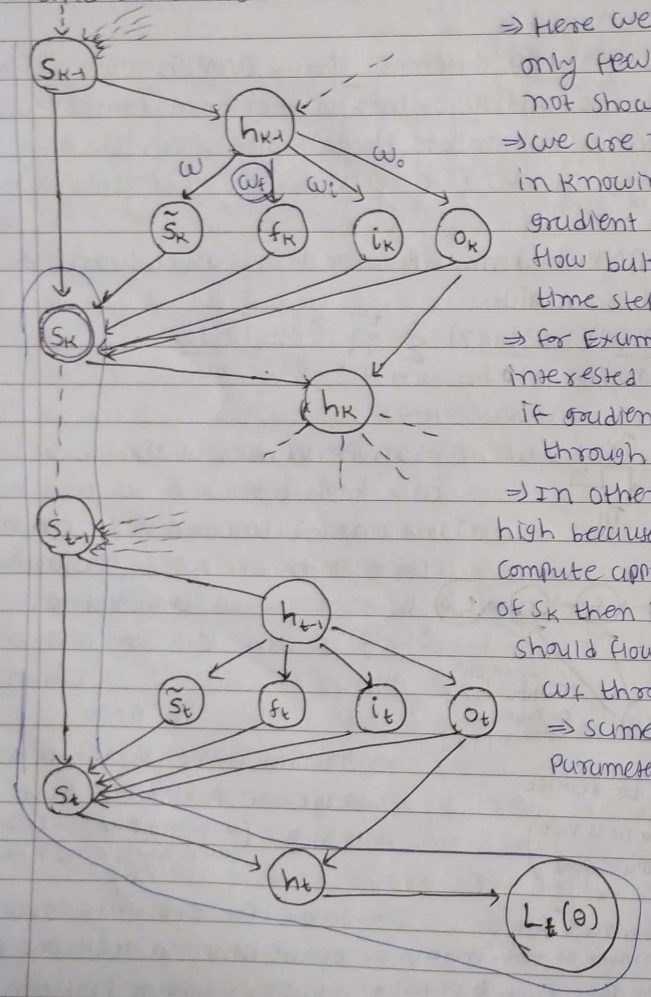
⇒ gradient is not vanish when atleast one path has not vanish (vanishing gradient problem occures when every path vanish)

⇒ same for exploding gradient occure when atleast one path has exploding problem.

in the case

⇒ LSTMs ✿ there exists at least one path through which the gradients can flow (and hence no vanishing gradients)

⇒ Starting from h_{k-1} and s_{k-1} we have reached h_k and s_k and the recursion will now continue till the last timestep



⇒ Here we are showing only few parameters. not showing U_i, U_o, U_f, U_i

⇒ we are now interested in knowing if the gradient from $L_t(\theta)$ flow back an arbitrary time step k

⇒ for Example we are interested in knowing if gradient flows to w_f through s_k

⇒ In other way $L_t(\theta)$ was high because w_f failed to compute appropriate value of s_k then this information should flow back to w_f through gradient

⇒ same for other parameters w_i, w_o, w etc

⇒ There are multiple paths (many many) from $L_t(\theta)$ to s_k (you just need to reverse the direction of arrows for back prop)

⇒ many path like s_{k+1} another path through h_k and further there is multiple path to reach h_k itself.

⇒ let consider one path and let denote gradient along this path to

$$t_0 = \frac{\partial L_t(\theta)}{\partial h_t} \frac{\partial h_t}{\partial s_t} \frac{\partial s_t}{\partial s_{t-1}} \cdots \frac{\partial s_{k+1}}{\partial s_k}$$

This will not vanish because h_t directly connected to $L_t(\theta)$ and there is no intermediate node which can cause gradient vanish

$h_t = o_t \odot s_t$

$h_t = [h_{t1}\ h_{t2} \cdots h_{td}]$

$o_t = [o_{t1}\ o_{t2} \cdots o_{td}]$

$s_t = [s_{t1}\ s_{t2} \cdots s_{td}]$

h_{t2} only depend on o_{t2} & s_{t2}

$\frac{\partial h_t}{\partial s_t}$ matrix contain $\frac{\partial h_{ti}}{\partial h_{tj}}$ so for $i \neq j$ this term zero

and only diagonal term remain where $i = j$

so

$$\frac{\partial h_t}{\partial s_t} = \mathbb{D}(o_t \odot s_t') \text{ diagonal matrix } \in \mathbb{R}^d$$

↑ representation of diagonal matrix with diagonal vector inside $\mathbb{D}$

$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$ ← $\tilde{s}_t$ depends on s_{t-1} so we can't treat as const

⇒ we dealing with ordered network so thus $\frac{\partial s_t}{\partial s_{t-1}}$ will be a sum of an explicit term and implicit term (treating $\tilde{s}_t$ as const)

⇒ now making worst case assumption implicit term vanish so explicit term f_t

⇒ gradient from explicite term is given as $\mathbb{D}(f_t)$

$$t_0 = \frac{\partial L_t(\theta)}{\partial h_t}\frac{\partial h_t}{\partial s_t}\frac{\partial s_t}{\partial s_{t-1}}\cdots\frac{\partial s_{k+1}}{\partial s_k}$$

$$= L_t'(h_t)\, D(O_t \odot S_t')\, D(f_t)\, D(f_{t-1})\cdots D(f_{k+1})$$

$$= L_t'(h_t)\, D(O_t \odot S_t')\, D(f_t \odot \cdots f_{k+1})$$

$$= L_t'(h_t)\, D(O_t \odot S_t')\, D\left(\bigodot_{i=k+1}^{t} f_i\right)$$

- This term does not vanish
- → contain multiple of forget gates
- → The forget gate thus regulate the gradient flow depending on explicit contribution of a state (S_t) to the next state S_{t+1}
- → If during forward pass S_t didn't contribute much to S_{t+1} (because $f_t \to 0$) then during back prop also gradient will not reach S_t

→ This is fine because if S_t did not contribute much to S_{t+1} then during back propagation (f_t does the same regulation during forward pass and back pass which fair)

⇒ so in LSTM gradient will vanish only when required otherwise it will not vanish like RNNs.

★ why LSTM do not solve exploding gradient

⇒ we will see the path through which the gradient can explode

⇒ lets

gradient along this path is $t_1 = \frac{\partial L_t(\theta)}{\partial h_{k-1}}$ which contain this gradient along that path

$$t_1 = \frac{\partial L_t(\theta)}{\partial h_t}\left(\frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial o_{t-1}}\right)\cdots\left(\frac{\partial h_k}{\partial o_k}\frac{\partial o_k}{\partial h_{k-1}}\right)$$

$$= L_t'(h_t)\left[D(s_t \odot o_t')\cdot w_0\right]\cdots\left[D(s_k \odot o_k')\cdot w_0\right]$$

This is diagonal matrix

$$\|t_1\| \le \|L_t'(h_t)\|\,(\|k\|\,\|w_0\|)^{t-k+1}$$

⇒ so depending on norm of w_0 the gradient may explode

→ so to solve this problem popular trick is to use gradient clipping

⇒ while back propg if the norm is exceed a certain value it is scaled to keep its norm within an acceptable threshold.

⇒ This is fine because gradient care about direction not for a magnitude. when gradients are not manageable in terms of their magnitude the we clip it to some manageable while be careful of the direction

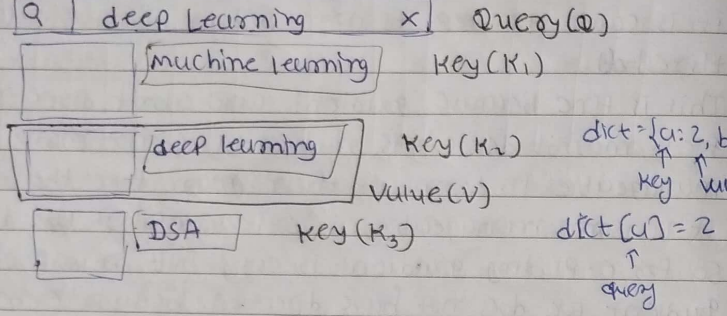
⇒ So for exploding gradient is easy but in case of vanishing gradient we does not have direction because either gradient became zero.

MIT 6.S191 2nd lec 2023

Intuition behind self-Attention

→ Attention is most important part of an input

⇒ If we want to find video on youtube about deeplearning then we have type in search bar



⇒ for every video in database there some key information related to that video its called **Key**

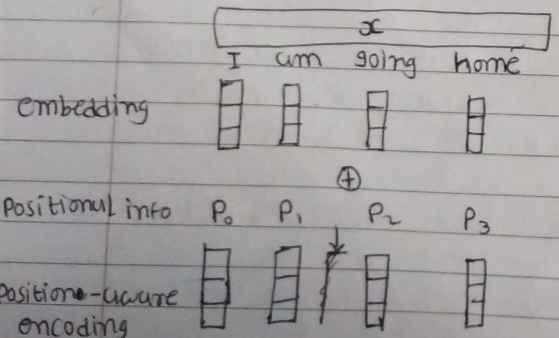
⇒ and now Algorithm will compare how keys and query are related to each other. This also do similarity computation

⇒ Final step now we know which key is relevant so extracting the information what we want to pay attention to. (video)

⇒ Learning self attention with neural networks

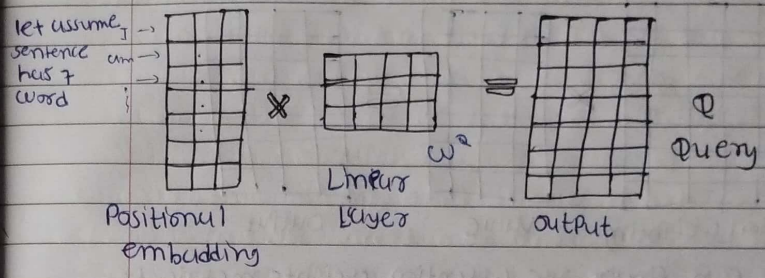
Goal: identify and atted to most important features in input

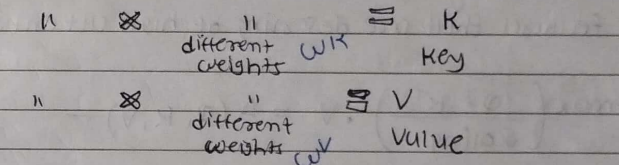
1) incode positional information



Data is fed in all at once! Needs to encode positional info to understand order

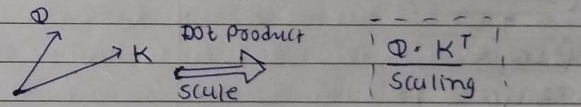
2) Extract query, key, value for search





3) Compute attention weighting

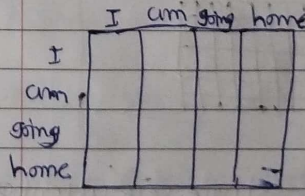
Attention score: Compute pairwise similarity between each query and key



similarity metric (Cosinesimilarity)

Attention weighting: where to attend to!

How similar is the key to the query?

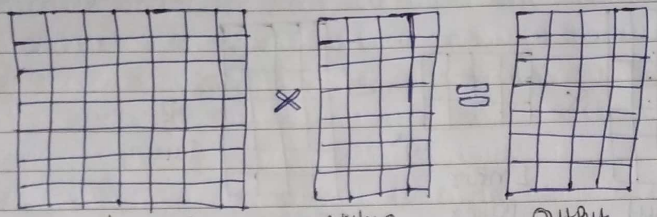


⇒ This operation gives us the score which define how the component of input data related to eachothers.

$$\text{softmax}\left(\frac{Q \cdot K^T}{\text{scaling}}\right)$$

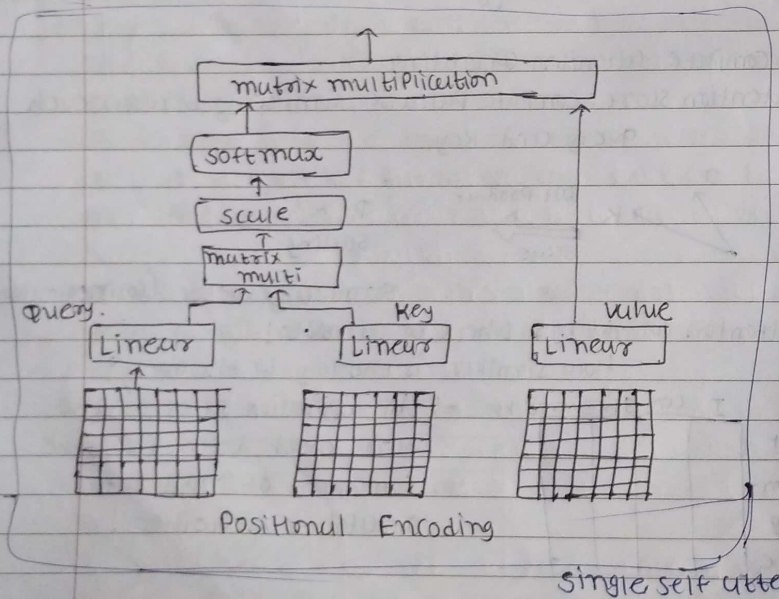
Attention weighting

4) Self-attend to extract features



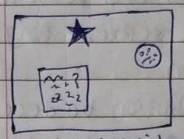
now we can finally use attention weights metric to extract features that are deserving of high attention

$$\text{Softmax}\left(\frac{Q \cdot K^T}{\text{Scaling}}\right) \cdot V = A(Q, K, V)$$

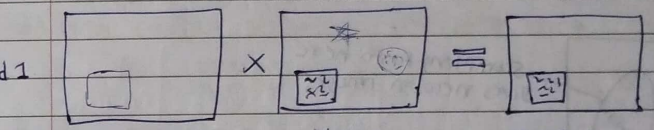


⇒ multiple self attention heads can be linked together to form larger network architecture where different heads trying to extract different informations different relevent

Parts of the ~~information~~ input to now put together a very very rich encoding and representation of data that we have working with.



Let consider this image and applying mutliple self attention heads



others and picking out others part of the data ★ and ⊛ etc.

→ ★ why do we scale self attention?

$$\text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V$$

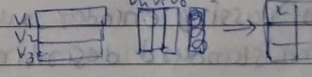
low dimention → dot product → low variance

high dimention → dot product → high variance

d_k = dim of weight matrix

⇒ means distribution of number is different (large range) for high dimention so softmax of it will neglect small values and give more value to big number

$$\text{softmax} = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$



→ and vanishing gradient occures so to reduce high variance to low softmax prob become computible so we need to scale.

1D → Var(x) x = random variable

2D ⇒ 2 Var(x)

3D ⇒ 3 Var(x)

dD ⇒ d Var(x)

we need variance const Var(x) for any dimension

$y \rightarrow \text{Var}(y) \rightarrow 2\,\text{Var}(x)$

$\frac{y}{\sqrt{2}} \rightarrow \frac{1}{2}\text{Var}(y) \rightarrow \frac{1}{2} \times 2\,\text{Var}(x)$

If $Y = cX$

$\text{Var}(Y) = c^2 \text{Var}(X)$

money bank grows

money = 0.7 money + 0.2 bank + 0.1 grows

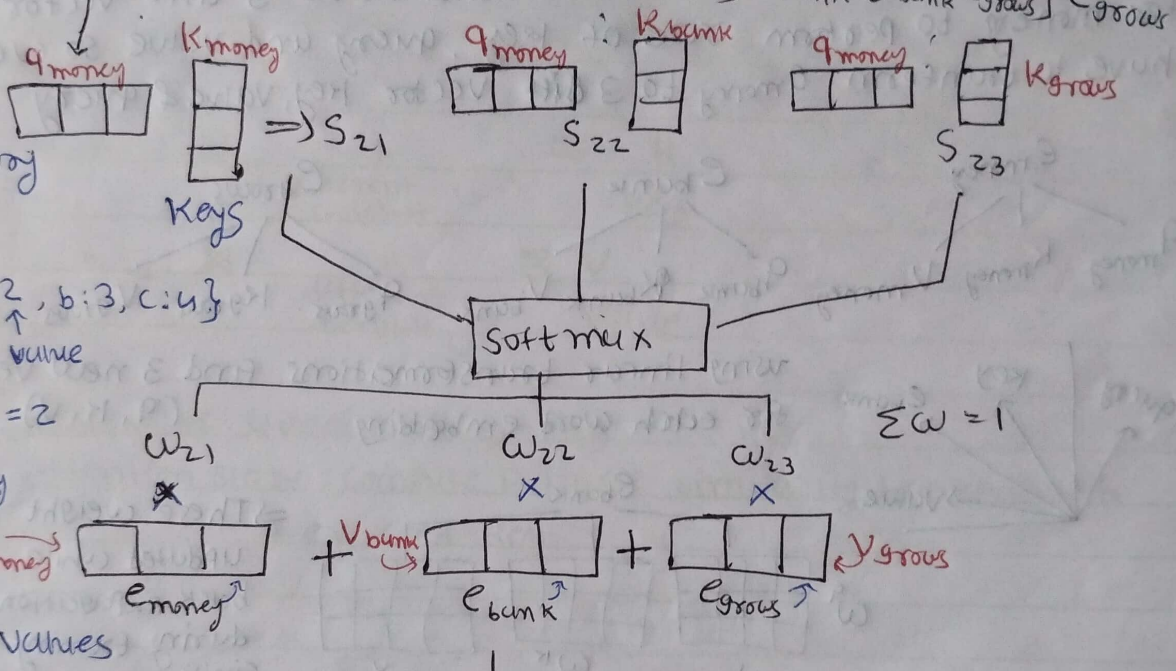
bank = 0.25 money + 0.7 bank + 0.05 grows

grows = 0.1 money + 0.2 bank + 0.7 grows

⇒ In mathematics form

$e^{new}_{money} = 0.7\, e_{money} + 0.2\, e_{bank} + 0.1\, e_{grows}$

$e^{(new)}_{bank} = [e_{bank} \cdot e^{T}_{money}]\, e_{money} + [e_{bank} \cdot e^{T}_{bank}]\, e_{bank} + [e_{bank} \cdot e^{T}_{grows}]\, e_{grows}$



dict

d = {a:2, b:3, c:4}

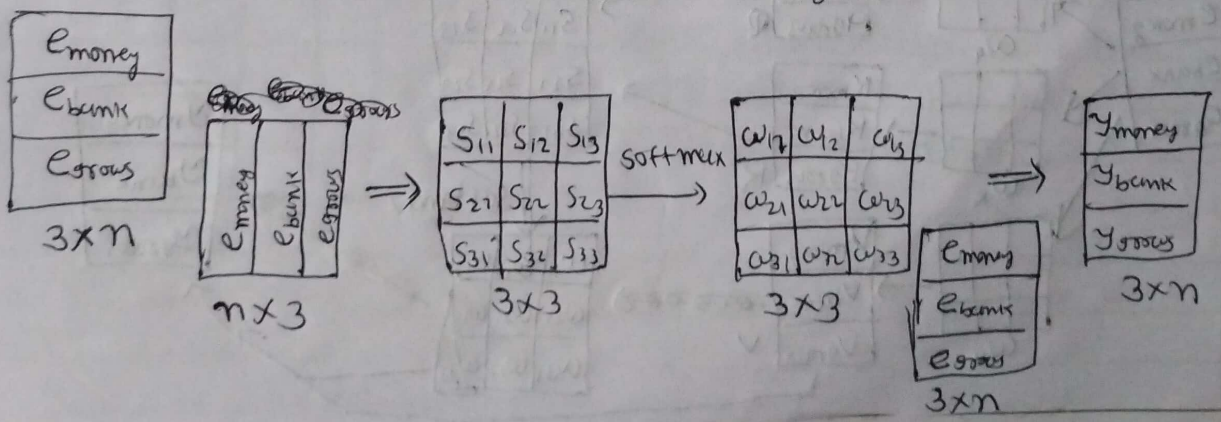
key value

d[a] = 2

query

$e^{new}_{bank} = y_{bank}$ contextual word embedding

same way can find y_{money} & y_{grows}



★ Points to consider

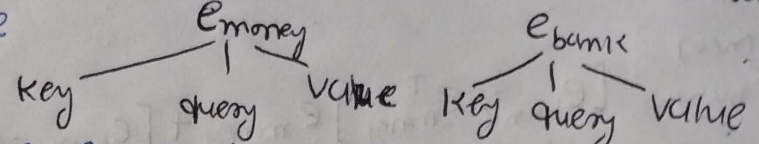
→ This operation is a parallel operation

→ There are no learning parameters involved

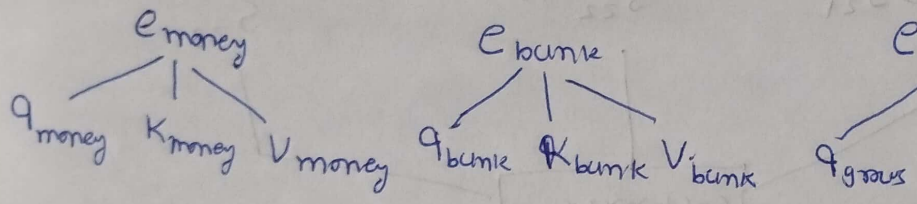
(generated new embedding is not learnt from data that we have)
so this simple attention model give general contextual embedding
but we need task-specific embedding for better result.

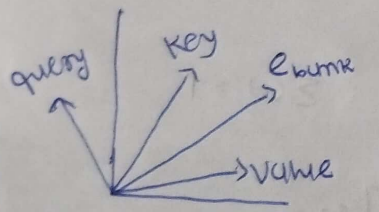
So we introduce w & b so self attention model can learn from data.

⇒ we use each word as 3 type embedding

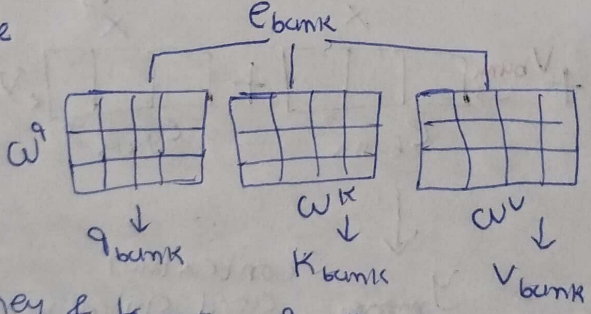


and used same vector for key, query and value but we should have 3 diff vector for money to perform role of key, query and value so we have to tranform e_{money} to 3 diff vector key, value & query



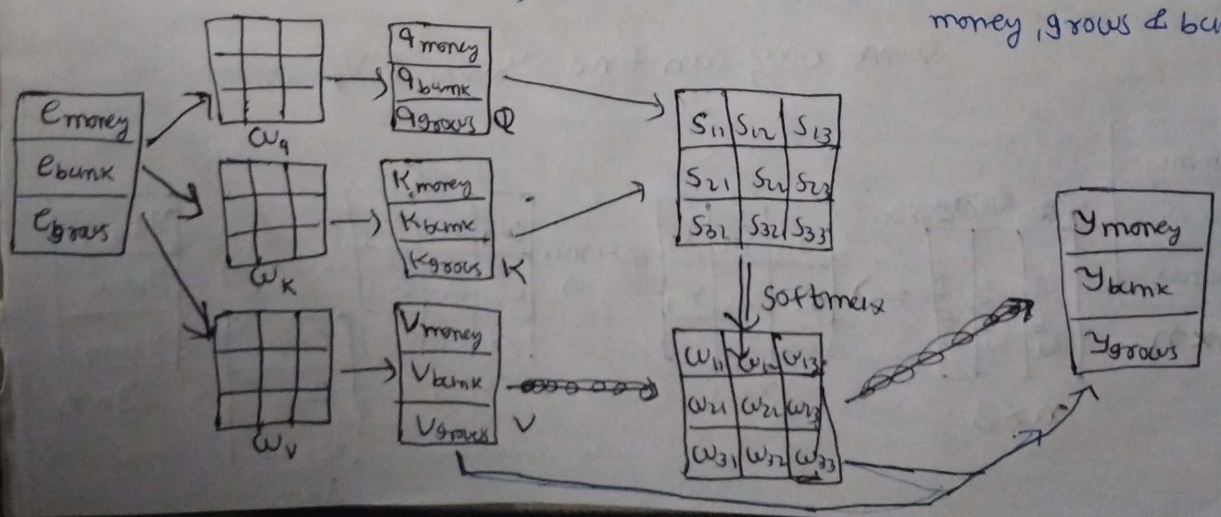


using linear transformations find 3 new vector (q, K, V) for each word embedding



⇒ These weight updated while back propagation during training (initial random values of weight)

same for money & grows, w^q, w^K, w^V are same for all three money, grows & bank



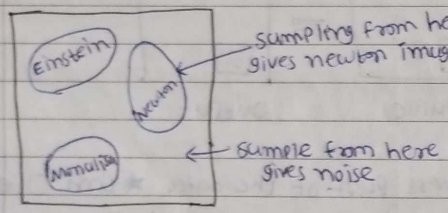
Variational Autoencoder

⇒ idea behind VAE is that instead of mapping any input to a fix vector we map our input into a distribution

⇒ The difference in VAE is that normal bottleneck vector z replace by two seperate vectors one representing the mean of distribution and other representing standard deviation of that distribution

⇒ and know fed a sample from distribution to decoder

⇒ Loss function is contain two terms Reconstruction loss and KL divergence



Latent distribution autoencoder

How to define the latent variable space?

⇒ Instead of mapping input to a fixed latent vector, we map it to a distribution

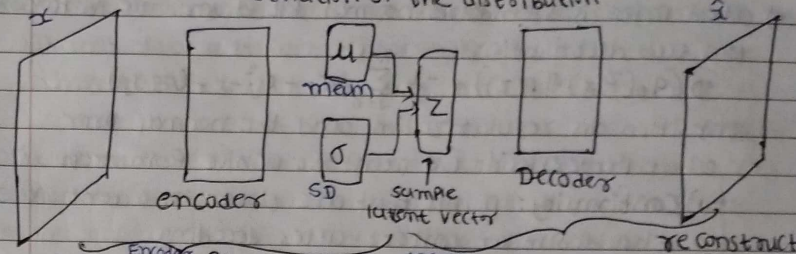
⇒ Force latent variable to be normaly distributed so its easy for us to define normal distribution using mean and variance

⇒ instead of passing encoder output to the decoder, use mean and standard describing the distribution.

⇒ now we have only two variable that need to be trained not entire distribution

⇒ Quantify the distance bet^n learned distri and standard normal distri using Kullback-Leibler (KL) divergence

⇒ so while training we are going to force q_ϕ normal distribution as close as posible to standard normal distribution minimize one of the Loss which is given by KL divergence, and there is other loss is due to reconstruction loss



⇒ Here we use reparameterization trick for back propagation

Learned parameters during back propagation

$z = \mu + \sigma \odot \varepsilon$ ⟷ standerd normal distribution

mean, standerd deviation → Fixed space where we randomly sampling

$\varepsilon \sim N(0,1)$ → not learned during BP.

$L(\phi, \theta, x)$ = reconstruction loss + regularization term

→ like log likelihood, $\|x - \hat{x}\|^2$ $\quad D(q_\phi(z|x) \| P(z))$

inferred latent distr; Fixed prior on latent distr (some initial hypothesis of guess about what the latent variable looks like)

Regularization term capturing dis bet^n encoding of latent variable and prior hypothesis

⇒ So during training we are trying that each of latent variables adopts a probability distr. that similar to that prior

⇒ common choice of prior is Normal Gaussian

$P(z) = N(\mu = 0, \sigma^2 = 1)$

Encourages encoding to distribute encoding evenly around the center of the latent space.

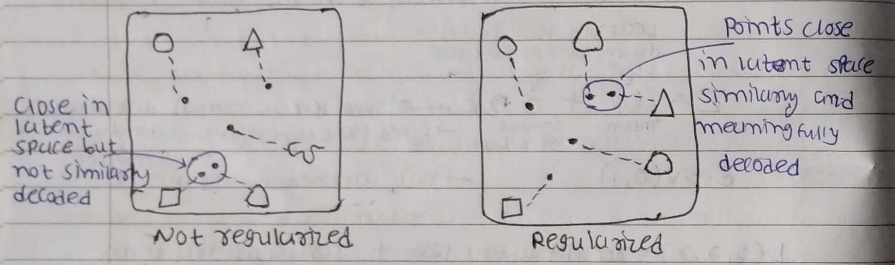
Punalize the network when it tries to cheat by clustering points in specific region (i.e by memorizing the data)

⇒ when prior is standard normal KL divergence between the two distribution

$$D(q_\phi(z|x) \| p(z)) = -\frac{1}{2}\sum_{j=0}^{K-1}(\sigma_j + \mu_j^2 - 1 - \log \sigma_j)$$

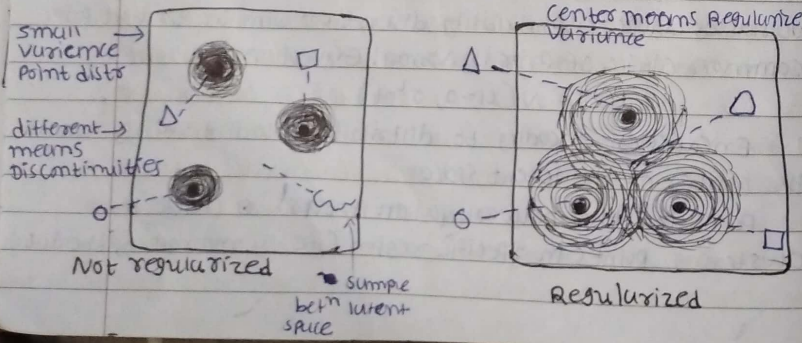
⇒ Intuition on regularization and the normal prior what properties do we want to achive from regularization?

1) **Continuity:** points that are closer in latent space should similar content after decoding
2) **completeness:** sampling from the latent space should have meaningful content after decoding



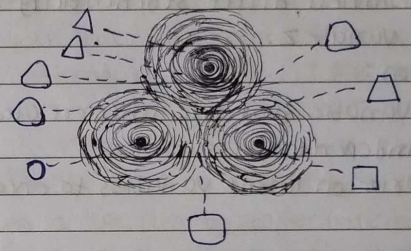
⇒ ~~with~~ with regularization we are able to achive that by ~~trine~~ toining to minimize regularization term its not sufficient to just employ the reconstruction loss along to achieve achieve this continuity and this completeness

⇒ without regularization just encoding and reconstructing does not ~~gua~~ guarantee the properties of continuity and completeness



⇒ Normal Prior allows is for Learned distr. of those latent variables to effectively overlap in latent space because everything is regularized to have according to this prior mean zero standard deviation one and that centers the mean regularizes the variances for each of the independent latent variable distributions.

⇒ by adding effect of Regularization we can achieve continuity and completeness in the latent space

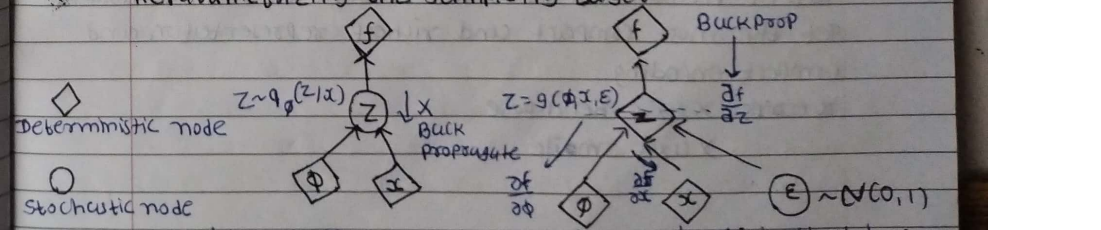


⇒ when we have notion of randomness of ~~th~~ Probability we we can't sample directly through that layer instead with reparametrization what we do is we redefine how a latent variable vector is sampled as a sum of the fix deterministic mean μ a fixed vector of SD σ.

⇒ and now we divert all the randomness all the sampling to a random constant EPsilon ϵ. that drawn from a normal distribution

$$z = \mu + \sigma \odot \epsilon \qquad \epsilon \sim N(0,1)$$

Reparametrizing the sampling layer



⇒ we can't propa ~~back~~ anything that has stochasticity that has randomness

⇒ reparameterization allows us to shift this diagram where now we completely diverted that sampling operation off to the side to this const ε which is drawn from normal prior.

⇒ now when we look back at our latent variable it is deterministic wrt that sampling operation about this mean is that we can back propagate to update neural networks weights completely end to end without worry about direct Randomness, direct stochasticity within those latent variable z.

Latent perturbation:

⇒ we want latent variable that are uncorrelated with eachother and independend

⇒ Enforce diagonal prior on latent variables to encourage independence

⇒ for example slowly increase or decrease a sing latent variable keep all other variable fixed we can see head pose is shifting and this is due to perturbation of single latent variable

⇒ turning the single variable and we can seeing how that particular latent varible affects the decoded reconstruction

⇒ different dimentions of z different interpretable latent features

⇒ we can even multiple latent variable simultaneously compare one agains the others and idealy we want those latent features to be as independent as possible inorder to get the most compact and richest representation and compact encoding

X axis head pose

Y axis smile

Latent space disentanglement with β-VAEs

β VAE loss:

$$L(\theta, \phi; x, z, \beta) = \underbrace{E_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{Reconstructed Term}} - \beta \underbrace{D_{KL}(q_\phi(z|x) \| p(z))}_{\text{Regu. term}}$$

⇒ weighting constant β controls how powerful that regularization term is in the overall loss of VAE

⇒ By increasing value of β you can try to encourage greater disentanglement more efficient encoding to enforce this latent variable to be uncorrelated with each other.

⇒ let look into face reconstruction as a task of interest

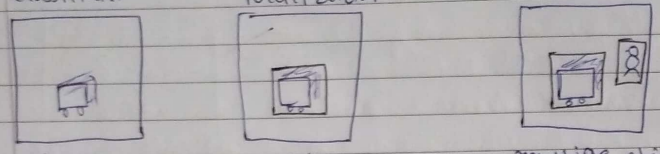
with standard VAE (β=1): pose and rotation of the head correlated with smile. as the head pose is changing current smile is also changing

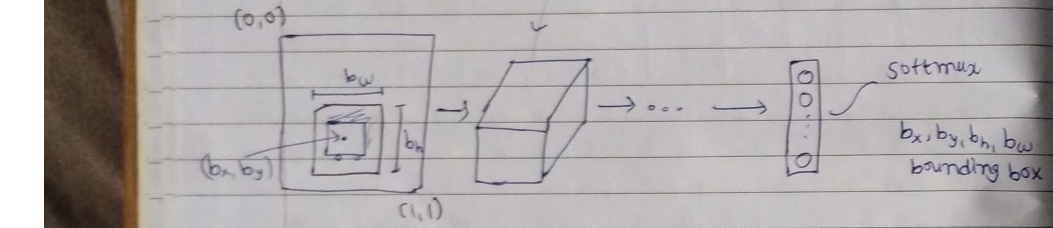
but with β VAE imposing β value much much greater than 1 we can try to enforce greater disentanglement

now we can consider only single latent variable head pose and the smile is constant compared to standard VAE.

Object Detection / localization

Image Classification | Classification with localization | Detection





Define the target lable y

1- pedestrain
2- car
3- motorcycle
4- background ← nothing

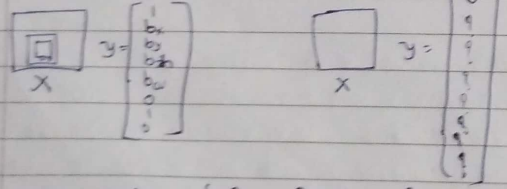
$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

P_c: If there is object class 1,2 or 3

y_2, y_3, y_4: b_x, b_y, b_h, b_w

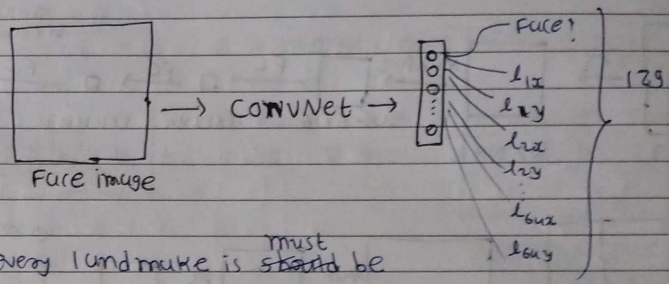
c_1, c_2, c_3: Class 1, 2 or 3

For example



$$\text{Loss} = L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2, & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2, & \text{if } y_1 = 0 \end{cases}$$

⇒ Landmark detection



⇒ Every landmake is ~~should~~ must be consistent in all images

Ex (l_{1x}, l_{1y}) contain left side of eyes corner landmark lable must be consistent

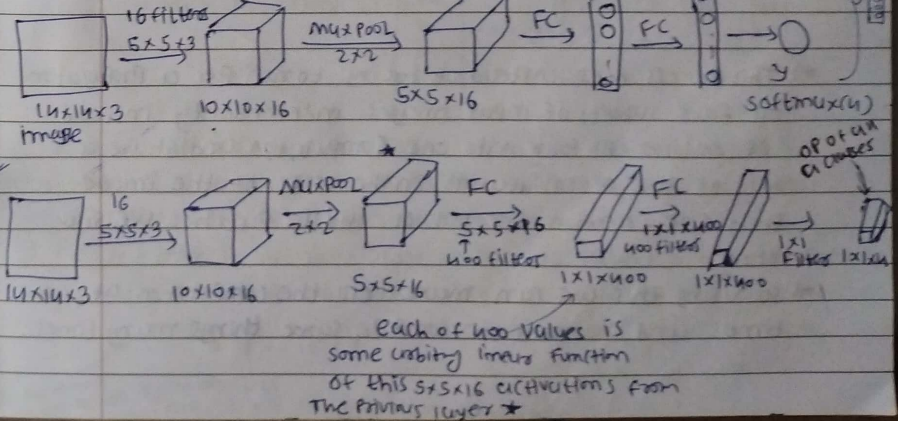
⇒ car detection example

in this we have to give many lubled data for train so we take different size of window and try to detect car in every position of given image (~~the~~ here we use some stride) This also called Sliding window detection.

⇒ disadvantage of sliding window algo is its take much more computation time

⇒ Turning FC layers into conv layers:

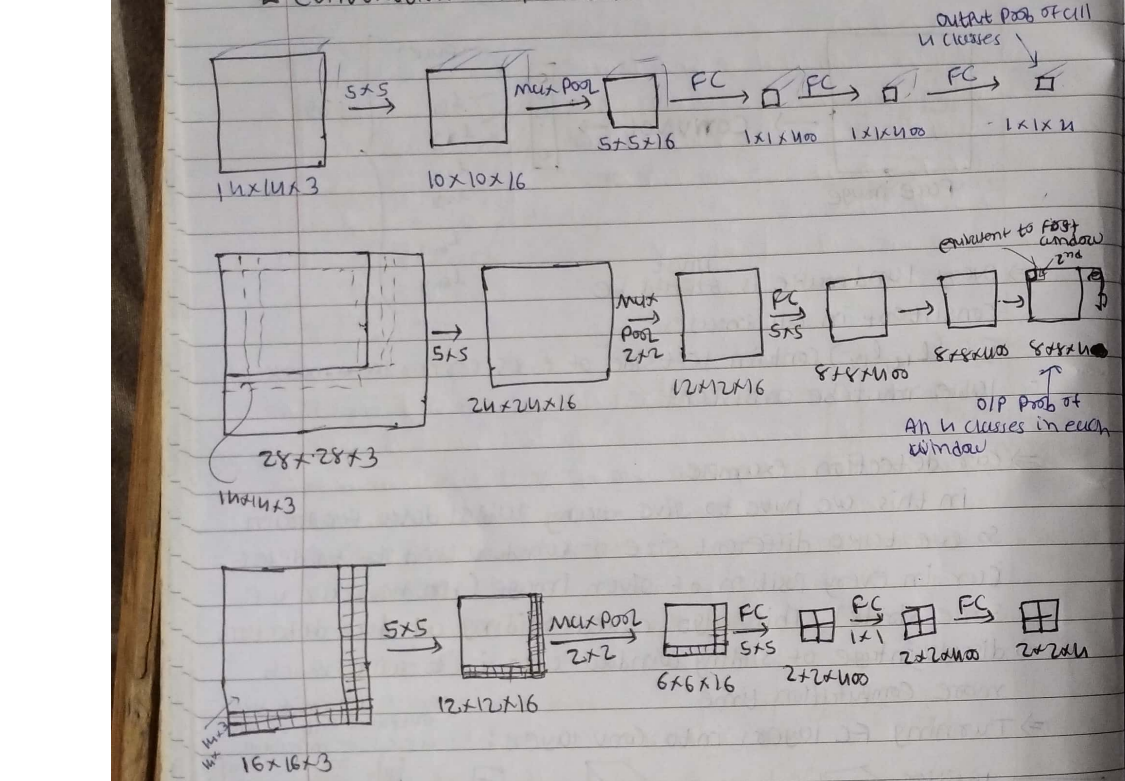
let consider y as output of prob of all 4 classes instead of one op



each of 400 values is some arbitary linear function of this 5×5×16 activations from the previous layer ★

⇒ ? means some numbers since $P_c = 0$ so no need to worry about that numbers that number does not signify anything

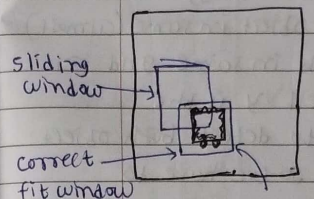
★ convolution implementation of sliding window



⇒ This process does instead of forcing to run for a propagation on four subsets of input images independently instead it combine all four into one computation and share a lots of the computation in the regions of the image that are common for all 4 14×14 patches we saw here.

⇒ so using this we can make all the computation at same times and no need to compute same things many times

★ Intersection over union (IOU)



Intersection over area = $\frac{\text{size of intersection}}{\text{size of union}}$

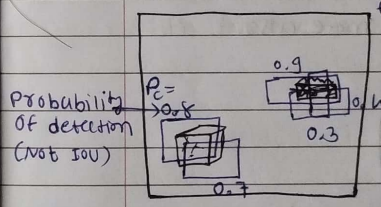
Sliding window is correct for given object if $IOU \geq 0.5$

Threshold value can take according to your choice

★ Non-max suppression

⇒ algo may find multiple detection of same object non-max suppression is a way to make sure that your algo detect the object only ones

⇒ take largest P_c box and highlight that from remaing who has higher IOU ($\geq$ Threshold) with highlighted suppose them as darker



⇒ let take example for only one object detection : car detection

Each Prediction is $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

Algorithm i) discard all boxes with $P_c \leq 0.6$ (low prob of box) ← Threshold

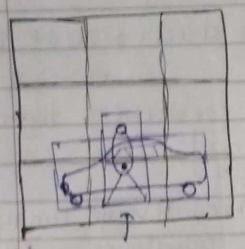
ii) while there is a any remaining boxes:

→ Pick the box with largest P_c output that as prediction (highest box)

→ Discard any remaining box with IOU ≥ 0.5 with the box output in the previous step.

Algo for same multiple object Ex many cars only car not car and pick both

★ overlapping objects — c_1 = pedestrian
c_2 = car
c_3 = background (nothing)



This grid contain mid point of both

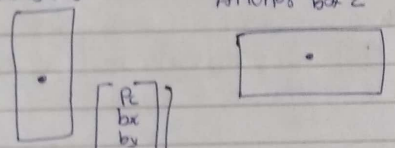
⇒ mid point of both object in same (almost) location and both in same grid cell
⇒ For that grid cell y is the not be able to detect both object

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

so we have to pick one of the two ~~detections~~ detection as output

⇒ with the idea of Anchor boxes we going to predefine two different shapes called Anchor boxes.
⇒ And now we able to associate two predictions with the two anchor boxes and can use more also

Anchor box 1 Anchor box 2

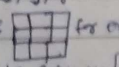


$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ \vdots \\ c_3 \end{bmatrix}$$

Anchor box 1
Anchor box 2

Previously: Each object in training is assigned to grid cell that contains that object's midpoint

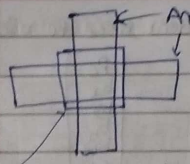
O/P $y = 3 \times 3 \times 8$

(where  for every box gride cell $y = \begin{bmatrix} p_c \\ b_x \\ c_3 \end{bmatrix}$ size 8)

with two anchor boxes:
Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for grid cell with highest IOU.

O/P: $y = 3 \times 3 \times 16$

Anchor box 2×8 (y size)



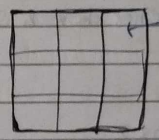
(anchor box) we see which have higher IOU with respect to grand truth bounding box and we assign that object to grid size and anchor box

For training labeled output

O/P $y = 3 \times 3 \times 16$

grid

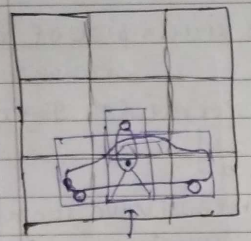
⇒ two anchor boxes with 3 object in same cell: this case is not handle by only two anchor box
⇒ two object in same cell and have same anchor box size: this case also does not handle by this algorithem

with Anchor box  every box has O/P

$$y = \begin{bmatrix} p_c \\ c_3 \\ p_c \\ c_3 \end{bmatrix}$$

anchor box 1
anchor box 2

★ Overlapping objects — c_1 = pedestrian
c_2 = car
c_3 = background (nothing)



This grid contain mid point of both

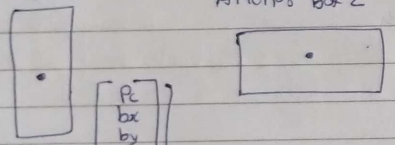
⇒ mid point of both object in same (almost) location and both in same grid cell
⇒ For that grid cell y is the ~~for~~ not be able to detect both object

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

so we have to pick one of the two ~~detections~~ detection as output

⇒ with the idea of Anchor boxes we going to predefine two different shapes called Anchor boxes.
⇒ And now we able to associate two predictions with the two anchor boxes and can use more also

Anchor box 1 Anchor box 2



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ \vdots \\ c_3 \end{bmatrix}$$

Anchor box 1
Anchor box 2

Previously: Each object in training is assigned to grid cell that contains that object's midpoint

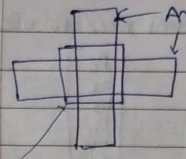
O/P $y = 3 \times 3 \times 8$

(there  for every box grid cell) $y = \begin{bmatrix} p_c \\ b_x \\ \vdots \\ c_3 \end{bmatrix}$ size 8

with two anchor boxes: Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for grid cell with highest IOU

O/P: $y = 3 \times 3 \times 16$

Anchor box 2×8 for y size



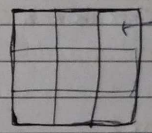
we see which have higher IOU with respect to ground truth bounding box and we assign that object to grid size and anchor box

For training labeled output

O/P $y = 3 \times 3 \times 16$

⇒ two anchor boxes with 3 object in same cell: this case is not handle by only two anchor box
⇒ two object in same cell and have same anchor box size: this case also does not handle by this algorithem

with Anchor box



every box has o/p

$$y = \begin{bmatrix} p_c \\ c_3 \\ p_c \\ c_3 \end{bmatrix}$$

p_c, c_3 — anchor box 1
p_c, c_3 — anchor box 2

★ YOLO Algorithm

C_1 = Pedestrian
C_2 = car
C_3 = motorbike

dont need to use background class now

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ \vdots \\ 0 \\ ? \\ ? \end{bmatrix}$$

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ C_3 \\ P_c \\ b_x \\ b_y \\ \vdots \\ C_3 \end{bmatrix}$$

Training labeled (given in training op table)

Anchor box 1

Anchor box 2

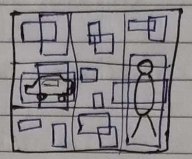
$$y = \begin{bmatrix} 0 \\ ? \\ \vdots \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

anchor box 2

⇒ Training labled has higher IOU with Anchor box 2 so we choose anchor box 2

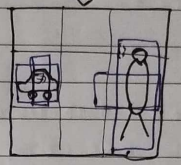
Image 100×100×3 → ConvNet → 3×3×16

★ Outputting the non-max supressed output

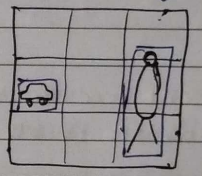


⇒ For each grid cell, get 2 predicted bounding boxes

⇒ get rid of low prob predictions. For object



⇒ For each class (Predestrian, car, bick) use non max suppression to generat final prediction



Reinforcement Learning

MIT 2020
MIT 6.S191 Reinforcement learning
57.33 min

Supervised Learning	unsupervised Learning	Reinforcement Learning
Data: (x, y)	Data: x	Data: State-Action pair
x is data y is label	no labels	
Goal: Learn function to map $x \to y$	Goal: Learn underlying structure	Goal: maximize future rewards over many time step
Ex 		
This is apple	This thing is like other things	Eat this thing because it will keep you alive

Key Concepts:

Agent: take actions

Environment: The world in which the agent exists and operates

Actions: a move that agent can make in environment

Action space A: The set of the all possible actions that agent can make in environment

Observation: of environment after taking actions.

State: a situation which agent perceives

Reward: feedback that measures the success or failure of the agent's action.

Total Reward (Return) $R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + r_{t+2} + \dots$

Discounted Total Reward (Return) $R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} + \dots + \gamma^{t+n} r_{t+n} + \dots$

γ: discount factor

$0 < \gamma < 1$

greedyness

⇒ γ is designed for make future rewards essentially worth less than rewards that we might see this instance

⇒ Total reward R_t is discounted sum of all rewards obtains from time t

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$$

$$Q = (s_t, a_t) = E[R_t / s_t, a_t]$$

(state, action)

⇒ Q function captures the expected total future reward an agent in state s, can receive by executing a certain action, a

⇒ Ultimately the agent needs a policy $\pi(s)$, to infer the best action to take at its state s

Strategy: the policy should choose an action that maximizes future reward

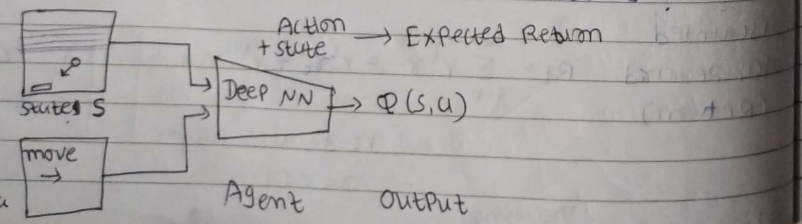
$$\pi^*(s) = \arg\max_a Q(s,a)$$

Value Learning	Policy Learning
Find $Q(s,a)$	Find $\pi(s)$
$a = \arg\max_a Q(s,a)$	Sample $a \sim \pi(s)$

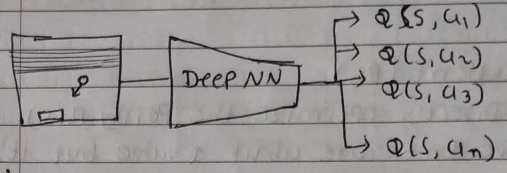
Value Learning algo

Deep Q Networks (DQN) ← approximate Q fun and use to infer the optimal Policy $\pi(s)$

How can we use deep neural networks to model input Q-functions ?



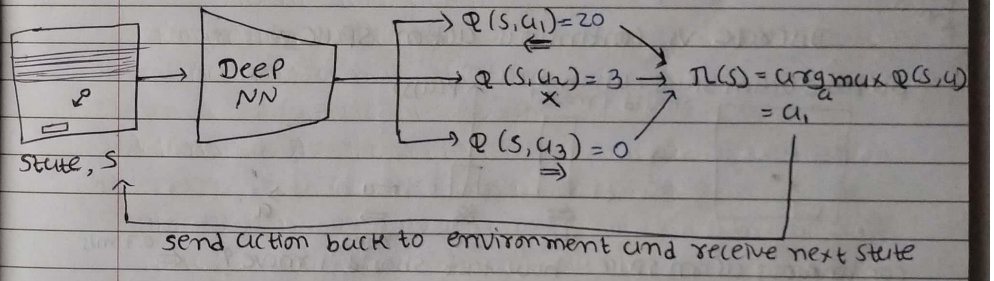
State: → Expected return for Each action



network Predicted

$$L = E\left[\left\| \underbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}_{\text{target}} - \underbrace{Q(s,a)}_{\text{Predicted}} \right\|^2\right]$$

target: initial reward; $\max_{a'}$: action that maximize expected return; Predicted: Q-Loss



★ Downsides of Q-Learning

take only those action which give best result at that time so it repeat same action and don't learn diff actions

Complexity: can model scenarios where the action space is discrete and small
cannot handle continuous action spaces

Flexibility: Policy is deterministically computed from the Q function by maximizing the reward
→ cannot learn stochastic Policies

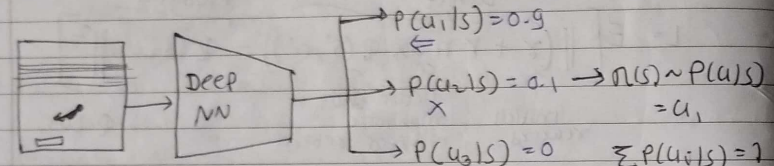
⇒ To address these consider a new class of RL training algorithms: Policy gradient methods

Policy learning algorithm

Policy Gradient (PG)

Policy gradient: Directly optimize the policy $\pi(s)$

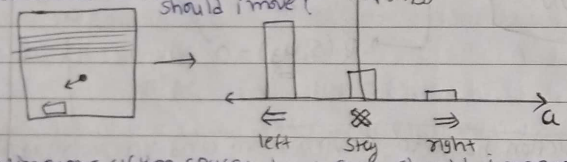
⇒ in this approach we don't care about a value but it's give probability of selecting that action gives desirable output



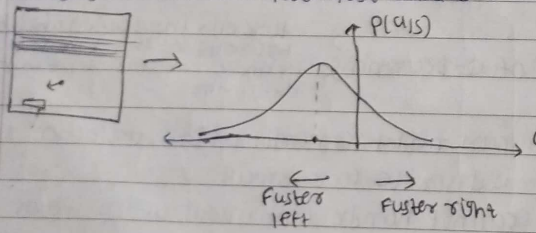
Easy to train than q fun

Discrete vs continuous action spaces

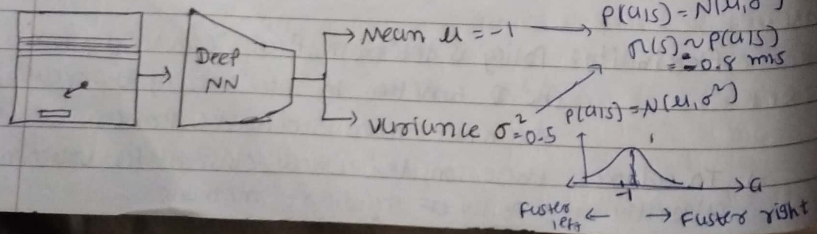
discrete action space: which direction should i move! P(a|s)



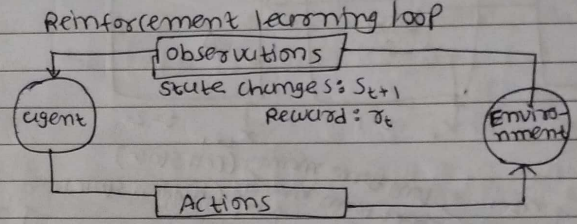
continuous action space: how fast should i move? ⇐ 0.7 m/s



$\int_{-\infty}^{\infty} P(a|s) = 1$



⇒ Training policy gradients



case-study: self driving cars

Agent: vehicle
state: camera, lidar, etc
Action: steering wheel angle
Reward: distance traveled

Training algorithm:

1) initilize the agent
2) Run a policy until termination
3) Record all state, actions rewards
4) decrease prob of action that resulted in low reward
5) increase prob of " " " " high "

$$loss = -\log P(a_t|s_t) R_t$$

log-likelihood of action — reward

gradient decent update

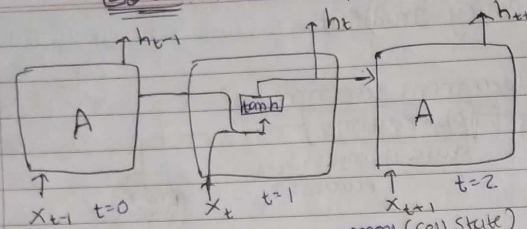
- $w' = w - \nabla loss$

$$w' = w + \nabla \log P(a_t|s_t) R_t$$

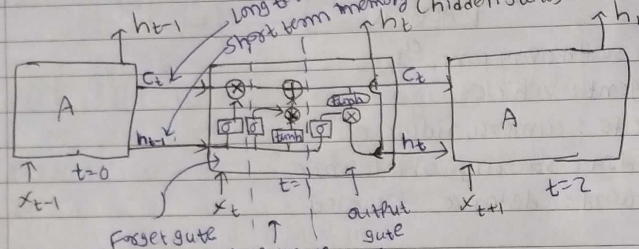
LSTM & GRU

7-10-23

RNN:



LSTM:



Remove information which is not required base of current input

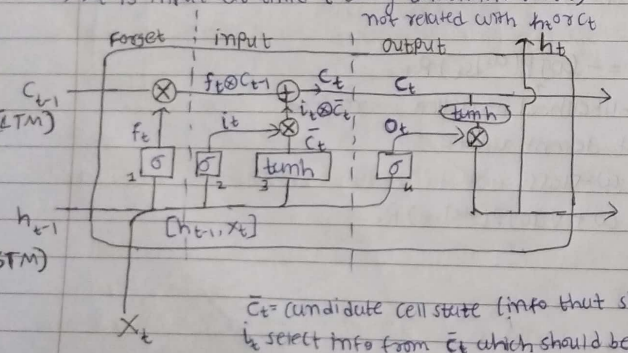
Add info base on current input

base on curr input decide what should be output

⇒ h_t & C_t is vector

dimention of h_t & C_t is same also $h_t, C_t, f_t, i_t, \bar{C}_t, O_t$ has same dimension

⇒ x_t is input at time t (any dimention vector)



$\bar{C}_t$ = candidate cell state (info that should be added in LTM)

i_t select info from $\bar{C}_t$ which should be add to LTM

⇒ σ / tanh neural network layers with activation function σ / tanh

No of node for every neural network 1→4 is same

NN has only one HL (neural network layer)

⊗ ⇒ [4 5 6] ⊗ [1 2 3] ⇒ [4 10 18]

⊕ ⇒ " ⊕ " ⇒ [5 7 9]

tanh ⇒ [1.2 3]

⇒ [tan1 tan2 tan3]

⇒ there are three pointwise operation ⊗, ⊕, tanh

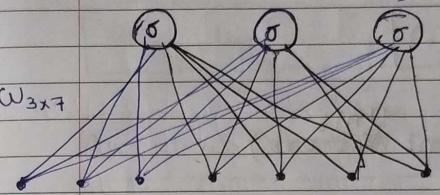
⇒ calculation for forgot gate: (let assume node = 3 & input dim = 4)

f_1 f_2 f_3 ⟶ $[f_1\ f_2\ f_3]$

$[b_1\ b_2\ b_3]$ $(3\times1) + (3\times1) = (3\times1)$ (b, f)

$(3\times7) \times (7\times1) = (3\times1)$

21 weight + 3 bias

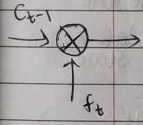


$$f_t = \sigma\left(W_f [h_{t-1}, x_t] + b_f\right)$$

3×

(3×7) (7×1) (3×1)

$(3\times1) + (3\times1) \longrightarrow (3\times1) \xrightarrow{\sigma} (3\times1)$



Forgot some info

(add info)

⇒ Calculation For input gate:

$\bar{C}_t$ is calculate same as above but use tanh activation fun

$$\bar{C}_t = \tanh\left(W_c [h_{t-1}, x_t] + b_c\right) \quad (3\times1)$$

i_t same as above $\quad i_t = \sigma\left(W_i [h_{t-1}, x_t] + b_i\right)$ (3×1)

Pointwise operation of $\bar{C}_t$ & i_t ⇒ $\bar{C}_t \otimes \bar{i}_t$

Now For C_t do ⊕ for $f_t \otimes C_{t-1}$ & $i_t \otimes \bar{C}_t$

$$C_t = f_t \otimes C_{t-1} \oplus i_t \otimes \bar{C}_t \quad (3\times1)$$

⇒ Calculation for output gate (OP for current time stamp h_t)

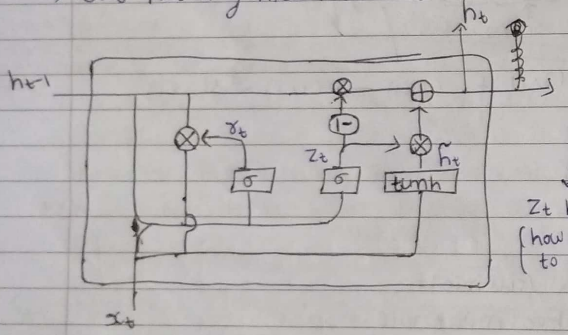
$$o_t = \sigma[W_o [h_{t-1}, x_t] + b_o]$$
3×1

$$h_t = o_t \otimes \tanh(C_t)$$
3×1

★ GRU (Gated Recurrent Unit)

⇒ LSTM has very high number of params so training is slow so GRU is use for fast training.

⇒ GRU has only hidden state h_t



(reset dim of which is not zero according to curr input)

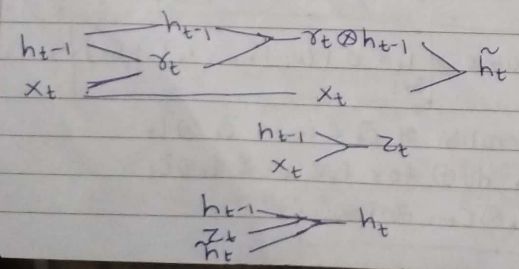
r_t: reset gate

z_t: update gate

$\tilde{h}_t$: candidate hidden state

z_t balance h_{t-1} & $\tilde{h}_t$ (how much importance should give to $\tilde{h}_t$ for calculate h_t)

⇒ $h_{t-1}, h_t, r_t, z_t, \tilde{h}_t$ has same dim



⇒ Choice betⁿ GRU & LSTM is depends on dataset and task try both and find best one

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

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

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

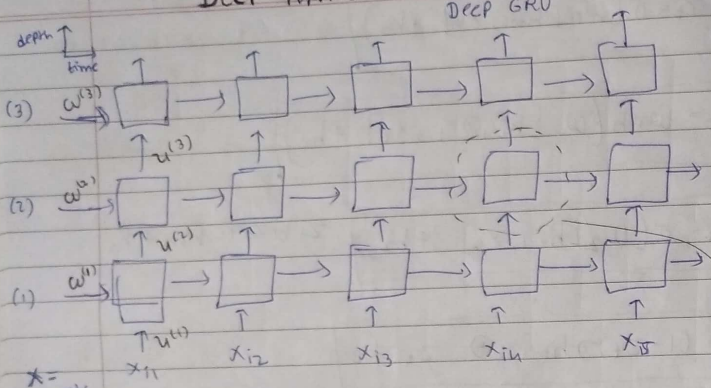
z_t balance h_{t-1} & $\tilde{h}_t$

$$h_t = (1 - z_t) \otimes h_{t-1} \oplus z_t \otimes \tilde{h}_t$$

LSTM	Vs	GRU
⇒ 3 gates: input (update), output, forgot		⇒ 2 gates: reset, update
⇒ cell state (C_t) hidden state (h_t)		⇒ hidden state (h_t)
⇒ more parameters for input size d & hidden size h LSTM has params: $4 \times ((d \times h) + (h \times h) + h)$ $+ (h \times h) + h)$		⇒ less params $3 \times ((d \times h) + (h \times h) + h) 3 \times ((d \times h) + (h \times h) + h)$
⇒ due to extra gates Computationally intensive		⇒
⇒ LSTM give better result the GRU		

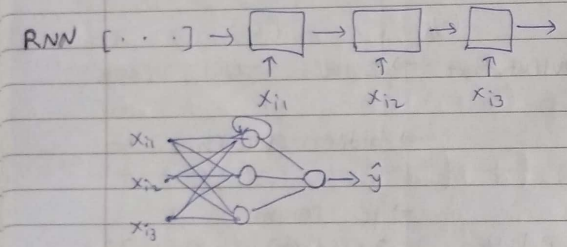
Deep RNNs

Deep LSLM
Deep GRU



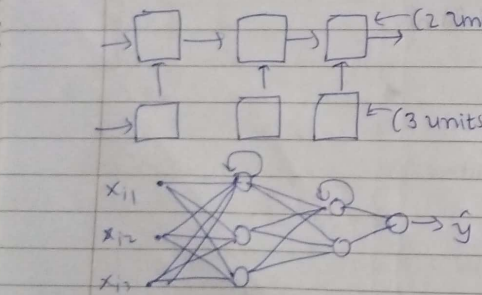
x = 5 words

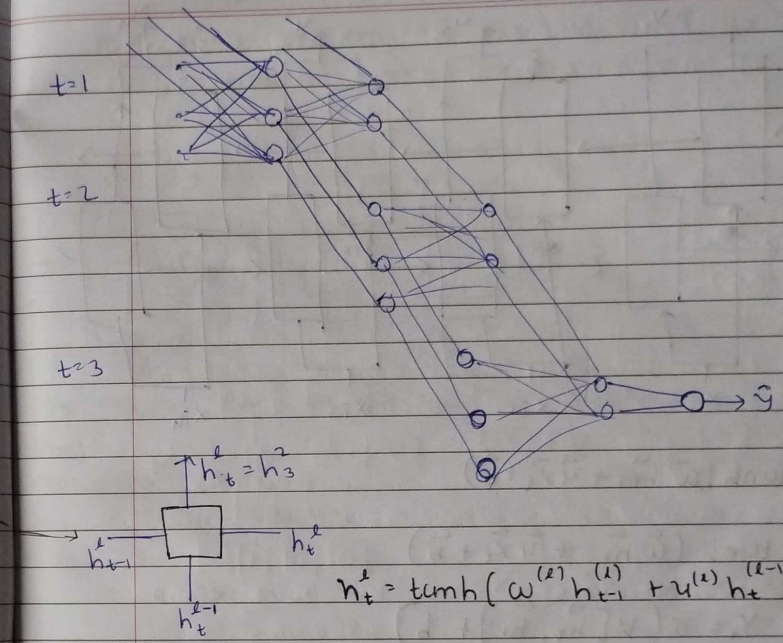
RNN [· · ·]



Paragraph Level
Sentens level
word level

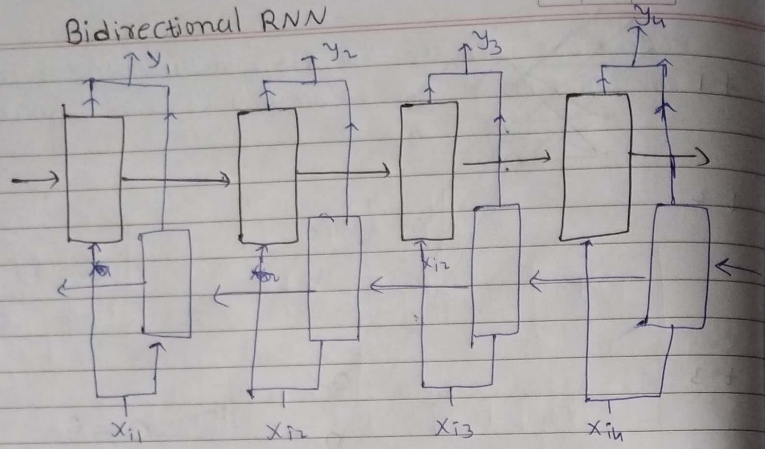
Deep RNN





$$h_t^l = \tanh\left(w^{(l)} h_{t-1}^{(l)} + u^{(l)} h_t^{(l-1)} + b\right)$$

Bidirectional RNN



$$\vec{h}_t = \tanh(\vec{w}\vec{h}_{t-1} + \vec{u}\vec{x}_t + \vec{b})$$

$$\overleftarrow{h}_t = \tanh(\overleftarrow{w}\overleftarrow{h}_{t+1} + \overleftarrow{u}\overleftarrow{x}_t + \overleftarrow{b})$$

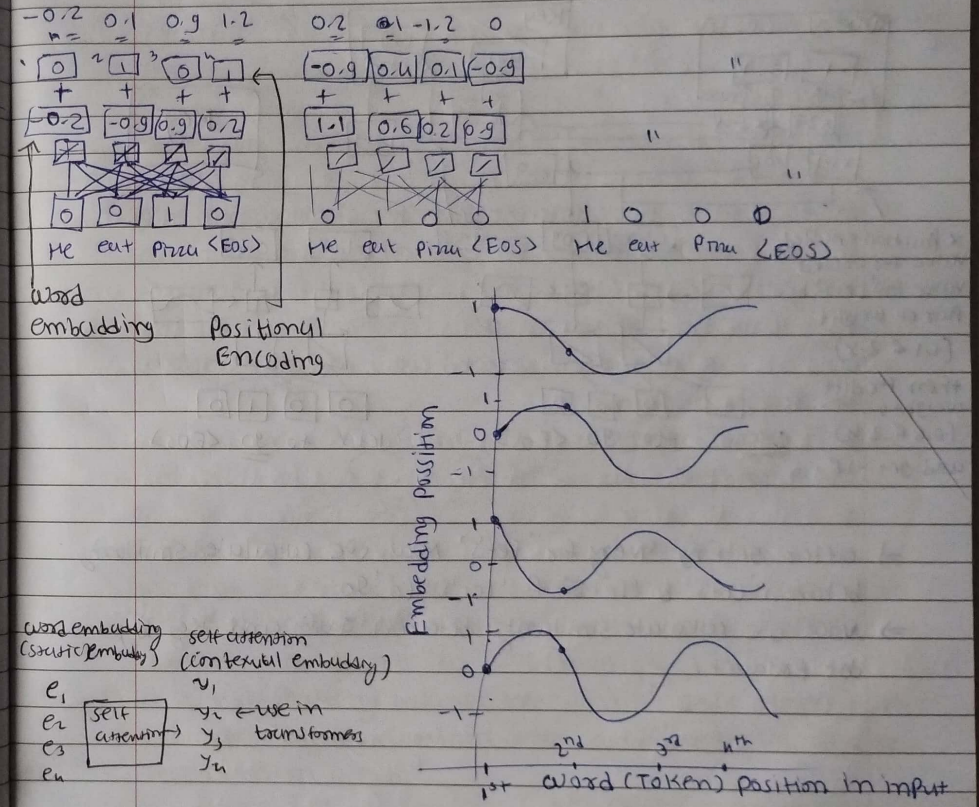
$$y_t = \sigma\left(V\left[\vec{h}_t, \overleftarrow{h}_t\right] + b\right)$$

V: weight, $[\vec{h}_t, \overleftarrow{h}_t]$: concatinated, b: bias

Transformers

Transformers

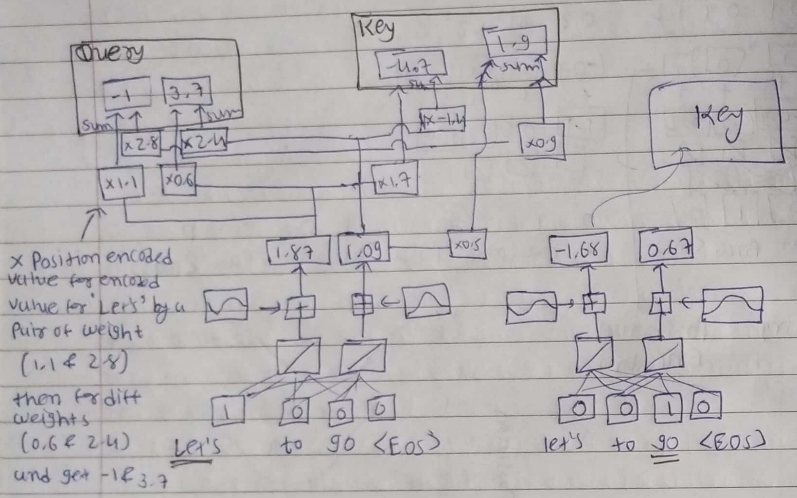
Position Encoding: position encoding allows a Transformer to keep track of word order.



Self Attention: self attention works by seeing how similar each word is to all of the words in the sentence including itself.

The pizza came out of the oven and it tasted good!

⇒ self attention calculate similarity of the 'The' & all of the words in sentense. same for 'Pizza' & all the words



⇒ after getting query for 'let's' now we calculate similarity between 'Let's' to 'Let's' & 'Let's' to 'go'

⇒ Now we calculate similarity between query and Key by dot product.

DeepLearning.AI

Sequence models

deeplearning.ai andrew ng

20Nov23

Week 2

Word to Vec

Week 2
w2vec

★ why Problem with softmax classification

target word $\theta_t^T e_c$ ← embedding vector

$$P(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$$

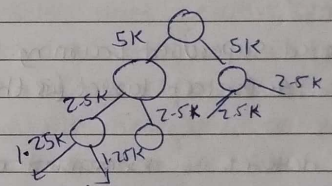
vocabulary size → 10000

Here Problem is : we need to calculate this complex calculation every time

For given context vector c
Predict target word vector in range ± window size

⇒ There are many solution one is Hierarchical softmax classifier

⇒ This Hierarchical soft max classifier instead of trying to categorize something into all 10000 carries on one go image you have one classifier which tell target word is in first 5000 or second 5000 and second classifier tell word in $\overleftarrow{2500}$ or $\overrightarrow{2500}$ & third $\overleftarrow{2500}$ or $\overrightarrow{2500}$ .....



until leaf node you get down to what word is exactly it is.

so each step it is binary classifier so we do not need to some all of 10000 at denominator and complexity become Vocab size to log (vocab size)

⇒ it is not Perfectly balance tree in practice tree has top leave node as common words and deeper leafs are less common words.

Negative Sampling

In this algo we creat new supervise learning problem given a pair of word are are going to predict is this a contex target pair?

	Context word (X)	target (Y)
context c →	orange ← c	juice ← t (target) 1 ← y
K=2	orange	king 0
	orange	book 0

I want a glass of orange juice to go along with my cereal.

so for create this dataset we choose word within ± window of word orange so for that target is 1.

→ For target 0 we choose random word from dictionary and so probabilly that is not in windows and target for this is 0.

→ and its ok if random selected word is from dictionary but we target 0.

→ so now we have created supervised Learning problem for given contex & word to predect target (is those words can put togather?)

K = 5-20 for small dataset

K = 2-5 large dataset

K = negative to positive exam ratio

Model: Softmax

$$P(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j}^{10000} e^{\theta_j^T e_c}}$$

$$P(y=1|c,t) = \sigma(\theta_t^T e_c)$$

orange 6257 → O_{6257} → E (embedding matrix) → e_{6257} → 0, 0, 0, 1, 0 (embedding vector)

O_{6257}: one hot vector

so by creating this now we are not predicting all for given word orange target word is juice or not? and so on for whole dictionary 10,000

So that have 10000 binary regression classifier but insted of all 10,000 of them every iteration we are only going to train K of them (here K=2) + target 1

⇒ so instead have 10,000 way giant softmax which is very expensive to compute we are instead turn it into 10000 binary classification problem each of very cheap to compute and every iteration we are only going to train K+1 of them. which is relatively cheap to do (negative example, positive example). on every iteration ruthe thum updating 10,000 way softmax classifier

This technique called Negative sampling.

⇒ How to select negative word?

one thing we can do is sample according to emphirical frequency of the word in corpus but problem is with that is that you end up with a very high representation of the, of, a, an...

→ one other thing we can do is $\frac{1}{|\text{vocab size}|}$ sample negative example uniformly random But that also very non representative of distribution

→ so researcher famd some heuristic value which is little bit in beteween the two extrimes of sumpling from the imperical frequencies. meaning from whatever is the observed distribution in english text to the uniform distribution. sample proportional to their frequencies of the word to the power of 3/4.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j}^{10000} f(w_j)^{3/4}}$$

w_i = freqn of word in corpus

Glove (Global vector for word representation)

I want a glass of orange juice to go along with my cereal

X_{ij} = number of times word j appers in contex of i

X_{ij} is caunt captures how often do words i & j appear with each other

Glove model:

$$\text{Minimize} \sum_{i=1}^{10000} \sum_{j=1}^{10000} f(x_{ij}) \cdot (\theta_i^T e_j + b_i + b_j' - \log x_{ij})^2$$

weighting term: $=0$ if $x_{ij}=0$; θ_t^T & e_c

Solve for parameter θ_i^T & e_c with using gradient decent to minimize above eqn

$f(x_{ij}) = 0$ when $x_{ij} = 0$

also $f(x_{ij}) \geq 0$ for stop words, this, is, a, an, the....

so this weighting function is function which give less weight to more freqn word and more to less freqn word.

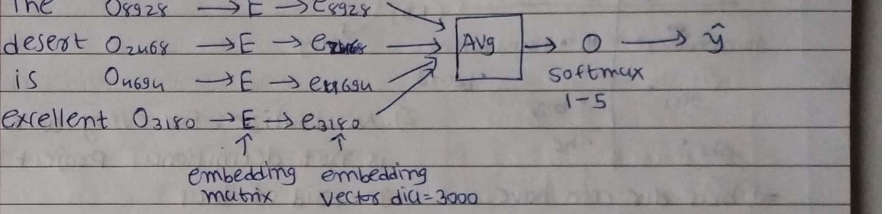
⇒ In this algo role of θ_i and e_j are completly symmetric

Sentiment classification model

21Nov23

The desert is excellent

8928 2468 4694 3180

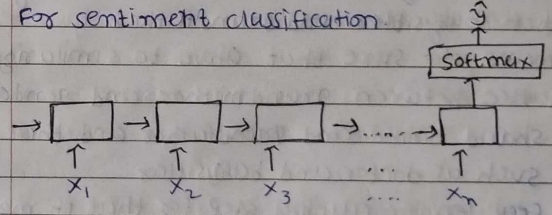


⇒ This algo avg all the word

⇒ problem with algo is ignor order of the word

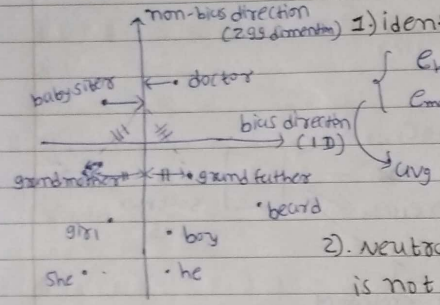
⇒ Not good sentence like: Completely lacking in good taste good service and good ambience

⇒ So we use RNN instead of summing all of word embedding for sentiment classification



week 3

Addressing bias in word embeddings



1) identify bias direction

$e_{he} - e_{she}$

$e_{male} - e_{female}$

avg

2). Neutralize: For every word that is not definitional project to get rid of bias

⇒ Project ~~grandmother~~ word like doctor and baby sitter onto axis to ~~reduce~~ element their component on the bias direction

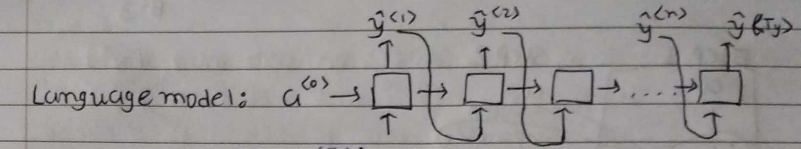
⇒ bias dir can have more then 1D dim by using PCA & SVD to handle it.

3). Equalize pairs.

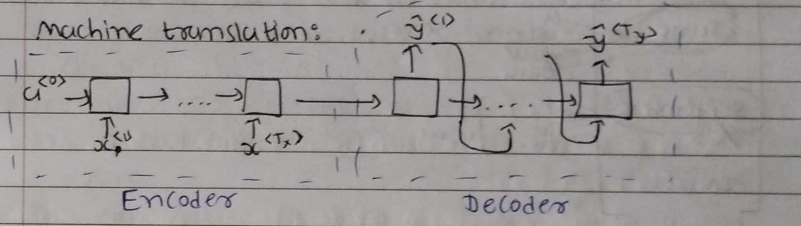
⇒ This example similarity bet^n grandmother and baby sitter is smaller then babysitter and grandfather

⇒ This step we make ~~the~~ sure that above to similarity is same. distance between grandmother and gender neural word should same and grandfather and that gender neutral word such as doctor and babysitter

⇒ So there are few lenear algebra step for that it move grandfather and grandmother from the axis in the middle



$P(y^{<1>}, y^{<2>} \dots y^{<T_y>})$



conditional language model

$P(y^{<1>} \dots y^{<T_y>} / x^{<1>} \dots x^{<T_x>})$

why not use greedy search?

$P(y^{<1>}/x)$ Find Prob of first of word given x

First of word — French sentence

choose most prob word the go for $P(y^{<2>}/x)$ choose most prob word for second word and so on...

greedy OP: Jane is going to be visiting Africa in september

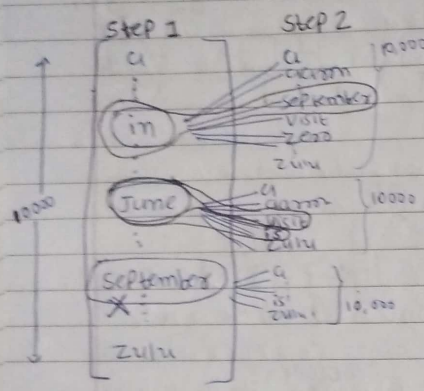
→ Here 'going' is most common after 'is' in English dictionary so greedy algo choose above sement for Frech to eng conversion of sentence X = Jane visite l'Afrique en septembre

→ but actual accurate OP should be Jane is going to be visiting Africa in september.

⇒ For greedy algo space complexity For 10,000 vocab and 10 word long prediction telke $10{,}000^{10}$ space so we use approx search algo and do $\arg\max\ P(\hat{y}^{<1>}, \hat{y}^{<2>} \dots \hat{y}^{<T_y>}/x)$

Beam search algoritham

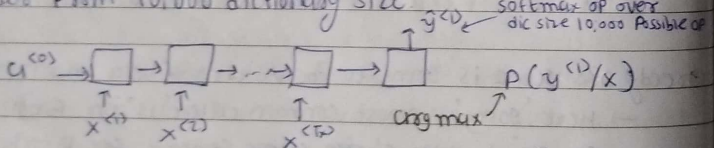
Here Beam width B=3



⇒ large B better result, slower
small B worse result, faster

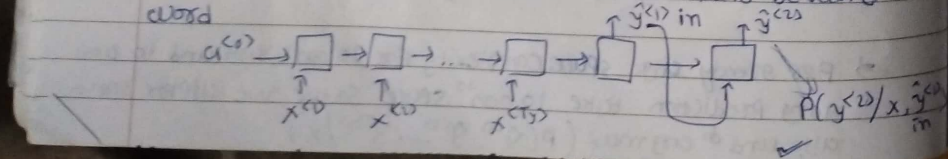
⇒ Unlike BSF & DFS Beam search run faster but is not guaranteed to find exact maximum for $\arg\max_y P(y|x)$

Step 1: Predicts most prob B words as output of first word in decoder from 10,000 dictionary size



let in June & September is highest first 3 Prob words

Step 2: For each of this 3 choices consider what should be second word



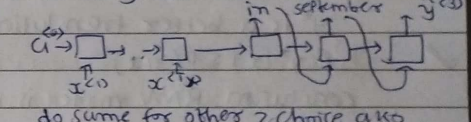
~~do same for June & September~~

and finally we select most prob first 3 for in, ~~June~~ ~~September~~

$$P(y^{<1>}_{in}, y^{<2>}|x) = P(y^{<1>}_{in}|x)\, P(y^{<2>}|x, y^{<1>})$$

⇒ do same for June and September and we get total 9 most Prob op out of those select most 3 likly op

⇒ let out of 30,000 choices most likly choices are
in, September
June, is
June, visit



do same for other 2 choice also

⇒ B=1, is greedy search algo

★ Length normalization

$$\arg\max_y \prod_{t=1}^{T_y} P(y^{<t>}|x, y^{<1>}, y^{<2>}, \dots, y^{<t-1>})$$

$$\arg\max_y \sum_{t=1}^{T_y} \log P(y^{<t>}|x, y^{<1>}, y^{<2>}, \dots, y^{<t-1>})$$

⇒ First eqn Π has long length and all Prob is rang 0 to 1 so first eqn become very small so we take log and make 2nd eqn

⇒ we can normilize by dividing by T_y = total word in op

$$\frac{1}{T_y} \sum_{t=1}^{T_y} \log P(y^{<t>}|x, y^{<1>}, \dots y^{<t-1>})$$

generally divide by T_y^{α} where α $(0 \le \alpha \le 1)$ $\alpha = 0.7$ generally

Error Analysis in Beam search

Error is due to one of those two RNN or Beam search

better translation → Human: Jane visits Africa in September. (y^*)
Algo : Jane visited Africa last September. ($\hat{y}$)

Case 1: $P(y^*|x) > P(\hat{y}|x)$
Beam search chose $\hat{y}$ But y^* attains higher $P(y|x)$
✓ conclusion: Beam search is at fault

Case 2: $P(y^*|x) \leq P(\hat{y}|x)$
y^* is a better translation than $\hat{y}$. But RNN predicted $P(y^*|x) \leq P(\hat{y}|x)$
✓ conclusion: RNN model is at fault.

⇒ RNN computes $P(y|x)$
so Here we compute $P(y^*|x)$ & $P(\hat{y}|x)$ with RNN and do above two case

Error analysis Process

Human	Algo	$P(y^*\|x)$	$P(\hat{y}\|x)$	At fault?
-	-	-	-	B
-	-	-	-	RNN
-	-	-	-	B
⋮	⋮	⋮	⋮	B

⇒ and we can found out what fraction of error is due to RNN & Beam search
⇒ RNN fault ⇒ deeper network
increase data...
⇒ Beam search ⇒ increase beam width
fault

Bleu score (used for translation)

Bleu score compute a score that How good that machine translation.

French: Le chat est sur le tapis
Reference 1: The cat is on the mat.
Ref. 2: There is a cat on the mat
MT OP : the the the the the the the ($\hat{y}$)

Precision = $\frac{7}{7}$ (∵ every words in MT OP appears on ref 1 or 2)

Modified precision = $\frac{2}{7}$ ← count "the"
count clip "the"

bleu score on bigrams:
MT OP: The cat the cat on the mat ($\hat{y}$)

$count_{clip}$ is that max input appears on ref 1 or ref 2

	count	$count_{clip}$
the cat	2	1
cat the	1	0
cat on	1	1
on the	1	1
the mat	1	1
	6	4

modified Precision = 4/6

$P_1 = P_2 = \ldots = P_n = 1$ when MT output is same as one of the ref

unigram: $P_1 = \frac{\sum_{unigram \in \hat{y}} count_{clip}(unigram)}{\sum_{unigram \in \hat{y}} count(unigram)}$

n gram: $P_n = \frac{\sum_{n\,gram \in \hat{y}} count_{clip}(n\text{-}gram)}{\sum_{n\,gram \in \hat{y}} count(n\text{-}gram)}$

P_n = Bleu score of n grams only $\quad P_1, P_2, P_3, P_4$

combine Bleu score = $BP \exp\left(\frac{1}{4}\sum_{n=1}^{4} P_n\right)$

BP = brevity Penalty

⇒ when MT OP is very short translation then Precision goes high so BP Penalizes translation system that out put translation are too short.

$$BP = \begin{cases} 1, & \text{if MT OP len} > \text{reference OP length} \\ \exp\left(1 - \frac{\text{ref OP len}}{\text{MT OP len}}\right), & \text{otherwise} \end{cases}$$

Rouge (use for summarization)

⇒ compares a summary to one or more reference summaries

Reference (human): It is cold outside

Generated OP : It is not cold outside

$$\text{Rouge-1 Recall} = \frac{\text{unigram matches}}{\text{unigram in reference}} = \frac{4}{4} = 1$$

$$\text{Rouge-1 Precision} = \frac{\text{unigram matches}}{\text{unigram in OP}} = \frac{4}{5} = 0.8$$

$$\text{Rouge-1 F1} = 2 \times \frac{\text{Pre} \times \text{recall}}{\text{Pre} + \text{recall}} = 2 \times \frac{0.8}{1.8} = 0.89$$

For Rouge-2 bigram instead of unigram

Rouge-L unigram ~~instead~~ of uses LCS (longest common seq)

⇒ Rouge clipping

human: It is cold outside

OP: It It It It

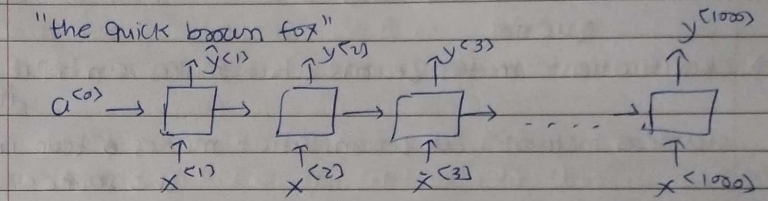
$\text{RouGE-1 precision} = \frac{4}{4} = 1$

$\text{Modified precision} = \frac{\text{clip(unigram matches)}}{\text{unigram in OP}} = \frac{1}{4} = 0.25$

Speech Recognition - Audio Data

1 march 24

★ CTC cost for speech recognition (connectionist temporal classification)



⇒ number of time step is very large for audio input time step is much bigger than output

Ex For 10 sec of audio comes at 100 HZ so we have 100 sample per second and 10sec audio clip has 1000 time step

But OP might not have 1000 alphabets it would be much lesser.

⇒ CTC cost function allows RNNs to generate ~~cat~~ output like this

the q

ttt_ h- eee ---␣--999-- ⇒ the q

Space, blank

★ Trigger word detection (Trigger words like: Alexa, okay Google)

⇒ So to detect trigger word we give input audio to train model which has label as below

For training: when trigger words comes in input audio mark output of model 1 otherwise 0 for that time interval and train model

DeepLearning.AI

NLP with sequence models

Week 3

1 March 24

Siamese Networks

: identify similarity between things

⇒ siamese network in NLP

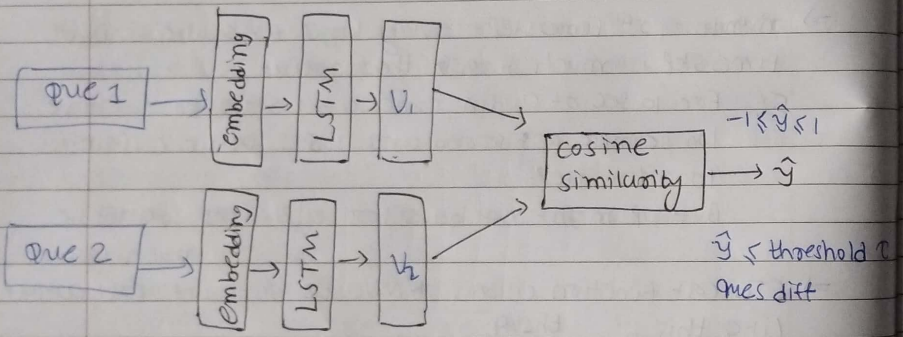
Hand written checks

question duplicate

queries

⇒ classification: ~~makes~~ Learns what makes an input what it is

siamese network: ~~a~~ Learns what makes ~~a~~ two inputs the same



same network

⇒ Not all siamese network use LSTMs.

Loss Function

How old are you? Anchor

what is your age? Positive

where are you from? Negative

$$\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\| \|v_2\|}$$

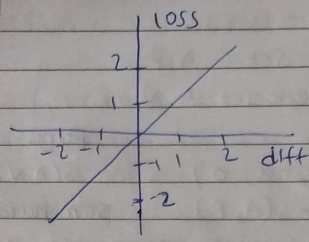
$\cos(A, P) \approx 1$

$\cos(A, N) \approx -1$

$-\cos(A,P) + \cos(A,N) \leq 0$

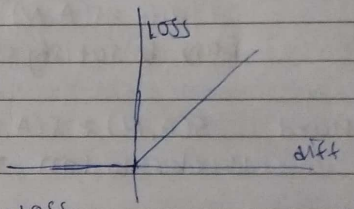
Simple loss:

$diff = S(A,N) - S(A,P)$



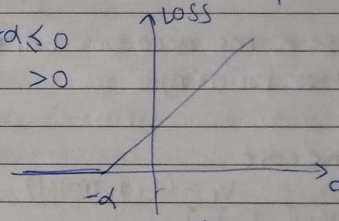
with non-linearity

$$L = \begin{cases} 0 & ; \text{if } diff \leq 0 \\ diff & ; \text{if } diff > 0 \end{cases}$$



with a margin

$$L = \begin{cases} 0 & \text{if } diff + \alpha \leq 0 \\ diff + \alpha & ; \text{if } " > 0 \end{cases}$$



⇒ you need your model confident enough and distinguish between negative and positive values ~~or~~ with ease So you need your model to learn until the model achives a specific value ~~of~~ for the difference between similarities so we need to ~~you~~ use margin α

It's ensures that learning goes on until the difference between similarity is equal or close to the margin value.

so Triple loss $L = \begin{cases} 0 & , \text{if } diff + \alpha < 0 \\ diff + \alpha & , \text{otherwise} \end{cases}$

$$L(A, P, N) = \max(diff + \alpha, 0)$$

↓ From neural network

You can use any similarity function or distance metric

Triplet selection

Triplet A, P, N { duplicate set A, P
non duplicate set A, N

Random $L = max(diff + \alpha, 0)$ Easy: $S(A,N) < S(A,P)$

$diff = S(A,N) - S(A,P)$ · semihard

Easy to satisfy Little to learn $S(A,N) < S(A,P) < S(A,N) + \alpha$

Hard $S(A,N) \approx S(A,P)$ Hard

Harder to train more to learn $S(A,N) > S(A,P)$

⇒ so instead of selecting random triplets select hard triplet for better training

Computing ~~loss~~ the cost

$V_1 = (1, d\_model)$

Batch 1

1	V_{11}	
2	V_{12}	
3	V_{13}	
4	V_{14}	

Batch 2 V_2

1	V_{21}	
2	V_{22}	
3	V_{23}	
4	V_{24}	

$S(V_1, V_2)$

V_2 \ V_1	-1	-2	3	-4
-1	0.9	0.8	0.3	-0.5
-2	-0.8	0.5	0.1	-0.2
-3	0.3	0.1	0.7	-0.8
-4	-0.5	0.2	-0.8	1

← Other than this : similarities for non duplicate question

← Similarity bet^n duplicate question

For well train model this value should be greater than off diagonal similarities.

⇒ we can use off-diagonal information to make some modification to the loss function & can improve model performance

⇒ to do that we used to concepts

mean negative: mean (avg) of all off-dia value in each row

closest negative : off diagonal value closest to (but less than) the value on diagonal in each row

mean negative: mean of off-diagonal values

~~Loriginal~~ $= max(\underbrace{S(A,N) - S(A,P)}_{diff} + \alpha, 0)$

$L_1 = max(mean\_neg - S(A,P) + \alpha, 0)$

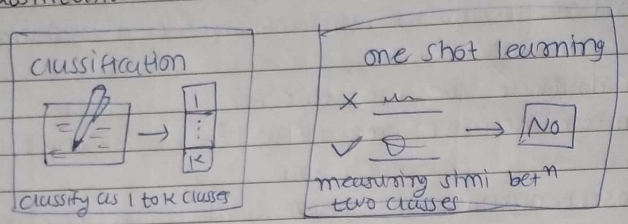
closest negative : closest off diagonal value

$L_2 = max(closest\_neg) - S(A,P) + \alpha, 0)$

$L_{Full}(A,P,N) = L_1 + L_2$

$$J = \sum_{i=1}^{m} \cdot L_{Full}(A^{(i)}, p^{(i)}, N^{(i)})$$

Classification vs one shot Learning



⇒ problem change which class ⟶ measuring similarity betn two classes

⇒ one shot Learning no need to retrain when ~~new~~ data e with new op comes to model. we need to just check similarity score betn signs.

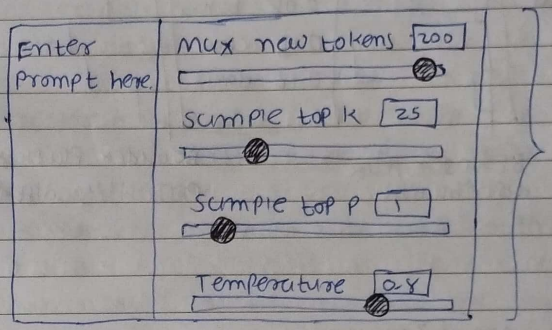
Deep learning. AI

Generative AI with Large Language Models.

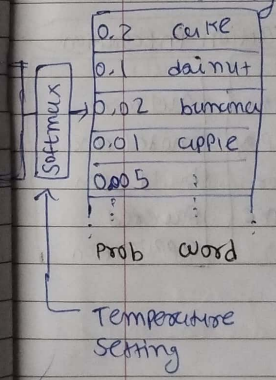
1 March 24

week 1

Generative configuration - inference parameters



greedy vs random sampling



greedy: The word/token with the highest prob is selected

random (-weighted) sampling: select token using random-weighted strategy across the probs of all tokens

Top K: select an op from the top K results after applying random-weighted strategy using the probabilities

Top P: Select op from the random weighted strategy with the top-ranked consecutive results by probability and with cumulative probabilities ≤ P

Temperature: Temperature setting is applies to softmax final level temp setting is scaling factor

⇒ Changing the temperature alters the predictions that the model will make.

⇒ T is low resulting prob distri from the softmax layer is more strongly peaked with the prob being concentrated in the smaller number of words.

more creative generation

Cooler Temp (e.g <1)

Prob	word
0.001	apple
0.002	banana
0.004	cake
0.012	donut
...	

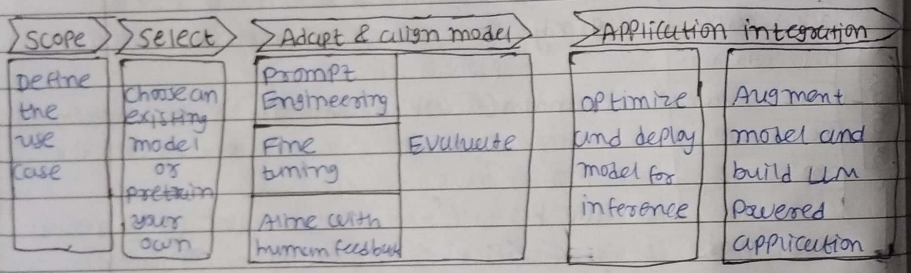
Stronger peak Prob distribution

Higher Temp(e.g >1)

Prob	word
0.04	apple
0.08	banana
0.15	cake
0.12	donut
..	.

Broader flatter probability distribution

Generative AI Lifecycle



Types of Transformers

⇒ There are three variance of Transformer model

i) Encoder only models
ii) Decoder only models
iii) Encoder decoder models

i) Encoder only models (Auto encoding models)

This models pre trains using ~~MSKE~~ Masked Language Modeling (MLM)

→ ~~This models pretrain using MLM~~

original text : The teacher teaches the students

→ Here token of input sequences are randomly mask and the training objective is to predict the mask token inorder to reconstruct original sentence.

→ It's use bidirectional representations of the input sequence that means model has understand full context of the tokens not ~~the~~ just of the words come before.

Mask Language modeling (MLM)

The teacher <MASK> the student

Encoder only LLM

Objective: Reconstruct text ("denoising")

The teacher teaches the student

→ ← bidirectional context

use cases: Sentiment analysis
Name entity recognition
word classification

Ex BERT
ROBERTA

ii) Decoder only (Autoregressive models)

Decoder only models pretrain using causal Language modeling (CLM)

→ Here the training objective is to predict next token based on the previous sequence of tokens.

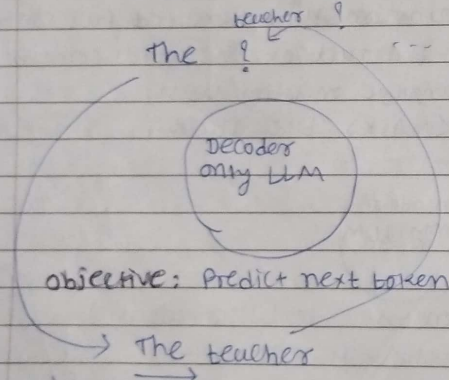
→ Predicting the next token is sometime called full language modeling by researchers.

→ This model mask the input sequence and can only see the input tokens leading up to the token in question. model has no knowledge of end of the sentence.

→ The model iterates over the input sequence one by one to predict the following token.

→ Decoder only model has only unidirectional context.

Causal Language Modelling (CLM)



use cases:
→ Text generation
→ other emergent behavior
- depends on model size

Ex GPT
BLOOM

iii) Encoder Decoder models (sequence-to-sequence models)

→ pretraining details ~~of~~ are varies from model to model.

→ popular seq-to-seq model T5 pretrains ~~the~~ the encoder using span corruption which mask random sequences of input tokens and those random sequences are replace with the unique sentinel token

→ sentinel tokens are special tokens add to vocab but do not correspond to any actual word from the vocab

→ Decoder is then tasked with reconstructing the mask token sequences auto-regressively

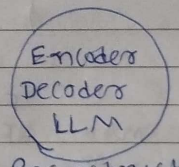
→ The out put ~~of~~ is the sentinel token followed by predicted tokens

Span corruption

The teacher <mask> <mask> student

The teacher <x> student

<x> ↖ Sentinal token



objective: Reconstruct span

<x> teaches the

use cases: Translation
summarization Text
Que answering

Ex
T5
BART (not Bert)

Computational challenges of traning LLMs

1 Parameter = 4 bytes (32-bit float)
(real number)

1B parameters = 4×10^9 bytes = 4GB

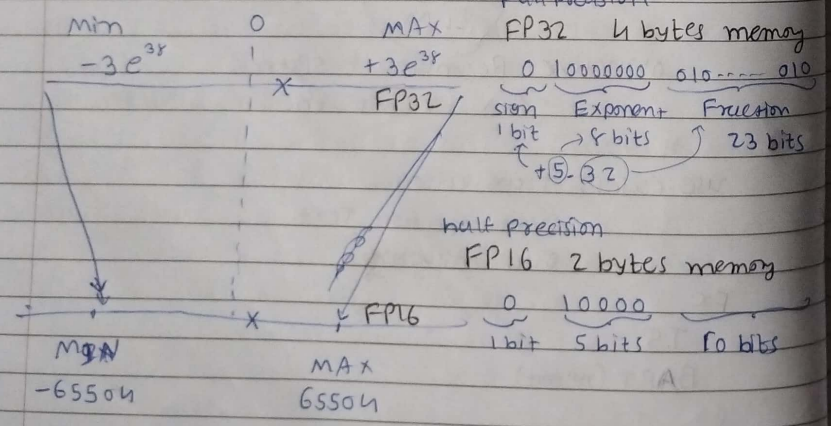
4 GB @ 32-bit
Full precision

Additional GPU RAM needed for train 1B Parameter
- Adam optimizer (2 states) + 8 bytes
- Gradients + 4 bytes
- Activation and temp memory + 8 bytes

= 4 bytes · per params
+20 extra bytes per param

So GPU needed is not 4GB
but 24 GB @ 32-bit
Full Precision

★ Quantization



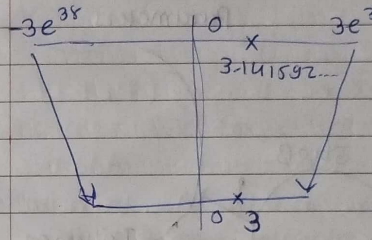
Full precision
FP32 4 bytes memory
0 10000000 010---010
sign Exponent Fraction
1 bit → 8 bits → 23 bits
+5-32

half precision
FP16 2 bytes memory
0 10000
1 bit 5 bits 10 bits

BFLOAT 16 | BF16 (not suitable for intiger calculation)

0 10000000 1001001
1 bit sign | 8 bits Exponent | 7 bit Fraction
$\sim 3e^{-38}$ $\sim +3e^{38}$

⇒ Let convert pi from FP32 to 8 bit space



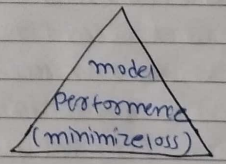
FP32
0 10000000 100---000

INT8
0 0000011 —

⇒ Quantization reduce required memory to store and train models
⇒ Projects original 32 bit floating point number to lower precision spaces

Scaling choices for pre-training

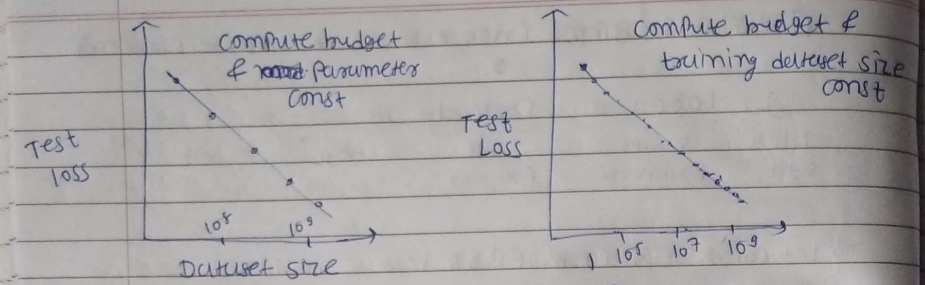
Goal: maximize model performance

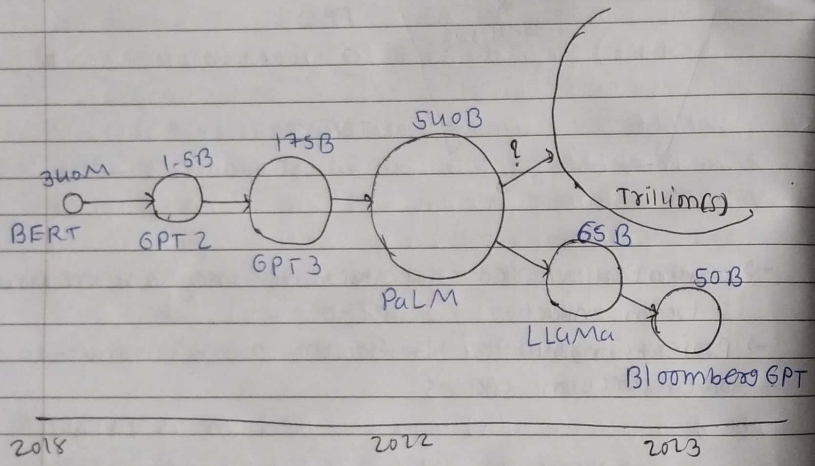


↑ Scaling choice : Dataset size
(number of tokens)

Scaling choice : Model size ↑
(# of params)

⇒ Compute resource constrains: → Hardware
→ Project time line
→ Financial budget

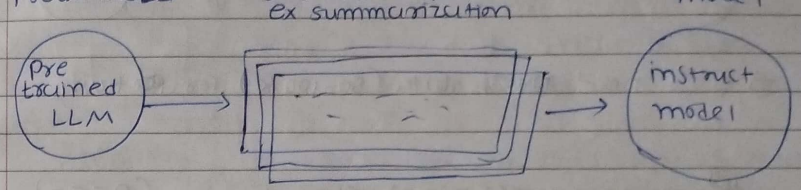




Fine-tuning LLMs with instruct

week 2

★ Fine-tuning on single task



⇒ Catastrophic forgetting happens here. because of full fine tuning process modifies the weights of original LLM while this leads to great performance on single fine-tuning task but it can degrade performance on other task.

Ex before fine tuning model give best performance on name entity recognition but after pretraining it will give bad result.

⇒ How to avoid Catastrophic forgetting.

→ If you need only single task model and do not need multi tasks then no need to do anything.

→ Fine tune on multi task at the same time

→ consider, Parameter Efficient Fine-tuning (PEFT)

↳ PEFT is set of techniques that preserves the weights of original LLM & trains only small number of task specific adapter layers and params.

★ Multi-task instruction fine-tuning

Pre trained model → instruction fine-tune on many tasks → instruct LLM

⇒ It's need much data (50-100 K) for train

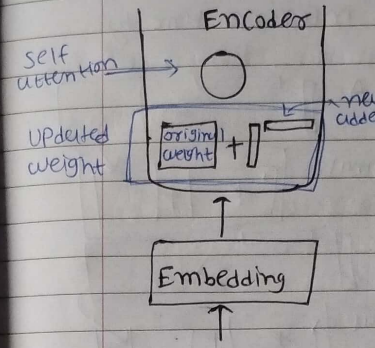
Parameter efficient fine-tuning (PEFT)

Selective	Reparameterization	Additive
select subset of initial LLM params to fine-tune	reparameterize model weights using a low-rank representation LoRA	Add trainable layers or parameters to model Two main Approaches i) Adapters ii) soft Prompts

⇒ Adapters methods add new trainable layers to the architecture of the model typically inside the encoder or decoder component after the attention or feed-forward layers

⇒ soft Prompts on the other hand keep the model architecture fixed and frozen and focus of manipulating the input to achive the better performance.
This can be done by adding trainable params to the prompt embeddings or keeping input fixed and retraining the embedding weight.

★ PEFT techniques 1 : LoRA (Low Rank Adaption)



1. Freeze most of original LLM weights
2. inject 2 rank decomposition matrices
3. train the weights of the smaller matrices

Steps to update model for inference:
1. matrix multiply the low rank matrices

B [ ] * A = B×A

2. Add to original weight

original weight + B×A

⇒ the model has same number of params as the original so there is little to no impact on inference latency.

⇒ Researchers has found that applying LoRA to just a self attention layer of the model is often enough to fine tune a task to achive performance gains.

⇒ However in priciple you can also use LoRA on other components like a feed forward layers. but since most of the Params of the LLM are in the attention layers you get a biggest savings in trainable parameters by applying LoRA to these weight matrices

⇒ 'base on paper Attention is all you need
Transformer weights d×k = 512×64 = 32768 trainable params
In LoRA with rank r = 8 :

A = r×k = 8×64 = 512 params

B = d×r = 512×8 = 4096 params

86 % reduction in params to train

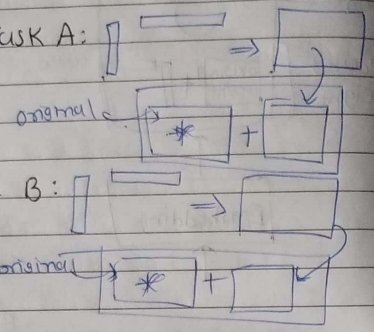
32768 ⟶ 4608

Prompt tuning is not prompt engineering

↑ not change weights but giving a prompt in such a way so performance will increase

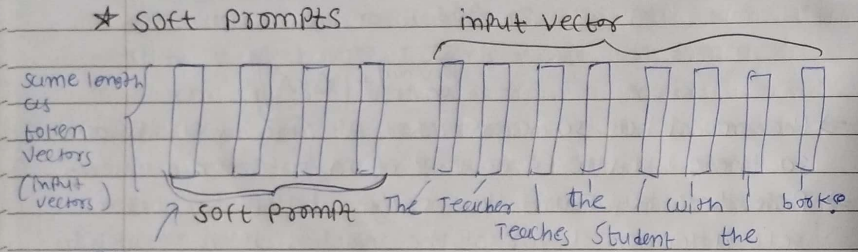
⇒ we can fine-tune to different set for each tesk and then switch them out at inference time by updating the weights.

1. Train different sunk decomposition matrices for different tesk
2. Update weights befor inference



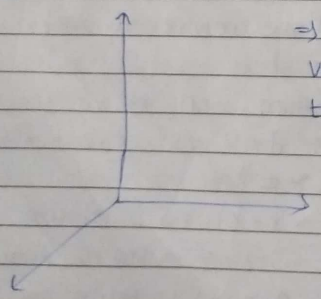
PEFT Techniques 2:

★ Soft prompts



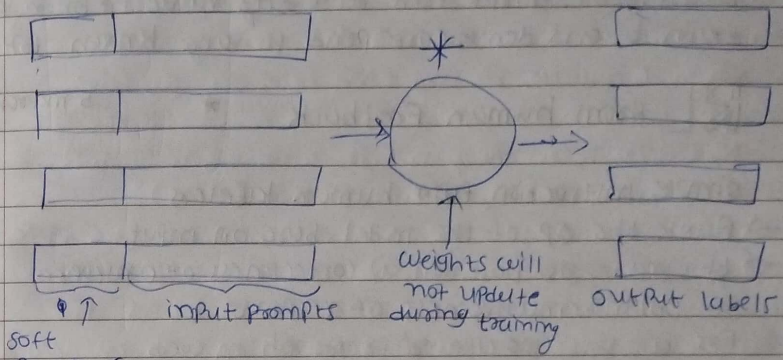
20-100 tokes

⇒ with prompt tuning you add additional trainable tokens to your prompt and leave it up to the supervised learning process to determine their optimal values



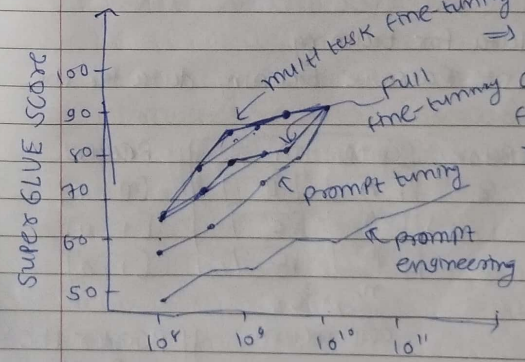
⇒ soft prompt can take any value in embedding space to maximize performance

weights of model frozen and soft prompt trained



⇒ similar as LoRA can train different soft prompt for different tesk and switch out soft prompt at inference time to change tesk!

Performance of prompt tuning



⇒ Prompt tuning can be as effective as full fine-tuning for larger models

PAGE NO.

week 3

5 march 24

interpretability of soft prompts

→ because of prompt can take any value in token space train tokens don't correspond to any known token.

RL from human Feedback ★ 5 march 24

simple instruction for human labelers

→ Rank the op of the model base on input.

→ this rank is base on a) correctness of answer
b) informativeness of response

For (a) you are allowed to search on web

→ If two response provide same correctness and informativeness and there is no clear winner you may rank them as a same but please only use this sparingly

→ If ans for given response is nonsensible, irrelevant and highly confusing or does not clearly respond to given prompt then label it with 'F' (for fail) rather than its rank. so poor quality ans can be removed.

Prepare labeled data for training

→ convert ranking into pairwise training data for the reward model

Prompt	Completions	Rank
	A	2
	B	1
	C	3

→

Completion	
A	B
A	C
B	C

Reward
[0, 1]
[1, 0]
[1, 0]

${}^{n}C_2$ ${}^{3}C_2$ combinations

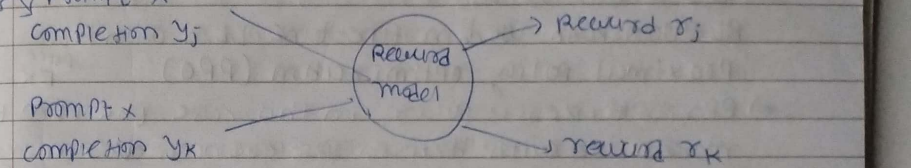
↓↓ Reorder the Prompts

Completion $\{y_j, y_k\}$	Reward
B A	[1, 0]
B C	[1, 0]
A B	[1, 0]

Train reward model

Train model to predict preferred completion from $\{y_j, y_k\}$ for prompt x

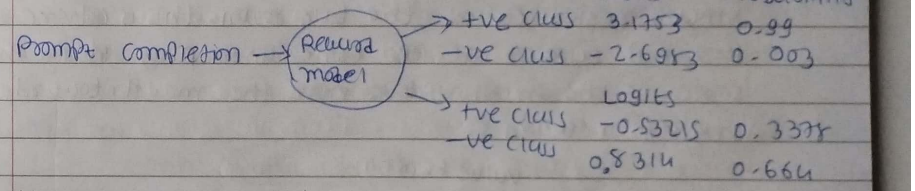
preferred completion is always y_j



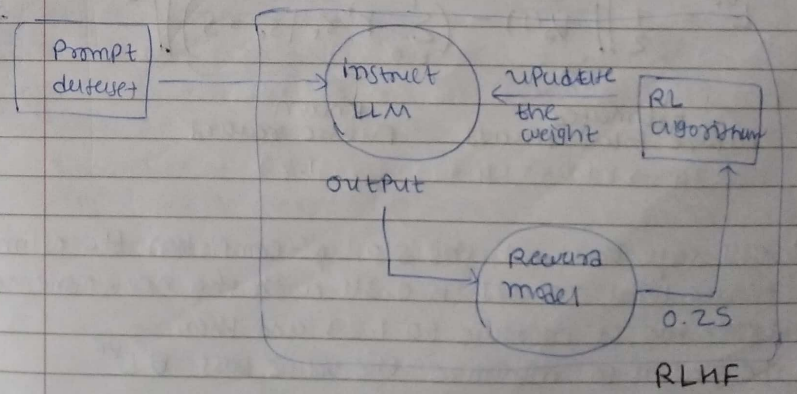
$$\text{loss} = \log(\sigma(r_j - r_k))$$

Use the reward model

→ Use reward model as a binary classifier to provide reward value for each prompt completion pair



Use reward model to fine-tune LLM with RL



⇒ We are repeating this process until we get threshold OR as human need or till number of epoches

⇒ In each epoches input data feed to LLM and then calculate reward and base on RL algo update the weight of LLM.

RL algorithem used in fine-tune LLM with RL

Proximal Policy Optimization (PPO) Proff EK

⇒ PPO optimizes a policy, in this case the LLM to be more aline with human preferences over many iteration. PPO make updates to LLM

⇒ The updates are small and within a bounded region, resulting in an updated LLM that is close to the previous version.

⇒ Keeping the changes within this small region result in a more stable learning.

⇒ PPO ensures to keep the model updates within a certain small region called the trust region. this is where the Proximal aspect of PPO comes into play. Ideally this series of the small update move the model towards higher rewards.

Value function

$$L^{VF} = \frac{1}{2}\left\| \underbrace{V_\theta(s)}_{\text{Estimated future total reward}} - \underbrace{\left(\sum_{t=0}^{I} r^t r_t \mid s_0 = s\right)}_{\text{Known future reward}} \right\|_2^2$$

0.34 → 1.23 → 1.4 →

1.87

⇒ Let's say that at this step of completion the estimated future total reward is 0.34 with the next generated token It increase to 1.23 and then....

⇒ The goal is to minimize the value loss L^{VF}

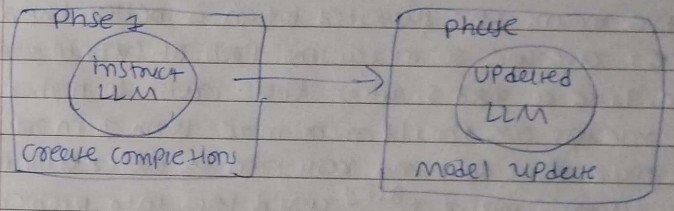
⇒ The value function then used in advantage estimation in phase 2.

⇒ This is similar to when you start writing a passage and you have a rough idea of its final form even before you write it.

PPO Phase 2: Model update

In phase 2 you make a small updates to a model and ev evaluate the impact of those updates on your alignment goal for the model.

→ The model weight updates are guided by prompt+completion, losses and rewards.



PPO phase 2 : calculate policy loss

$$L^{POLICY} = \min\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \cdot \hat{A}_t,\ \text{clip}\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}, 1-\varepsilon, 1+\varepsilon\right) \cdot \hat{A}_t\right)$$

model's Prob distri over tokens

trust region

Probs of the next token with initial LLM (which is frezz)

Advantage term

Probs of next token with updated LLM (which we can change for a better reward)

$\hat{A}_t$: Estimated advantage terms

estimates how better or worse the current action is compare to all possible actions at data state

⇒ Notice that second expression defines a region where two Policies are near each other.

second term is

Guardrails: keeping the policy in the trust region

why we have choose min of 1st & 2nd term:

⇒ A_t +ve means the suggested token is better then the ~~aver~~ average so increasing Prob of current token seems like a good strategy that leads to higher rewards

⇒ This means we should maximize term 1 in order to get our reward. and result in a better aligned LLM

⇒ directly maximizing L^{POLICY} would lead into problems because our calculations of are reliable under the assumption ~~the~~ that our advantage estimation are valid

★ ⇒ this advantage estimation are valid only when old and new policy are close to each other

PPO phase 2: calculate entropy loss

$$L^{ENT} = \text{entropy}(\pi_\theta(\cdot|s_t))$$

⇒ while the policy loss moves the model toward the alignment goal entropy allows the model to maintain creativity.

There is othe technique like PPO is Q-Learning

PAGE NO.

★ 6 march 24

⇒ If you kept entropy low you might end up always completing the prompt in the same way.

⇒ higher entropy guid the LLM toward more creativity This is similar to temperature setting of LLM.

⇒ the diff that ~~them~~ temperature influences model creativity at the inference time while the entropy is during training.

$$L^{PPO} = L^{POLICY} + C_1 L^{VF} + C_2 L^{ENT}$$

L^{POLICY}: Policy loss, L^{VF}: value loss, L^{ENT}: entropy loss

C_1 & C_2 : hyperparameters

⇒ The PPO objective updates the model weights through back propagation over several steps.

⇒ onces the model weights are updated PPO starts a new cycle. For the next iteration LLM is replaced with updated. LLM and new PPO cycle starts.

⇒ after many iterations you arrive at the human-aligned LLM.

★ 6 March 24

Reward Hacking

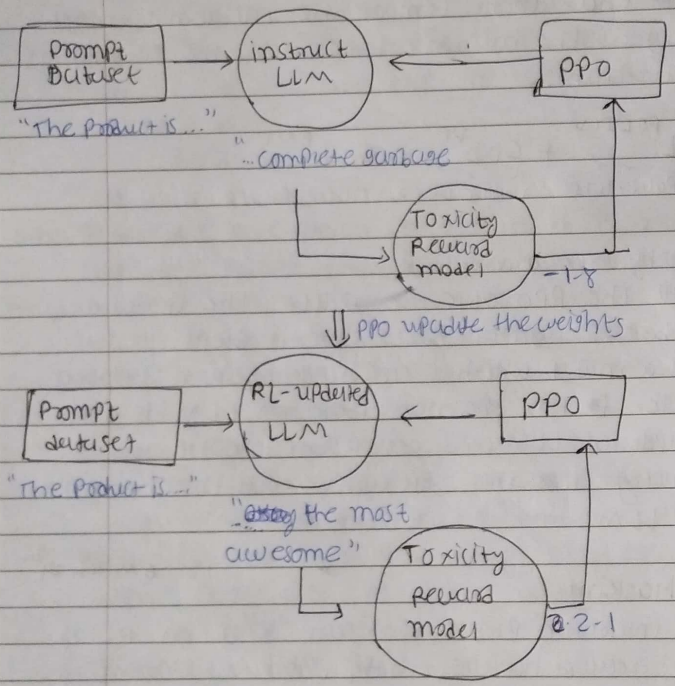
An interesting problem that can emerge in RL is known as reward hacking where the agent learns to to cheat the system by favoring actions that maximize the reward received even if those actions don't align well with the original objective.

Ex: suppose we are using RLHF to detoxify and instruct model, you have reward model that can carry out sentiment analysis and classify model completion as toxic or non toxic.

you select prompt from training data and pass it to the instruct LLM which generate the completion and reward model

gives a bad reward and ~~so~~ PPO update the model weight to create less toxic responses

→ as the policy tries to optimize the reward it can diverge too much from the initial language model



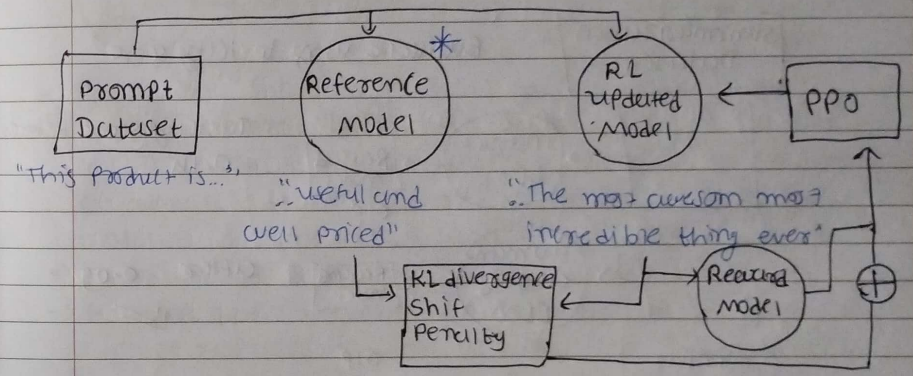
→ The model also can generate nonsensical, grammatically incorrect text that just happens to maximize the reward in a similar way.

Avoid Reward hacking

⇒ to avoid reward hacking we can use the initial instruct LLM as performance reference lets called it reference model. weight of it's are frozen.

⇒ During training each prompt is pass through both model reference LLM and intermediate LLM updated model at this point you can compare the completion and calculate KL divergence

⇒ KL divergence are statistical measure of how diff two prob distri are.

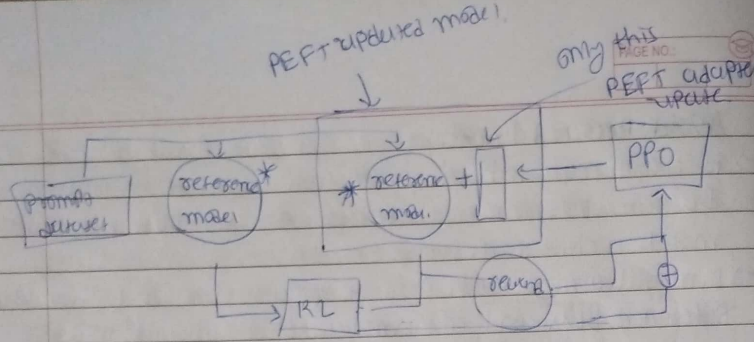


⇒ KL divergence is calculated for each generated token across the whole vocabulary of the LLM.

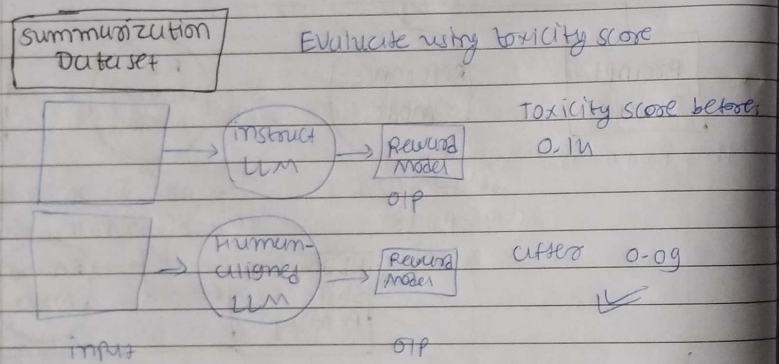
⇒ once KL divergence bet^n two model is calculated add acid term to the reward calculation

⇒ This penalize RL updated model if it shift too far from the reference LLM.

⇒ By the way you can benefit from combining our relationship with PEFT in this case only update the weight of PEFT adapter



Evaluate the human-aligned LLM



Constitutional AI

Constitutional AI is a method for training model using a set of rules and principles that govern the model's behavior.

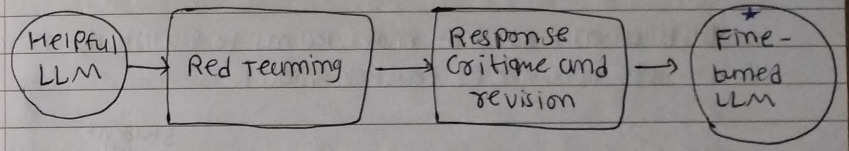
Principles like → choose helpful, honest and harmless responses
→ choose ethical and moral ...

⇒ training model will doen in two phases.
i) supervised learning stage
ii) RL stage

i) Supervised Learning Stage

Prompt the model in a ways that try to get it to generate harmful responses this is called red teaming and then tell to model to critique its own harmful responses according to constitutional principle and revise them to comply with those rules

→ once done you fine tune the model using the pairs of red team prompt and revised constitutional responses



ii) Reinforcement learning Stage

This stage is similar to RLHF, except that instead of human feedback we know use feedback generated by model.

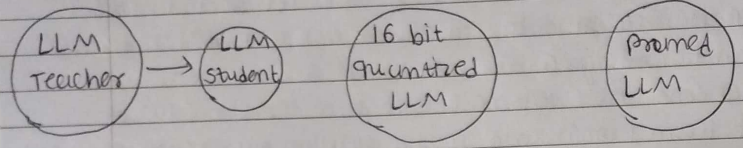
→ This is called RL from AI Feedback (RLAIF)

→ Here we used fine-tuned model* from (i) to generate a set of responses to prompt. then ask the model which of the responses is preferred according to the constitutional principled. The result is a model generated preference dataset that you can use to train reward model.

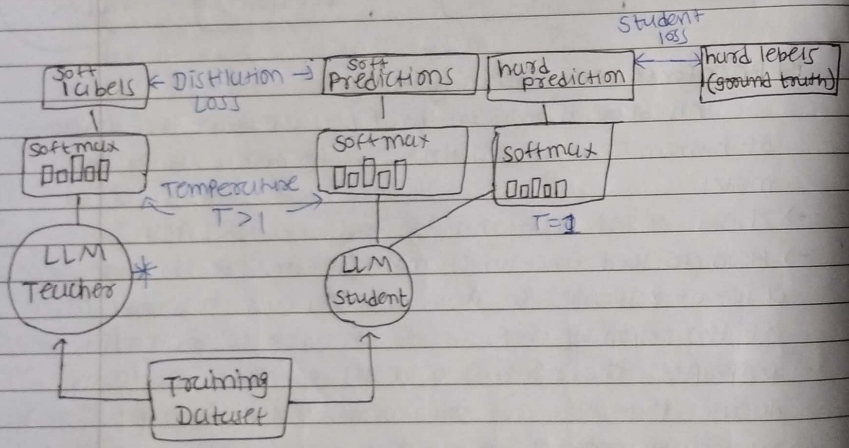
→ using this reward model can now fine-tune model Further using RL algo like PPO.

Model optimizations for deployment

Distillation Quantization Pruning



Distillation: student model learns to statistically mimic the behavior of teacher model



→ The teacher model is already fine tuned on training data so Prob distri likely closely matches the ground truth data and won't much variation in tokens so they also distillation applies Temperature parameter to the softmax function. (T>1)

→ Combine Distillation & student losses are used to update the weights of student model through a back prop.

→ In Practice distillation not as effective for generative decoder models. It typically more effective for encoder only model like Bert burt.

→ Distillation we train new smaller model we are not reducing model size of initial LLM in any way

Pruning: element the weights that are not contributing much to overall model performance these are the weights with value very close to zero, or equal 0

→ however there may not much impact on the size and performance if only a small percentage of the model weights are close to zero.

LLM Powered applications

8-3-24

★ Programming-aided language (PAL) models

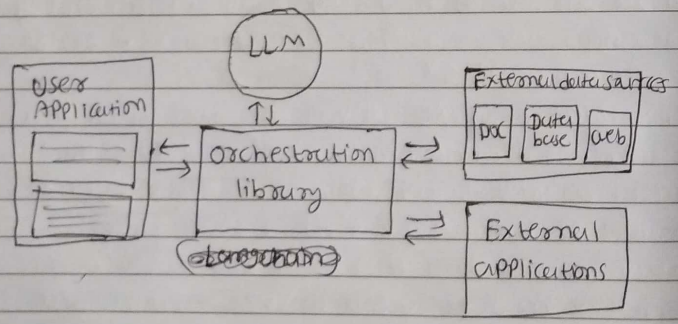
PAL Example

que: roger has 5 balls he buys 2 more cans of tennis ball each can has 3 balls. How many tennis balls does he have now?

Ans: # Roger started with 5 ball
balls = 5
# 2 can of balls each is
bought = balls = 2 × 3
# tennis balls. ans is
answer = balls + bought - ball

⇒ step of each reasoning step starts with # so line can be skip as a comment by python interpreter

⇒ we can give this example as one-shot.

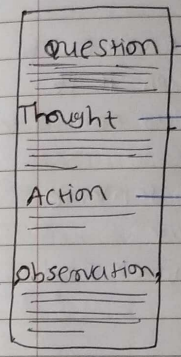


⇒ Orchestrator manage the flow of info and initiation of calls to external data data sources or applications.

⇒ It can also decide what actions to take based on the info contained in the output of LLM

★ ReAct : Combining reasoning and action.

ReAct is a prompting strategy that combines chain of thought reasoning with action planing



Question → Problem that requires advanced reasoning and multiple step to solve

Thought → a resoning step that identifies how the model will tackle the problem and identify an action to take

Action → An external task that model can carry out from an allowed set of actions

→ the authors created a small python API to interact with wikipedia the 3 allowed actions

i) search (entity): which looks for a wikipedia entry about particular topic

ii) lookup (string): which searches for a string on a wikipedia page.

iii) finish (answer): which the model carries out when it decides it has determined the answer.

→ Observation : the result of carrying out the action.

→ This is where the new information provided by the external search is bought into the context of the prompt for the model to interpret the prompt.

⇒ The prompt then repeat the cycle { as many time as it necessary to obtain the final answer.

⇒ solve a question answering task with interleaving Thought Action observation steps.

→ Thought can be ~~action~~ reason about the current situation and action can be three types:

1) search [entity] : which searches the exact entity on wikipedia and return the first paragraph if it exist. If not it will return some similar entities to search
2) Lookup [keyword] : which returns the next sentence containing keyword in the current passage
3) Finish [answer] : which returns the answer and finishes the task, ~~done~~

→ The ReAct frame work shows one way to use LLMs to power an application through reasoning and action planing.

→ This strategy can be extend for your specific use case by creating examples that work through the decision and ~~correct~~ action that take place in your application.

Ex: Langchain

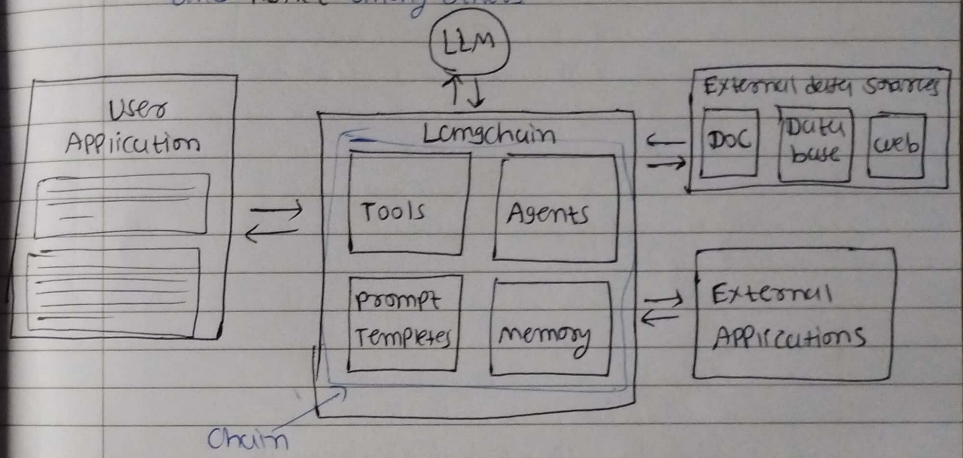
Langchain framework provides you with modular pieces that contain component to work with LLMs.

→ These component includes prompt templates for many different use cases that you can use to format both input examples and model completions and memory that you can use to store interaction with LLM. The framework also include pre-built tools that enable you to carry out a wide variety of tasks, including calls to external dataset and various APIs

→ Langchain has some predefined chains that have been optimized for diff use cases.

→ It also inclued agent which interpret the input from the user and determine which tools to use to complete the task.

→ Langchain currently includes agent of for both PAL and ReAct among others



LLM application architectures