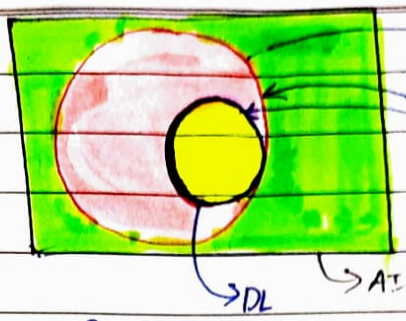
Date: _____

Topic: NATURAL LANGUAGE PROCESSING

Krish Naik

Agenda:

- → Roadmap of Natural Language Processing
- → Why NLP ?
- → Lots of examples
- → Tokenization, stemming, Lemmatization
- → Bag of words.



Pre-requisites
- Python
- Stats
- ML Algo
- ANN / Optimizers, Loss fn

NLP can be used both in ML and DL because we deals with text dataset in NLP.

Make the machines to do tasks efficiently

Our aim w.r.t NLP should be that we need to create a model in such a way that the model should be able to understand text and based on this it should give us some output.

eg: text Summarization, Chatbot, What's next sentence? Language translation, etc.

Roadmap
- BERT
- TRANSFORMER
- Bidirectional, LSTM, Encoder Decoders, Attention models
- Advance Text Preprocessing - Word embedding - Word2Vec
- RNN, LSTM, RNN, GRU, RNN → DL
- ML Use-cases (Solve)
- removes disadvantages with below → Text Preprocessing 3 → Word2Vec, Avg Word2 Vec
- Convert word into Vectors → Text Preprocessing 2 → BOW, TFIDF, Unigram, Bigram
- Text Processing 1 - BOW, TF-IDF, Word2Vec
  - → Tokenization, Lemmatization, Stopwords, Stemming
  - → Cleaning the data in this step.

BOW - Bag of Words

- Google Translation (Example)
- Positive Sarcasm / Negative Sarcasm (Yet to understand)
- → AI application.

Libraries
- Tensorflow, Pytorch → DL
- NLTK, Spacy, Text Blob → ML
- Hugging faces

(I) Tokenization:

Consider one ML Usecase.

Gmail → Spam Classifier Independent features: Email body, Email subject

Dataset

f1 Email body	f2 Email subject	o/p (Spam / Ham)
You won 1000 Rs $	Billionaire	Spam
Hey, How are you?	Hello	Ham
Credit card worth	Winner	Spam

Here you cannot simply give these sorts of text dataset to machine for processing, you should want to perform some tasks.

Step-I : Tokenization

Step-II : Stemming → Lemmatization
Stopwords

Tokenization: Process of converting sentence into words.

for example. You won 1000 $

Here | You | won | 1000 | $ | } Converted the entire sentence into words separately

Application created with NLP :
- → Spam classification
- → Chatbot
- → Text Summarization
- → Recommendation Systems

Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

(II) Stopwords

Example-2: Hey buddy I want to go to your house.

Here in above sentences, "to" is repeated and removing words like these may not change the idea of sentence.

So in-order to remove these words. We use technique called "Stopwords"

"[redacted]" // We can also have our own "keywords".

not ✗ → Important word → Changes idea of any sentences.

(III) Stemming:

Process of reducing words to their base word stem.

historical, history → histori

base word stem may not have any particular meaning.

Extracting root word (or) Baseform.

finally, final, finalized → fina

Disadvantage of Stemming

Advantages

↳ Stemming is really fast.

Here in-order to overcome disadvantages of stemming we use concept called "Lemmatization"

going, goes, gone → go

↳ Here we got meaningful base word.

(IV) Lemmatization:

Here we get meaningful word.

Lemmatization has entire dictionary of words it will check w.r.t base word and then compare.

history, historical → history

finally, final, finalized → final

Advantages

↳ provides meaningful word

Disadvantages

↳ Slow performance.

Use-cases of Stemming, Lemmatization

(i) Spam classification
(ii) Review classification
→ stemming

(iii) Text Summarization
(iv) Language Translation
(v) Chatbot
→ lemmatization

#1 Text-Preprocessing involves following steps:

(I) Tokenization
(II) Stopwords
(III) Stemming &
(IV) Lemmatization

↳ Here we have cleaned the entire words in the form of various methods.

#2 Convert words to Vectors

(I) Bag of Words
(II) TF-IDF (Term Frequency, Inverse Document Frequency)
(III) Word2Vec

Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

Bag of Words:

These words doesn't play big role they can be removed.

D1: He is a good boy
D2: She is a good girl
D3: Boys and girls are good.

encircled red words are generic words and should be removed. ⇒ stopwords

remove unnecessary words.

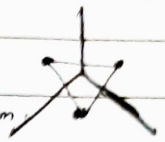
D1: good boy
D2: good girl
D3: Boys girls good

Vocabulary: good, boy, girl

Vocabulary	Frequency
good	3
boy	2
girl	2

	f1 (good)	f2 (boy)	f3 (girl)
Doc1 :	1	1	0
Doc2 :	1	0	1
Doc3 :	1	1	1

Classification ML algorithm



→ Euclidean distance
→ Cosine similarities

BOW → Binary BOW

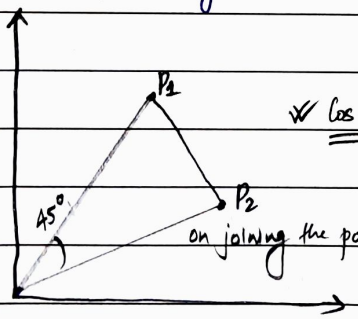
Advantages

1) Simple and Intuitive

Disadvantages

i) Sparsity
ii) OOV (out of vocabulary)
iii) Ordering of word has completely changed.
iv) Semantic meaning not able to capture

Cosine Similarity



Here if we need to find distance between P_1 and P_2 we can do it by two ways.

(I) Euclidean distance (II) Cosine Similarity

$\cos 45^\circ = 0.53$

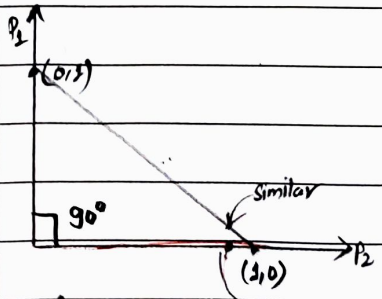
on joining the point to base lets suppose angle is 45°.

In-order to find similarity between P_1 and P_2, we use;

We have, $\cos 45^\circ = 0.53$

$1 - 0.53 \Rightarrow$ Cosine Similarity

$= 0.47$ [$\because P_1$ & P_2 are only 0.47 Similar]

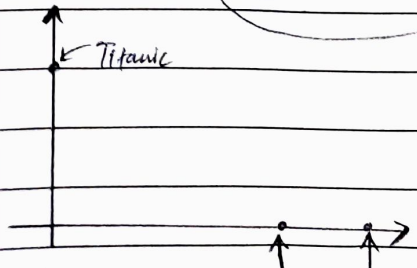


$\cos 90^\circ = 0$

$1 - 0 = 1$ [Cosine Similarity]

$\cos 0^\circ = 1$

$1 - 1 = 0$ ✓



In movie recommendation system we can relate that one movie Avenger which may be similar to Ironman must be similar with the cosine as well.

Deepakydv

So, disadvantage of BOW is that it's weak for capturing Semantics meaning. Hence ngrams comes into picture.

Teacher's Signature _____

Date: Day-2

Topic: _____

Agenda:

(i) Text Preprocessing → Words → Vectors
(a) a) One (One-hot encoding) c) TF-IDF
b) Bag of Words → ngrams
c) TF-IDF → Term frequency Inverse Document frequency
d) Word2Vec

Basic Terminology used in NLP:

1. Corpus → Paragraph → [D1, D2, D3, D4]
2. Documents → Sentence → 1st document D1, 2nd " D2, etc
3. Vocabulary → 10K unique words (Total Unique words)
4. Words → Word

Sentiment Analysis:

Text	O/P
D1: The food is good	1
D2: The food is bad	0
D3: Pizza is amazing	1
D4: Burger is bad	0

Dictionary Book
10K Unique words

Dataset ⇒ Sentiment Analysis Model

Dataset → Text Preprocessing 1 → Text Preprocessing 2 → Words → Vectors

- Text Preprocessing 1: Tokenization, Lowering case of word
- Text Preprocessing 2: Stemming, Lemmatization, Stopwords
- Words → Vectors: BOW, TF-IDF, Word2Vec

One-Hot Encoding:

Suppose a Corpus as below:

Document 1-D1 → [A man eat food
D2 Cat eat food
D3 People watch KRISH YT]

Corpus

Vocabulary:
9

"A man eat food cat people watch KRISH YT." ← These are unique Vocabularies

[[1 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 1 0 0 0]
. . .
. . .]

In place of word replaced by 1 remaining all zeros.
"ONE-HOT Encoding"

Advantages
- Simple to implement
- Intuitive

Disadvantages
- Sparse Matrix
- OOV (Out of Vocabulary)
- Not fixed size
- Semantic meaning b/w Word is not captured.

D1 → [[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]]

D2 → [[1 0 0]
[0 1 0]
[0 0 1]]

Here Size is decreasing we cannot train the model.

Teacher's Signature _____

Deepakydv

Date: _____ Topic: _____

Ngrams :- ⇒ (Bigrams, Trigrams . . . .)

{ comparing with Doc sentences from BOW example. for eg. • in the doc 1 there is not word good girl so "0" but there is good boy so "1".

	f1 good	f2 boy	f3 girl	f4 goodboy	f5 goodgirl
Sent-1	1	1	0	1	0
Sent-2	1	0	1	0	1
Sent.-3	1	1	1	0	0

Example :

DOG CHASE CAT

1 1

= total ⇒ 2 bigrams only

DOG CHASE CHASE CAT

"I am not feeling well"

1 1 1

I am not = 1
am not feeling = 1
not feeling well = 1
} 3 trigrams

Exc-2 : KRISH IS NOT FEELING WELL (1,3) this means from unit to tri any grams can be used

f1 f2 f3 f4 f5

Day-3 NLP :

Agenda

(1) Bow (Bag of words)
(2) TF-IDF
(3) Practical Implementation
(4) Dawgs

Bag of Words (BOW)

Technique to convert text into vectors.

(I) Text → Vectors

Ex: Sent-1 : He is a good boy
Sent 2 : She is a good girl
Sent 3 : Boy and girl are good.

Step-1st : Apply stopwords, Stopwords will help you remove all the unnecessary words such as he, is, she, is, a, and, are.

↳ Lowercasing.

Step-2nd = After lowercasing we obtains result shown alongside.

Step-3rd - find out frequency of Vocabulary w.r.t unique words.

Step-4th :- Divide Sentences into features and put for present word in Sentence is 1 and not present is 0. Which is Sparse Matrix.

(II)

Sent 1 good boy
Sent 2 good girl
Sent 3 boy girl good

(III) Frequency (Vocabulary)

good	3
boy	2
girl	2

unique words

(IV)

Sentences	(good) f1	(boy) f2	(girl) f3
Sent 1:	1	1	0
Sent 2:	1	0	1
Sent. 3:	1	1	1

Disadvantages

→ If there are many words your Sparse Matrix will be very large and its difficult in computation.

→ OOV (out of Vocabulary).

→ Semantic Meaning is missing.

Lets consider two sentences:

I: The food is good
II: The food is not good

food	good	is	not	The
1	1	1	0	1
1	1	1	1	1

Here Sentences are not similar but if we find the cosine similarity for the sparse matrix we will come to know they are similar but infact they are not. This is disadvantage.

This can be fixed with TF-IDF.

Teacher's Signature _____

Deepakydv

Date: _____ Topic: _____

Term Frequency - Inverse Document Frequency (TF-IDF)

Lets take both the sentences used in BOW.

Sentences after cleaning the text.

Sent 1: good boy

Sent 2: good girl

Sent 3: boy girl good.

The issue with BOW is that it had given same weightage to all the words irrespective of meanings. For boy and girl same weightage, good also same weightage. But its not fair. Boy is repeated on sentence 1 and sent. 3 and girl in sent 2 and sent 3. So weights should be assigned accordingly.

Whichever words are rarely present we should allocate more weightage.

For instance in ex-2, cat, Bat and Krish is unique (rare) words. So they must be assigned with more weightage.

Ex: 2

Sent 1: Cat eats food

Sent 2: Bat eats food

Sent 3: Krish eats food

Rare words will be captured by "Term Frequency"

Common words will be captured by "Inverse Document Frequency"

On combining both we would be able to identify most rare and

$$\text{Term Frequency (TF)} = \frac{\text{No. of rep of words in sentence}}{\text{No. of words in sentence}}$$

→ Sentences (TF is calculated per sentences)

$$\text{IDF} = \log_e\left(\frac{\text{No. of sentences}}{\text{No. of sentences containing the word}}\right)$$

→ Words (IDF is calculated per word)

Term Frequency:

	Sent 1	Sent 2	Sent 3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

* (multiply)

Inverse Document Frequency

Words	IDF
good	$\log_e(3/3) = 0$
boy	$\log_e(3/2) =$
girl	$\log_e(3/2) =$

	f1 good	f2 boy	f3 girl
Sent 1	1/2 * 0 = 0	1/2 * log(3/2)	0
Sent 2	0	0	1/3 * log(3/2)
Sent 3	0	1/3 (log(3/2))	1/3 log(3/2)

Since good is present in all the sentences it is saying that it is playing not crucial role

boy is present rarely in 2 sentences hence boy has some value similarly for girls.

Some kind of Semantic Information is being displayed using TF-IDF method.

Advantages

→ Intuitive

→ Word importance is getting captured.

Disadvantages

→ Sparsity

→ OOV (Out of vocabulary)

* max_features = 3 ⇒ This is going to take top 3 max-occurring features (frequency).

Teacher's Signature _____

Deepakydv

In NLP Word embedding is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encode the meaning of word such that a words that are closer in the vector space are expected to be similar in meaning.
#Wikipedia.

Date: Day 4 **Topic:**

Agenda

(1) Word Embeddings

(2) Word2Vec → CBOW (Continuous Bag of Word)
→ Skip-gram

Word Embeddings :-

: It is a technique which converts word in Vectors. Word embedding can be done on two ways:

- Word Embedding
  - Count or frequency
    - i) BOW
    - ii) TF-IDF
    - iii) One-hot encoding (OHE)
  - Deep Learning Trained Model.
    - i) Word2Vec (Pre-trained)
      - CBOW
      - Skip-grams

All disadvantage of ^ are corrected through deep learning trained model.

Word2Vec :- Feature Representation

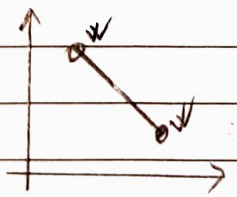
Ex 1:

	f1	f2	f3	f4	f5 ...	f6 ...
	Boy	Girl	King	Queen	Apple	Mango

→ Vocabulary

In Word2Vec, for every word we try to create Vector. but always remember this vector will be of limited dimensions. Also Sparsity would be reduced.

This Word2Vec also make sure semantic meaning is maintained.



Suppose similar words honest and good is converted into vectors with the help of Word2Vec. We will get vectors which are very near to each other. If you try to subtract these vectors, you will get minimal difference.

So, from Ex 1:

	f1	f2	f3	f4	f5	f6 ...
	Boy	Girl	KING	QUEEN	APPLE	MANGO
Gender	-1	1	-0.93	+0.93	0	0.1
Royal	0.01	0.02	0.95	0.96	-0.02	0.01
Age	0.03	0.02	0.7	0.6	0.95	0.96
Food	...	...	...	...	...	...

Suppose these are our feature representation

300 dimensions
All words are represented by 300 dimensions.

↑ These values would be generated by model any pre-trained models etc.

In case of deep learning we basically use embedding layer. This is basically used for converting text to Vectors.

The major disadvantages with TF-IDF and BOW was that semantic meaning is not captured.

Semantic meaning → word depicting similar meaning. { Honest / Good }

Another disadvantage was that TF-IDF Sparse matrix, because of which huge dimensions were created. Many more Vectors were created which is computationally expensive in terms of processing, model training etc also may lead to overfitting.

These all the disadvantages will be solved by Word2Vec.

Here we are assigning some vectors to our feature representation for eg, Boy -1 and girl by +1 similar in case of King & Queen but apple and mango are totally related with one another in case of gender so it getting some random values.

Lets Suppose King, Queen, Man, Women is created by following vectors.

King [0.96 0.95] Man [0.95 0.93]

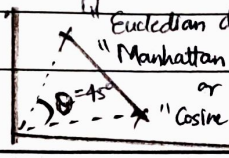
Queen [-0.96 0.95] Women [-0.94 -0.93]

Matching

King - Man + Women = Queen.

So, what ever Vector you get from result that would be matching obtained feature value.

This is done by "Cosine Similarity".



Distance = 1 − cosine Similarity

The more value towards 0 more similar entities are.

$= 1 - \cos\theta = 1 - \cos 45° = 0.29$

Word2Vec is a technique for natural language processing published in 2013. The word2Vec algorithm uses neural network model to learn word association from large corpus of text.

As name implies, Word2Vec represents each distinct word with a particular list of numbers called a Vector.

One trained such a model can detect synonymous words or suggest additional words for a partial sentence.

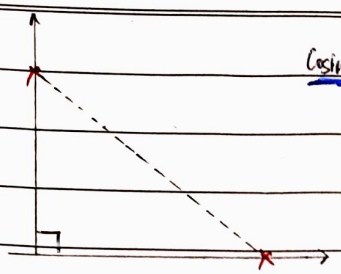
Teacher's Signature

Deepakyadu

Date: _____

Topic: _____

*Word2Vec is based on "Supervised Machine Learning".



Cosine Similarity = 1 - cosine = 1 - cos θ = 1 - cos 90°

= 1 - 0

= 1 (This means the points are nowhere similar to one another.

Value towards 0 → more similar

Value towards 1 → Not similar

Suppose ex: Movie Recommendation

Movie name - Vector

Vocabulary

*Titanic

{Action, Comic, comedy → features

Iron man

Avengers

Word2Vec (CBOW): (Continuous Bag of Words)

lets suppose we have corpus.

CORPUS: Dataset Training

[KRISH CHANNEL IS RELATED TO DATA SCIENCE]

Word2Vec

(I) → Centre Word (O/P or Target)

II → Centre word after sliding right (O/P : Target)

→ Context word (Since this word would bring some similarity in sentence. So its context word

Take any no. Preferably odd. → Window size = 5 ⇒ This Window size is helpful for creating training data (i/p and o/p data)

for Word2Vec, we must have Window size.

This Window size would be helpful for creating "Training data"

W.r.t training data, we have 2 features.

(I) Independent feature	O/P
KRISH, CHANNEL, RELATED, TO	IS
CHANNEL, IS, TO, DATA	RELATED
IS, RELATED, DATA, SCIENCE	TO

Center word

The main aim of Word2Vec is that it will take the text and based on the o/p as shown in example it will try to generate Vectors.

Simple BOW Representation or OHE

Word1	KRISH	1	0	0	0	0	0
Word2	CHANNEL	0	1	0	0	0	0
	IS	0	0	1	0	0	0
	RELATED	0	0	0	1	0	0
	TO	0	0	0	0	1	0
Word3	DATA	0	0	0	0	0	1
Word4	SCIENCE	0	0	0	0	0	1

#Process in ANN (I) Assign weights

(II) Initialize the weights

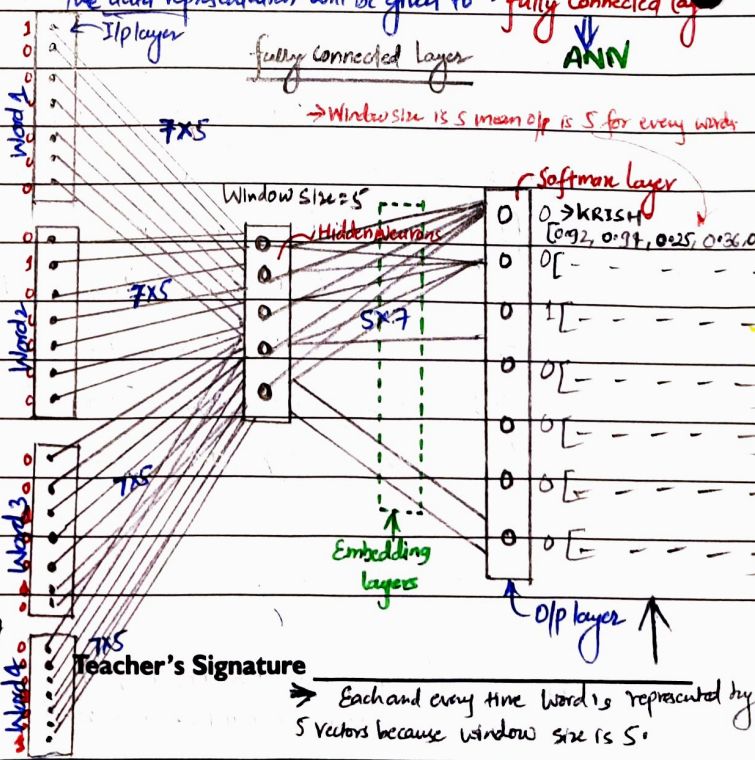
(III) one Hidden layer

(IV) f/w Propagation

After f/w propagation loss $(y - \hat{y})$ is calculated and again b/w propagation would happen by updating weights

This f/w and b/w propagation keeps happening until we get loss value very less.

This data independent features along with the o/p features and the data representation will be given to "fully Connected layer" → ANN



Each and every time word is represented by 5 vectors because window size is 5.

Teacher's Signature _____

Deepakydv

Date: _____ Topic: _____

(II) **Skipgram :-** { reverse of CBOW }

* Window size is define by hyperparameter tuning.

IP	OP
IS	KRISH, CHANNEL, RELATED, TO
RELATED	CHANNEL, IS, TO, DATA
TO	IS, RELATED, DATA SCIENCE

Neural network will be reverse of CBOW ✓

Day-5:

Agenda

(I) Practical implementation { Spam Classification Dataset }

↳ Text preprocessing : Tokenization, Stopwords, stemming, lemmatization, NLTK

↳ Text → Vectors : BOW, TF-IDF, Word2Vec, AvgWord2Vec

Word2Vec (2 types) — Skipgram, CBOW

gensim - library (from scratch)

Word2Vec

- Pre trained Model (Google new 300 dimension) ✓
- Train this model from scratch → gensim ✓

If you have any new dataset and if it capture 75% of entire word by pre-trained model then definitely go for Pre-trained model.

But if you see inside this pre-trained model if any such text are used wherein those text are not present in pre-trained model at that time you can train your model from scratch.

for example in case of Word2Vec when we are converting word to vector. Every single word in case of 300 dimension forms with 300 vectors.

Our IP is in this format

IP	OP
I want to eat pizza	Spam/Ham

↳ convert into 300 dimensions

This problem can be actually fixed with new concept called as "AvgWord2Vec"

Lets suppose we have one sentence.

"Welcome to Zambia" → we need to convert this into 300 dimensions but acc" to Word2Vec individual conversion happens like Welcome gets converted to 300 dimension likewise "to" of "Zambia".

But our main aim is to convert them in totality.

Hence "AvgWord2Vec" is used for this.

Entire words

This should be our aim.

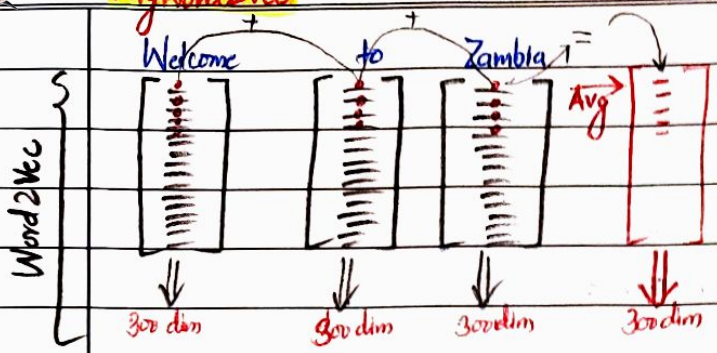
But on using pre-trained model from scratch by using Word2Vec for each word we are getting that many no. of dimensions.

Teacher's Signature _____

Deepakydv

Date: _____ Topic: _____

AvgWord2Vec

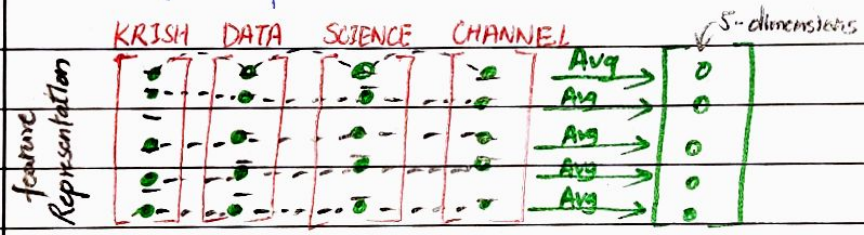


Here in case of Avg Word2Vec. All the individual Word2Vec elements of dimensions are taken average and new Vector is created with single 300 dim

Once this is done our average 300 i/p feature and Spam/ham is o/p features

Ex 2: KRISH DATA SCIENCE CHANNEL

Word2Vec (feature representation) ⇒ 5 dimensions



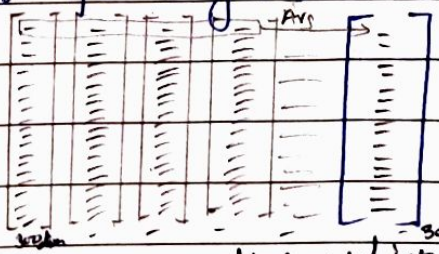
Example 3

Text	O/P
D1: The food is good	1
D2: The food is bad	0
D3: Pizza is amazing	1

Vectors (Avg)	O/P
[- - - - - - -]	1
[- - - - - - -]	0
[- - - - - - -]	1

Google Pre-trained Word2Vec Model

[The food is good] → D1



Here every single word is converted into Vectors and taken average of it.

Our main aim should be w.r.t avgword2Vec is that I can take this as an input (avg word2vec) and our provided o/p which is given above and we can basically train the model.

→ Since we are combining all the words the Semantic meaning would also be maintained. This is another advantage.

Teacher's Signature _____

Deepakydv

Date: Day 6

Topic: NLP with DL

Deep learning: (1) RNN (2) LSTM RNN (3) GRU RNN (4) Bidirectional LSTM RNN (5) Encoders-Decoders (6) TRANSFORMERS (7) BERT

Recurrent Neural Network [RNN]

Text → Vectors

Word Embedding ← ← Word2Vec, AvgWord2Vec

Lets take an example as "Chatbot"

In Chatbot we are asking specific question and getting specific answers to it.

Question and Answer

Chatbot should always understand proper text of the question and then answer it.

The sequence of words are very important in case of Chatbot.

Let us take an example of "Google Translate"

In case of google translate also sequence of word is very important on top of that grammetical corrections are equally important.

Language translation is very important thing.

Language translation → Hindi → English

Text Generation → A sentence → Suggestions

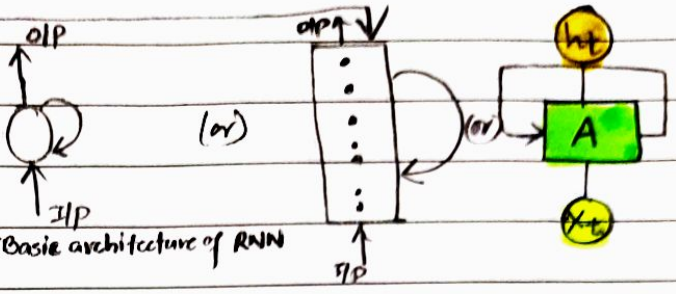
↓

Gmail, LinkedIn Auto-Suggestion

Grammetically Correct and Sentence completion, are most imp. factors.

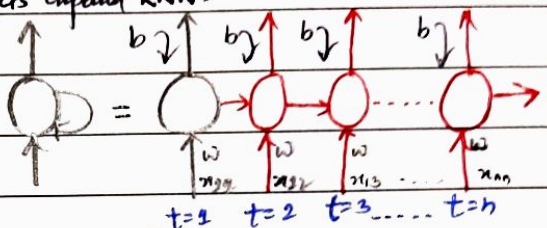
All these applications can be only trained well with the help of neural networks present in Deep Learning.

Word2Vec is a technique in deeplearning which we use to train a model or we can use pre-trained model which actually converts word into Vectors. This is important because "Semantic meaning is captured".



Basic architecture of RNN

Lets expand RNN.



So, what ever o/p we are getting we are giving same as an input to same Neural Network in case of RNN

Sentiment Analysis Example:

Sentences	o/p
The food is good	Positive

x_{11} x_{12} x_{13} x_{14}

Lets understand how do we send this information to RNN.

So the sentence is basically indicated with x_{11}, x_{12}, x_{13} x_{14} notation. for example if "The" word need to be sent then its notation is assigned to the RNN as shown above. Some process happens inside RNN after that which is "weights are initialized". but before that word is converted to vector and dimensions are initialized.

So, when we are providing x_{11} we will convert first x_{11} by using Word2Vec.

~~x_{11} → Vector → Then Passed to RNN~~

x_{11} → Word2Vec → Vectors of dimension (d)

(300)

This dimension is initialized to RNN.

Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

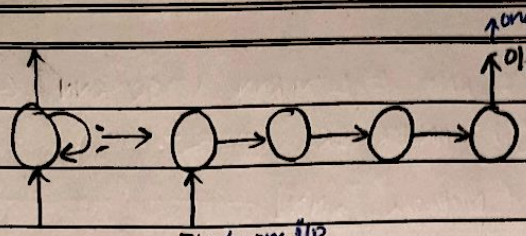
Types of RNN :

(i) One to One RNN

(ii) One to Many RNN

(iii) Many to One RNN

(iv) Many to Many RNN



Use case (example)
Image classification

Here we are passing i/p but not retreiving it anywhere except one
So its "One to One" RNN

There are 2- important features of deep learning.

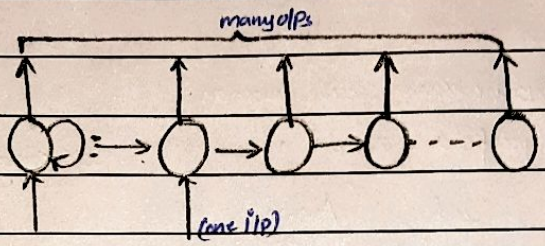
(I) Forward Propagation

(II) Backward Propagation

Any complex problem that are solved by google or insta specifically worst NLP or text data where sequence is involved, they have used RNN.

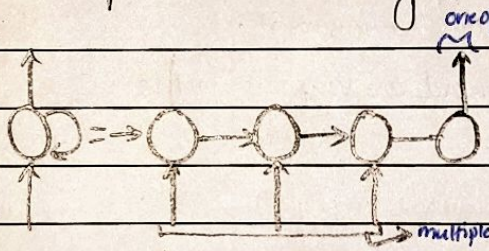
RNN is can not only used for text data.

It is also used for "Time series Data".



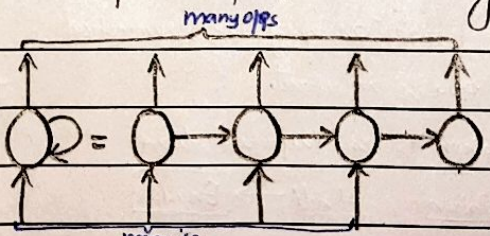
Examples
- Music Generation
- Text Generation
- Google search Suggestions
- Movie Recommendations

Here we pass one i/p but retreives many o/p. This is "One to Many" example -



Examples:
- Sentiment Analysis
- Predict next day sale.

Here we passes multiple i/p but retreives only one o/p "Many to One" RNN

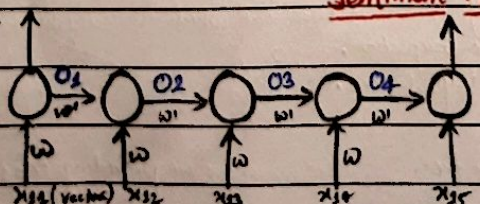


Examples
- Language translation
- Question Answering
- Chatbots etc...

Here we pass multiple i/p and retreives multiple o/ps, "Many to Many" RNN.

Forward Propagation (Many to One RNN)

Sentiment Analysis



Taking previous example of Sentiment Analysis.

The food is very good

x11 x12 x13 x14 x15

O/P
Positive

Passing $x_{11}, \dots, x_{15}$ to RNN in the form of vectors

We do that using Word2Vec Technique

In case of forward propagation we apply some specific operation. We multiply i/ps and weights

$$O_1 = f(x_{11} * w) + b$$

$$O_2 = f(x_{12} * w + O_1 * w_a') + b$$

$$O_3 = f(x_{13} * w + O_2 * w')$$

$$O_4 = f(x_{14} * w + O_3 * w')$$

In case of "Many to One" we apply "Sigmoid" $\rightarrow \hat{y}$

"Softmax" as its multi-class-classification.

*: for binary classification we apply "Sigmoid". $\rightarrow \hat{y}$

Teacher's Signature _____

Deepakydv

Date: Day-7

Topic:

Backward Propagation:

Once we get $\hat{y}$ after applying activation function.

We would try to calculate loss function with $\hat{y}$ and our original value y.

Loss fn" = $(y-\hat{y})$ (Suppose)

W.r.t we update all the weights (w') shown in previous fig.

$\frac{\partial L}{\partial w'} \Rightarrow w'$ (first updation) $\frac{\text{derivative of loss}}{\text{derivative of weights}}$

Weight updation formula.

$$w'_{new} = w'_{old} - \eta \frac{\partial L}{\partial w'}$$

η = learning rate

$\frac{\partial L}{\partial w} = \frac{\text{derivative of loss}}{\text{derivative of } w'}$

$\frac{\partial L}{\partial w'}$ can be found using simple chain rule.

$$\frac{\partial L}{\partial w'} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w'}$$

$$w'_{new} = w'_{old} - \eta \frac{\partial L}{\partial w'}$$

Again $\frac{\partial L}{\partial w'}$ can be found by chain rule.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_t} \cdot \frac{\partial O_t}{\partial w}$$

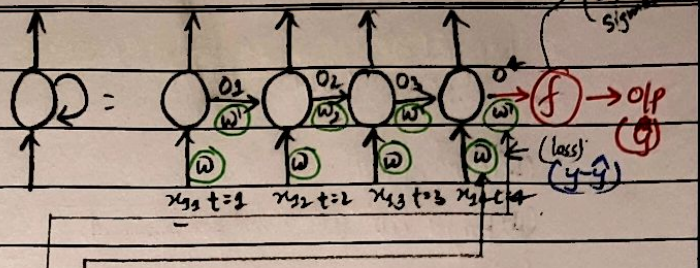
(chain rule)

Similarly for $t=3$ w', chain rule would be.

$$\frac{\partial L}{\partial w'} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial O_t} * \frac{\partial O_t}{\partial w'}$$

As we move back in backward Propagation there are chances that loss value will keep on getting smaller and smaller and if it is getting smaller & smaller weight updation is negligible.

In order to fix this issue we need to use LSTM RNN.



Let say there's one sentence,

"My name is Krish and I want to eat Pizza"

When we use RNN, RNN will try to make sure that each and every word will read correct in orders and ensure every word is important. Here "eat" is dependent on "Pizza". In this case "eat" is just before "Pizza". So it will be able to capture content by "I" is related to "Krish" "I" & "My" are also related. It signifies that O_3 can be sometime dependent on O_1 and O_4 may be dependent on O_1. This sometimes because of length it will not be ~~any~~ able to capture content properly. That is why we specifically use LSTM RNN.

⇒ RNN only remembers short term memory.

Problems with RNN :-

Suppose Predict the next word.

- On Sunday I want to eat Pizza
- On Monday I want to eat ____

Q-Which RNN would be used here?

⇒ Many to One

RNN is good for short sentences, not for long.

LSTM RNN

→ LSTM RNN remembers long term memory.

→ Good for longer sentences.

Teacher's Signature _____

Deepakydv

Date: Day-8

Topic:

* Colah blog → for detail on LSTM

Long Short Term Memory: (LSTM)

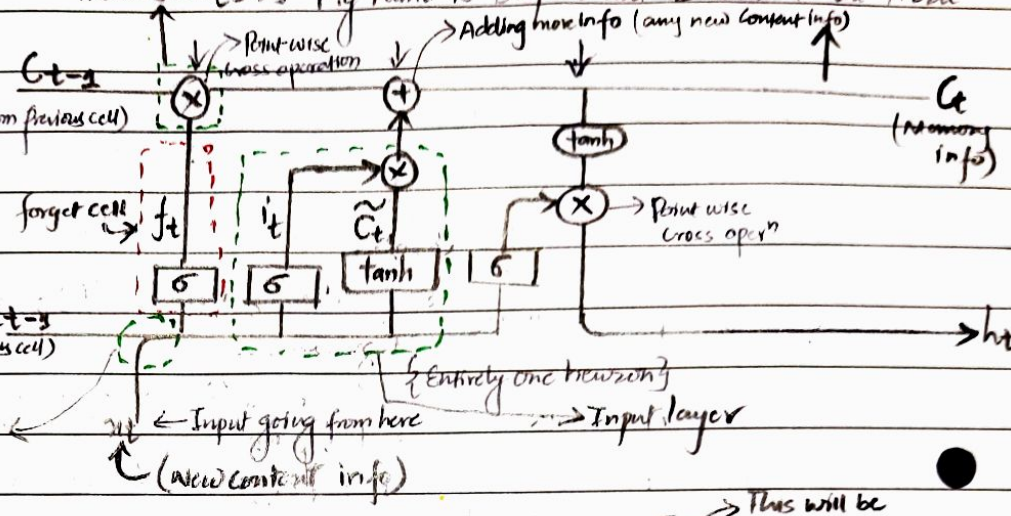
(RNN)

Suppose an example below:-

memory cell {R} {S} {V}

My name is Deepak and I want to eat Pizza

(I) Memory Cell
(II) Forget cell
(III) Input cell



Krish like DS → One content → lets suppose previous cell we pass this content → This will be forgeted

Yann lecun like CNN → another content → Here content changing → This will be remembered.

→ new content info.

(18) # Why LSTM RNN instead of RNN?

→ Vanishing Gradient or Dead Neuron

→ Content info. Deep RNN ['Huge'] ⇒ Gaps.

tanh → activation function.

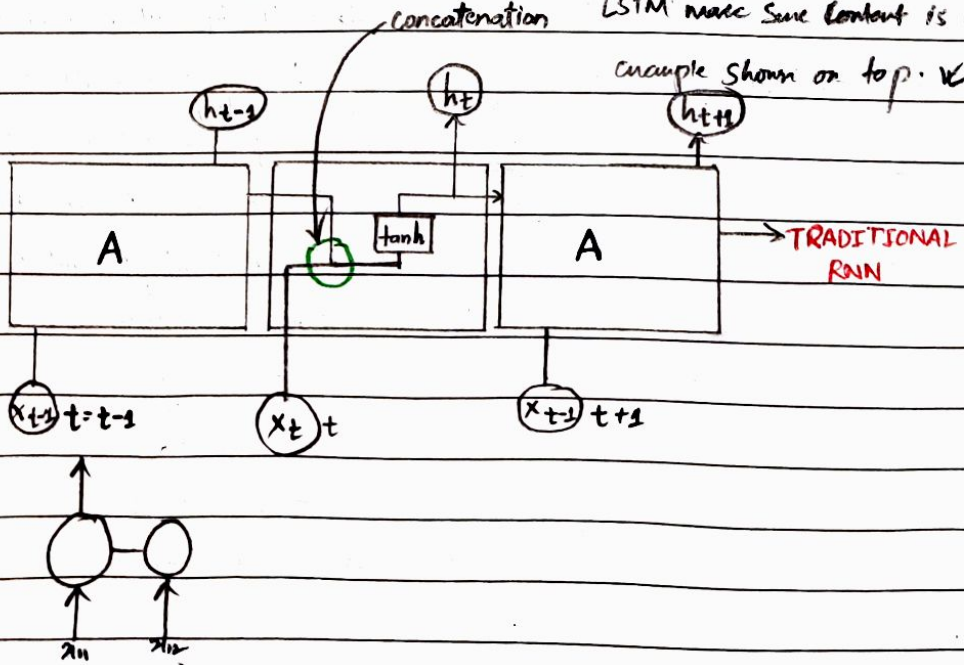
from previous state whatever o/p we get we are combining it with input we provides.

lets say if you have very deep neural network and some of the output is really dependent on first word then this content cannot be easily captured with the help of RNN, so we need LSTM RNN.

LSTM is important way to remember any important content.

LSTM make sure content is captured.

example shown on top. ↑



Deepakydv

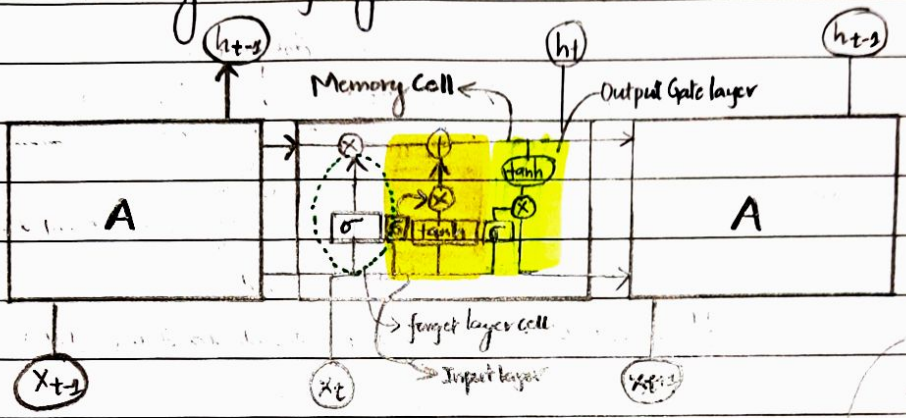
Teacher's Signature

Date: _____ Topic: _____

(I) Why LSTM RNN?

⇒ Whenever we have long sentences and suppose there is dependency of the o/p probably with the first word, or second word, and we want to predict 3rd word, Content will be switching in case of longer sentences.

t=1 t=2
lets consider one long sentence,
t=1 t=2
My name is KRISH and my friend name is
Here on reading LSTM model will find dependency
Krish friend name can be related to Krish.
→ Content switching and dependency is happening



→ Here Content switch is happening

σ → neural network with sigmoid activation function.

tanh → neural network with tanh activation function.

KRISH like pizza but his friend likes burger
forget this info

→ Concatenate

→ Copy

→ Vector transform

○ → Point wise operation

Memory Cell will act as conveyer belt → Airport luggage Chain
Similarly in memory cell you can add info or you can remove info.
You can add / remove luggage.

Forget layer cell: In this cell if previous info and passed info are similar then most of values will be nearer to 1 If this two will not be similar, then most of values will be nearer to zero. If it is nearer to zero that means we are making memory cell to forget the the previous info.

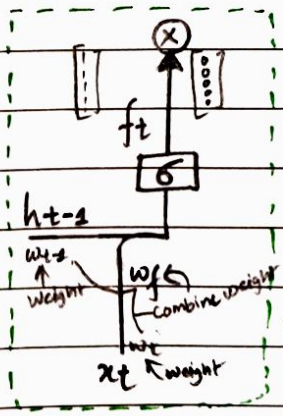
σ-sigmoid transforms values between 0 to 1

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

bias

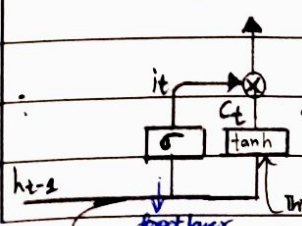
Lets consider 2 sentences
① Krish like pizza but he doesn't like burger
② Krish like pizza but his friend like burger.

Here in Sent 1 Content switching is not happening in Sent 2 Content Switch is happening.
Incase of sent 1 info will not be lost because Content switching is not happening.
But in sent 2 the content switching is happening so it will forget the Previous Content. "Krish like Pizza" will be forgotten.



forget gate layer

Input layer cell: let us consider same example-2
"KRISH like pizza but his friend like burger.
Since the content is changing o/p should be nearer to zero.
but on apply tanh o/p should be [+1 to -1]
≈ → only newer info will be passed after doing point wise operation. This passed info will merged with above cell.



Important! Point wise operation ⇒ dot product ⊗
⊕ → addition.

your o/p is b/w [-1 to +1]

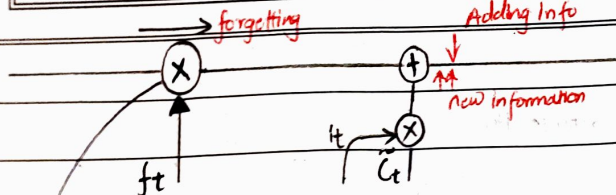
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$

Deepakydv

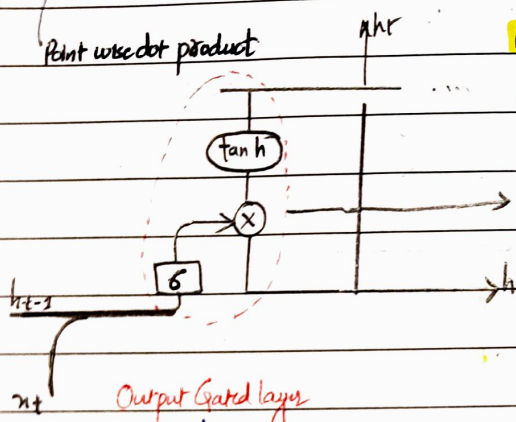
Teacher's Signature _____

Date: _____

Topic: _____



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



Output Gated Layer : from the memory cell apply tanh operation so we get recent content info then it will get point wise dot product.

In short here we are trying to combine both the info previous forgotten info and the recent info and that is actually passed as an o/p to next cell. This will keep going with different - different timestamp.

In short all useless data is removed, only important data is considered and forwarded to the next cells.

Output Gated layer

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

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

Deepakydv

Teacher's Signature _____

Date: Day - 33

Topic:

LSTM

Agenda

(I) Bi directional LSTM RNN

(II) Encoder and decoder

(III) Transformer, Bert, GPT

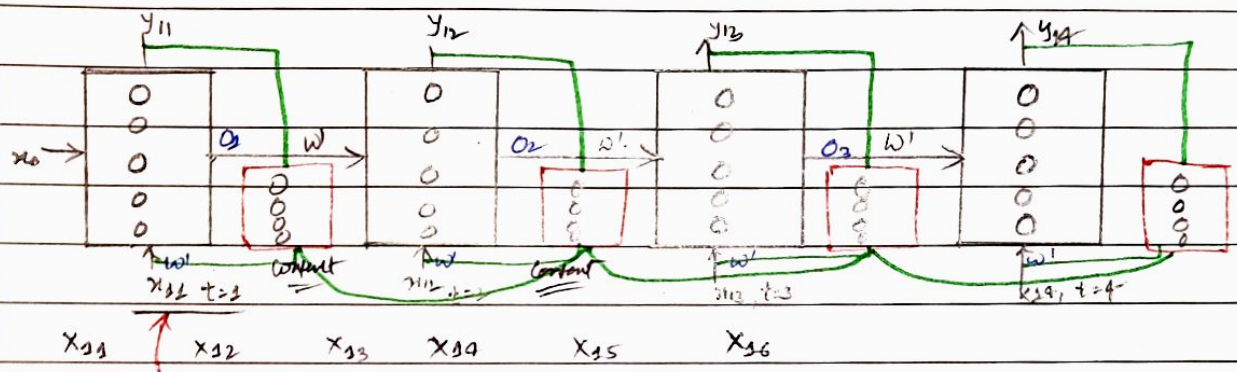
(IV) Practical Application.

Bi-directional LSTM RNN:

Predicting the next word:

Example: KRISH LIKES TO EAT ____ IN BANGALORE

Simple RNN



Embedding layers. (Many to Many RNN)

Here in case of simple RNN if we need to find y_{13} it will have content of all the previous layers like x_{11}, x_{12}, etc. But y_{13} will not have content of forward words (y_{14}). But in our example model should predict the next word based on our forward word, fo eg, if its bangalore, as forward word model might predict food as Dosa or if its Lucknow as forward word it might predict the next word as Kabab.

Since our model has to predict the next word with respect to the content of the forward word. Simple RNN fails here and Bi-directional RNN comes into the Picture.

Here bidirectional RNN is designed adding extra node next to every node of simple RNN model and adding content of previous o/p to the forwards as shown in fig:

Now on constructing bi-directional neural network Now y_{13} will have content information of forward words and will have also content information of Previous words. Now when training this model in correct way its possible to predict next of next words.

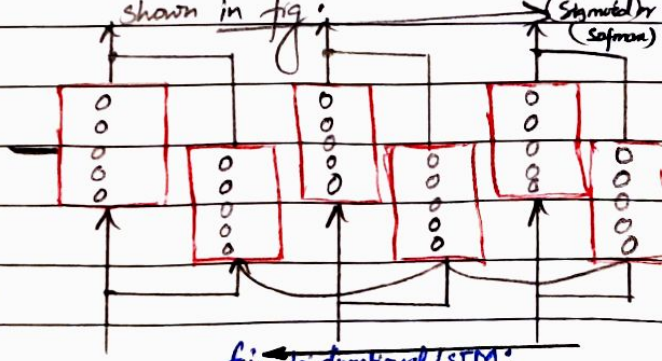


fig: bi-directional LSTM:

Have content of both
- forward Words
- Backward words

Teacher's Signature

Deepakydv

Date: Day 9

Topic:

Mangalam

Agenda

(i) Word Embedding layer → Tensorflow & Python

(ii) LSTM Sentiment Analysis.

Word2Vec
Word embedding layer → (Words → Vector)
Training

Suppose we have a sentence like below:

[KRISH LIKES PIZZA] (Vocabulary size = 500)

[SHYAM LIKES BURGERS]

Vocabulary size means no. of unique words.

steps

(1) Sentences

(2) OHE - One Hot Encoding → Vocabulary size

(3) Padding → Post Padding, Pre-Padding

(4) OHE → Vectors

We do not use any pre-trained model, instead we're using dataset then we're going to train it using embedding layer and then find out the relationship.

Embedding layer and convert to Vector.

#Convert each and every value with some vectors

[0 0 0 0 0 120 150 160]

feature size = 10

⇒ All the sentences length must be same in-order to train into neural network.

Let's look back to steps once more:

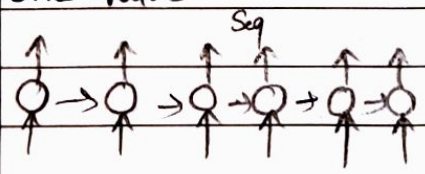
(I) Sentences

(II) Perform OHE (One Hot Encoding)

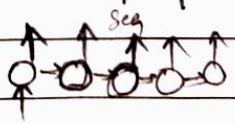
(III) Padding — (Pre and Post)

(IV) OHE - Vectors

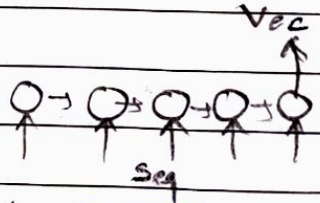
(V)



Seq - Seq (many to Many)



Seq - Vec



Seq - Vec

Deepakydv

Teacher's Signature

Date: NLP (27/03/23)

Topic: Encoders and Decoders:

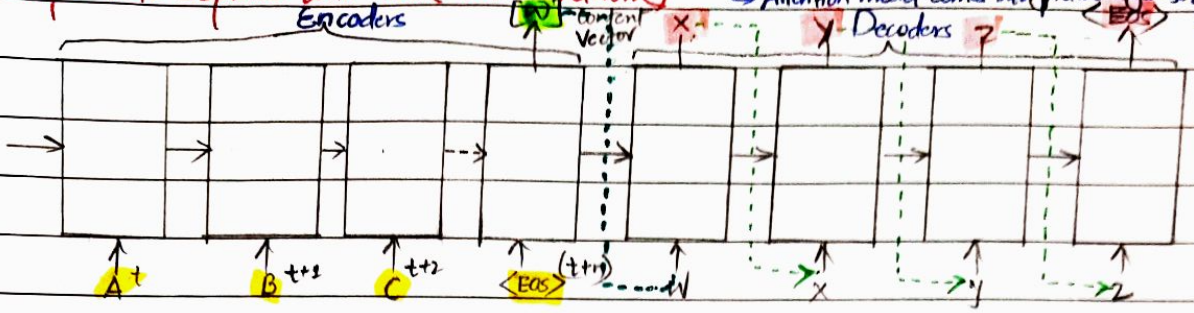
Must Know
- RNN
- LSTM
- GRU

*

$X = \langle x_1, x_2, x_3 \rangle$ $y = \langle y_1, y_2, y_3 \rangle$ unrival and sequence.

Sequence to Sequence with NN (Neural Network)

Disadvantage :- Not Good accuracy with long sentences
↳ Attention model comes into picture which use Bidirectional LSTM.



In sequence to sequence the model, takes in sequence of information and output sequence of information, that's one of many example.

for ex: language translation:
for eg, english to french

If you given i/p as an image and if that particular image converted to text which is also called as "Image Captioning".

for ex: google image search.

If you search "Dog Image" in google search you can see images of dogs which is great example of "Image Captioning".

Also automated reply of "LinkedIn" is an example.

Let's suppose ABC as english character and XYZ as french character.

Here encoder and decoder will be trained by pass A, B, C at different timestamps like t, t+1, t+2 etc finally end of string will be passed, we will be getting some content Vector (CV) where all these 3 words A, B, C will be represented in the form of content Vector. After the representation it will be passed to decoder as an input.

In case of image captioning in case of LSTM we use advance CNN (VGG16, RESNET) or any transfer learning etc. at flatten layer pick up all vectors and pass them to decoders.

We can build encoder-decoder model with RNN, LSTM and GRU, mostly LSTM is considered best to build "encoder-decoder" model. Here the above diagram is build with LSTM model. Encoder doesn't contains o/p but decoder should contain both encoder i/p as well as o/p.

In language translation we need to pass text in the form of vectors. We may use various technique for that purpose like embedding layer, OHE, Word2Vec etc.

All the operations remains same as LSTM but only thing to remember is that there's no o/p for encoders.

After that first decoder will be giving o/p as X which will be passed to next decoder as an i/p. and then second decoder will give o/p as Y which would be passed to third decoder as an i/p and so on... and at last we get end of string.

#LSTM actually works on probability it is computed as below.

$Pr\left(\hat{y} / \langle x_1, x_2, x_3, \cdots x_n \rangle\right)$

$\hat{y}$ is combination of $\hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4 \cdots \hat{y}_n$

loss will be calculated b/w y and $\hat{y}$ for this we can use optimizers (Adam) ↓ loss by back Propagation.

Applications:
- grammarly
- whatapp
- Gmail.
- Sentiment Analysis

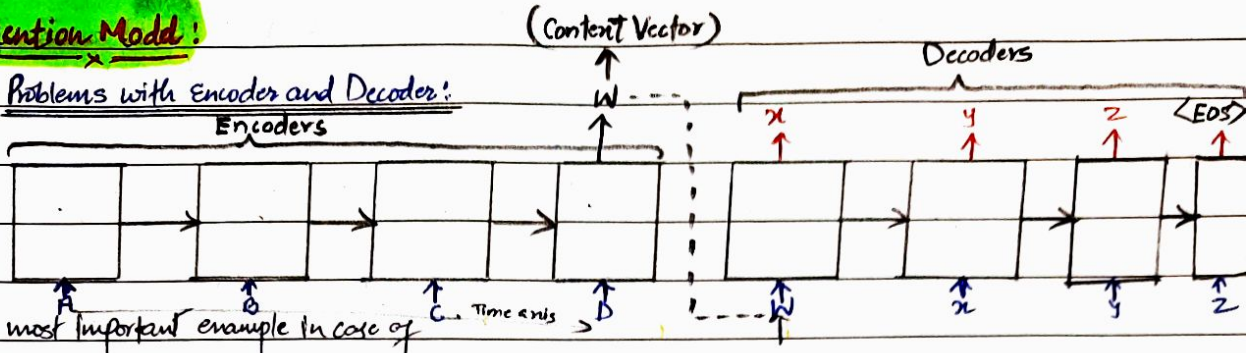
Teacher's Signature

Deepakydv

Date: _____ Topic: _____

Attention Model !

Problems with Encoder and Decoder:



The most important example in case of encoder-decoder is "Machine-Language-Translation".

Sequence to Sequence.

As we know encoder takes i/p and finally we take this particular vector "w" which we pass it to decoder for the translation.

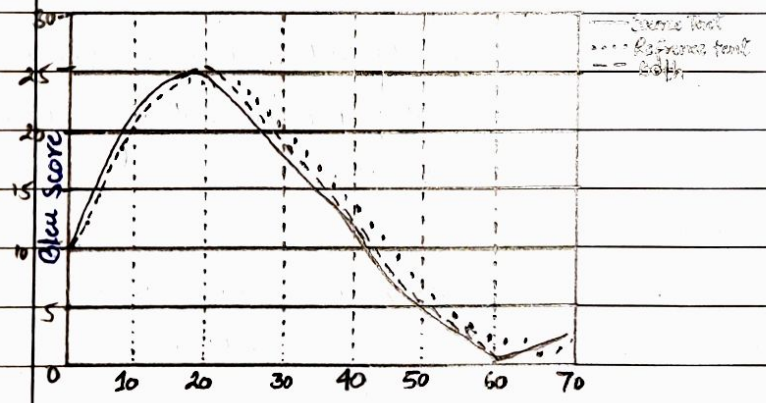
The "w" is basically called as "content Vector".

This vector is generated at the end of the string <EOS>.

Suppose if we take sentence as an i/p then at the end of the string, for that particular string processing would be happening in the LSTM and finally we get 'w' vector.

This w vector is then passed to decoder layer and based on that we're getting o/p w.r.t the other language till the end of statement.

Bleu Score:



"loss in Model skill with increased sentence length"

"Sentence length ↑, accuracy ↓"

When sentence length is increasing up to specific length that time the accuracy of translation is decreasing.

The major problem is we are not taking o/p w.r.t every i/p sentences. If we would take o/p of every i/p sentences it would capture the content properly w.r.t every individual sentences, which will obviously have context to nearest word.

This problem could be fixed using "bi-directional LSTM".

As we know that $\vec{w}$ (content vector) is created based on the i/p words that we're given.

We've take just one vector at the end of the all input sentences. that we've not take o/p w.r.t every i/p sentences. This single content vector cannot justify the content of every i/p sentences we've passed.

It is basically not being able to represent the content of whole sentences.

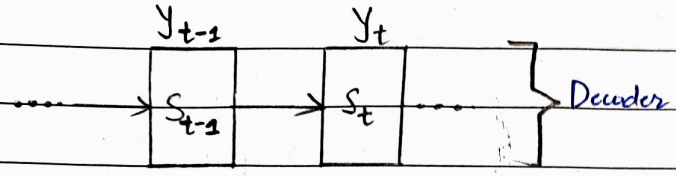
Which is the reason when we passed to decoder its not being able to translate sentences properly.

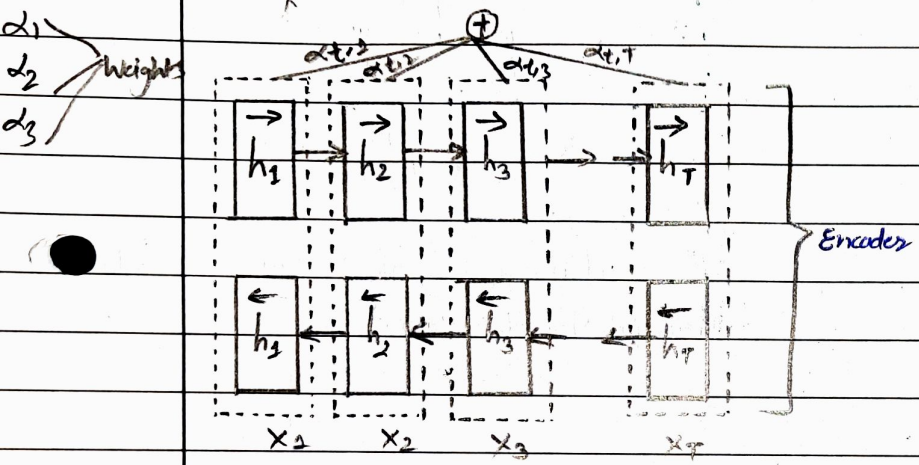
Deepakydv

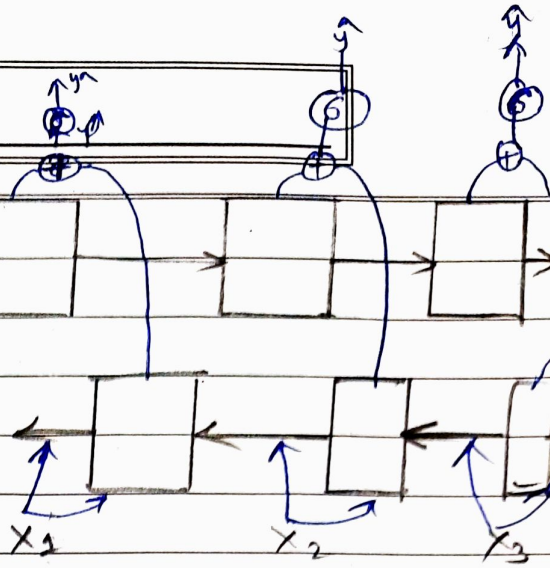
Teacher's Signature _____

Date: _____ Topic: _____

In case of normal sequence to sequence model both encoder and decoder are uni-directional.





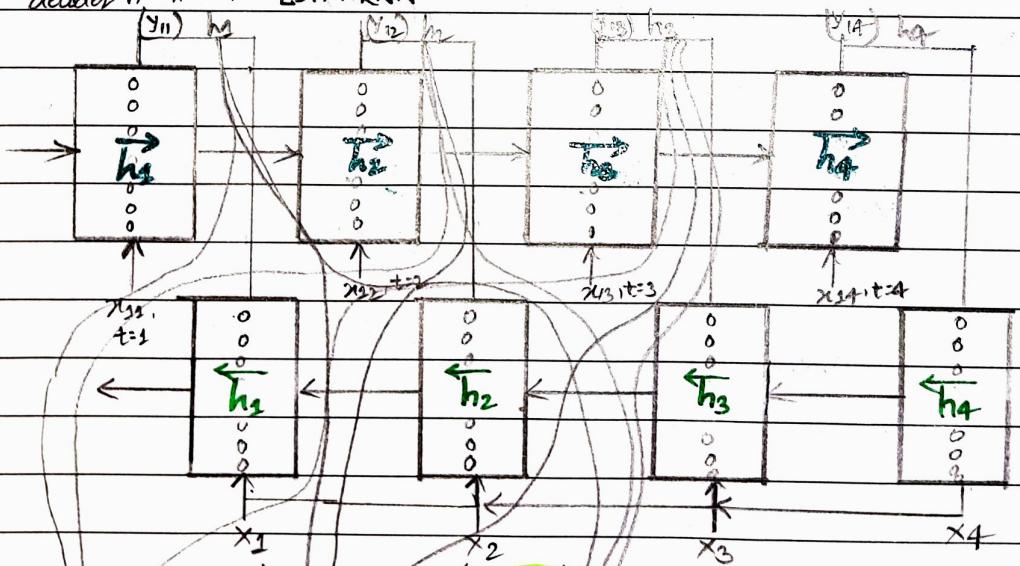


Using bi-directional LSTM we can not only predict previous content but also the future content of words.

In encoder we will use "bi-directional LSTM RNN"

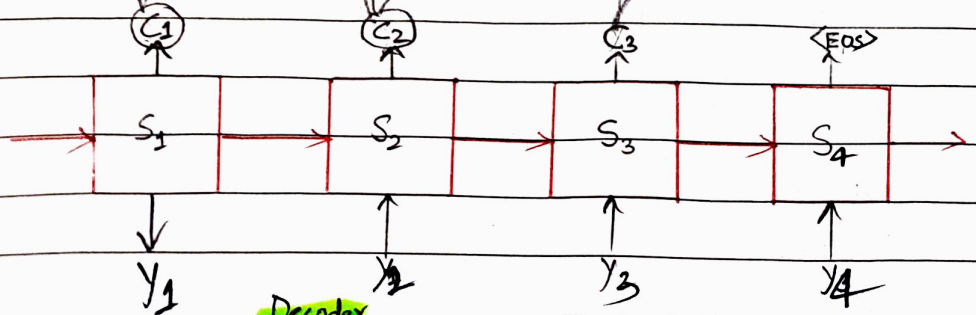
In decoder " " "LSTM RNN".

$h_1, h_2 \cdots h_n \rightarrow$ Hidden layer



Bidirectional LSTM (encoder)

$x_1, x_2, x_3 \cdots x_n \rightarrow$ Words



Decoder

* Language Translator

Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

→ LSTM RNN Works on probability

→ When T=3, (Window size is 3)

→ We get 3 α values. $\alpha_1, \alpha_2, \alpha_3$ for first word X_1

→ α values will be initialized by formula,

$$\sum_{T=1}^{Tx} \alpha_i = 1$$

$\alpha_1, \alpha_2, \alpha_3 \Rightarrow$ Sum is always 1

$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$

The Context Vector is generated by below formula.

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

(Multiplying o/p of bi-directional LSTM with α value)

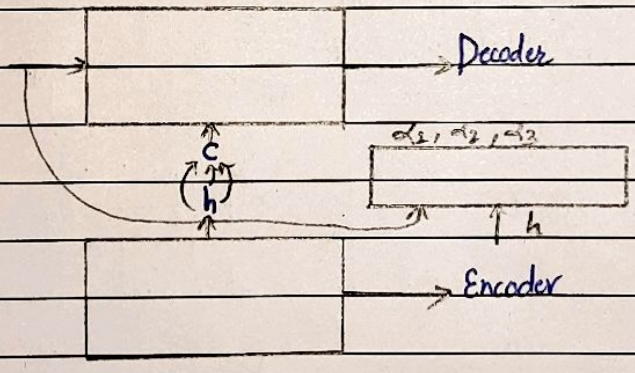
This is the property of feed forward neural network.

The main reason of applying feed forward neural n/w is for back propagation, because we need to update value of $\alpha_1, \alpha_2, \alpha_3$.

Unless and until we don't update weights we can't get good accuracy.

Feed forward neural network is same like ANN (Artificial Neural Network)

Let's Summarize,



Here, consider below block as encoder and upper block as decoder. In decoder, we are giving some context vector 'C_2' and in encoder, we're getting o/p as 'h'

This 'h' based on T_x will be generating some value $\alpha_1, \alpha_2, \alpha_3$ and so on. The based on h We will be assign $\alpha_1, \alpha_2, \alpha_3$ as weights.

We will desire these weights through feed-forward-neural network.

This neural n/w will take 2 input one is h and another as previous state of decoder.

After taking i/p it will desire $\alpha_1, \alpha_2, \alpha_3$ based on T_x value. Once it desires this then multiply h with $\alpha_1, \alpha_2, \alpha_3$ to get context vector 'C_1'

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

↳ Context Vector.

As we said earlier sum of $\alpha_1, \alpha_2, \alpha_3$ is 1. We can use this Softmax formula to compute such value.

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

Where. $e_{ij} = a(S_{i-1}, h_j)$

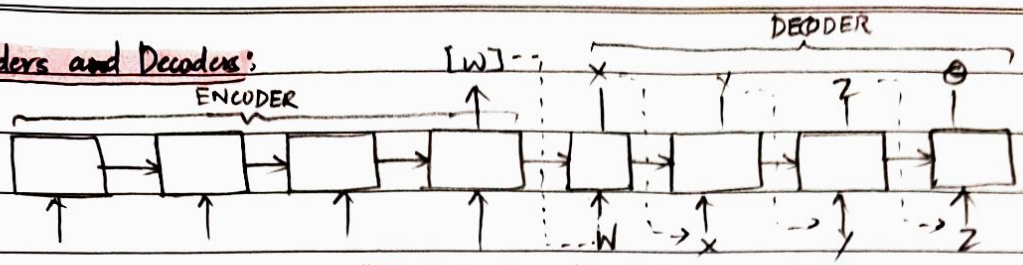
a = function.

Teacher's Signature _____

Deepakydv

Date: _____ Topic: _____

Encoders and Decoders:



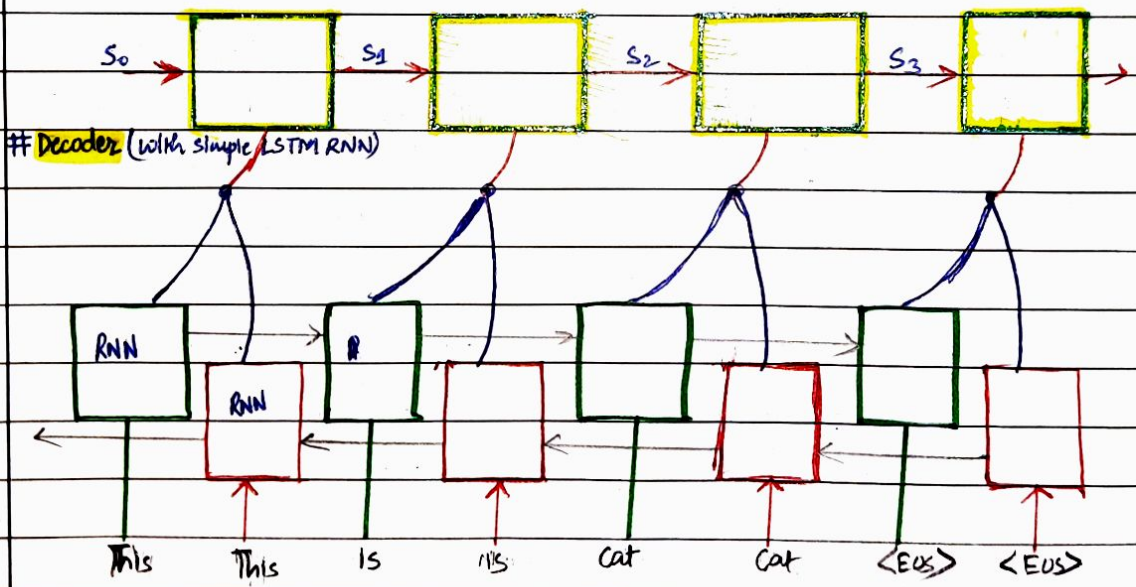
Encoder-Decoder model with Simple LSTM RNN

Disadvantages

→ Uni-directional RNN (Won't understand sentence in backward direction).

→ Collecting content at the end. (qns-ans, text-summarization, text abbreviation, - in these cases all input won't contribute)

So we need some kind of a network which will be able to understand that what I need as an i/p to give some output. This is where concept of attention comes into picture.



Decoder (with simple LSTM RNN)

Encoder (with bidirectional LSTM)

Encoder-Decoder model with Bi-directional LSTM RNN.

In case of translating one language to another, in case of question answer system or fill in the blanks model, system which generate new story by understanding existing story. System that would tell a story about any particular image or about video or video analytics.

In all these cases we should look for a model which will be able to focus on certain set of i/p and based on that focused i/p it should be able to give some o/p.

This model may get failed because we're use one-to-one mapping. (Input → O/P) but we're not focused anywhere.

Teacher's Signature _____

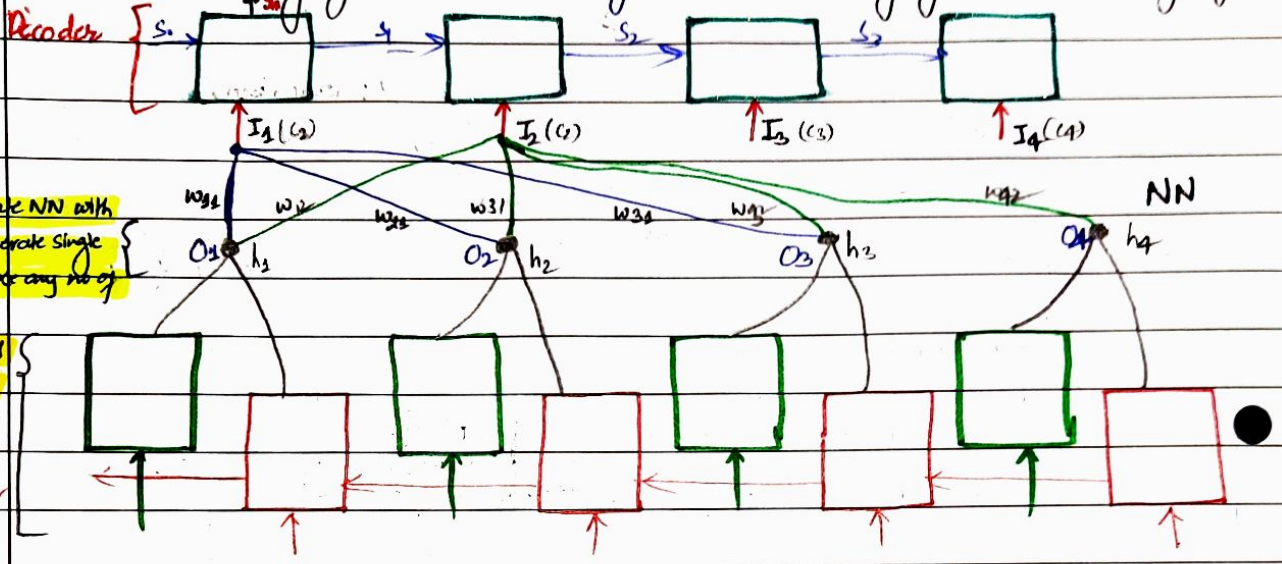
Deepakyadav

Date: _____

Topic: ATTENTION MODEL

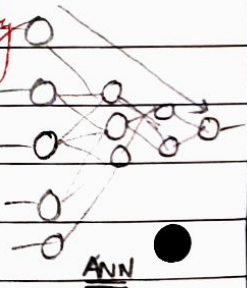
This is where "attention model" comes into picture. In attention model "neural network" added along with the encoder-decoder. "neural network" helps in focusing, which i/p we would like to focus.

Ex:- The bull is running high <<Stock Market Rising>> Here "is" is not having significance. "bull" "running" "high" should be focused.



As we pass "This is Cat" or "The bull is running high" "Here what we want the neural network to do is that focus on all the words and find out the significance. "In case of translation" it will focus on entire sentence and finds out significance like noun, pronoun, verb, adjective etc and priority of focusing would be different. This is what NN will do.

for every i/p to the decoder (cell) we have one neural network, which is trying to pick and choose, which is trying to decide that which one is having a high priority and based on that it will be generating i/p to decoder and decoder will generate $\hat{y}_i$ finally.



Because of applying these neural network researchers have named this concept as "attention model" So that it will be able to focus which word is responsible for giving some kind of o/p we desired.

Input inside a decoder cell (one single decoder cell) is nothing but its basically a weight. Which we're trying to multiply with weight and o_{ij} (o/p - combined encoder)

The context Vector c_i is given by:

$$c_i = \sum_{j=s}^{T_x} \alpha_{ij} h_j$$

c_i = Context Vector i/p.

T_x = no. of attention i/p that are injected in neural network. (In how many words we're supposed to pay attention)

$c_i = I_i$, $T_x = 4$ } $h_j = O_i$ } ✱ {above case} $\alpha_{ij} = w_{ij}$

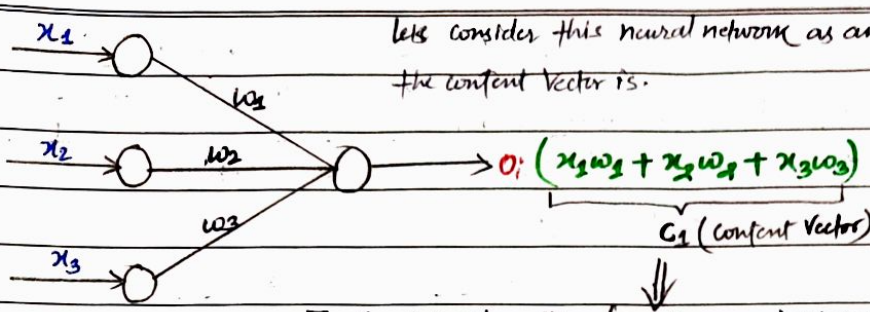
h_j = Combined output (outcome) from encoder side.

$\alpha_{ij} = (w_{ij})$ = Weights $\{w_{11}, w_{12} \dots\}$ ✱

Teacher's Signature _____

Deepakydv

Date: _____ Topic: _____



Lets consider this neural network as an example to see what's the content vector is.

The o/p that we're getting from the neural network which will go as i/p to the decoder network, which is responsible for giving final o/p.

$C_1 = I_1 = O_1w_{11} + O_2w_{21} + O_3w_{31} + O_4w_{41} \ldots\ldots$ as per previous diagram.

Whenever we're trying to generate weight, we're supposed to choose an equation with activation factor in such a way that it will not be able to overshoot or learn weight by any means.

Keeping this in mind researchers decides that $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$

$e_{ij} = a(s_{i-1}, h_j)$ → attention function.

→ outcome of decoder network

→ ~~Input~~ outcome from encoder n/w.

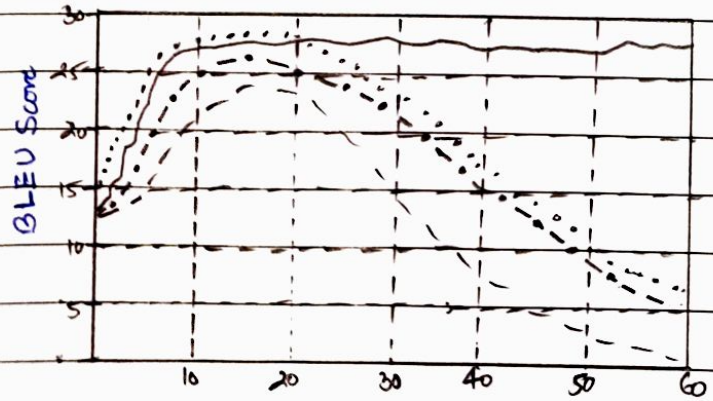
S → feedback

$\Sigma \alpha_{ij} = 1$

$\alpha_{ij} \geq 0$

'α' is nothing but its weight as we concluded earlier.

So the major thing is "How would I be able to control the weight?" because weight should not expand itself or shrink itself. for that we are supposed to put some kind of restrictions although its a trainable parameter. That means network would try to train it through backward propagation.



Teacher's Signature _____

Deepakydv

Date: _____ Topic: _____

Steps of using transformers in wdc:

(I) Install the transformer ⇒ `! pip install transformers`

(II) Call the pre-trained model (eg. Distilbert)

(III) Call the tokenizers (for specific pre-trained more, we have specific tokenizers) if you're using pre-trained related to "Distilbert" then use tokenizers relating to same.

⇒ `from transformers import DistilBertTokenizerFast`

`tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')`

(IV) Convert encodings into Dataset objects (eg, tensors in case of tensorflow)

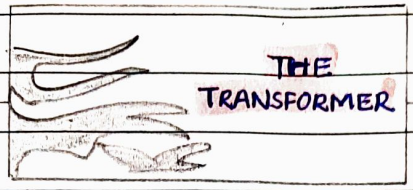
Deepakydv

Teacher's Signature _____

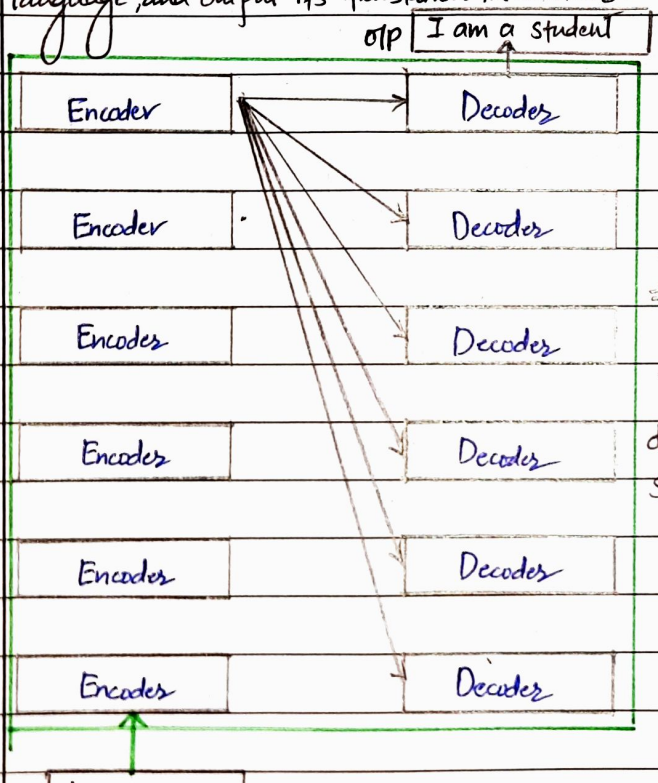
Date: _____

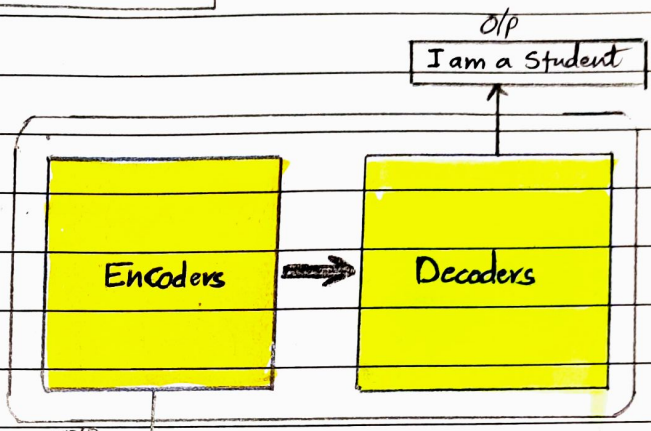
Topic: TRANSFORMER & BERT

TRANSFORMER



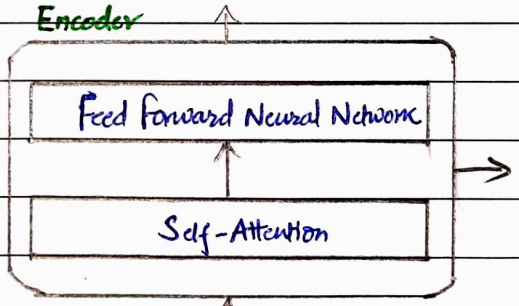
Let's begin by looking at the model as a single black box. In a machine translation application, It would take a sentence in one language, and output its translation in another.





encoding component, decoding component and connection bw them.

The encoders are all identical in structure (yet they do not share weights). Each one is broken down into two sub-layers.

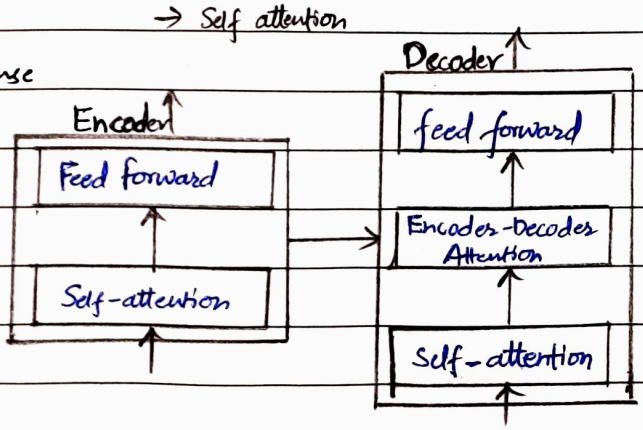


The stack of 6-encoders and 6-decoders provided good result to the researchers.

This is kind of hyper-parameter, you may use them at your convinient.

→ Feed forward neural network (Similar to ANN)

→ Self attention



Deepakydv

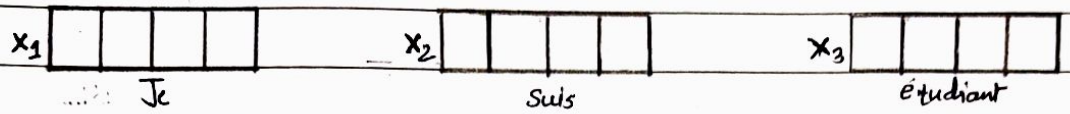
Teacher's Signature _____

Date: _____ Topic: _____

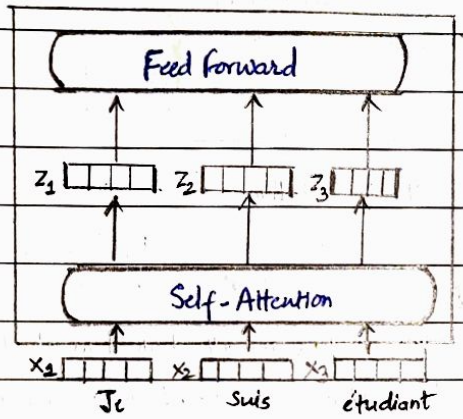
Bringing The Tensor Into the Picture.

As in the case of NLP applications in general we begin by turning each input word into a vector using an embedding algorithm.

> An encoder receives a list of vectors as input. It processes this list by passing these vectors into a 'self-attention layer', then into a feed-forward neural network, then sends out the output upwards to next encoder.



Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.



Here each words like "Je" "Suis" "étudiant" is converted into vector through Word2Vec with 512 dimension and this is basically passed to self-attention. In this process all the words are passed parallelly, all at one time. After that it goes to feed-forward neural network.

Self-Attention at a High level

Here lets take an example of below sentence.

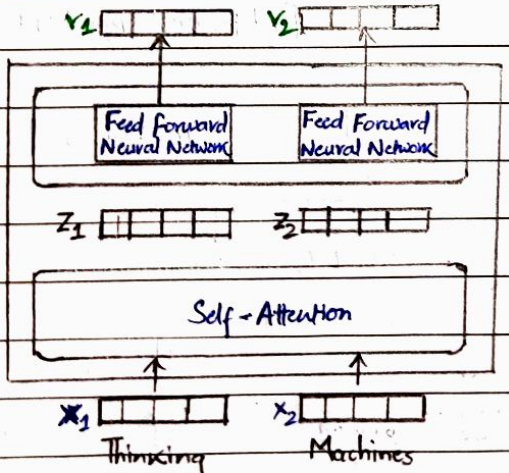
"The animal didn't cross the street because it was too tired"

What does it in the sentence refer to? Is it referring to the street or to the animal? It's a simple question to human, but not as simple to an algorithm.

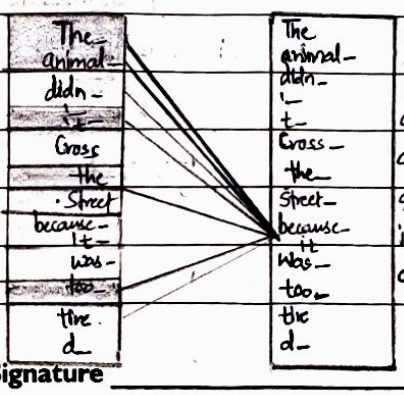
When then model is processing the word "it", self-attention allows it to associate "it" with "animal".

Encoder2

Encoder1



The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it seprately.



As we are encoding the word "it" in encoder. The animal" are strong correlated and baked a part of its representation into the encoding of "it".

Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

Self Attention in Detail

-> Word2Vec embedding (converted word to 512 dim)

Step I

Input (512 dim) Thinking Machines

Embedding x_1 x_2

Queries (64 dim) q_1 q_2

Keys K_1 K_2

Values V_1 V_2

W^Q W^K W^V

Why it is getting multiplied
→ Because we know that formula input × weight + bias and activation function. applied in case of DL models
→ Concept of ANN.

But in this case we just taking this i/p and multiplying by W^Q we get queries q_1 parameters which would be 64 dimensions

Multiplying x_1 by the W^Q weight matrix produces q_1 the "query" vector associated with that word. We end up creating a "query" a "key" and a "value" projection of each word in the input sentence.

Similarly x_1 Vector gets multiplied with W^K and then we get K_1 values. and x_2 with W^K get K_2 values.

Same in case of V_1 and V_2

$x_1 \times W^V \rightarrow V_1$

$x_2 \times W^V \rightarrow V_2$

Step-II

Input	Thinking	Machines
Embedding	x_1	x_2
Queries	q_1	q_2
Keys	K_1	K_2
Values	V_1	V_2
Score	$q_1 \cdot K_1 = 112$	$q_1 \cdot K_2 = 96$

In the second step we will be creating score for each and every words by multiplying queries with the keys.

The score is calculated by taking the dot product of the query Vector with key Vector of the respective word we're scoring. So if we're processing the self attention of the word in position #1. the first score would be the dot product of q_1 and K_1. The second score would be the dot product of q_1 and q_2.

Step-III

Input	Thinking	Machines
Embedding	x_1	x_2
Queries (64 dim)	q_1	q_2 (64 dim)
Keys	K_1	K_2
Values	V_1	V_2
Score	$q_1 \cdot K_1 = 112/8$	$q_1 \cdot K_2 = 96/8$
Divide by 8 ($\sqrt{64}$)	14	12
Softmax	0.88	0.12

What ever score we get we are dividing by 8 (Sq root of the dimension of key Vector) that is $\sqrt{64} = 8$.

This leads to have more stable gradients. There could be other possible values here, but this default. After that we're passing result to softmax operation. Softmax normalizes the scores. so they're all positives and add up to 1.

Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

Step-IV As discussed we're apply Softmax activation function in step-IV. Whenever you provide any I/p to softmax the total value is 1.

Here we got value after dividing score by 8 are 14 and 12 applying softmax activation to these value we get 0.88 and 0.12 and summing them we get (0.88+0.12)=1. ✓

As per softmax score more importance is given to "Thinking".

Step-V

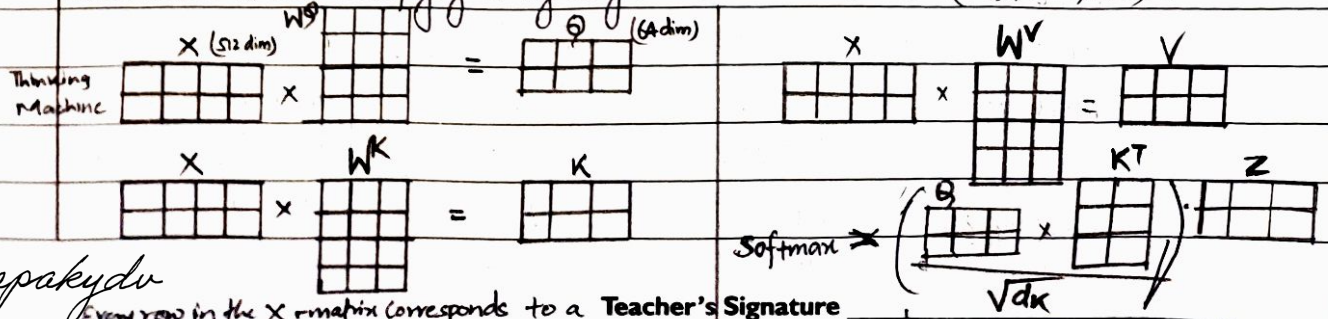
Input	Thinking	Machines	In this step whatever softmax value we got in step-IV is multiplied by Values, we get new vectors V_1' and V_2'
Embedding	X_1	X_2	
Queries	q_1	q_2	
Keys	K_1	K_2	
Values	V_1	V_2	
Score	$q_1 \cdot K_1$	$q_1 \cdot K_2$	Finally we add V_1' and V_2' to get Z_1.
Divide by 8 ($\sqrt{dk}$)	14	12	
Softmax	0.88	0.12	
Softmax X Value	V_1'	V_2'	
Sum	Z_1	Z_2	

Step-VI Z_1 will be the output of the self attention layer when you pass "Thinking" and Z_2 will be the o/p of the machine word when you are passing through the "Self attention".

That concludes the 'self-attention calculation - The resulting vector is one we send along to feed-forward neural network. In the actual implementation, however this calculation is done in matrix form for faster processing. So let's look at that now.

Matrix Calculation of Self-Attention

The first step is to calculate the Query Key and Value matrices, We do that by packing our embeddings into a matrix X and multiplying it by weight matrices we've trained (W^Q, W^K, W^V).



Every row in the X matrix corresponds to a word in the input sentences.

Teacher's Signature

→ O/p of self-attention layer.

weights will be initialized randomly.

Deepakydv

Date: _____ Topic: _____

We are considering W^Q, W^K, W^V the same weights for all the words, we will call it as 'Single head attention'.

This is further improved when we use "multi-headed-attention".

The Beast with Many Heads:

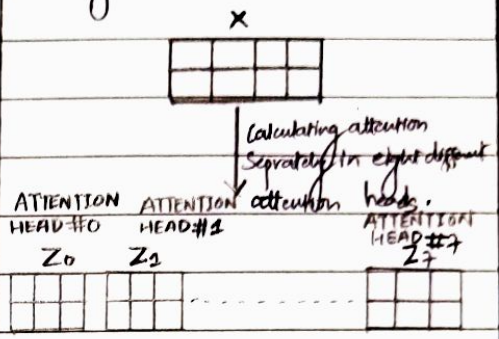
"Multi-headed" Attention

→ By using multi-headed attention, improves the performance of the attention layer in two ways.

(1) It expands the model's ability to focus on different positions. In the example above "Z_1" contains a little bit of every other encoding, but it could be dominated by the actual word itself. If we're translating a sentence like "The animal didn't cross the street because it was too tired", it would be useful to know which word "it" refers to.

(2) It gives the attention layer multiple "representation subspaces". As we'll see next, with multi-headed attention we have not only one, but multiple sets of Query/Key/Value weight matrices (the transformer uses eight attention heads, so we end up with eight sets for each encoder/decoder). Each of these sets is randomly initialized. The after training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace.

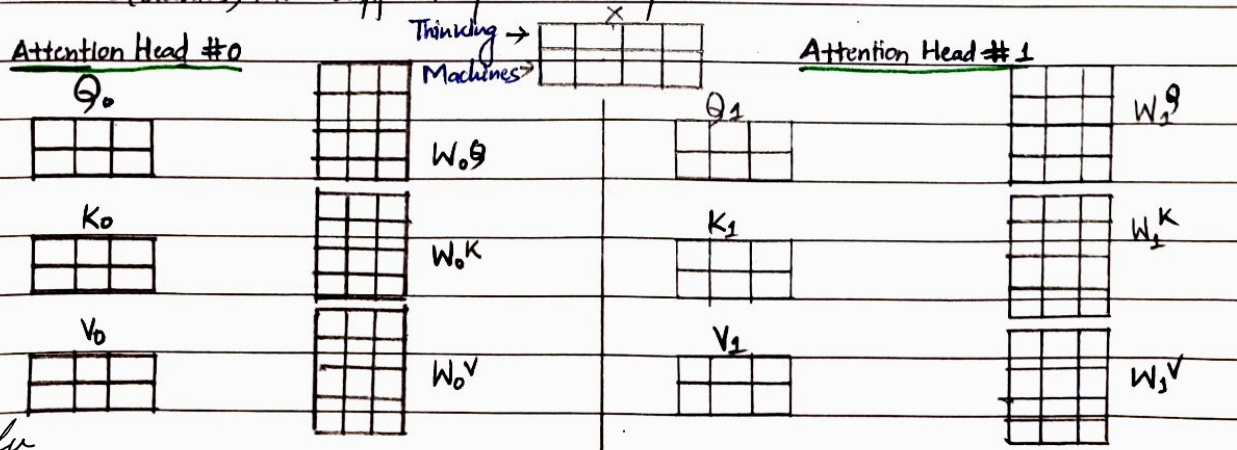
If we do the same self-attention calculation we outlined above, just 8-different times with different weight matrices, we end up with eight different Z-matrices.



This leaves us with a bit of a challenge. The feed-forward is not expecting 8 matrices - it's expecting a single matrix.

We concat the matrices then multiple them by an additional weights matrix W^O.

P.T.O



With multi-headed attention we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before we multiply X by $W^Q/W^K/W^V$ matrices to produce Q/K/V matrices.

Deepakydv

Teacher's Signature _____

Date: _____

Topic: _____

(I) Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7

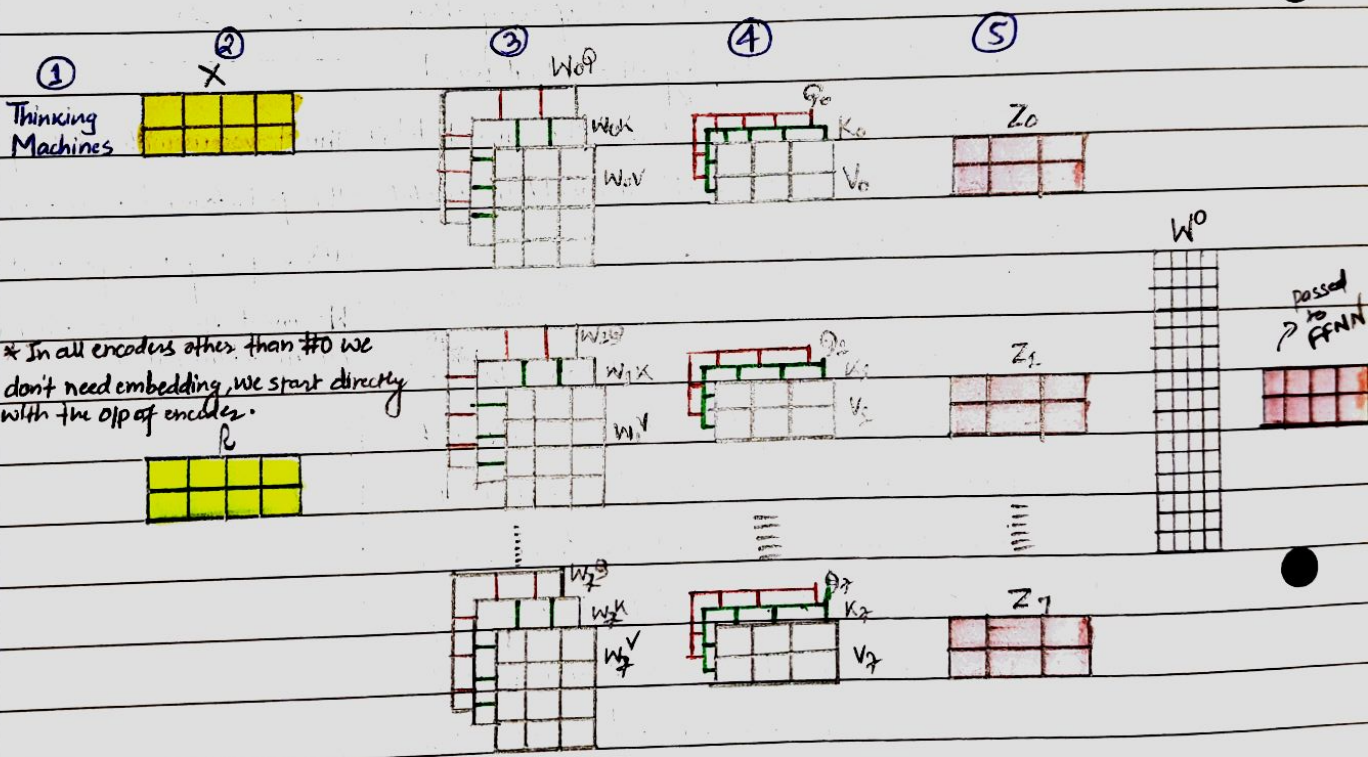
↳ Concatenate all the attention heads.

(II) Multiply with a weight matrix W^0 that was trained jointly with the model.

W_0

(III) The result would be the Z-matrix that captures information from all the attention heads. We can send this forward to the FFNN.

Z



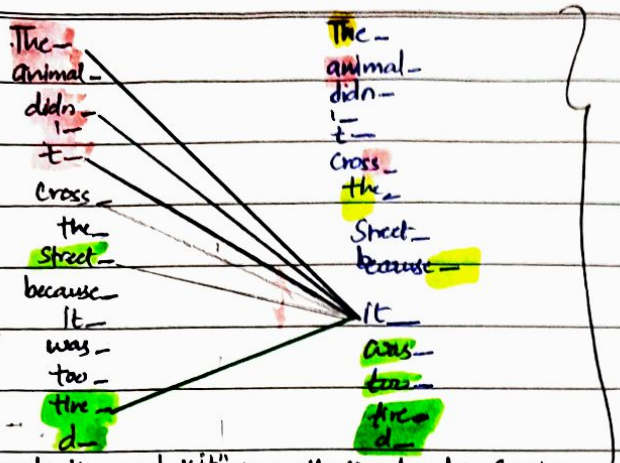
1. Input Sequence
2. We embed each word
3. Split into 8 heads. We multiply X or R with weight matrices (attention head 1 to 7)
4. Calculate attention using the resulting Q/K/V matrices
5. Concatenate the resulting Z matrices, then multiply with weight matrix W^0 to produce the output of the layer.

Deepakydv

Teacher's Signature _____

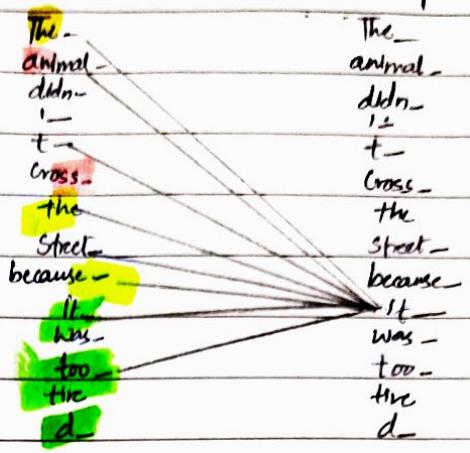
Date: _____

Topic: _____



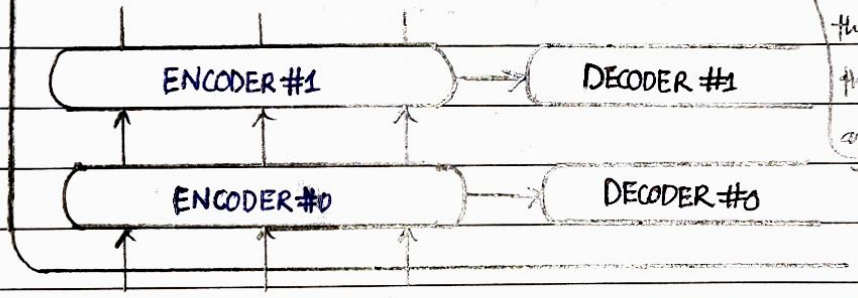
As we encode the word "it" one attention head is focusing most on "the animal", while another is focusing on "tired"-- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

If we add all the attention heads to picture,



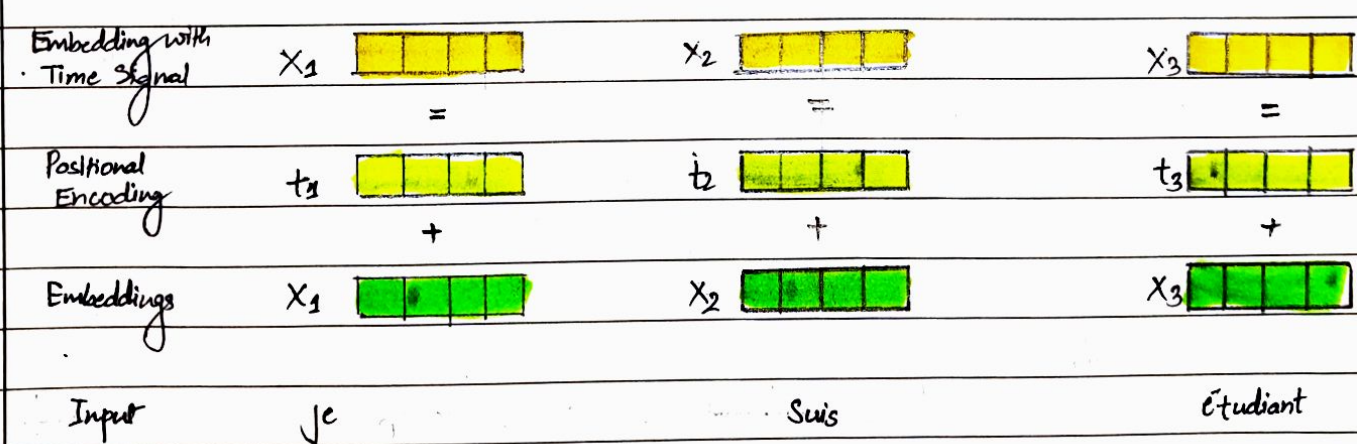
* check blog for more clear picture.

Representing the order of Sequence using Potential Encoding :



Positional encoding deals with calculating distances between the words, lesser the distance the much closer the words are and vice-versa.

→ Ordering of word is also important.



To give the model a sense of the order of the words, we add positional encoding vectors. the values of which follow a specific pattern.

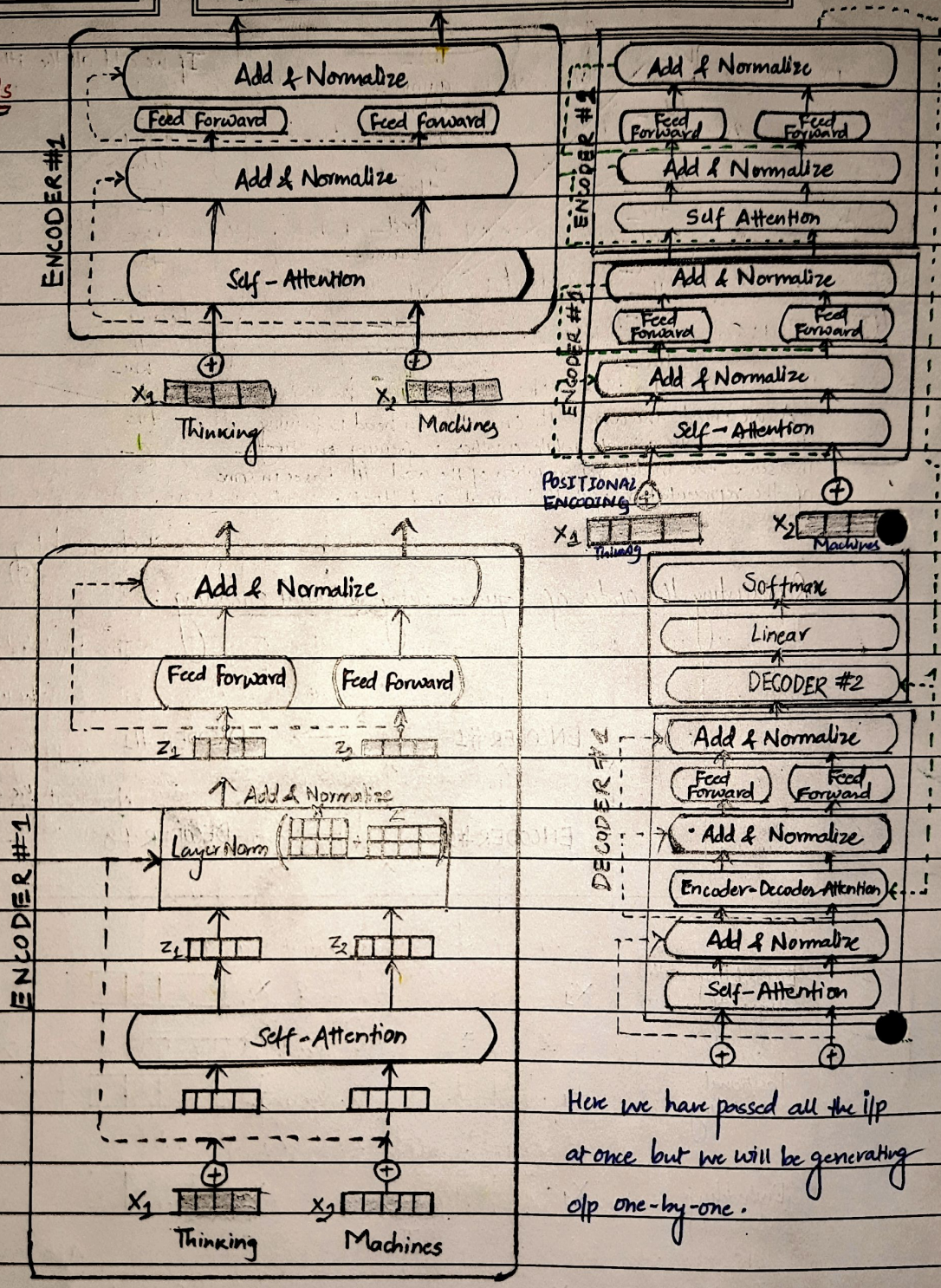
If we assumed the embedding has a dimensionality of 4, the actual positional encoding would look like this.

Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

The Residuals



Here we have passed all the i/p at once but we will be generating o/p one-by-one.

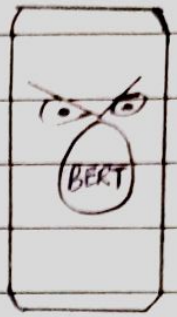
Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

BERT

BERT (Bidirectional Encoder Representations from Transformers) is a recent paper published by researchers at Google AI Language.



BERT's key technical innovation is applying the bi-directional training of transformers, a popular attention model, to language modelling.

The paper's results shows that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper researchers detail a novel technique named Masked LM (MLM) which allows bi-directional training in models in which it was previously impossible.

BERT makes use of transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its Vanilla form, Transformer includes two separate mechanisms - an encoder that reads text input and a decoder that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary.

Application (Uses of Bert)

BERT can help with the following tasks:-

(I) Text encoding / Similarity retrieval

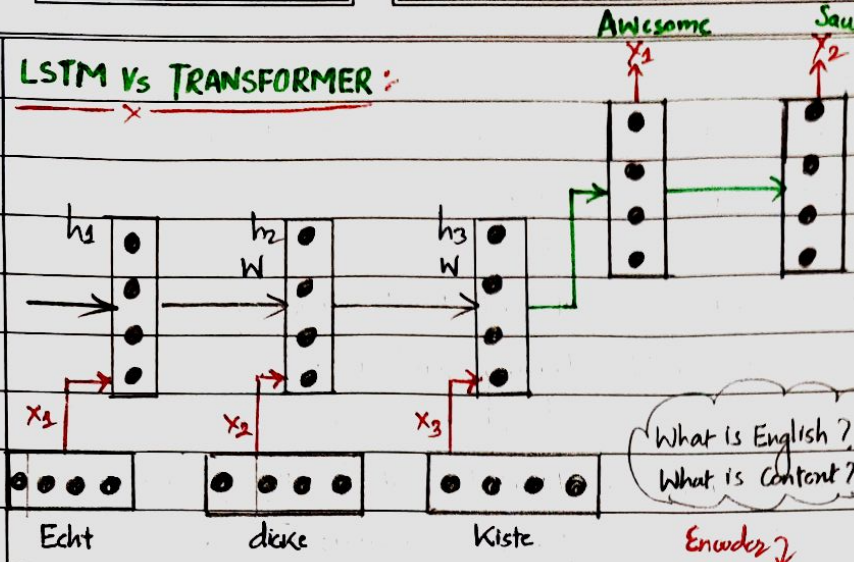
(II) Text Summarization / Response Selection.

(III) Question Answering

Deepakydv

Date: _____ Topic: _____

LSTM Vs TRANSFORMER :



LSTM Network

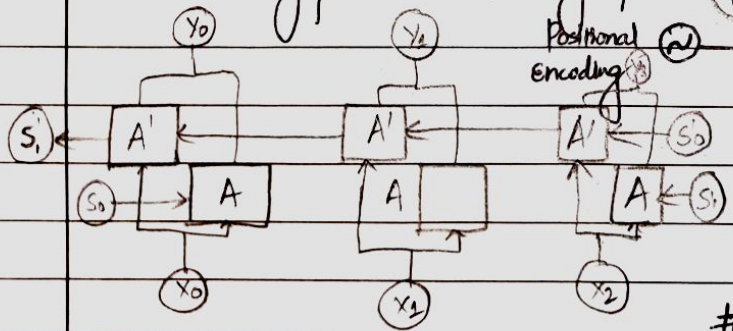
Before the transformer, LSTM was used for the Machine translation process.

LSTM Networks has below problems:-

→ Slow to train

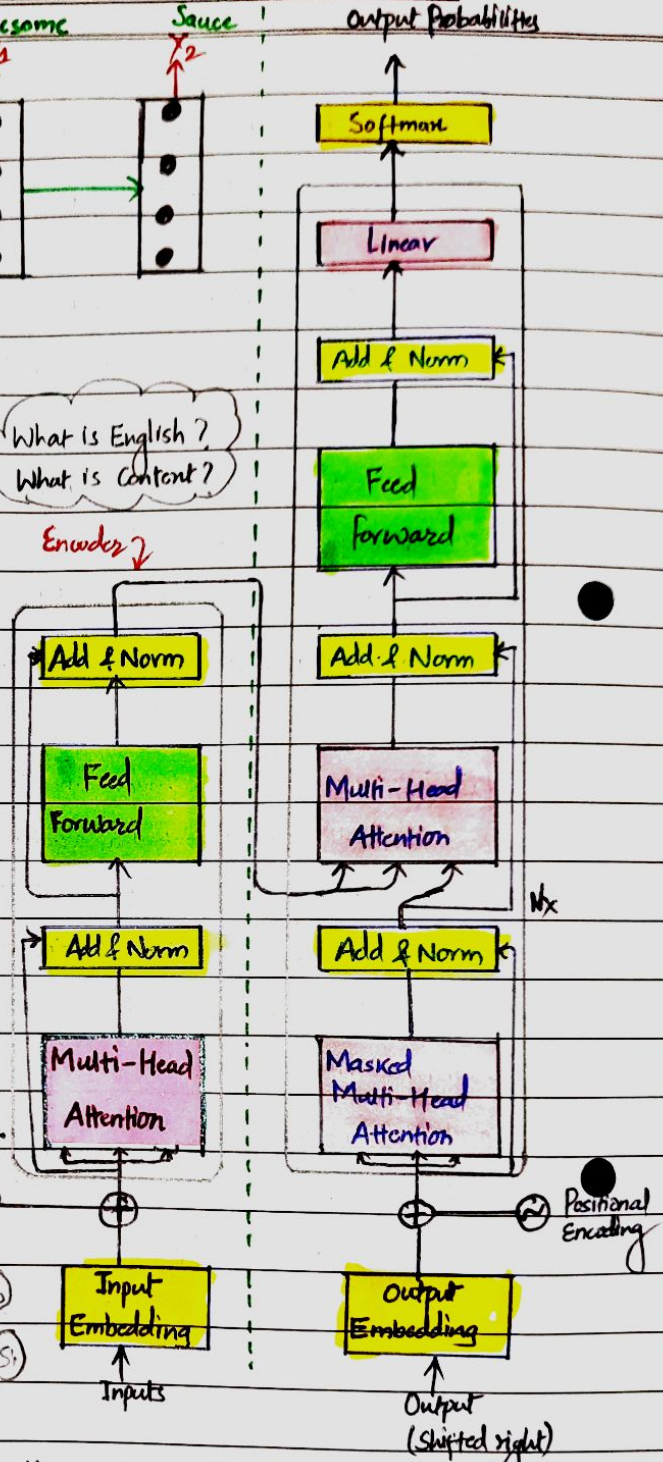
→ "Words are passed sequentially and are generated sequentially, it can take significant no. of timesteps for neural network to learn (time-consuming)

→ True meaning of words are not entirely captured.



Even bidirectional LSTM "technically learning left to right and right to left." content seprately and concatenating them so true content is slightly lost.

But transformers architecture addresses some of these concerns.



Transformer Neural Network Architecture initially created to solve problem of Neural Machine Translation

→ first they are faster as words can be processed simultaneously.

→ Second content of word is better learnt

→ As they learns from both direction simultaneously.

Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

Lets see transformer in action.

Lets take an example of language translation "english to french"

The transformer consists of 2 key components "Encoder" & "Decoder".

"Encoder" takes english words simultaneously and it generates "embeddings for every words", simultaneously

These "embeddings" are vectors that encapsulate the meaning of the word.

"Similar words" have closer numbers in their vectors.

"The Decoder" takes these embeddings from encoder and the previously generated words of the translated french sentence and then uses them to generate next french word, and we keep generating the french translation one word at a time until the end of sentence is reached.

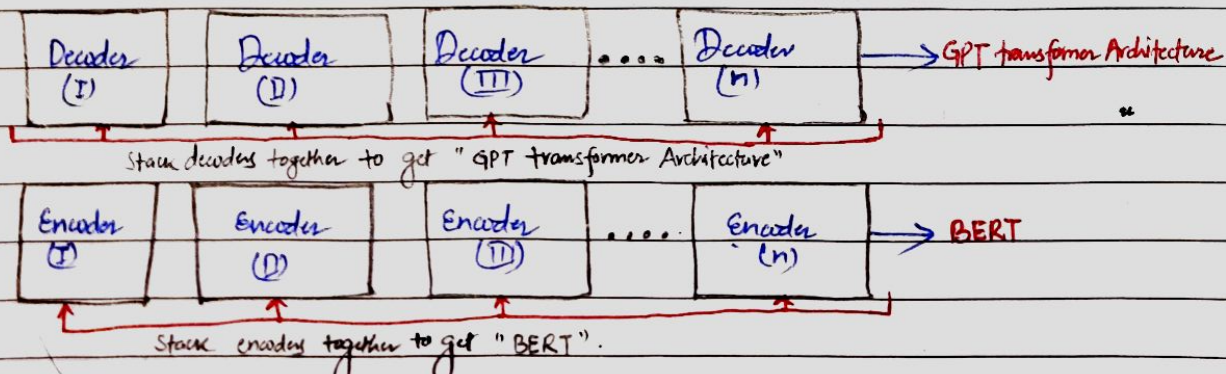
What makes this conceptually so much more appealing than some LSTM cell is that we can physically see a seprations in tasks. The encoder learns what is "English" what is "grammar" and more importantly "what is "content".?

The decoder learns how do "english word relates to french words".

"Both of these encoder and decoder have some underlying understanding of language.

We stack the decoders and we get the GPT transformer architecture.

Conversly if we stack just the encoders we get "BERT" (Bidirectional Encoder Representation from Transformers)



The OG transformer solves problem of "Neural Machine Translation".

We can use BERT for followings:-

(I) Language Translation
(II) Question Answering
(III) Sentiment Analysis
(IV) Text Summarization and many more...

Needs language understanding

BERT Training

We can train BERT to understand language and fine tune BERT to learn specific task.

> Pre-training
> fine-tuning

Teacher's Signature _____

Deepakydv

Date: _____ Topic: _____

Pre-training:

The goal of pre-training is to make BERT learn

Pre-training (Pass 1): "What is language?, What is content?"

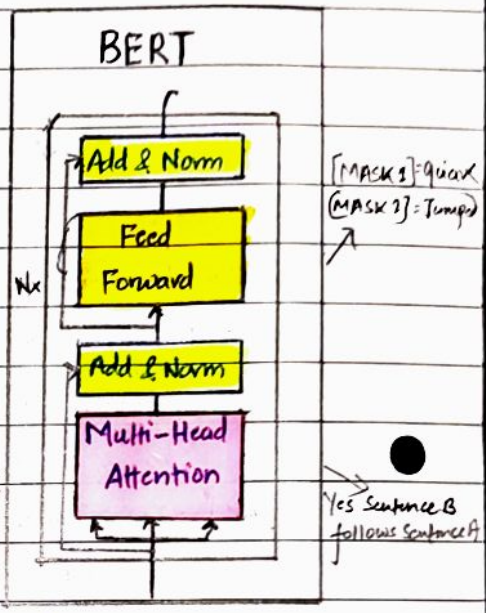
BERT learns language by training on two "Unsupervised tasks"

Simultaneously they are:

Masked Language Model (MLM)

Next Sentence Prediction (NSP)

THE [MASK1] brown [MASK2] over the lazy dog



For MLM BERT takes in a sentence with random words filled with MASKS. The goal is to output these masks tokens.

It helps BERT understand a bi-directional content within a sentence.

A: Ajay is cool dude
B: He lives in Okla

In the case of Next Sentence Prediction (NSP), BERT takes in 2 sentences and it determines if the second sentence actually follows the first.

In kind of like binary classification problem.

This helps BERT understand content across different sentences themselves, and using both of these together BERT gets a good understanding of language.

~~Pre-training~~ Pre-training (Pass-2)

~~Fine-tuning~~ (Pass 2): "How to use language for specific task?" (FTD)

The goal of pre-training is to make BERT learn what is language and what is content. BERT learns language by training on 2 unsupervised tasks simultaneously. They are:

Masked Language Model (MLM)

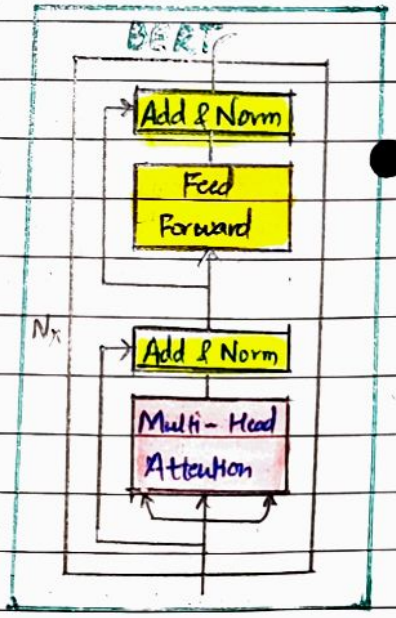
Next Sentence Prediction (NSP)

For MLM BERT takes in a sentence with random words filled with masks the goal is to output these Masks tokens.

The i/p is the set of 2 sentences with some of words being masked each token is a word. We convert each of these words into embedding using "pre-trained embeddings".

C → binary o/p for next sentence prediction

T_N → Word vectors



for fig refer to "BERT overview" Pdf research Paper by Anna Rogers. . . .

Deepakydv

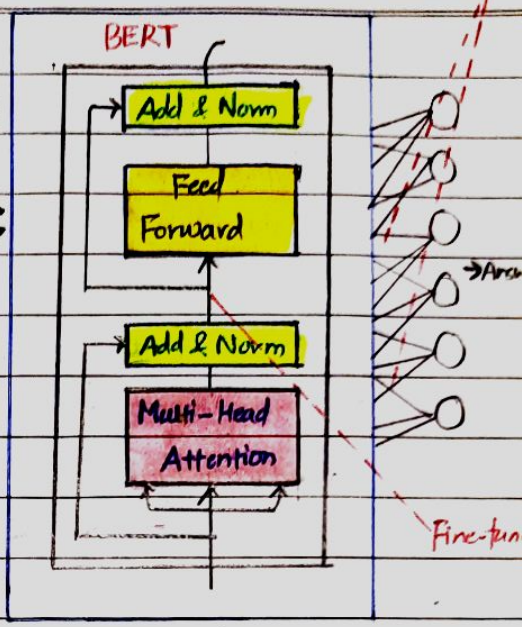
Teacher's Signature _____

Date: _____ Topic: _____

Fine-Tuning

Fine tuning (Pass 1): "How to use language for specific task?"

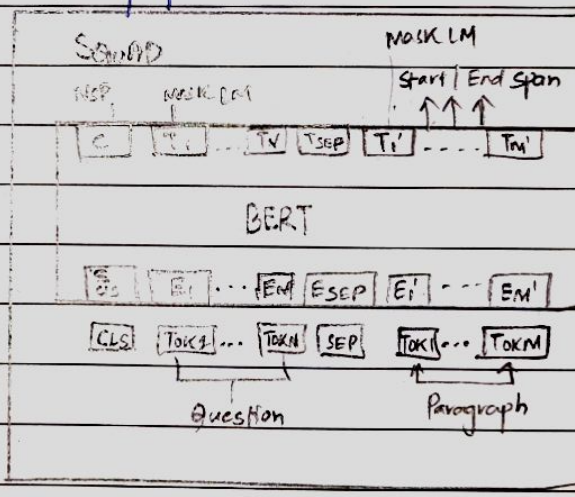
We can further train BERT on very specific NLP tasks for eg. "Question Answering".



All we need to do is to replace fully connected output layers of the network with fresh set of output layers that can basically output the answer to the question we want.

Then we can perform Supervised using a question answering dataset. It won't take long since it's only the output parameters that are learned from scratch. The rest of the model parameters are just slightly fine-tuned and as a result training time is fast. We can do this for any NLP problem. That is replace the o/p layers and train with the specific dataset.

Fine tuning (Pass 2)



If we wanted to perform Question answering we would train the model by modifying the inputs and the output layer.

We pass in the question followed by a passage containing the answer as inputs and in the output layer we would output these start and end words that encapsulate the answer assuming that the answer is within the same span of text.

Deepakydv

Teacher's Signature _____

Date: _____ Topic: _____

Pre-training

(Pass-3)

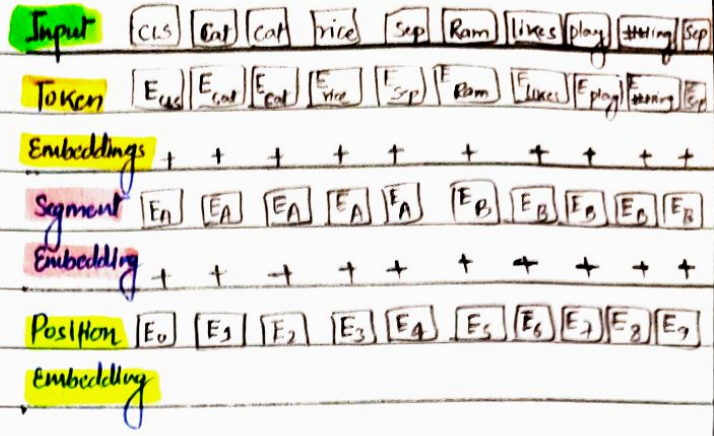
On the i/p side how we going to generate these embeddings from the word token inputs. Well the initial embedding is constructed from three vectors

Token embeddings are pre-trained embeddings

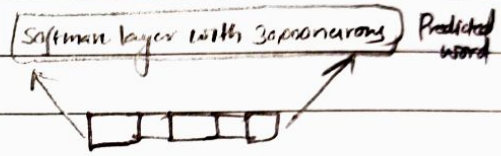
Segment embedding is basically the sentence no. that is encoded into a vector and the

positional embeddings is the position of the word within that sentence that is encoded into a vector.

Adding these 3 vectors together we get an embedding vector that we uses as input to BERT



Segment + Position embeddings add "ordering" for inputs



C → o/p of binary and bunch of word vectors, but with training we need to minimize loss.

Two key things to note:

(I) Word vectors T_i have the same size.

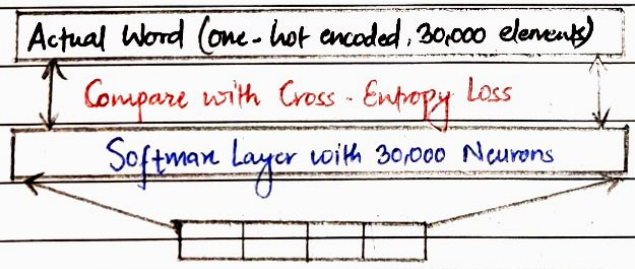
(II) Word vectors T_i are generated simultaneously.

We need to take each word vector and pass it into fully connected layer o/p with same no of neurons equal to the no. of tokens in the vocabulary.

So that would be o/p layer corresponding to 30K neurons in this case. We will apply Softmax activation. This way we would convert word vector to a distribution. The actual label for this distribution would be a one-hot encodding vector for the actual word.

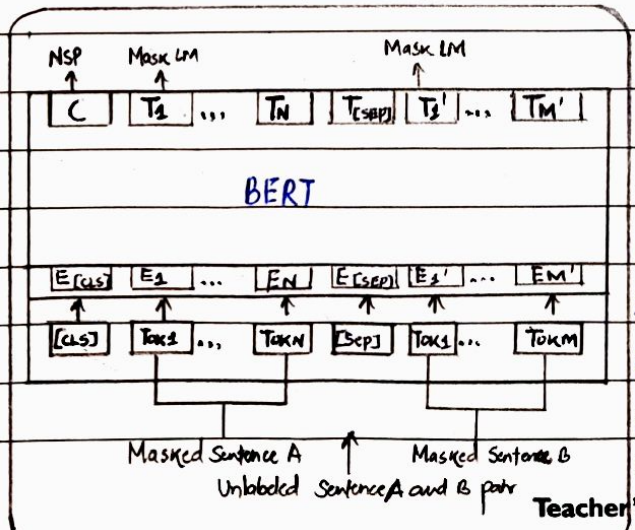
So we compare this two distribution and then train the network using "Cross-entropy-loss".

But note that the o/p has all the words. Eventhough those outputs weren't masked at all. The loss through only considers the prediction of the masked words and it ignores all the other words that are o/p by the network. This done to make sure more focus is given to predicting those mask value so that it gets them correct and it increases context awareness.



For [MASK]ed Word only

Predicted Word.



Teacher's Signature _____

Deepakydv

Date: _____ Topic: _____

Key-Takeaways:

#Pre-training (Summary)

→ We pre-train BERT with Mask language modeling and Next Sentence Prediction

→ For every word we get the 'Token embedding' from the pre-trained word piece embeddings, add the position and segment embeddings to account for the ordering of the inputs.

→ These inputs are then passed into BERT which under hood is a stack of transformer encoders and it outputs a bunch of word vectors for mass language modeling and a binary value for next sentence prediction.

→ The word vectors are then converted into distribution to train using "Cross-entropy-loss".

→ One training is complete 'BERT has some notion of language.

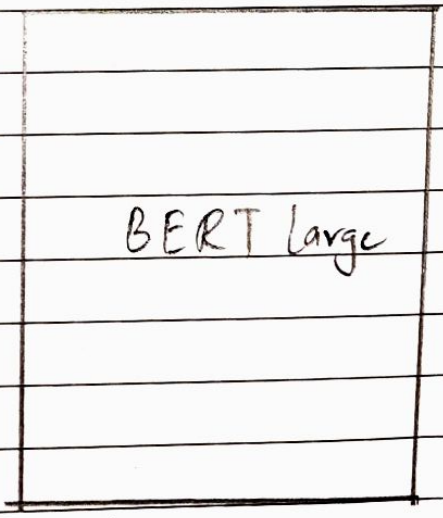
#fine-tuning (Summary)

→ In fine-tuning we perform supervised training depending on the task we want to solve and this should happen fast in fact the BERT SQUAD ie Standford Question and Answer model, only takes about 30 mins to find tune from a language model for a 91% performance.

#Performance

Performance depends on how big we want BERT to be. For BERT large model which 340 Mn parameters can achieve way higher accuracy than BERT base model which has only 110 parameters.

110 M Parameters



340 M parameters

Deepakydv