NATURAL LANGUAGE PROCESSING

→ TEXT DATA

NLP can be used in both machine learning and deep learning

Why NLP ?

Till now we have just seen numerical data which can be easily solve by ML and DL algorithm. but when it comes to textual data it become very hard to solve thats why NLP was introduced

Human → Communication → Machine

→ Here we need NLP

Roadmap:

Text preprocessing → ML Model → RNN, LSTM, GRU → Word embedding

↓

BERT ← Transformer ← Bidirectional LSTM, Encoder, Decoder

Tokenization

Converting sentences into words

Eg: Hello, How are you ? → Sentence

Hello How are you → Word

Stopwords

The words which help to form sentences

Eg: tea with toast

→ Stopword.

Stemming

Process of reducing words to their base word stem.

Eg:

Creation
Creativity
Creating
→ Creat → Stem word

going
goes
gone
→ go → Stem word

Stemming usecase → Spam classification
→ Review classification

Cons of stemming

→ Removing meaning of the word

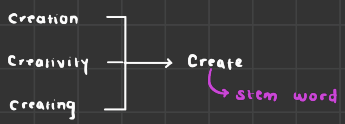
Pros of stemming

→ Faster

lemmatization

process of reducing the words to their base form with meaning

Eg:



Cons of lemmatization

- slower than stemming

pros of lemmatization

- meaningful word

lemmatization usecase → Chatbot, Text summarization.

Basic Terminology used in NLP

1. CORPUS → paragraph
2. Documents → sentence
3. Vocabulary → unique words
4. Words → all words

Text Preprocessing

Steps:

① Tokenization → ② stopwords → ③ stemming → ④ lemmatization

Convert word into vectors

1. One hot encoding

[A man eat food
Cat eat food → CORPUS
People watch movie] vocabulary - 8

Document 1 →

	A	man	eat	food
[[	1	0	0	0
	0	1	0	0
	0	0	1	0
	0	0	0	1]]

- Sparse matrix

Advantage

- Simple to implement

Disadvantage

- Sparse matrix
  - majority of zero present
- Out of vocabulary → extra word not handle
- Size is not fixed
- Semantic meaning between word is not captured.

Document 2 →

Cat	eat	food
1	0	0
0	1	0
0	0	1

→ Vocabulary size is decreasing ↓
↓
cannot train model
↓
Input size not same

2 Bag of words (BoW)

Removed ← Stopwords

Document 1 → He is a good boy → good boy

Document 2 → She is a good girl → good girl

Document 3 → Boys and girls are good → Boy girl good

Vocabulary	Frequency
Good	3
Boy	2
Girl	2

	feature1	feature 2	feature 3
	Good	Boy	Girl
document 1	1	1	0
document 2	1	0	1
document 3	1	1	1

→ Binary Bow

Advantage

Simple and intuitive

Disadvantage

- Sparsity will be these
- Out of vocabulary
- Ordering of word get change
- Semantic meaning between word not captured
  - → Cosine Similarity

In order to capture the semantic information → we use ngrams

NGRAM → Bigram, Trigram, — Ngram

Bigram → Combination of two feature

	feature1	feature 2	feature 3	feature 4	feature 5
	Good	Boy	Girl	good boy	good girl
document 1	1	1	0	1	0
document 2	1	0	1	0	1
document 3	1	1	1	0	0

will check combo of two words are present or not in document 1, 2, 3 if present than we will fill value as 1 Otherwise 0.

Example :

Anirudh eats food ⟶ Bigram ?

Anirudh eats , eats foods } ⟶ 2 possible bigram

I am not feeling well ⟶ Trigram ?

I am not , am not feeling , not feeling well } ⟶ 3 Possible Trigram

ngram(1,3)

Anirudh is not feeling well Unigram ⟶ Bigram ⟶ Trigram

f1	f2	f3	f4	f5	Bigram	Trigram
Anirudh	is	not	feeling	well	— —	— —

TF-IDF

Bow is not able captwe exact meaning or semantic meaning so to solve this. we use Concept of TF IDF

Term frequency – Inverse document frequency (TF-IDF)

Example

Sent1 - Good boy

Sent2 - Good girl

Sent3 boy girl good

Good present in all

boy and girl present in only few sentences

we will give some weights to boy and girl in that Sentances only where the word is present

$$\text{Term frequency (TF)} = \frac{\text{no of repetation of words in Sentences}}{\text{no of words in sentence}}$$

$$\text{Inverse document frequency (idf)} = \log_e\left(\frac{\text{no of Sentences}}{\text{no of sentences containing the word}}\right)$$

TF * Idf = Tfidf

Sent1 - Good boy

Sent2 - Good girl

Sent3 - boy girl good

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

*

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	$\frac{1}{2} * 0$	$\frac{1}{2} * \log_e(3/2)$	$0 * \log_e(\frac{3}{2})$
Sent 2	$\frac{1}{2} * 0$	$0 * \log_e(3/2)$	$\frac{1}{2} * \log_e(3/2)$
Sent 3	$\frac{1}{3} * 0$	$\frac{1}{3} * \log_e(3/2)$	$\frac{1}{3} * \log_e(3/2)$

Over here some amount of semantic meaning is captured.

Common in all sentences

Advantage

- Intuitive
- word importance is getting captured

Disadvantage

- Sparsity
- out of vocabulary

Word Embedding

technique which converts the words into vectors

- Word embedding
  - Count or frequency
    1. BOW
    2. TF-IDF
    3. One hot encoding
  - deep learning trained model
    1. Word2vec
       - CBOW
       - SKIP GRAM

Word2vec

feature representation

	f1	f2	f3	f4	f5	f6
	Boy	Girl	King	Queen	Apple	Mango
Gender	-1	1	-0.92	+0.93	0	01
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						
300 dimension						

Related word (semantic mean)

Vectors

King [0.96 .95] Man [.95 .98]

Queen [.96 .95] Woman [.94 .96]

King - man + woman = Queen

How?

Cosine similarity ←

→ Euclidian distance

distance = 1 - cosine similarity

cosine similarity = Cos θ

Cos 45° = 0.7071

= 1 - 0.7071

= 0.29 → not that much similar

Advantages:

- In word2vec we create vector for each word with limited dimension.
- Sparsity is reduced.
- Semantic meaning is captured

CBOW - Continous Bag of words

Corpus : Training Dataset → center word → output word

[Machine learning is related to data science]

Window Size = 5

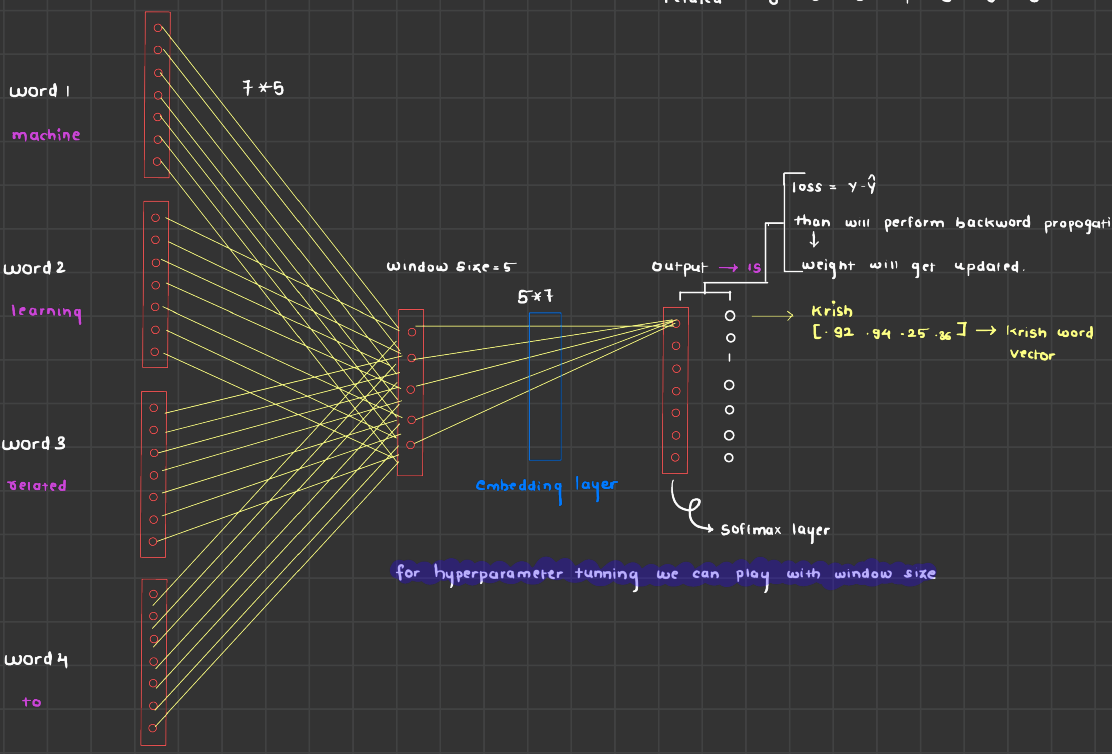
Context

Slide the window

Independent features	output
Machine, learning, related, to	is
learning, is, to, data	Related
is, related, data, science	TO

Bag of word

Machine	1	0	0	0	0	0	0
learning	0	1	0	0	0	0	0
is	0	0	1	0	0	0	0
related	0	0	0	1	0	0	0



for hyperparameter tunning we can play with window size

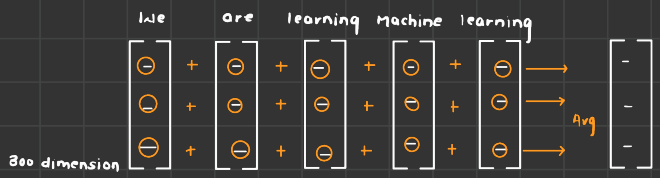
Skip Gram Reverse the CBOW network will get skipgram

Input	Output
IS	Machine, learning, related, to
Related	learning, is, to, data
To	is, related, data, science

AvgWord2vec

↳ used to represent the vector of entire sentence or document not single word.

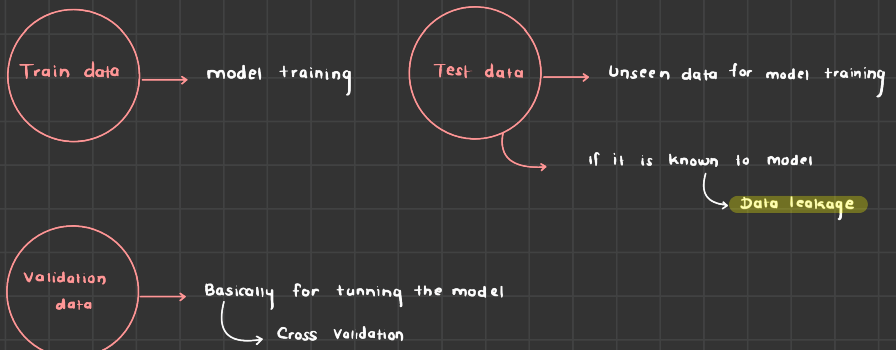
Example : We are learning Machine learning



↳ Representation of entire document

Interview Question

A. Train vs Test vs Validation



B. Why Random forest instead of decision tree?

Reduce overfitting and bias · Generalized · Higher Accuracy

Recurrent Neural Network

Use case:

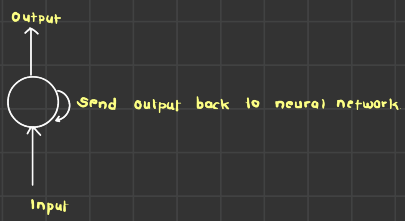
Question → Chatbot → Answer

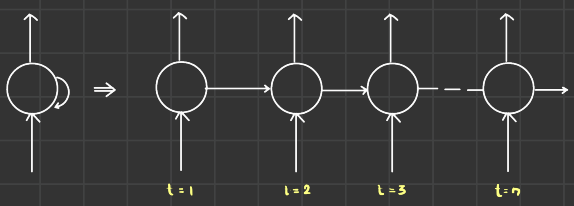
Sequence of word matter

Text Generation → A senfences → Auto complctes the sentences

Sequence of word matter

Basic Architecture of RNN





t = time

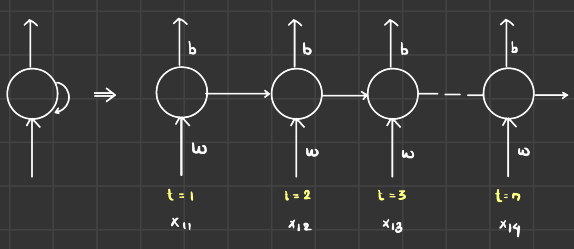
Example

Sentiment Analysis	Output
The food is good	positive

At time t_1 will pass x_{11} and weight w will initialize.

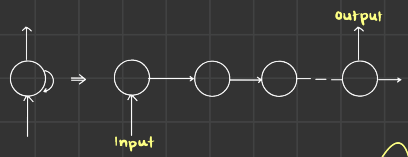
x_{11} → Word2vec → vector

→ dimension = d = 9



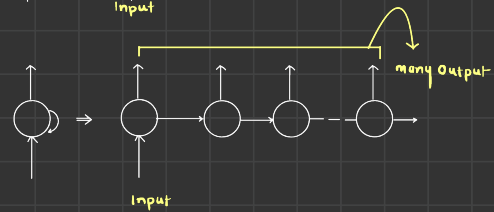
Types of RNN :

1. One to One Rnn



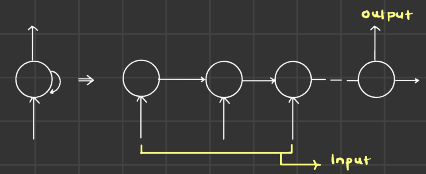
2. One to many Rnn

Eg: Text generation



3. Many to one Rnn

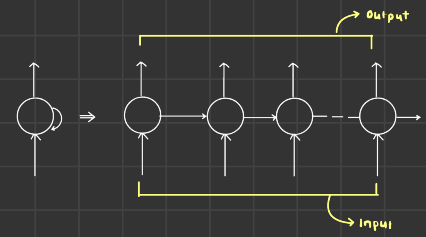
Eg: Sentiment analysis



4. Many to many Rnn

Eg: language translation

chatbot

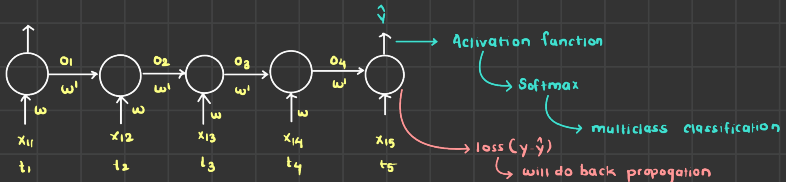


Forward Propogation in Rnn:

Eg: Sentiment Analysis

The food is very good → positive

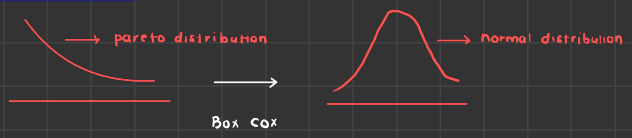
x_{11} x_{12} x_{13} x_{14} x_{15}



$O_1 = f(x_{11} * w) + b$ $\quad O_2 = f(x_{12} * w + O_1 * w) + b$

$O_3 = f(x_{13} * w + O_2 * w' + O_1 * w') + b$

Interview Question

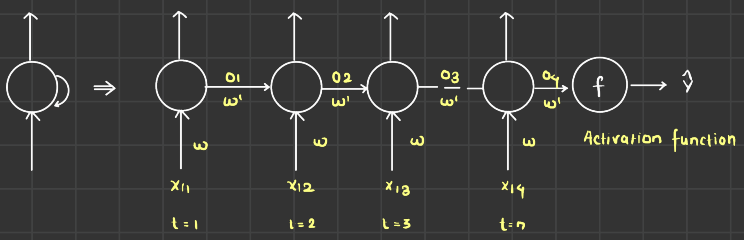


Transformation can convert pareto to normal

How to know this graph is normal or gaussian

→ QQ Plot

Back Propogation



$o_1 = f(x_1 * w_1)$

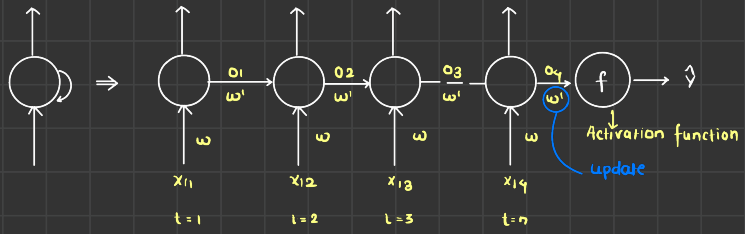
$o_4 = f[(x_{14}w_1) + (o_3 w_1)]$

$o_3 = f[(x_{13}w_1) + (o_2 w_1)]$

$o_2 = f[(x_1 w_1) + (o_1 w_1)]$

In back propogation we update the weights

first we will calculate the loss than we will update weights



weight updation formula

$$w'_{new} = w'_{old} - \eta \left[\frac{\partial L}{\partial w'}\right]$$

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

Chain Rule

$$w'_{new} = w_{old} - \eta \left[\frac{\partial L}{\partial w_{old}}\right]$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_4} \cdot \frac{\partial o_4}{\partial w}$$

Chain Rule

$$w'_{new} = w'_{old} - \eta \left[\frac{\partial L}{\partial w_{old}}\right]$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial o_4} * \frac{\partial o_4}{\partial w'}$$

Chain Rule

In back propogation we may face problem of vanishing gradient so to solve this we use LSTM RNN

LSTM stand for long short term memory

In LSTM RNN we have four main section
1. memory cell
2. forget cell
3. input cell