NLP

Next revolution will come when we are able to interact with machines the same way we interact with humans.

Natural Language Processing is a branch of artificial Intelligence (AI) that helps machines to understand and process human languages either in text or audio form.

The functionality of search engines is powered by NLP. Other modern applications include

1) Contextual Advertisements
2) Social Media: removing adult content, opinion mining
3) Search Engines.
4) Chatbots. (let's say in Swiggy or Amazon : Customer Service)

Try thinking more use yourself!!

Approaches to NLP

1) Heuristic Approach — "Jugaad" approach

Rule based approaches were created by NLP practitioners.

Eg: Let's say for sentiment analysis for a movie review, we count the number of positive words & no. of negative words.

Examples:
1) Regular Expressions
2) Word Net
3) Open Mind Common Sense.
   - ↳ Open for everyone (open source) community
   - ↳ Anyone can contribute common English facts to it. Basically a database was created to be used for NLP tasks.

These are quick approaches, error free because humans are constructing it.

2) Machine Learning Approach

Good for open ended problems. We cannot make general rules everywhere.

Here, our algorithms construct the internal patterns required by Input and Output.

ML Algos used:
1) Naive Bayes
2) SVM
3) Logistic Regression
4) LDA
5) Hidden Markov Models

They don't care about sequential data → which is fixed by RNN and (DL) others.

Bottleneck of Heuristic Approach

3) Deep learning Approach → Automatic feature and parameter generation.

1) RNN
2) LSTM
3) GRU/CNN
4) Transformers
5) Auto encoders.

Challenges to NLP

1) Ambiguity in the language
2) Contextual words.
   Eg: River bank & Money Bank
3) Colloquialisms & slang. eg: Piece of Cake
4) Synonyms.
5) Irony, Sarcasm & tonal diffn
6) Spelling Errors.

NLP Pipeline

The following steps are necessary for constructing any NLP application

1) Data Acquisition: Acquiring relevant data
   Even if we use Heuristic Approach, we need extra data for better performance. (More data is always good)

2) Text Preparation: Eg.
   i) Text cleanup: fixing spelling mistakes, incomplete phrases
   ii) Basic preprocessing: Eg. Tokenization of words.
   iii) Advanced preprocessing
   Eg: Parts of Speech Tagging, Chunking.

3) Feature Engineering: We need data in correct format
   We need to change our text data into numerical data. Eg: Word2Vec, TF-IDF, One-hot encoding.

4) Modelling: Actual Algorithm is applied

5) Deployment: Monitoring & Model Update

The above pipeline is not universal. For elementary NLP tasks like POS, Text Representation, this works fairly well.

Data Acquisition
- Available with you
  - If its less in quantity, we have to go for "DATA AUGMENTATION" Let's say
    1) Replacing words with its synonyms.
    2) Back translate: Convert words from Eng → Hindi and then Hindi → Eng. The sentence structure changes in this process.
    3) Add noise: Adding words to the sentence without changing the sentiment of the sentence
    4) Bigram flip:
       "I am walking on the road"
       "I am walking the on road"
- Data Available but not with you (Some other company)
  - public dataset → like github/kaggle/some major university
  - Webscrapping → using beautiful soup (Python library)
  - API → Use Rapid API for finding APIs related to your domain
  - pdf/Image → Some python library
  - Audio → Audio to Text then start preprocessing.
- No one has the data (You are the first one to work on this) then Start Collecting data and manually label them

Text Preparation

- Cleaning
  - ↳ html tag removal
  - ↳ emoji from social media are removed
  - ↳ Spelling checker (↳ lib called Textblob is used)

  None of them are mandatory (task dependent)
- Basic preprocessing:
  - Basic
    - → Tokenization (word & Sentence Tokenization)
  - Optimal
    - → Stop word Removal. (., ;, ,) don't have semantic meaning
    - → Stemming (brings words to their root forms) (dance, dancing, danced) (All 3 of them have same semantic meaning)
    - → Removing digits, punctuations
    - → Lowercasing
    - → language detection.
- Advanced Preprocessing
  - → Pos tagging
  - → Parsing
  - → Coreference resolution

If we do stop word removal, we can't do parts of speech tagging (POS). So, we have to be very careful & modify our pipeline based on our end goal.

Feature Engineering

- When we are using Machine learning approach
  - * We have to do the feature engg. ourselves.
  - (Model's Interpretability is high)
- Deep l approach
  - * feature engg. is done by the neural networks.
  - (Model's Interpretability is low)

Modelling

- modelling
  - data is ready, now we have to fit a model to find patterns.
  - possible ways:
    - heuristic : if data is less, we try to fit a general rule to define trends. → let's say we are making a spam email classifier. For heuristic approach, we take a note of the email id from which we are continously receiving spam emails
    - ML Algo : Data (more data)
    - DL Algo : Data (even more data) → use Transfer Learning. (BERT) We have a model trained on a very large dataset and it has large capabilities.
    - Cloud API :

  Decided by nature of data & Nature of Problem
- evaluation
  - intrinsic (model centric)
  - extrinsic (business why product is not being used by)

google (let's say auto complete)

Text Preprocessing

Some of the common text preprocessing/cleaning steps:

* lower casing
* Removal of Punctuations
* Removal of Stop words
* Removal of Frequent Words
* Removal of Rare Words
* Stemming
* Lemmatization
* Removal of emojis / emoticons
* Conversion of emoticons to words
* Removal of URLs
* Removal of HTML tags
* Chat words conversion
* Spelling correction

Check Out
"Text-Preprocessing_AlWinter.ipynb"

Feature Engineering / Text Representation

We convert texts into numbers (numerical data)
↓
those are represented by vectors
So, this step is also called "text vectorization"

The numbers should capture the semantic meaning of the text
(hidden meanings should be represented)

Terms:

1) Corpus (C): when we concatenate all of our strings
let's say we have movie reviews.
When we combine all movie reviews together (many words might be repeating)
2) Vocabulary (V): Unique words from corpus.
3) Document (D): each movie review
4) Word (W): Each word of document

Eg: Review 1: "I loved SRK in Jawaan" (loved = word)
Review-2: "All actors had great chemistry"
Review 3: "Boring & predictable story line" (predictable = another word)

	Sentiment
Review 1	1
Review-2	1
Review 3	0

I haven't removed stop words here
Do remove it if required

Corpus (C): I loved SRK in Jawaan All Actors had great chemistry Boring & predictable story line → 15 words.

Vocabulary (V): 15 words (unique words)
Documents (D): Rev 1, Rev 2, Rev 3

(1) One hot encoding (OHE)

	Sentiment
S1: IITK is my college	1
S2: My college is IITK	1
S3: College is IITK	0

So, Corpus: IITK is my college My College is IITK College is IITK

"Stop words not removed. You might choose to remove"

Vocabulary:
IITK, is, my, college

We represent words in terms of vectors of length V.

V = 4

	IITK	is	my	college
IITK =	[1,	0,	0,	0]
is =	[0,	1,	0,	0]
my =	[0,	0,	1,	0]
college =	[0,	0,	0,	1]

So, (S1) is converted to: [[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]] 4 x 4

S3 : [[0,0,0,1], [0,1,0,0], [1,0,0,0]] 3 x 4

Very logical, intuitive, easy to code.

Flaws: 1) Sparsity

Let's say our vocabulary is 50000 (corpus of some million words)

So, IITK would be [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -- -- 0, 0, 0, 0]
(Sparse array)

Leads to overfitting (lots of zeroes in it)

2) Sentence is represented by Words X Vocabulary
All sentences don't have same size. Input size has to be kept same for any ML Model.

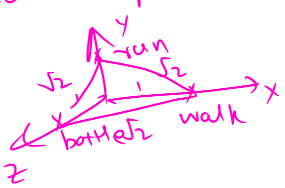
3) Out of Vocabulary (OOV) Problem:
Let's say we have to predict the sentiment of the sentence:
"IITB is better"
Out of Vocabulary Words. So, we want predict the Sentiment.

Let's say:

walk, run, bottle = Vocabulary

walk: [1, 0, 0]
run: [0, 1, 0]
bottle: [0, 0, 1]

Vector Representation



So, distance b/w run-walk = dist. b/w bottle walk.

But, run & walk have a closer meaning compared to the other pair.

So, the Semantic meaning is not being captured.

Which kills our purpose of vectorization.

2 Bag of Words (BoW)

Core Idea:

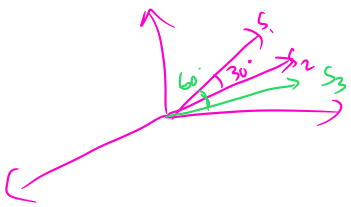
Sentences with same semantic meaning will have same type of words that appear in similar frequencies in the sentence.

Order of the words does not matter

content does not matter (but at the end its captured indirectly) while writing the vectors

All text is converted to V- dimensional vector.

Let's say V=3



Sentence is given a vector in the V-dimensional

If the dot product is high, they are similar to each other.

let's ∠le b/n S_1, S_2 : 30°

∠le b/n S_1, S_3 = 60°

$S_1 \cdot S_2 = \cos 30° = \frac{\sqrt{3}}{2}$

$S_1 \cdot S_3 = \cos 60° = \frac{1}{2}$

S_1, S_2: more similar (Because they are close to each other)

∠le is lesser.

[I haven't removed stop words in the example remove it if required]

Scikit learn has {count_vect}

functionality → binary (generally [binary = false; if true then freq > 1 will be make 1]

Sentence [1,0,1,2,3] → [1,0,1,1,1] vector

max_features

S_1: Bauder is a boy

S_2: Bauder is Bauder

S_3: Boys hate Bauder

	freq. of 'boy' in S1	Bauder	is	a	hate
S_1	1	1	1	1	0
S_2	0	2	1	0	0
S_3	1	1	0	0	1

Vocabulary = 5

So, 5-(V) dimensional vectors for each sentence

max_features: when many rare words are present. It decides the number of words considered for analysis

max_features = 1 So, top freq. words are observed

max_fea = 2 So, top 2 freq words are analyzed.

Bottlenecks:

1) Sparsity: The dim. of vectors will be very large (let's say 50,000) So, many zeroes will be present, end up overfitting

2) OOV: Improvement from OHE

This technique won't report an error.

3) Ordering: We are dealing with frequencies

4) S_1: Bauder is a very good singer and a nice boy

S_2: Bauder is not a very good singer and a nice girl.

Vectors might be close but meaning is semantic very different.

3 Bag of n-grams

Single word were creating issues
'not good' and 'good' have different meanings

$n=2$ (Bigram)

We take pairs.

[I haven't removed stop words here, remove if required if required]

Example

S_1: Barudev is your mentor

S_2: Mausi is your mentor

S_3: Khushal is your mentor and mausi is your mentor.

Tokens Vocab:

Barudev is | is your | your mentor | Mausi is | khushal is | mentor and | and Mausi

S_1:	1	1	1	0	0	0	0
S_2:	0	1	1	1	0	0	0
S_3:	0	1	1	0	1	1	1

We used bigram that's why we can create greater angles bln 'not good' & 'good'
If we use trigrams, difference would be more clear.
So, Semantic meaning is captured better of the sentence

Benefits

1) Easy to implement
2) Able to capture

Disadvantages

1) We still ignore OOV
2) uni-gram, bi-gram, tri-gram → dimensionality ↑ as token increase
So, this increases the model training time.

4 Tf-Idf

Term frequency — Inverse-document frequency

Core Intuition:
If a word appears less (rare) in the corpus, then its weight increases in the sentence its present in.

S1: Good old Barudev boy
S2: Barudev nice boy
S3: Old Barudev good boy

nice is rare in the corpus. → rare word [Here, Stop words have been removed]

For S1:
TF(Barudev) = 1/6, IDF(Barudev) = $\ln\left(\frac{3}{3}\right) = 0$
TF(nice) = 0, IDF(nice) = $\ln\left(\frac{3}{1}\right) = \ln 3$

(Rare words will have higher IDF)

$$TF(t,d) = \frac{\text{No. of occurrences of term } t \text{ in doc } d}{\text{Total no. of terms in doc } d}$$

$0 < TF < 1$ (Probability) [If TF↑ : So, term t is more frequent]

$$IDF = \ln\left(\frac{\text{Total no. of doc. in Corpus}}{\text{No. of docs with term } t \text{ in them}}\right)$$

Why ln is used?

$Idf = \left(\frac{N}{n}\right)$??

If $N = 10000$

$n = 2$

then $\frac{N}{n} = 5000$ value is very large

So, when we are considering IDF*TF, larger IDF values will dominate over TF values

We want both of them to contribute equally.

So, $IDF = \log_e\left(\frac{N}{n}\right) = \ln(5000)$

↳ Here, $\log_e$ normalizes the value

Sometimes, $IDF = \ln\left(\frac{N}{n}\right) + 1$ [In sklearn library]

If any word is appearing in every sentence (doc); its

$IDF = \ln(1) + 1 = 1$ instead of (0).

TF-IDF Calc for S1

Each term of word in vector is represented by TF * IDF

Vocab for corpus: [Good, old, Basudev, boy, nice]

$TF(Good) = \frac{1}{4}$, $IDF(Good) = \ln\left(\frac{3}{2}\right)$

$TF(old) = 1/4$, $IDF = \ln\left(\frac{3}{2}\right)$

$TF(Basudev) = \frac{1}{4}$, $IDF = \ln\left(\frac{3}{3}\right)$

$TF(boy) = \frac{1}{4}$, $IDF = \ln\left(\frac{3}{3}\right)$

$TF(nice) = 0$, $IDF = \ln\left(\frac{3}{1}\right)$

So, S1 is represented by

$= \left[\frac{1}{4}\ln\frac{3}{2}, \frac{1}{4}\ln\frac{3}{2}, \frac{1}{4}\ln 1, \frac{1}{4}\ln 1, 0 \times \ln 3\right]$

$= \left[\frac{\ln 1.5}{4}, \frac{\ln 1.5}{4}, 0, 0, 0\right]$

$= [0.101, 0.101, 0, 0, 0]$

Advantages

1) Used for Information Retrieval

Search Engines go through millions of webpages and give suggestions to you.

Disadvantages

1) Sparsity (High dimensions, so lots of zeroes) → leads of over fitting again

2) OOV

3) Semantic Meaning is still not captured in some cases.

"Einstein was genius"

"Einstein was brilliant"

They kinda have similar meanings. But TF-IDF treats them differently.

(5) Miscellaneous Custom features

Here, domain knowledge plays an important role.

Eg $\frac{\text{No. of Positive Reviews}}{\text{No. of Negative Reviews}}$

Word Embeddings

frequency types → BoW, Tf-Idf, Glove

&

prediction types → Word2Vec

Word 2 Vec

1) Semantic Meaning is captured
2) low dimensional vectors are formed (300-500)
3) dense vectors (non-zero elements) ∴ overfitting!

We can use pretrained as well as self-trained word2vec models.
↓
training it on our own dataset

→ Representing the Vector in 300 dimensional space.

→ Vector Arithmetic works here (Addition, Subtraction, Dot Product)
↳ Dot product also works.

The features would be anonymous to us.

We can never find out what they mean exactly

	king	queen	Man	Woman	Monkey
gender f_1	1	0	1	0	1
Wealth f_2	1	1	0.3	0.3	0
Power f_3	1	0.7	0.2	0.2	0
weight f_4	0.8	0.4	0.6	0.5	0.3
speak f_5	1	1	1	1	0

king = [1, 1, 1, 0.8, 1]

Internally, Word2Vec creates features based on the vocabulary. These features might be different for different vocabulary.

king − man + woman ≈ queen

We use automated techniques to create features.

We might have a vocabulary of 50lakh words. So, manually generating features is quite tough.

So, we use Neural Networks

All 300 numbers represent unique features, but we can never find out what they represent exactly.

Core Intuition

The underlying assumption of Word2Vec is that two words sharing similar contexts also share similar meaning and consequently a similar vector representation from the model.

Types of Word2Vec:

CBOW → Skip-gram

Continuous Bag of Words (CBOW)

let's say: "Choose Architecting Intelligence for deep learning"

we have.

We want to write the word embeddings (vectors) for each word.

let's first decide the size of these vectors for each word (dimension)

(let it be 3

So, Choose = [− , − , −]

We create a dummy problem and try to solve it. Now,

"Choose Architecting Intelligence for deep learning" → Sentence

Window keeps on moving → window-2

We start by considering windows of size=3 [It can be 5, 7, 9 ... also]

1st window

central word: target word. (Architecting)

left & right words: context words (Choose, Intelligence)

We created a "fill in the blanks" problem "Choose ___ Intelligence"

It is Supervised ML

By taking context words as inputs, we are trying to predict the target word

So,	X (Input)	Y (target)
	Choose, Intelligence	Architecting

Now we take OHE for our vocabulary

Choose : [1,0,0,0,0,0]

Architecting : [0,1,0,0,0,0]

Intelligence : [0,0,1,0,0,0]

for : [0,0,0,1,0,0]

Deep : [0,0,0,0,1,0]

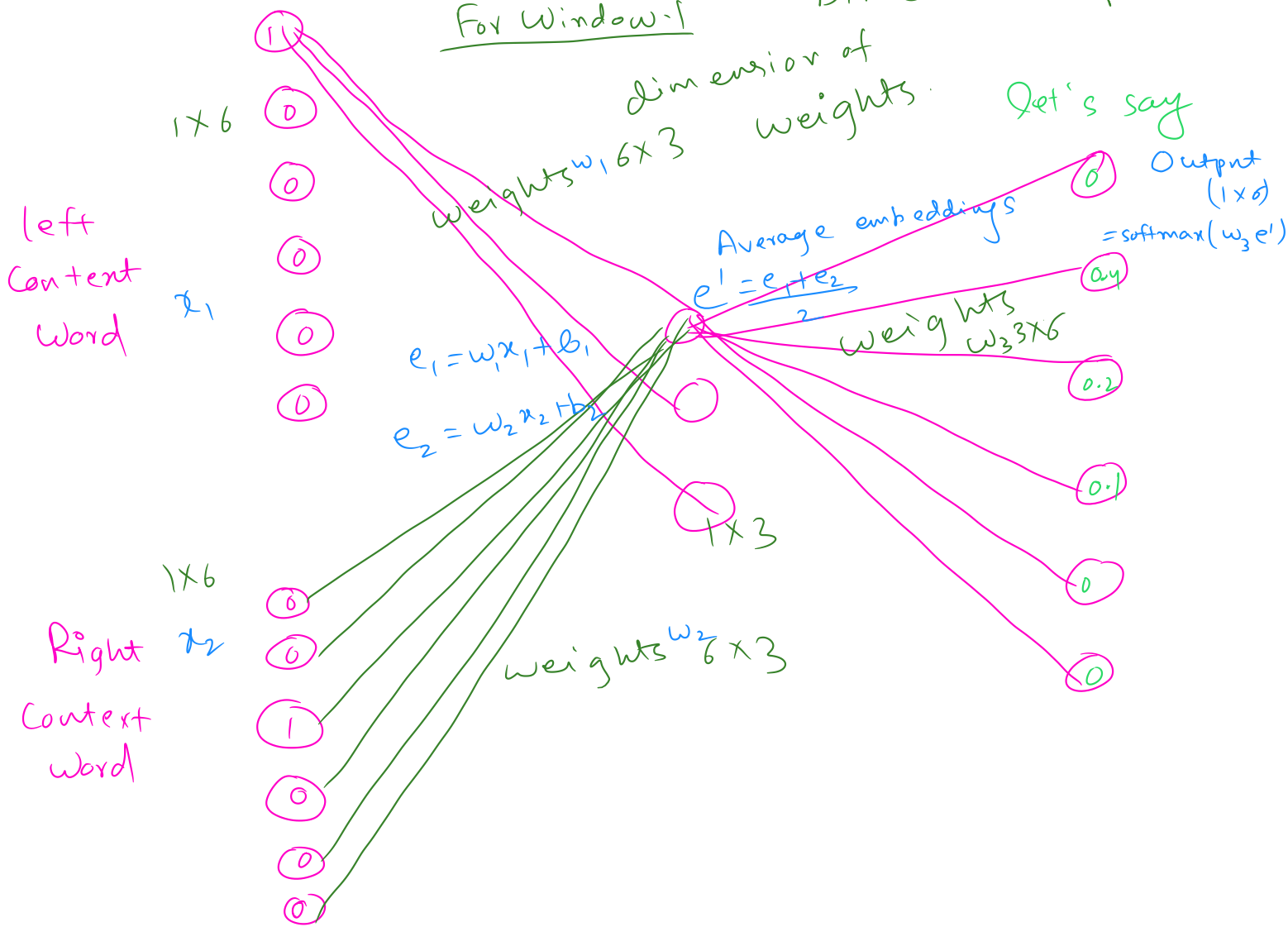
learning : [0,0,0,0,0,1]

For Window-1

Dimension of input : 1×6

dimension of weights

Let's say

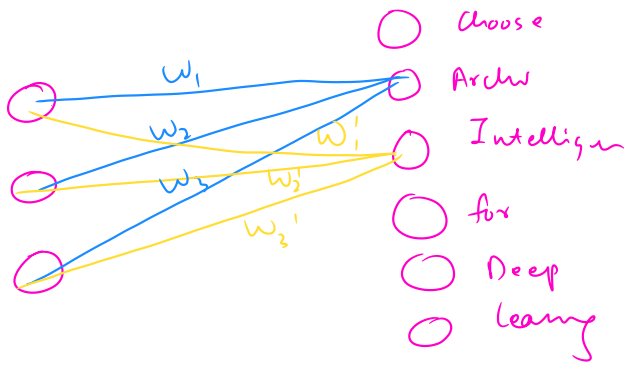


Expected output → [0, 1, 0, 0, 0, 0]

Output we got → [0, 0.4, 0.2, 0.1, 0, 0]

So, we calculate loss and do back propagation

So, vector representation for words:



After backpropagation is complete

Architecture: $[w_1, w_2, w_3]$

[We wanted 3 dimensional representation initially]

Intelligence: $[w_1', w_2', w_3']$

Skip gram

"Choose Architecting Intelligence for deep learning"

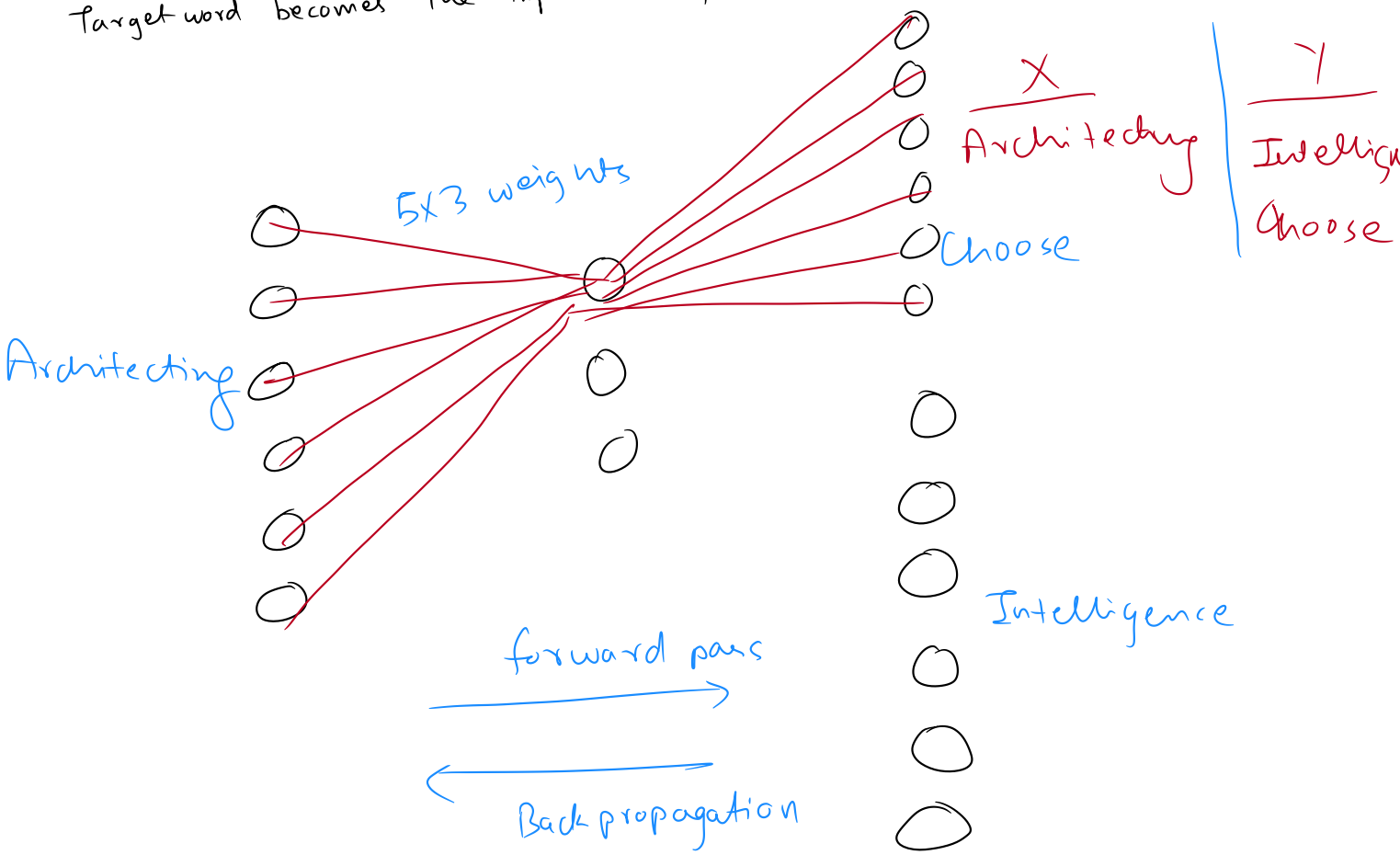
Window-size = 3

Reverse our problem this time

" ____ Architecting ____ " Window-1

predicting

Target word becomes the input here, context words become the output



For Small data : use CBow } Research proven.
large data : Skip gram

How can we improve Word2Vec?
→ Increase the training
→ Increase the dimensions of final vector embeddings (The Hidden layer)
→ Increase window size

New topic
Text Classification

Classification: Supervised Machine learning task
↳ On textual data : Email Spam Classifier
Movie Review Sentiment Analysis.

Types of Classification:

Binary — Eg Entertainment, business

Multi Class

Multilabel — Single news can be part of multiple Sources. (let's say → SRK, → Bollywood, → Entertainment.)

Pipeline:

Data Acquisition → text preprocessing (BoW, TF, Idf, Word2Vec) → text Vectorization → modelling (ML → Naive Bayes; DL → RNN, LSTM, CNN, BERT → pretrained) → evaluation (accuracy, confusion matrix)

Approaches:
Heuristic
APIs (Google, Aws)
ML : Bow, TfIdf, n-gram
DL : Word2Vec

Word2Vec gives vector embeddings for each word
Avg Word2Vec : for sentence

I love India . Overall Sentence: $\left(\frac{w_1+k_1+n_1}{3}, \frac{w_2+l_2+k_2}{3}, \frac{w_3+l_3+k_3}{3}\right)$

Obtained from Word2Vec: I → (w_1, w_2, w_3), love → (l_1, l_2, l_3), India → (k_1, k_2, k_3)

General Guidelines:
1) ensemble techniques instead of Naive Bayes
2) Heuristic features should be combined with ML
3) Start ML techniques, then go for DL
4) Avoid Imbalanced dataset.

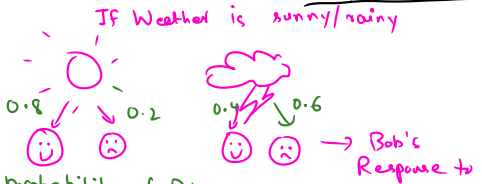
(POS) Parts of Speech Tagging

In simple words, we can say POS tagging is a task of labelling each word in a sentence with its appropriate part of speech.

POS tagging is a preprocessing step.
Used in Named Entity Recognition.
Question Answering System
Word sense diambiguation
Chatbots.

Check out my Jupyter notebook!

HIDDEN MARKOV MODEL



If Bob says that he was

Mon	Tue	Wed	Thurs	Fri	Sat

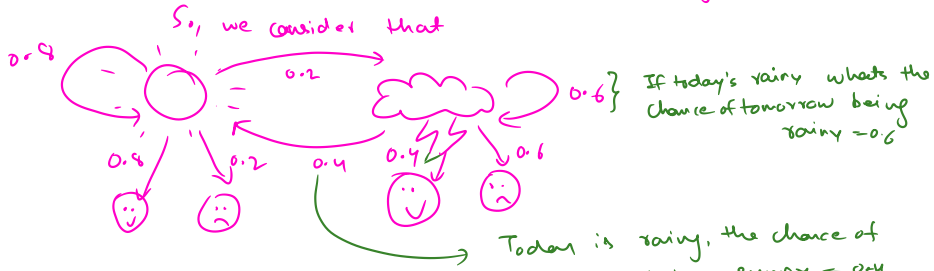
Based on Bob's review we have to guess the weather. So, it can be

S R S R S R

S: Sunny
R: Rainy

But the weather can't change so much.

So, we consider that

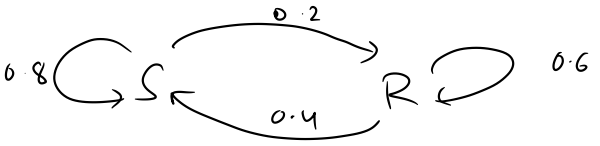


We can only see Bob's reaction, we can't see the whether.

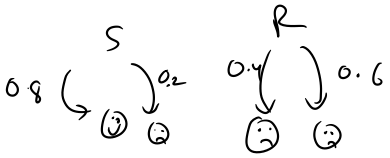
[Bob talks to us over phone and tells us his mood. Bob leaves in France, we live in India].

So, Bob's Reactions: Observations.
Weather: Hidden States.

And its called the "Hidden Markov Model"



These are transition probabilities



Emission Probabilities

Q1) How we found these probabilities.
Let's say the data of weather:

S → S → S → S → R → R → R → S → S → S → S → R → R → S → S

How many times S goes to R on the next day?

S → R = 2
Similarly, S → S = 8
R → R = 3
R → S = 2

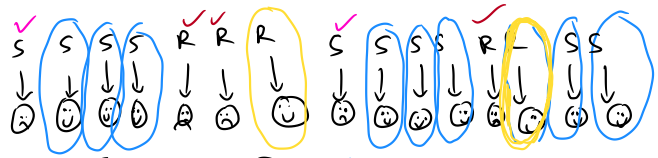
So, from S

R: $= \frac{2}{2+8} = 0.2$
S: $8/8+2 = 0.8$

R

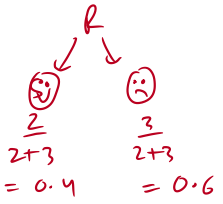
S: $= \frac{2}{2+3} = 0.4$
R: $\frac{3}{2+3} = 0.6$

Weather to Mood Data:



So, S → 😊 → 8
R → 😊 → 2
S → ☹ → 2
R → ☹ = 3

So, $P(\frac{😊}{S}) = \frac{8}{8+2} = 0.8$

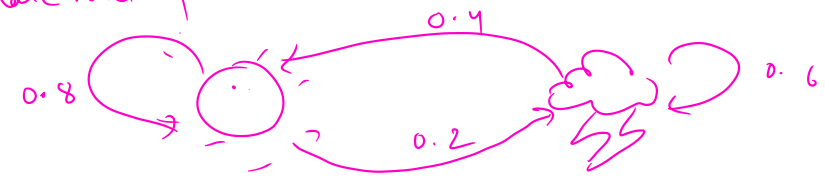


Q2) What is the probability that Random day is Sunny Rainy

No. of S days: 10
No. of R days = 5

(If Bob gave us no inputs).
$P(S) = 10/15 = 2/3$, $P(R) = 5/15 = 1/3$

Alternate Method



$S = 0.8S + 0.4R$

So, $0.2S = 0.4R$

$\boxed{S = 2R}$ —(1)

$S + R = 1$
$S = 1 - R$
So,
$R = 0.2S + 0.6R \Rightarrow 1 - R = 2R$
$R = 1/3$, $S = 2/3$

Q3) If Bob says "Happy 😊" he is

So, Probability of Sunny ↑↑ from 0.67

$$P\left(\frac{\text{Sunny}}{😊}\right) = P\left(\frac{😊}{\text{sunny}}\right) \cdot \frac{P(\text{sunny})}{P(😊)}$$

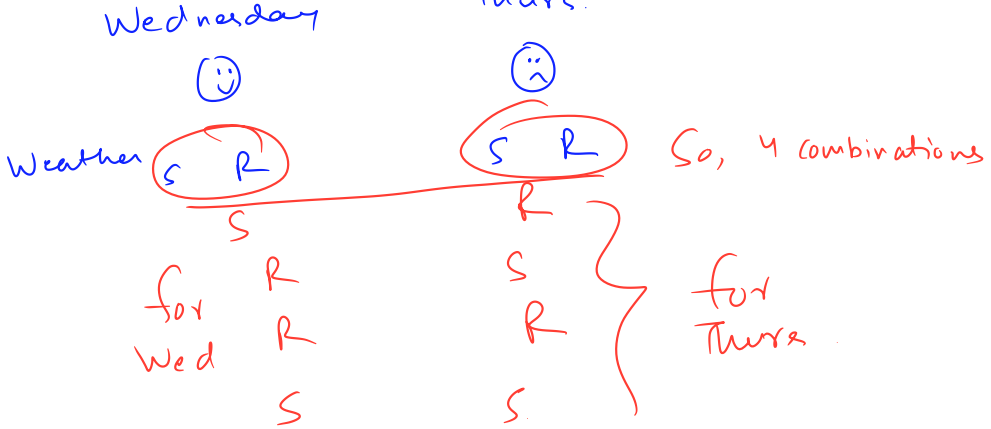
$$= (0.8) \times \left(\frac{10/15}{10/15}\right) = 0.8$$

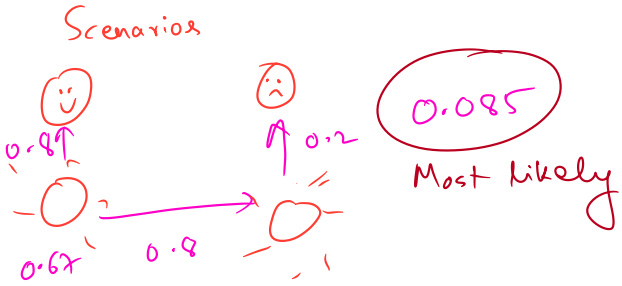
(Bayes Theorem)

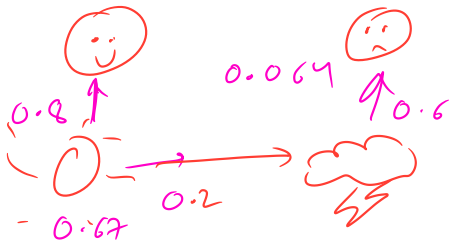
Q4) If for three days, Bob is "😊 ☹ 😊", what was the weather??

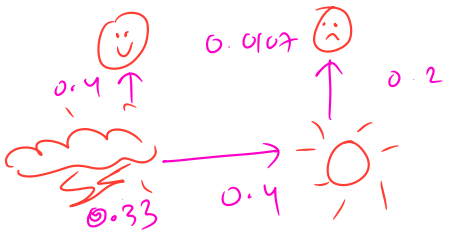
So, we have to guess the weather on all three days.

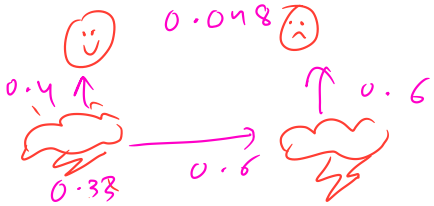
Let's start with 2 days,











So, Wed : Sunny
Thurs : Sunny

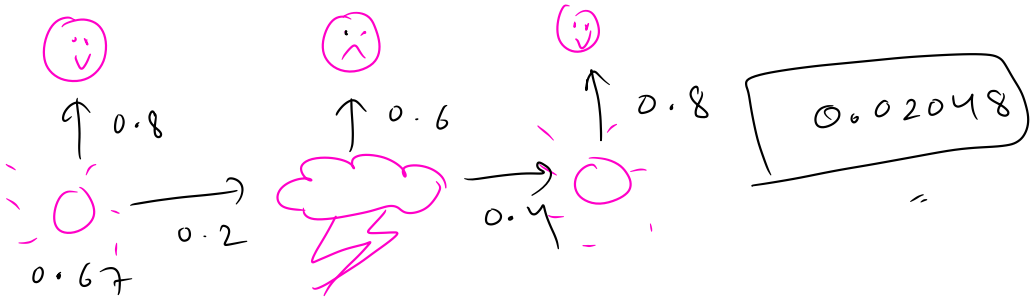
For three days

Wed Thurs Fri

8 combinations

S	S	S
S	R	S
S	S	R
R	R	R
R	S	S
R	S	R
R	R	S
S	R	R

for    Combinations



Check all 8 combinations → Best combination (SSS)

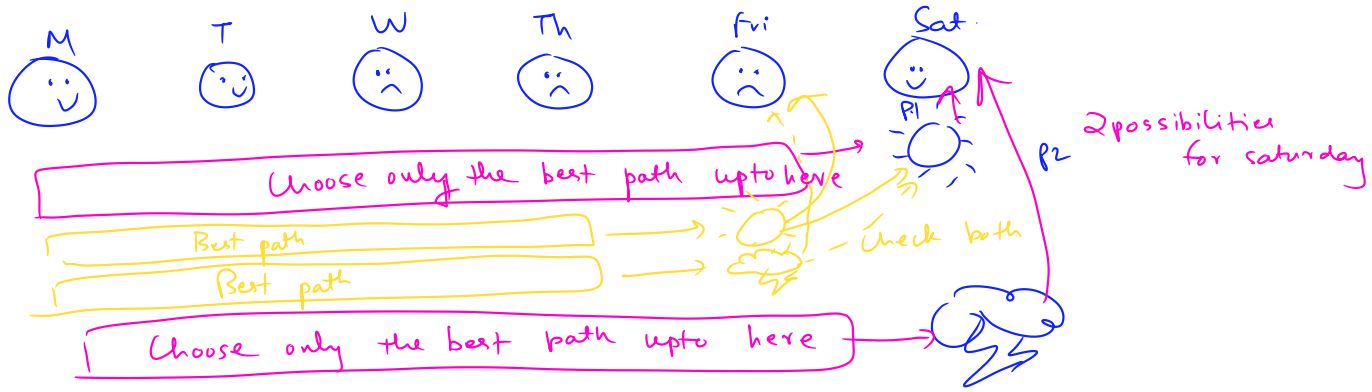
For 4 days, 16 (2^4 combinations)

The number of combinations grows exponentially (2^n)

So, to optimize, we have

VITERBI ALGORITHM

We need to optimize our solution



Check p_1 & p_2 and find out the better probability

Continuing POS tagging.

POS on the sentence

" Will will google Caribbean"

S: Start
N: Noun
V: Verb
M: Model
E: End

Data

S Jack N loves V Caribbean N E

S Can M Jack N google V Caribbean N E

S Will M Elizabeth N google V Caribbean N E

S Elizabeth N loves V Will N E

S Will N lover V google N E

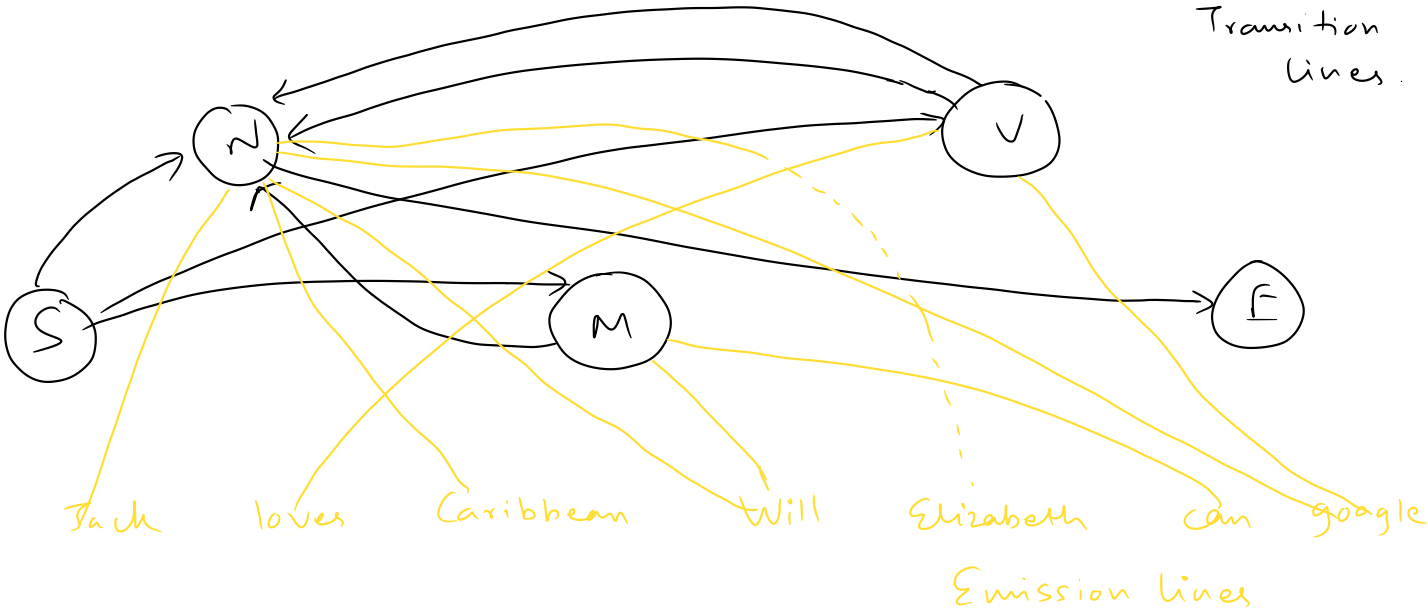
$P\left(\frac{\text{Jack}}{\text{Noun}}\right) = \frac{2}{2+3+1+2+2} = 2/10 = 0.2$

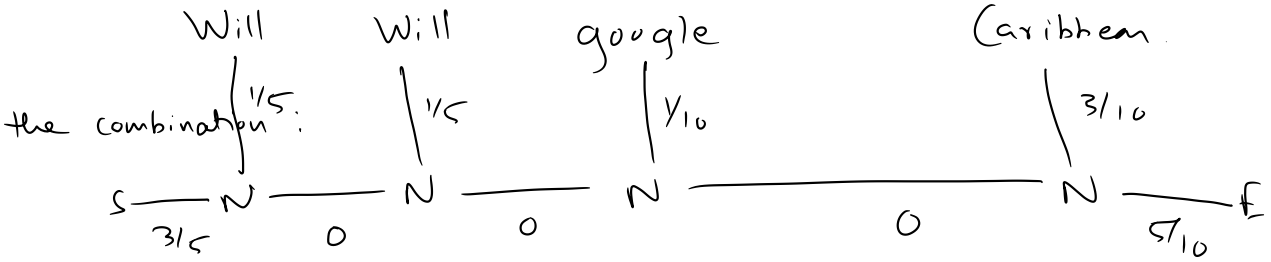
	N	M	V
Jack	2/10	0/2	0
lover	0/10	0/2	3/5
Caribbean	3/10	0/2	0
google	1/10	0/2	2/5
will	2/10	1/2	0
Elizabeth	2/10	0/2	0
Can	0/10	1/2	0

(Emission Probabilities are obtained)

For Transition Probabilities

Transitions	N	M	V	E
S	3/5	2/5	0	0
N	0/10	0/10	5/5	5/5
M	2/2	0	0	0
V	5/5	0	0	0





So, Probability of Combination-1 = (0)

Check all:

Probability of best combination

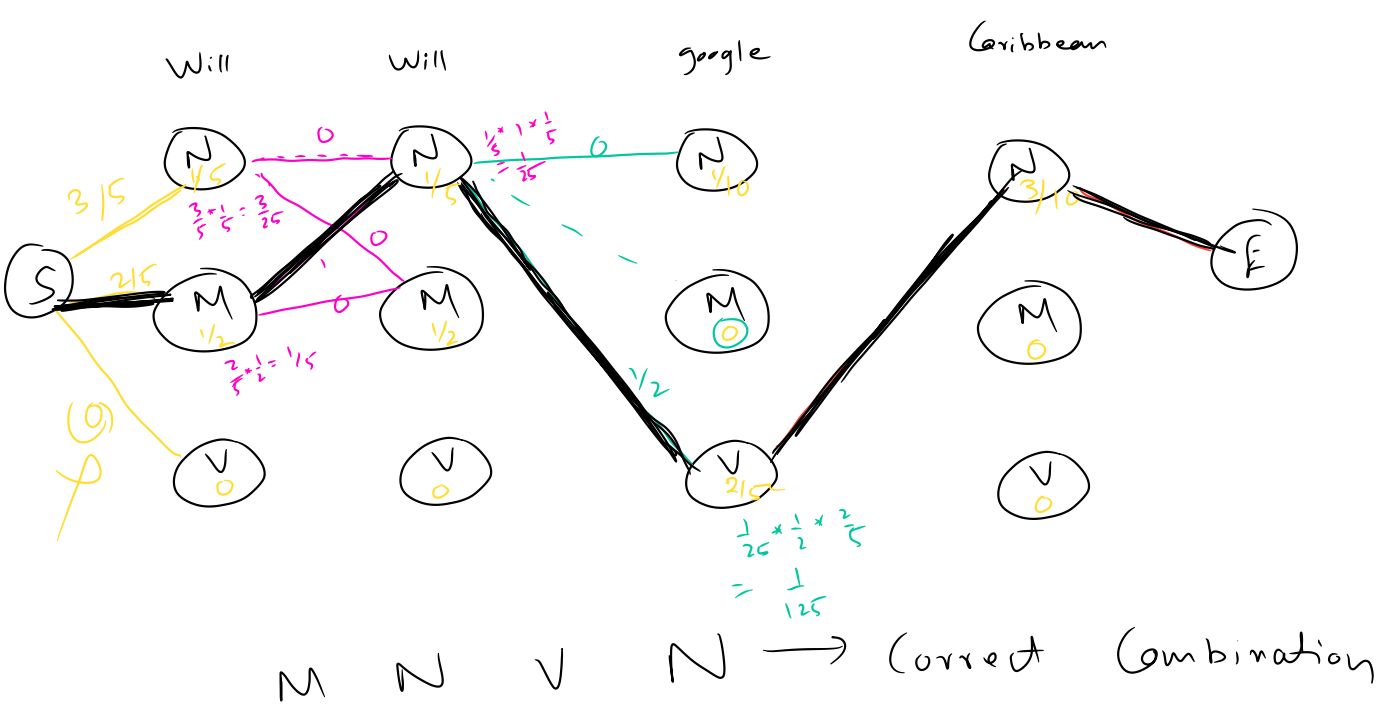
$= 0.4 * 0.5 * 0.2 * 0.4 \times 0.5 \times 0.3 \times 0.5$

$= \boxed{0.012}$

So, total possible combinations = $(3^4) = (81)$

But no. of POS >> 3

So, its optimized by Viterbi



M N V N → Correct Combination