Natural Language Processing (NLP)

Contents

Why NLP?

- Content recommendation like news ads based on profile and interest.
- Sentiment analysis
- Prediction of next text for Generative AI
- Text summarization, language translation, auto suggestions
- Spam classification
- Chatbot

NLP can be used both machine learning and deep learning. At the end of the day we are creating a AI application.

We will see how words can be converted to vector how machine is able to understand text and give output.
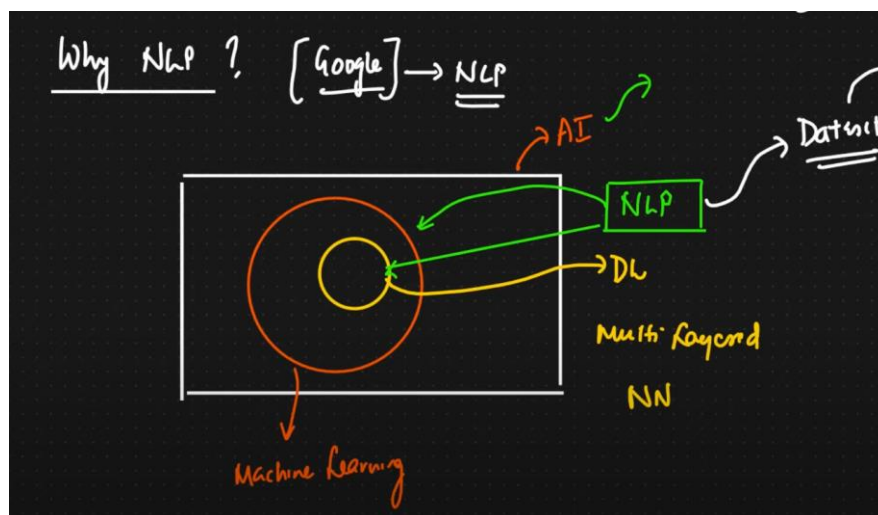
Example: Say we are searching in google as ram temple
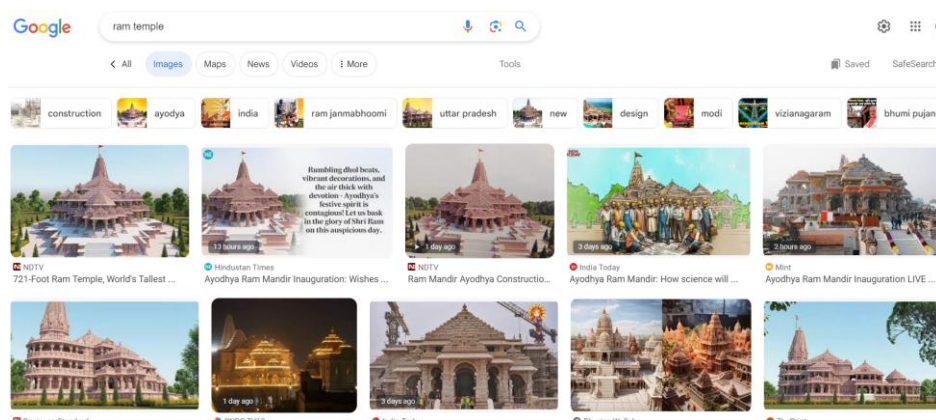
Now how all the images are associated with these images mean in the background text to image conversion is happening that might be using NLP.
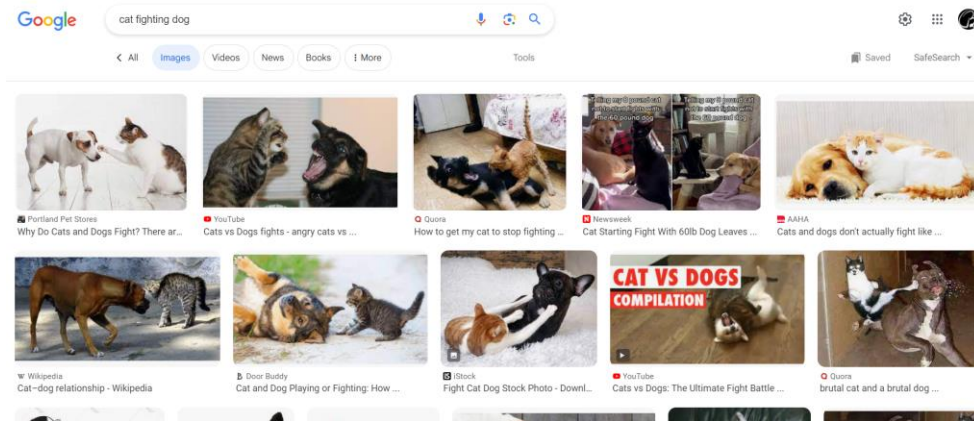
Current limitation of NLP

NLP not able to understand sarcasm in text. Machine is not able to understand sarcasm properly, so research is going on.

Roadmap of NLP

Basic Terminology used in NLP

1. CORPUS
2. Documents
3. Vocabulary
4. Words

CORPUS: It's like a **paragraph** like entire data set if we club together. Say my CORPUS is [D1,D2,D3,D4]

Document: Each **sentence** in the paragraph can b called as Document in NLP like D1 D2 etc. Though it is just a sentence but for NLP it is document. e.g. The food is good => D1
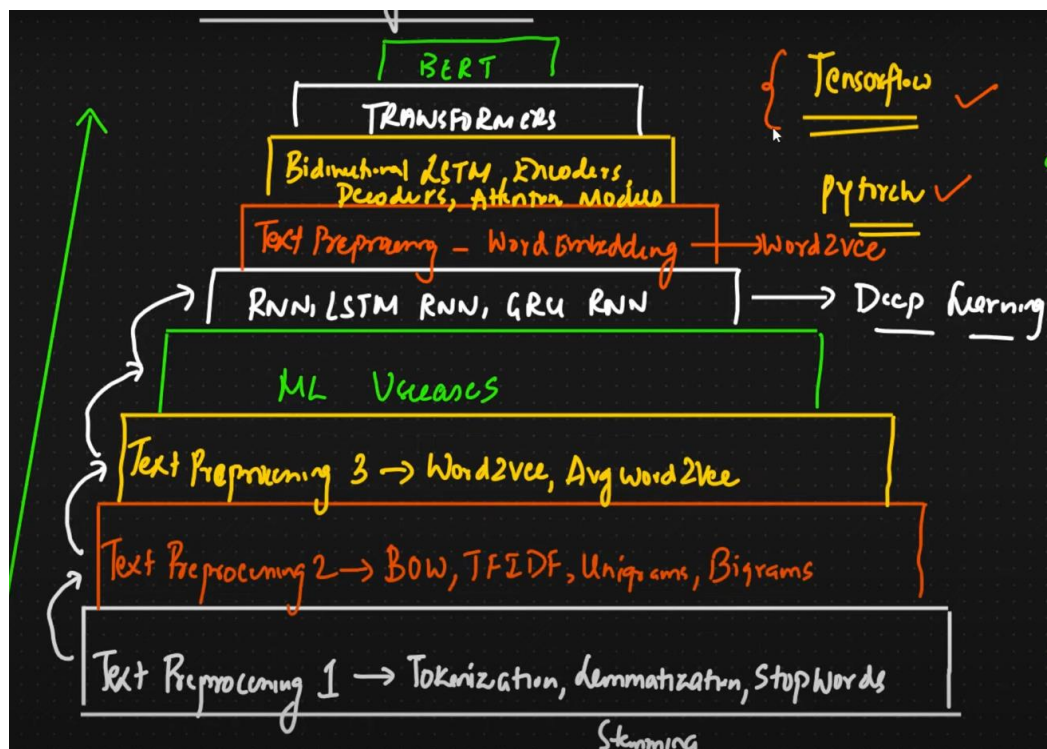
Vocabulary: It is the collection of all **unique words** in the CORPUS.

Say in my CORPUS we have 10K unique words in this case my vocabulary will be

The, food, is, good, bad, Pizza, Amazing, Burger

Words: All words in the CORPUS.

Preprocessing of text data

We need to do preprocessing of raw text before using it for NLP we will be using **NLTK (Natural Language Toolkit)** libraries for doing pre-processing. Here is the way we can do,
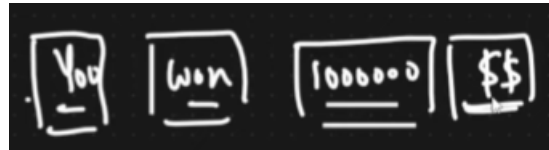
Code Link: https://github.com/gourabb8273/ML-Model-HandsOn-Hub/tree/master/NLP

Tokenization

Say we are using spam classifier. Our data set will be like

The first step here would be tokenization. It's about converting sentence into word so that those should be understandable by machine learning algorithm.

Sentence: You won 1000000 $$ after tokenization it will be

Stop words

Next step is stop words. Say we have a sentence like

"Hey buddy I want to go to your house" here we can see some words like **to** are not making any sense in prediction so we can remove those words.

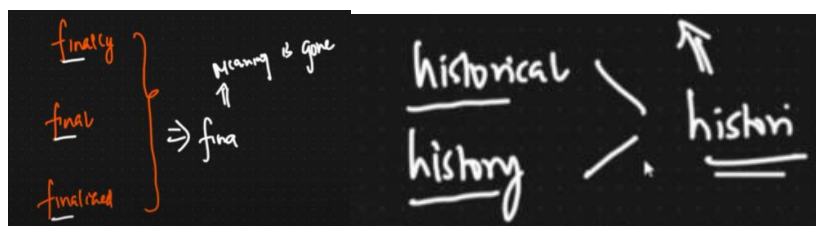Example of stop words: to, he she, is for of etc.

To remove all these words which are insignificant in output prediction using stop words. We can create our own custom stop words list as well. However we may need this in text summarization but not needed for spam classification or sentiment prediction.

While removing stop words you have to be very careful as it may delete the entire sentence so imbalance will create in target and input feature better to have custom stop word

Stemming

Here if we have say many similar types of words in different form we can convert those in base or root word thus we can reduce the number of words. It's a process of reducing words to the base word.

But base word may or may not have any meaning.

Advantage: It's very fast can be applied on large dataset

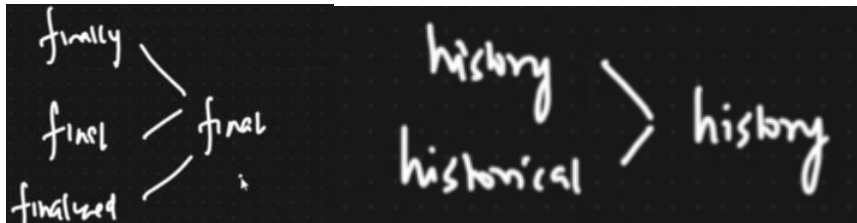**Disadvantages**: It is removing the meaning of the word

Use cases:

- Spam classification
- Review classification

To overcome it we have lemmatization.

Lemmatization

It has entire dictionary of words it will work the same way like stemming but it will give meaningful word.

Advantage: We will be able to get meaningful word.

Disadvantages: It is slow as it has to do lots of comparison.
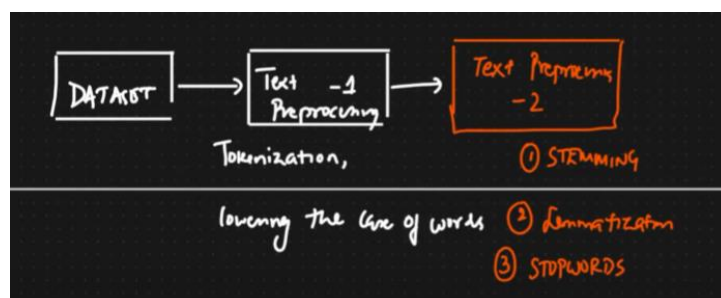
Use cases:

- Text summarization
- Language translation
- Chatbot

Lowering the case of Words

We will be now lowering the case of all word.

Word Embedding

In the next step we will be converting words to vector using different techniques, Word embedding is a technique to convert **words into vector**.

1. OHE (One Hot Encoding)
2. Bag of words (BOW)

3. TF-IDF (Term Frequency Inverse Document Frequency)
4. WordtoVec
    a. CBOW (Continuous Bag of Words)
    b. Skip Gram

One Hot Encoding

Say my CORPUS is

Now each word will be represented as one hot encoding. After doing this we will get as follows

For first document D1 – **A man eat food** we will get.

Here we have 4 words

For the second document D2 – **Cat Eat Food** we will get.

Here we have only 3 words so if vocabulary size is decreases then we can't train the model. Input feature should be fixed constant this called out of vocabulary.

Advantage:

- Simple to implement.
- Intuitive

Disadvantage

- For large set of vocabular size of sparse (many zero values) matrix will be large difficult to compute
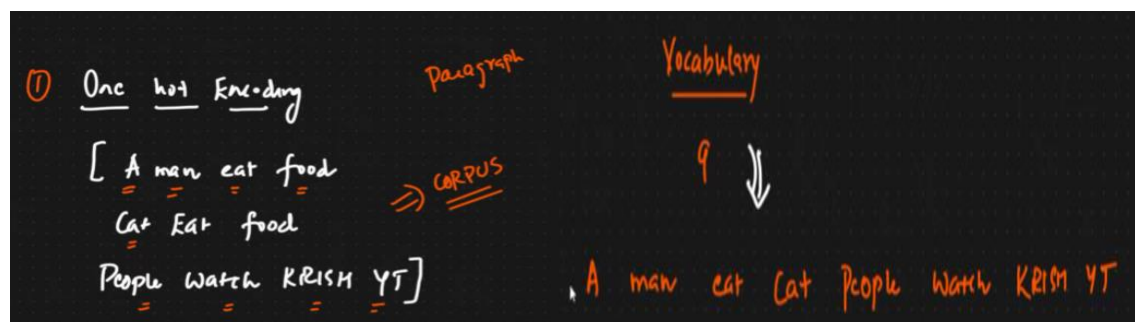- OOV (Out of vocabulary), my sentence is not fixed size. Any new extra words if we get can't be handled.
- Between the word semantic meaning is not captured like how two words related as each row only one column is 1 rest zero.

Bag of Words (BOW)

It is a technique of converting **text to vector**. Say my CORPUS is as follows,

Now we will use stop words and remove unnecessary words also will lower all case.

Now my vocabulary unique words will be my feature. Let's see that with frequency like no of time words are repeating,

Order of the feature will be based on frequency e.g. if **good** is repeating more time than other then first feature f1 will be **good**.

We will just increase the count in the feature based on the document. Like first doc was good boy so f1 and f2 became 1 and f3 is 0.

Say in Document 1 we have one more good like – **good boy good** then count of **f1** will be increased by 2.

Binary Bag of Words: In Bag of Word there is an option called **Binary Bag of Words**. Like if any count more than 1like f1 then also we will make it 1.

Advantage:

- Simple and Intuitive

Disadvantage

- Sparsity is still there though improved but not fixed.
- OOV (Out of vocabulary), if we get any new word like say **cat** then we won't be able to handle in feature so entire word will get rejected so meaning will get changed.
- Ordering of the word completely changed as we are doing ordering based on max frequency so won't be able to make semantic relationship.
- Semantic meaning not captured well like good and better are similar word so how similar words are.

If we take those features of 3 dimensions and plot it we will be able to calculate distance between them to find the similarity

- Euclidian Distance
- Cosine Similarity

Cosine Similarity

Say we have two point and we need to find out the similarity we will find the angle between them and subtract from 1. So P1 and P2 is 47% similar.

Say another example,

If the cosine similarity is close to 0 value then words are very similar that happens when angle will be less between them

This is heavily used in **recommendation engine** here avenger and Iron man are similar, but Minions is different as they are making 90-degree angle between them.

In the above example both the sentence or **documents are different like opposite** but the vector **looks similar** if we find cosine similarity that's the main problem **semantic meaning is missing** like which words should I focus more that will be **fixed using TF-IDF.**

Capturing semantic Information using Ngrams

- Bigrams (combination of two words)
- Trigram (Combination of three words)
- Ngrams (Combination of n words)

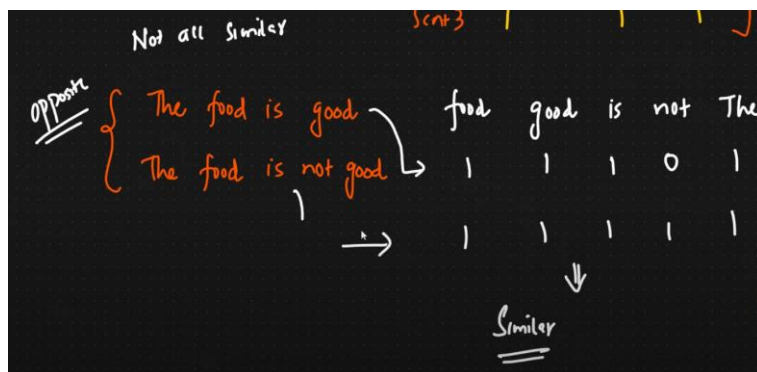Apart from single feature we will be using combinations of features to understand semantic meaning of words.

We are adding two more features f4 and f5 using **Bigrams**.

How many **Bigrams** will be possible for word **I love food**? **I love and love food** only two bigram I food can't be possible.

How many **Trigram** will be possible for word **I am not feeling well**?

If we say combination of **Ngrams is (1,3)** then we will include single feature (unigram), Bigram and trigram 1 to 3. Say doc is I am feeling well my feature will be.

- f1 – I,
- f2 – am,
- f3- feeling,
- f4- well,
- f5-I am,
- f6-am feeling,
- f7-feeling well,
- f8-I am feeling,
- f9- am feeling well

TF-IDF (Term Frequency - Inverse Document Frequency)

It has two important concept **Term Frequency** and Inverse **Document Frequency**. Here we are trying to solve the issue i.e. semantic meaning of the words to know which word we need to focus more on.

Whichever words are **rarely presen**t in the sentence we will give **more weightage**.

Here rare words are Cat, Bat and Krish rarely present in the sentences so we need to give more weightage.

- These **rare** words will be captured by **Term Frequency** which will have more weightage.
- **Common** words will get captured by **Inverse Document Frequency.**

When we combine both semantic meaning will be there.

Term Frequency:

No of repetition of word in sentence / No of words in sentence

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

Inverse Document Frequency:

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

By multiplying both we will get TF-IDF

Example

Find Term Frequency

Here is my sentence.

Sent 1 : good boy

Sent 2 : good girl

Sent 3 : boy girl good

Term Frequency

	Sent1	Sent 2	Sent 3
good	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$
boy	$\frac{1}{2}$	0	$\frac{1}{3}$
girl	0	$\frac{1}{2}$	$\frac{1}{3}$

Here for word good in sentence 1 only one time is present and total word in sentence 1 is 2 hence TF is ½.

For boy in sentence-3 only one time is present total word in sentence is 3 so TF is 1/3.

Find Inverse Document Frequency

This gets computed for every sentence.

Here no of sentence is fixed that is 3 and good is appearing in all 3 sentences so good is common word and log (3/3) =0 so no weightage.

Similarly for word boy it is appearing only in sentences so log (3/2)

If we multiply both then we will get TF-IDF

Now we will multiply TF and IDF in feature space like for good in sentence 1 we had ½ and from IDF it is 0 so it will be ½*0 =0

✓ Thus, we are removing or reducing weightage for common words and increasing weightage for rare word so some semantic meaning is captured.

Advantage

- Intuitive

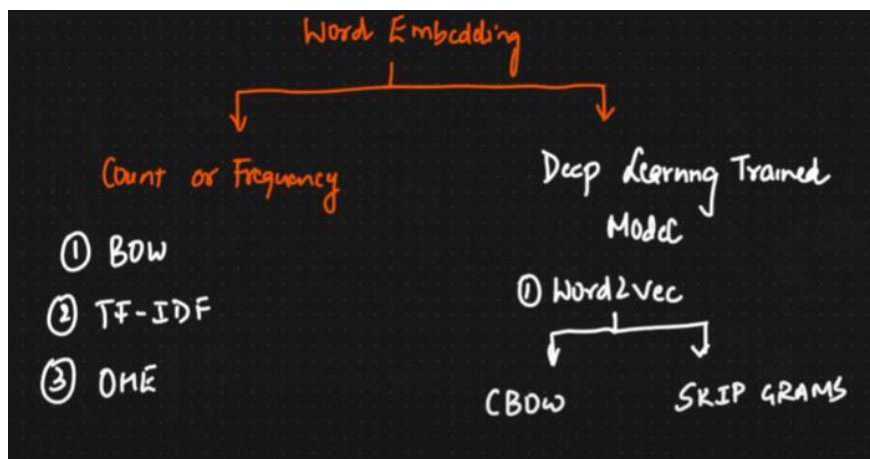- Word important is getting captured like semantic meaning.

Disadvantage

- Sparsity is still there for large data set though better than BOW.
- Out of Vocabulary is still there.
- Semantic meaning not captured well like good and better are similar word so how similar words are.

Word2Vec

https://github.com/gourabb8273/ML-Model-HandsOn-Hub/blob/master/NLP/Practicle/NLP_Pre_Processing_Machine_Learning/text_preprocessing_Word2Vec.ipynb

Now we have seen few technique for creating word embedding based on count of frequency now it's time to see another popular technique using Deep Learning Trained Model. Those are CBOW and Skip Gram.

The main two problem that is **sparsity** and **semantic meaning** of words like **how similar two words** will be fixed in word2Vec technique. We will have embedding layer that convert text to vector.

Say we have some features like words in sentences.

- Every word we will create a vector but of **limited dimension** like 100 or 300 dimensions within that many dimensions we will represent the entire word.
- Sparsity will be reduced we won't found zero value all will be non-zero value.

- Semantic meaning will be captured like if two words say good and better are similar then in vector also difference in value will be very less.

Feature Representation

Every word like Boy Girl will be represented as some no of features.

- Say we have 300 dimensions of word or feature like Gender, Royal Age etc and every word of my input document will be represented as weightage of those feature in vector format or relate with those 300 dimensions of feature. Those 300 dimensions of feature will be created by word2vec algorithm.
- Google has done this with more than **3 billion** no of dimension of feature.

- In the row we have feature or vocabulary from our corpus.
- In the column we have 300 dimension of feature those are created by word2vec algo
- We are having weightage or how my vocabulary is related to those features. **Royal is related to King so 0.95 say.** Thus, we are identifying semantic meaning of the words.

- Here each of my words like **Boy represented as vector of having limited dimensions like here 300** through we are representing based of 300 features. Ideally words will be very high dimensions like billion.

Say assuming we have only 2 feature dimensions and we have words like King, Queen, Man, Women so we will get **King - Man + Women = Queen** using vector addition and subtraction or **cosine similarity** we can also use Euclidian distance or Manhattan distance.

CBOW (Continuous Bag of Words) Architecture

Here we predict the target word from context.

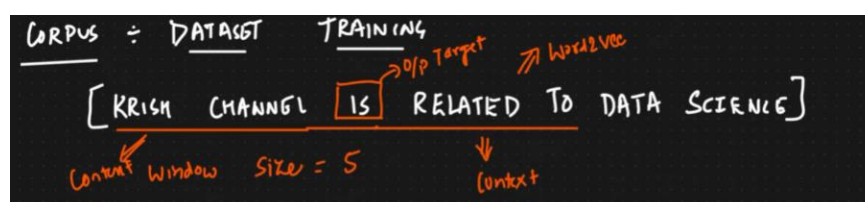Say we have a Copus collection of sentences.

- We need **window size** for creating our training data.
- With respect to training data we will have independent feature and output feature.

Example:

Assume window size is 5 (choose odd number).

So according to CBOW **centre word** will be the **output** or target and before and after words will be independent feature or context.

Now I will **slide my window** by one step. So next centre word or target is **Related.**
Similarly, we will get the independent feature and target.

Now I can't slide because I won't get 5 words. In real google take entire dictionary 13 million words to do this with millions of words.
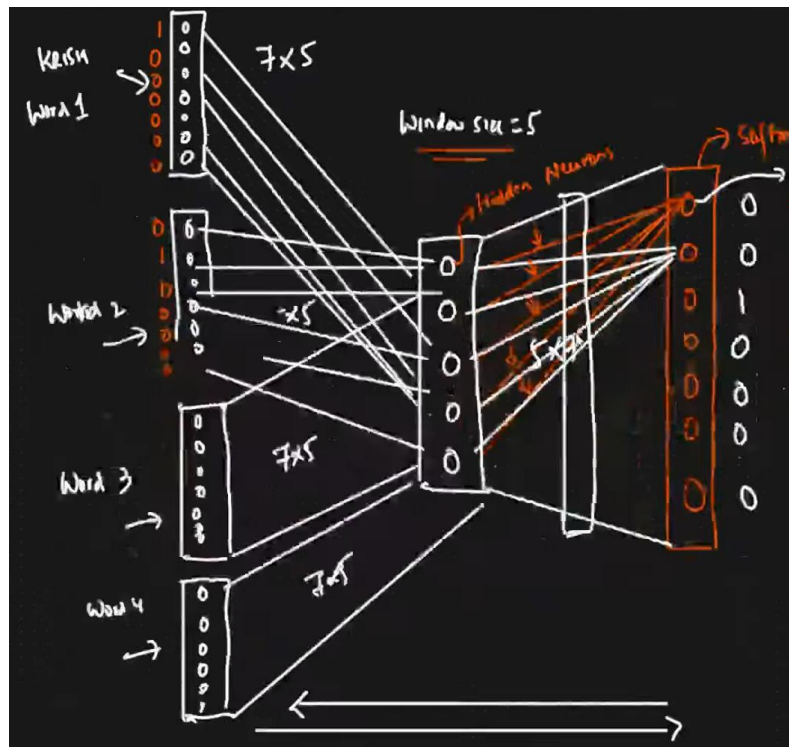
Now we also need to represent the words as One Hot Encoding like OHE. Then both OHE and our feature data will be fed to Deep Learning Neural network i.e. fully connected layer to learn each word representation as vector of feature.
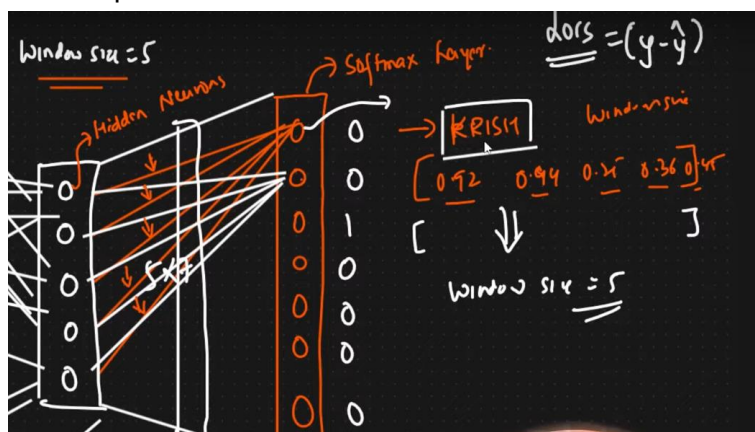

Here input size is 4 words due to 5 window size and each vector is represented by 7 vector using OHE.

- In first sample 4 words like Krish, Channel, Related, to each words will be represented as 7 vector as input feature i.e. 7*5 matrices in ANN.
- **Window size is 5 so hidden layer will be having 5 neurons so each word will be having 5 vector weightages** with 5*7 dimension. Word vector dimension not always same as window size it could be more but not dependent on vocabulary length
- Out put layer is having 7 nodes as IS is having 7 vectors in OHE. Here we will use **SoftMax** layer.
- In the out put we have 7 vectors so the weightage of hidden neuron connection will be the actual vector representation of each words of vocabulary. Like first index in output layer is Krish in our vocabulary which is currently 0. Now after training it will have weightage like the how all hidden neurons are connected to first output index.

- Thus, we are getting semantic meaning and as those weightages are not zero hence it is not a sparse matrix.
- We can increase the hidden layer thus we can do hyper parameter tuning.

Skip Gram
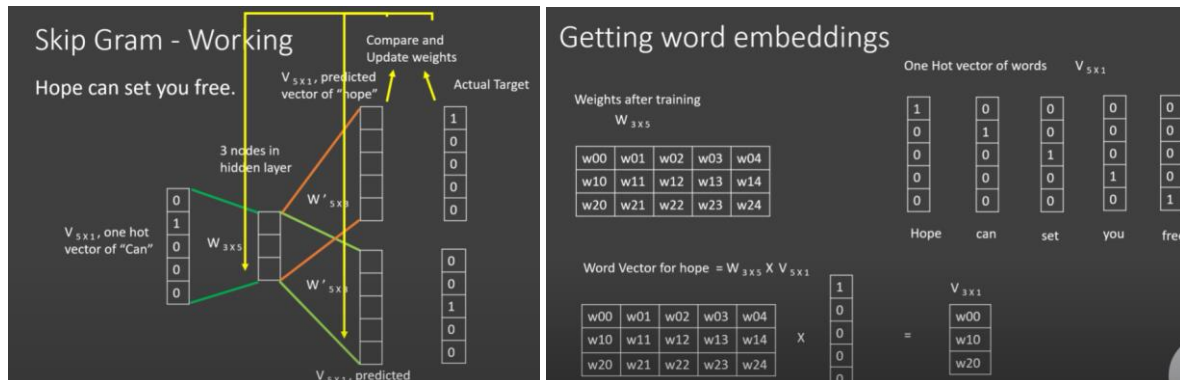
Here we are doing opposite like predicting the context words from target word.

- Consider window size as same as 5.
- Here input will be the centre word and output will be the context words.
- Just reverse the ANN and we will get Skip Gram
- If google is creating a word to vec as 300 dimensions like every word will have 300 values related to other feature word then window size is 300.

I/p	O/p
IS	KRISH, CHANNEL, Related, TO
Related	Channel, Is, To, Data.
To	IS, Related, Data, Science

- For large data use Skip Gram other wise use CBOW

More window size means very good model more semantic information can be captured.

Gensim is a Python library for natural language processing, offering tools for document similarity, topic modeling, and word embedding. It enables efficient analysis of large text datasets for extracting meaningful patterns and insights.

```python
from gensim.models import Word2Vec, KeyedVectors
```
✓ 0.0s

We will be using pre trained model made using googole news information more 3 billion data by Google for word embedding

```python
import gensim.downloader as api
wv = api.load('word2vec-google-news-300') # this eill be of size in 1662MB
```
✓ 6m 3.3s

[===] 100.0% 1662.8/1662.8MB downloaded


```python
wv.most_similar("king")
```
✓ 3.2s

```
[('kings', 0.7138045430183411),
 ('queen', 0.6510956883430481),
 ('monarch', 0.6413194537162781),
 ('crown_prince', 0.6204220056533813),
 ('prince', 0.6159993410110474),
 ('sultan', 0.5864824056625366),
 ('ruler', 0.5797567367553711),
 ('princes', 0.5646552443504333),
 ('Prince_Paras', 0.5432944297790527),
 ('throne', 0.5422105193138123)]
```

```python
wv.most_similar("man")
```
✓ 3m 41.2s

```
[('woman', 0.7664012908935547),
 ('boy', 0.6824871301651001),
 ('teenager', 0.6586930155754089),
 ('teenage_girl', 0.6147903203964233),
 ('girl', 0.5921714305877686),
 ('suspected_purse_snatcher', 0.571636438369751),
 ('robber', 0.5585119128227234),
 ('Robbery_suspect', 0.5584409832954407),
 ('teen_ager', 0.5549196600914001),
 ('men', 0.5489763021469116)]
```

```python
wv.similarity("boy","man")
```
✓ 0.0s

0.68248713

```python
vector= wv["king"]-wv["man"]+wv["woman"]
```
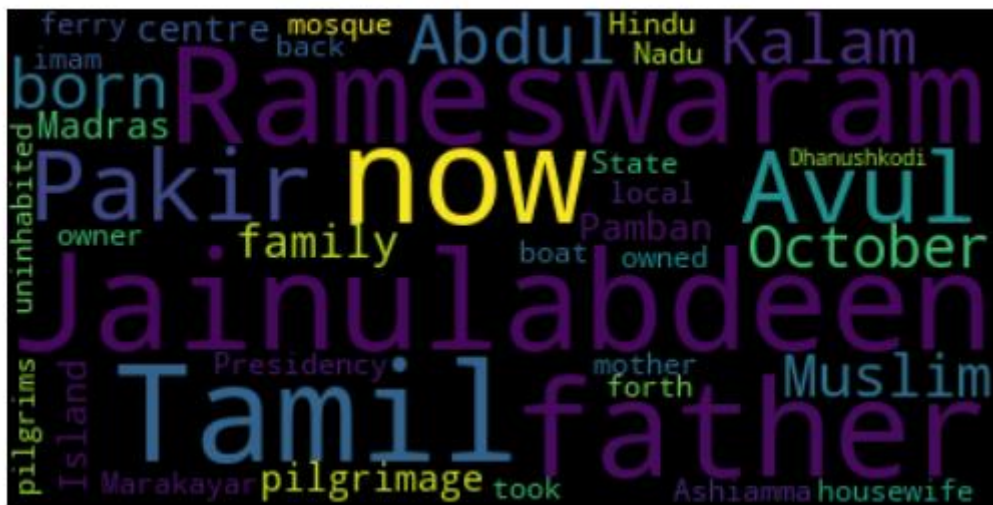✓ 0.0s

```python
wv.most_similar([vector])
```
✓ 0.3s

```
[('king', 0.8449392318725586),
 ('queen', 0.7300517559051514),
 ('monarch', 0.645466148853302),
 ('princess', 0.6156251430511475),
 ('crown_prince', 0.5818676352500916),
 ('prince', 0.5777117609977722),
 ('kings', 0.5613663792610168),
 ('sultan', 0.5376775860786438),
 ('Queen_Consort', 0.5344247817993164),
 ('queens', 0.5289887189865112)]
```

Word to Cloud

If I convert those 300 dimensions to 2D then it will look as word to Cloud.
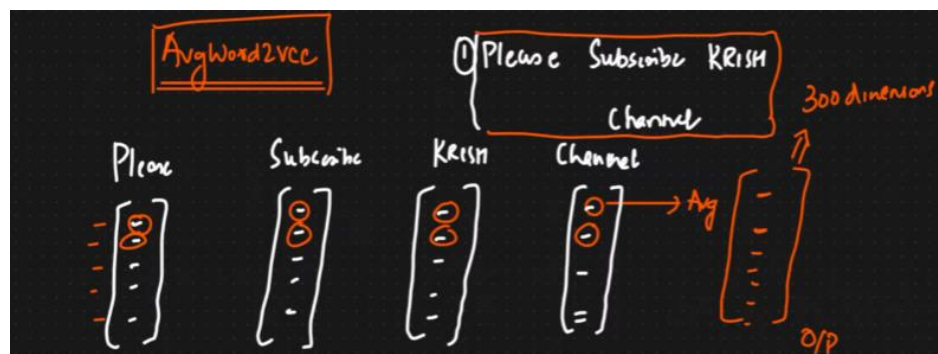
wordcloud is a Python library that creates visual representations of word frequency in a text corpus, generating "word clouds" where the size of each word reflects its frequency or importance in the given text.

Average Word2vec

While training a model using word2Vec from scratch not by using any pretrained model there will be one problem. Here say for spam filtering input would be my message of any no of words.

Now if we use 300 as window size then each word of the message will be of 300 dimensions, but we need 300 as a total dimension for the input feature as **input dimension should be fixed i.e. 300**. Hence, we will be taking the average of vector for all words.

Spam Classification using Different Techniques from scratch

https://github.com/gourabb8273/ML-Model-HandsOn-Hub/tree/master/NLP/Practicle/Spam-Classifier

Now we have seen general machine learning algorithms for NLP. Let's start NLP with Deep learning.

Going from **Machine Learning to Deep Learning**.

NLP with Deep Learning

Now we will see.

1. RNN (Recurrent Neural Network
2. LSTM (Long-Term Short-Term Memory) RNN
3. GRU RNN
4. Bidirectional LSTM RNN
5. Encoders-Decoders
6. Transformers
7. BERT
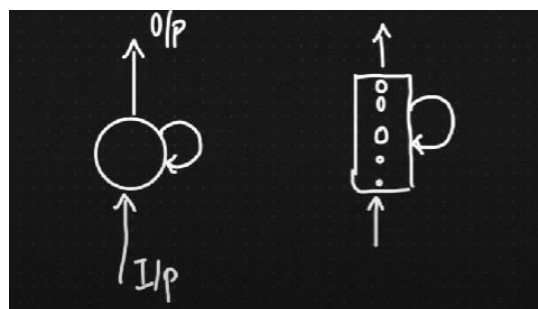
Recurrent Neural Network (RNN)

So far for converting text to vector we have found most efficient way is **Word2Vec** i.e. **AvgWord2Vec** the main reason was semantic meaning was captured. For all the real word complex applications like language translation, Text generation, chatbot etc we can't only rely on machine learning we have to go for Deep Learning technique like RNN, LSTM RNN, BERT etc to train the model and converting word to vector.
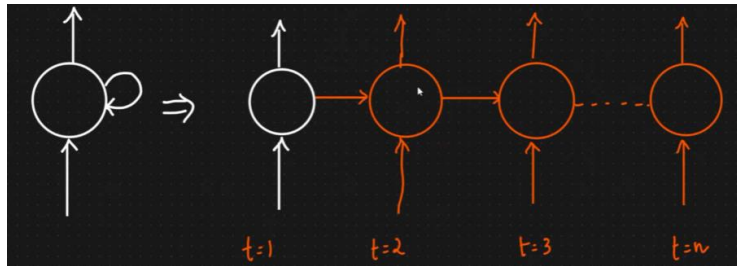
RNN is super useful for Time series Data where sequence is very important.

RNN Basic Diagram

Output of Neuron is given back to the same network. We can have N no of Neuron all output will be given back to the same network.
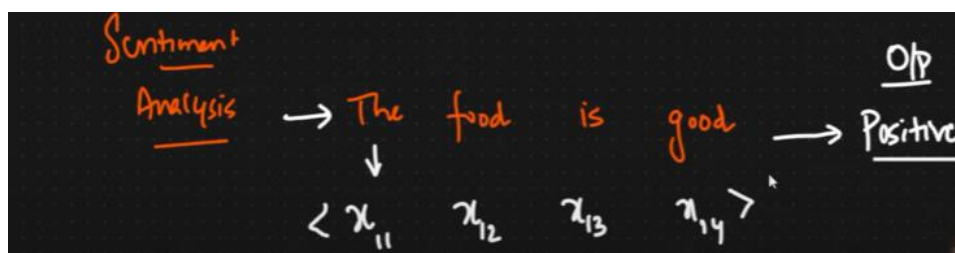

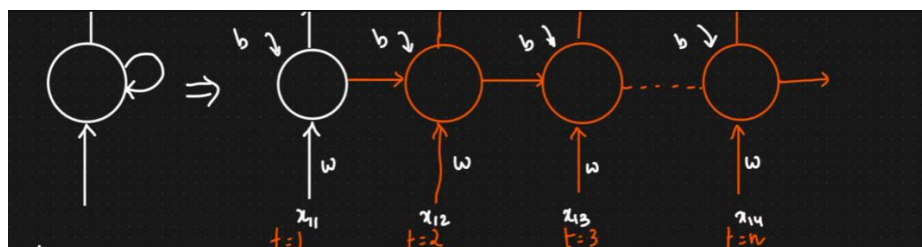Let's **expand** this circuit as below,

Whatever output is generating we are just giving it to the next timestamp of the same neural network.

Let's say an example **Sentiment Analysis**, We have a document and we converted it to work x11 to x14.
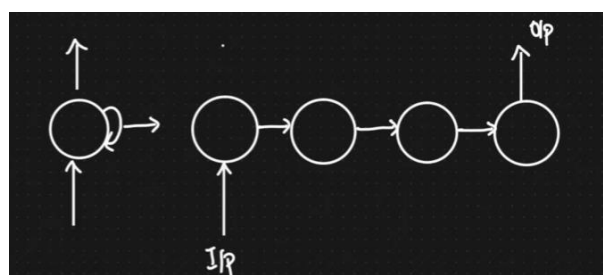

Fed the Input to RNN

Types of RNN

- One to One RNN
- One to Many RNN
- Many to One RNN
- Many to Many RNN

One to One RNN

Here we are giving **input to first Neuron** and the **output instead of retrieving being passed to next Neuron and output is retrieved from the last Neuron**.
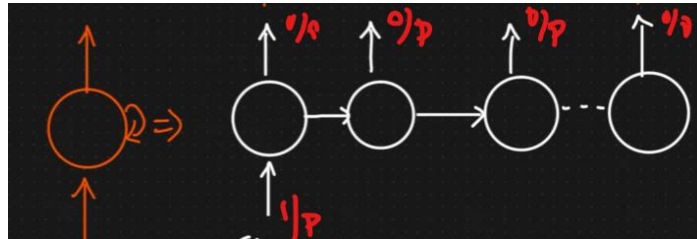
Example: Image Classification – Input in Image as pixel Output is label
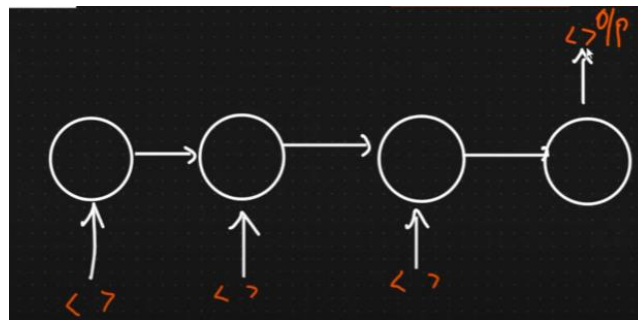
One to Many RNN

Here we are giving **input to first Neuron** and the **output is retrieving from each Neuron and also being passed to next Neuron**.

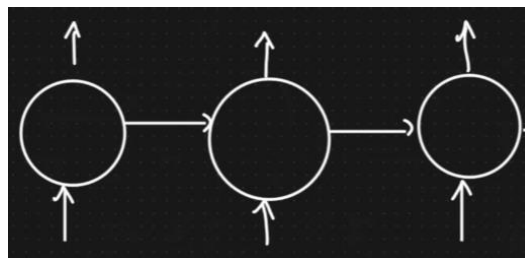**Example**: Google Search Suggestion one input and multiple output.

One to One RNN

Here we are giving **input to all Neuron** and the **output is instead of retrieving from each being passed to next neuron and retrieved from last Neuron**.

Example: Sentiment Analysis, Predict next day sale

Many to Many RNN

Here we are giving **input to all Neuron** and the **output is retrieving from each Neuron and also being passed to next Neuron**.

Example: Chatbot, Text Generation, Language translation, Auto suggestion
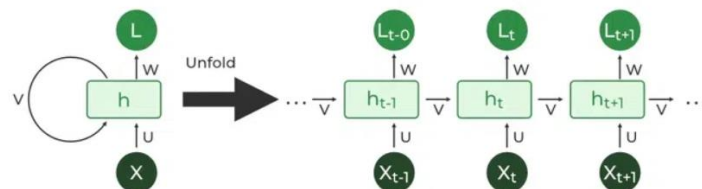
Forward Propagation in RNN

 Say for **Many to One** like sentiment analysis the Forward Propagation Diagram looks like as below,

$$O_1 = f(x_{11} * \omega)$$
$$O_2 = f(x_{12} * \omega + O_1 * \omega_i)$$
$$O_3 = f(x_{13} * \omega + O_2 * \omega')$$

Here the **x11** Is say the word The. Now it has a weight say **w** goes to neuron and output is o1 i.e. $\boldsymbol{f(x_{11} * o_1)}$ This now sent to next neuron with another weight as **w'** also and input say x12 already coming. Thus, every output of neuron depends on input as well as hidden feature coming from previous neuron in forward direction. So called **forward propagation**.

Another example for Many to Many

Now in the output we can use activation function like Sigmoid SoftMax and retrieve the labels.
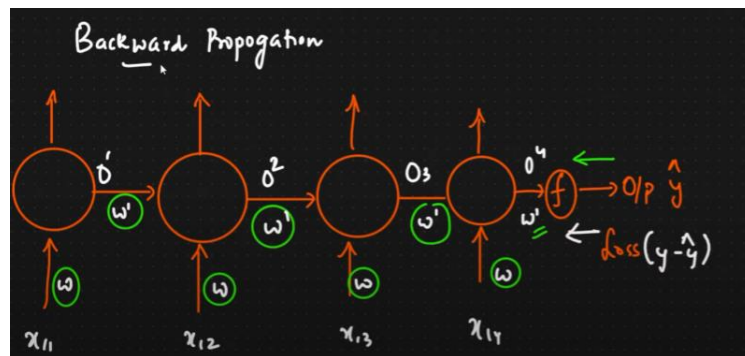
- **Sigmoid** Function for **Binary** class Classification
- **SoftMax** Function for **Multiclass** Classification

Now we will get y^ and we can calculate loss function and update all the weight.

Back Propagation in RNN

In the forward propagation we have got the weight and calculated the loss function now in Back Propagation we will try to go back and update the parameters to reduce the cost.

All the weights marked in green will be updated now based on the loss.

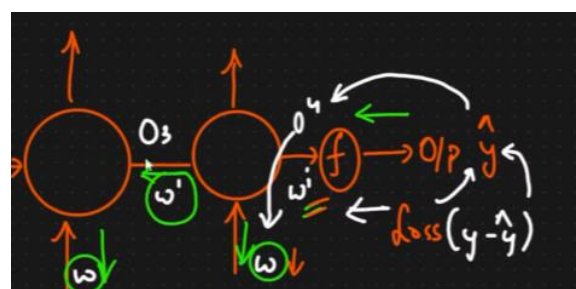Let's see how we update each weight based on loss function.

Updating Parameter Based on Cost Function

Say we need to update w' which is the weight of last neuron while doing back propagation using weight updation formula.

Learning rate (n) will be initialized. The main thing we have to find out is **derivative of Loss by derivative of w'**.

We can find this by using simple chain rule. **dl/dw' = dl/dy^ * dy^/dw'**

Now same way we update other parameters like **w** using the same chain rule. To go to **w** param first we go from **y^ to O4** then from **O4 to w**. Thus, we go as chain and w update dependent on previous output and weight.
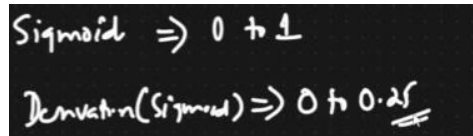

This back propagation will happen for certain no of epoch then once the loss is stagnant we will do early stopping

Problem of Back Propagation

Now, there is a problem for large sentence (only if RNN is very depth for deep) say the timestamp over there is about 300 like t1 t2 to t300 and now if keep on updating one by one then derivative will become smaller and effect also will become small.

Say for **sigmoid activation** function value is **0 to1** and **derivative** of its value is **0 to 0.25**.

As **derivative value is small** so as **we go back (t300 to t100)**, the value will keep on getting **smaller** and smaller so **weight updation will be negligible**. To fix this issue we have to specifically use another neural network called **LSTM RNN (Long Short-Term Memory and inside that Recurrent Neural Network).** This super amazing as entire neural network will be able to remember the context of the network. It has component like Memory Cell, Forget Cell, Input Cell etc.

Understand the Problem with Example

Say we have a word "**My name is john and I want to eat pizza**". Here in this sentence eat comes before pizza this context my RNN can capture easily but **My** and **I** also related but we have many words in between so many neuron due to **effect** of **smaller derivatives RNN wont be able to capture the context between I and My** as there are **many neuron in between and weight will become negligible**.

This problem is called **Vanishing Gradient** and **Dead Neuron**.

Hence, we need to go for LSTM RNN.
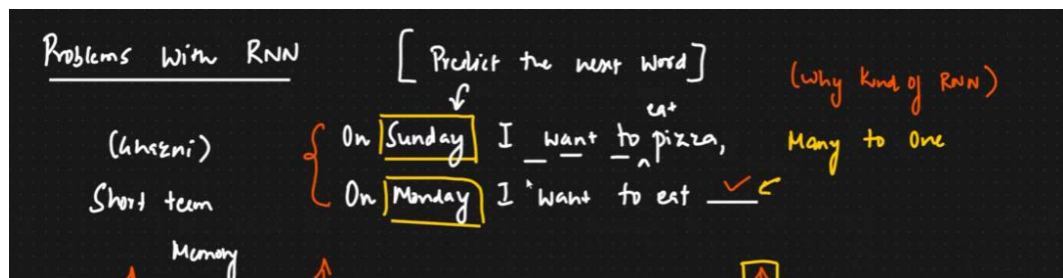
Long-Short-Term Memory RNN (LSTM RNN)

We need LSTM RNN to solve two major problems.

- Vanishing Gradient or Dead Neuron
- Capturing Context Information for Deep RNN Network

RNN can only capture context from previous if gap is small. But when we need to have more context and gap is also huge then RNN will fail to work.

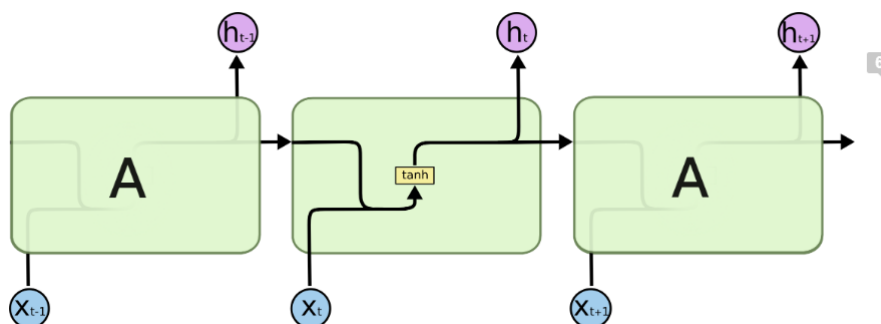https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN can only remember Short Term Memory but we need to remember Long Term Memory as well, so we need LSTM

Here I will eat on Monday will depend on Sunday as well not only Monday as I can have something else on Monday. This context will be hard to capture in normal RNN.

Traditional RNN

A tanh **activation function tanh** is applied after combining the previous output and new input with weight. This tanh is applied in all neurons.
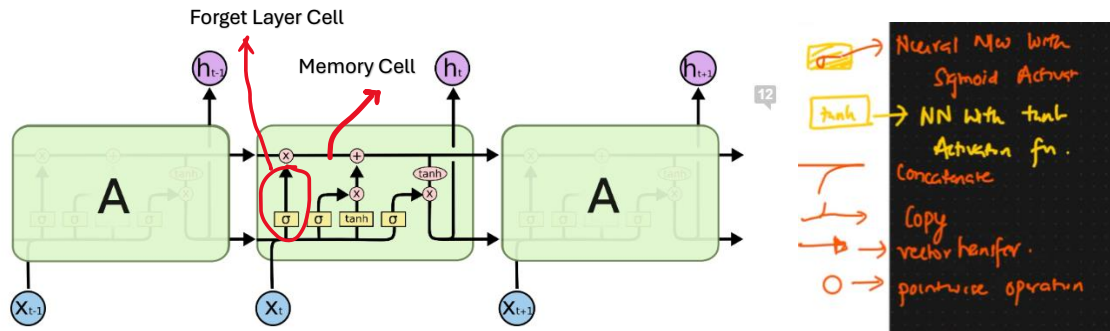

LSTM RNN

It has an important property called Long Short-Term Memory. When we have long sentences, and the **context** is **continuously** switching.

Say here friend might be the context of the prediction output but name is having context with Krish hence context is switching continuously which we can't do with RNN.
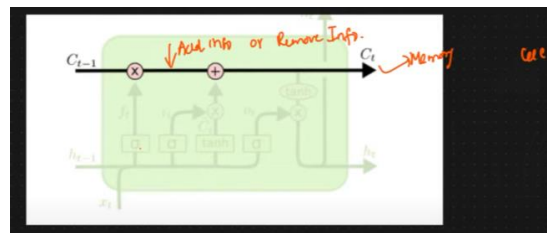

So what exactly changes is happening in RNN so it is able to remember long term memory as well?
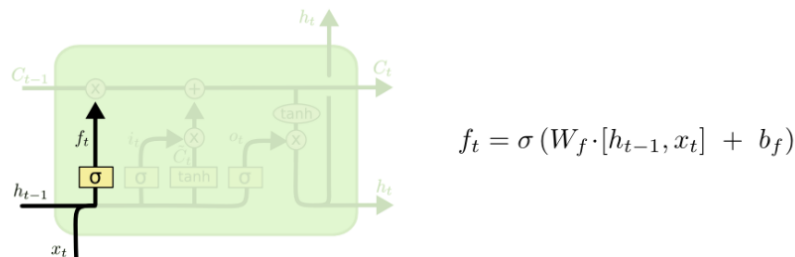
Memory Cell:

It works like conveyer belt in Airport. We can **add Info** and **remove info**.
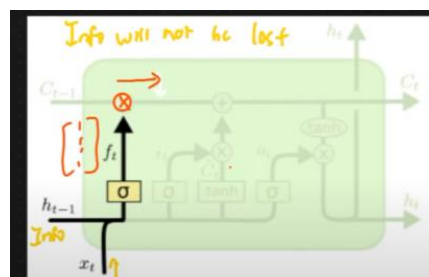

Forget Layer Cell:

When context switch happens our memory cell **should forget about the previous information**. In that case sigmoid will transform the value to 0.

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

Example -1 with out context switching:

Here context switching is not happening so in forget layer sigmoid will get 1 value. And information will be added in memory cell with point wise multiplication.
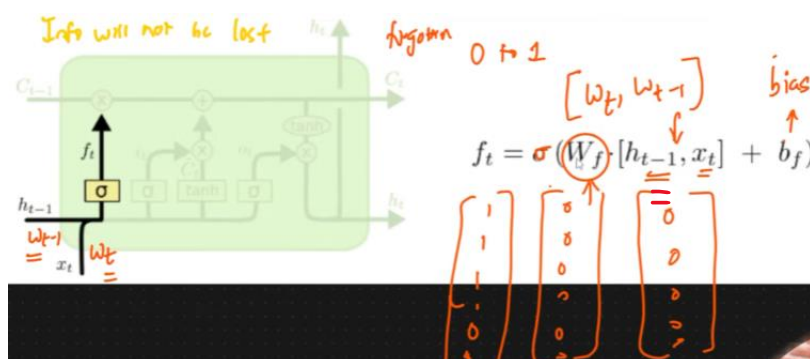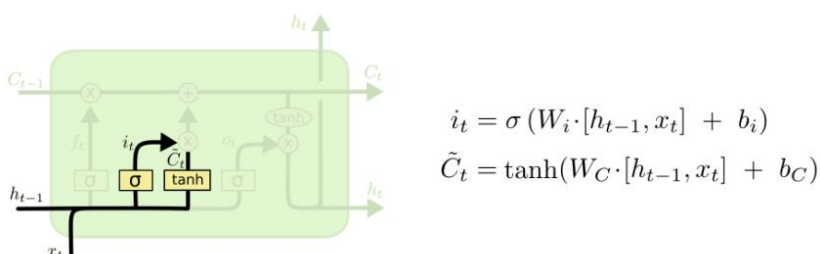

Example -2 with context switching:

Here when we are at friend ther privious context like Krish needs to be removed or forgot as it has not relation with Burger.

Here most of the value after sigmoid will be 0 and after point wise multiplication with previous values the new value will be zero hence context will be forgotten.

Input Gate Layer

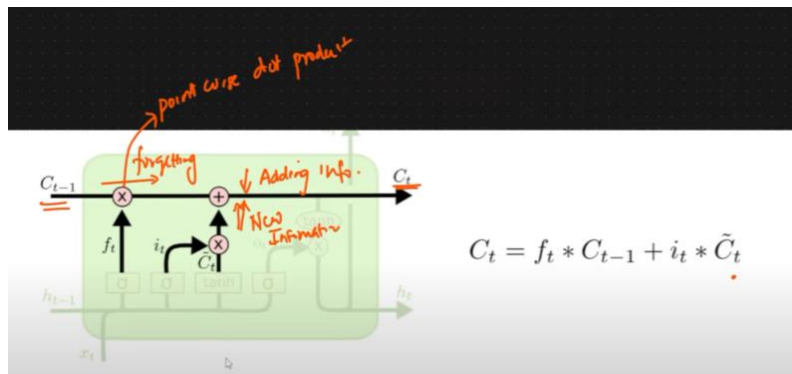Now as we have forgotten the old context we also need to capture the new context in the memory cell.

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

Example:

 Here context switch is happening and we forgot the old context that is say krish like pizza in forgot layer now we also need to capture or remember the new context which is **friend**. So same information with **sigmoid** (0 to 1) activation function will get passed and **tanh** (-1 to 1) activation also will pass then combine that only will pass to memory cell. Thus new context information will be captured.

Combination of sigmoid and tanh will help us to pass only new information or context. So, in input gate layer **we are capturing new information or context**.

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

Output Gated Layer

Now we have to decide what output we are going to pass. Recent info from memory cell will be passed to tanh activation function. Then it will combine with previous forgotten thing with sigmoid function and new output will be passed to next cell. Here all useless data removed and all important data will be passed to next cell.

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$
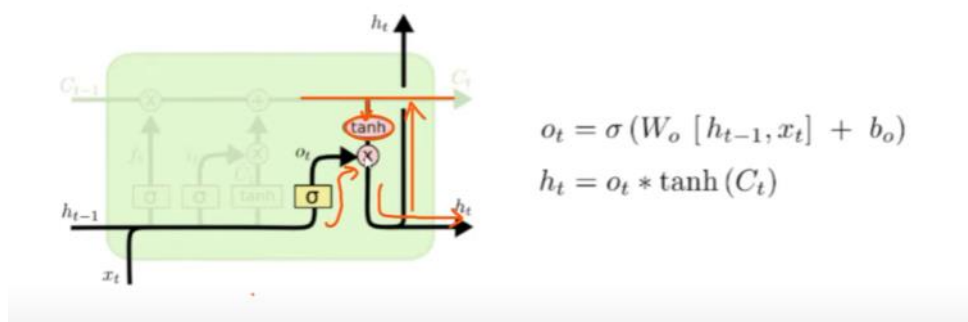
Work Embedding Layer & LSTM Practical Implementation

In NN or LSTM we have a layer called Word Embedding. Word Embedding Layer also converts word to vector. We will see demo like how to train or convert to word2vec using word embedding with TensorFlow and python.

We will be doing it in below steps,

- We will have sentences consisting with words.
- Convert to One Hot Encoding, we have to take vocabulary size this is hyper parameter tuning.
- Padding like pre or post padding
- One Hot Encoded values will be converted to vector.

Code Link

https://colab.research.google.com/drive/1NkJmlUZWK21flAkqsxV86tXbaORRH5ys?authuser=5#scrollTo=-uqVOAju1zTa

LSTM Practical Implementation

Advanced NLP

Bidirectional LSTM RNN

Let's say we have a sentence "I like to eat ___ in Kolkata" here in the sentence my Neural Network should be able to predict the food name.
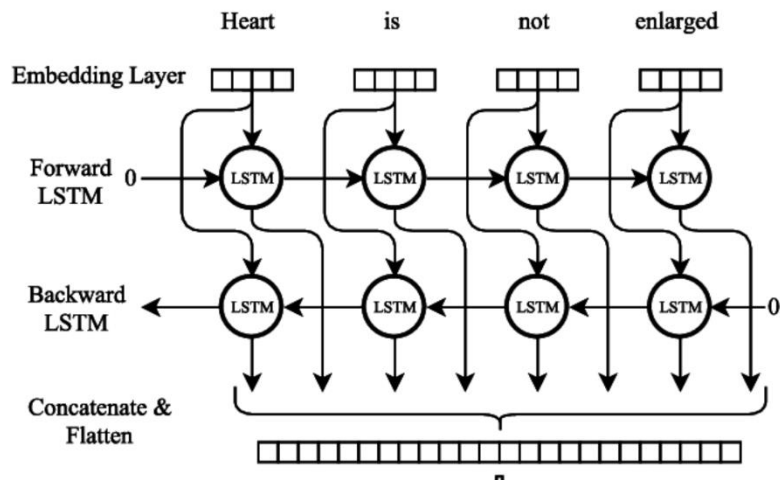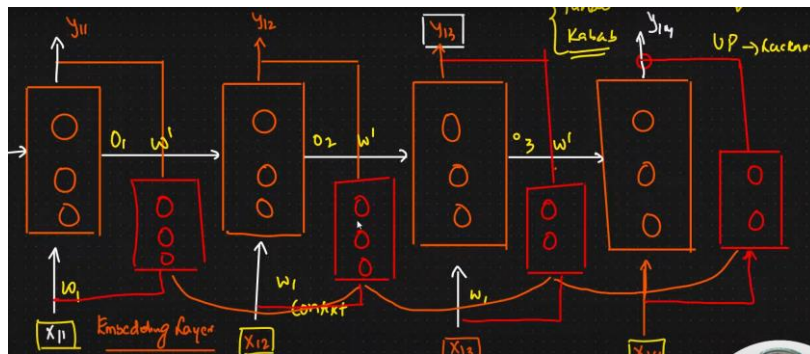
 Let's see a normal LSTM RNN.
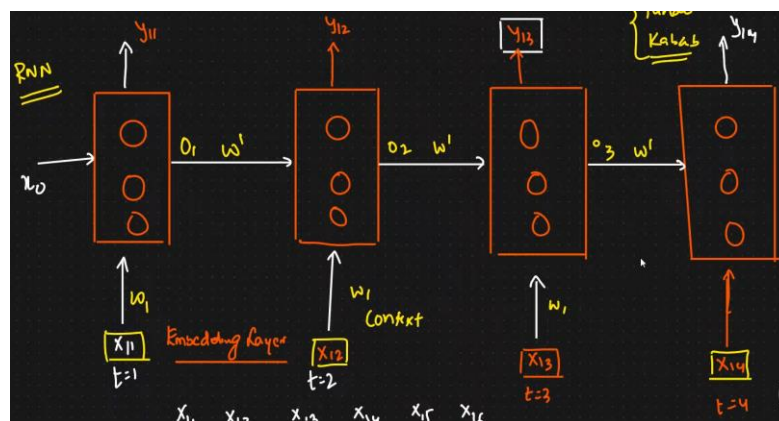
Now say y13 will have the output which is name of the food and y13 has the context of all previous context like x11, x12, x13 which is say I like to eat. However sometimes we may need context of next words as well here say Kolkata to predict the food name.

That's the **main problem with normal one-way LSTM RNN which has only context of previous words not the next words**.

In Bidirectional Neural networks we will have o**ne more neural network in the reverse direction** so that **one word can get context of both previous and next words**.

So bidirectional LSTM RNN means **two-way network forward and reverse** just to give context of previous and next words to each word for better prediction.

Code Link :