1 Background Studies

1.1 Word Embedding

Word embedding simply refers to vector representation of words. In our problem the input is a news article or some words. Now every machine learning model are not capable of processing string or text as input. These models expects vectors or numbers as input. So, transformation of word to vector is a crucial part. There are several techniques to perform this task. But these techniques are of two types.

1. Frequency based
2. Prediction based

1.2 Frequency based Word Embedding

Frequency based Word Embedding techniques mainly focus on the frequency of a word in a text or article to embed a particular word in a vector. There are several such techniques like TF-IDF, Countvectorizer etc.

1.2.1 Countvectorizer

In this technique a whole article is converted to a vector, where the dimension of the vector is the number of unique words in the corpus Countvectorizer simply counts the occurrence of each word in an article and puts this count in the respective field of that word in the vector. Lets consider the following two sentences.

1. Abul is very talented but lazy(D1)
2. Babul is hardworking but not very talented(D2)

So, the unique words in our corpus are
['Abul' , 'Babul' , 'but' , 'is' , 'very' , 'not' , 'talented' , 'lazy' , 'hardworking'] According to Countvectorizer technique the dociments D1 and D2 would be vectorized as following

	Abul	Babul	but	is	very	not	talented	lazy	hardworking
D1	1	0	1	1	1	0	1	1	0
D2	0	1	1	1	1	1	1	0	1

1.2.2 TF-IDF

Here TF stands for 'Term Frequency' and IDF stands for 'Inverse Document Frequency'. It is used in text mining as an weighting factor for features. TF is going to be upweighted by the number of times a term occurs in an article. And IDF is upweighted by the number of times a term occurs in the whole dataset. The equation representing the TF-IDF weight of

a term **t** for a particular document **d** (given the dataset **D** and the number of documents in the dataset **N**) is given by the equation

$$tf - idf(t,d) = tf(t,d) * idf(t,D)$$
$$Here,$$
$$tf(t,d) = \text{Frequency of } t \text{ in } d$$
$$idf(t,D) = log(\frac{N}{|\{d \epsilon D : t \epsilon d\}|})$$

Like Countvectorizer, TF-IDF also converts a whole article or document to a vector. But in stead of frequency count each field contains the TF-IDF value of the word in the document.

1.3 Prediction based Word Embedding or Word2Vec

Word2Vec is the process of transforming words to vectors preserving some of their syntactical and semantic correlations. In CountVectorizer and TF-IDF the vectors of two words are similar or closer to each other based on their occurrence in a particular document and they do not preserve any syntactical and semantic correlations. Word2Vec tries to determine the meaning of a word and understand its correlation with other words by looking at it's context. For example, lets take two sentences "Range Rover is great car." and "Range Rover is a wonderful vehicle.", then Word2vec will map similarities between the words 'great' and 'wonderful' and the words 'car' and 'vehicle'.
Word2Vec uses cosine distance over euclidean distance to measure the similarity or distance between two words. Now, the relation between two words is defined by the vector offset between two words.
Lets take some pair of singular-plural words like 'cat' and 'cats', 'dog' and 'dogs'.

Here the singular-plural relation between the words cat and cats is represented by the cosine difference between the two words V_{cat} and V_{cats} is given by the equation below

$$cosine(V_{cat}, V_{cats}) = \frac{V_{cat} * V_{cats}}{||V_{cat}|| * ||V_{cats}||}$$

And now the word pair dog and dogs have the same singular-plural relationship between them like cat and cats**??**. So according to Word2Vec

$$V_{dog} - V_{dogs} = V_{cat} - V_{cats}$$
$$=> V_{dogs} = V_{cats} - V_{cat} + V_{dog}$$

So, if we know the particular relationship between two words and know one of the words, Word2vec can predict the other word.
Word2Vec is actually a combination of two algorithms or techniques. They are:

1. CBOW(Continuous Bag Of Words)
2. Skip-gram model

1.3.1 CBOW(Continuous Bag Of Words)

CBOW is a neural network with one hidden layer. The aim of CBOW is to predict the next word given a sequence of words. Lets take a sentence 'Maradona is a great footballer'. So, if CBOW is given the word sequence 'Maradona is a great' it will predict the next word 'footballer'. The given word sequence is called context. Now, this context can be a single or multiple words. Lets consider a con text of four words 'Maradona is a great'.
At first the words are encoded into a one hot vector like following.

	Maradona	is	a	great	footballer
Maradona	1	0	0	0	0
is	0	1	0	0	0
a	0	0	1	0	0
great	0	0	0	1	0
footballer	0	0	0	0	1

Now each of these vectors are given as a input field in CBOW model. For this example the CBOW model will look like following.
(image : CBOW.png)
The activation function between input and hidden layer is *linear* and between hidden layer and output layer is *softmax*. A set of context-target pairs are fed to CBOW model. The CBOW model learns the weights between layers like normal neural network. Now the weights between the hidden layer and output layer acts as a vector for the target word.

1.3.2 Skip-gram model

The idea of Skip-gram model is somewhat reverse of CBOW. Given a word Skip-gram model predicts the possible contexts for this word. So the diagram of the model is like the reverse of CBOW model.
(image : SkipGram.png)
The main idea is like CBOW. The words in an article or document are encoded into one hot vectors. One hot vector of one word acts as the input of the model. The contexts are the output. Like CBOW between input layer and hidden layer the activation function is *linear* and between hidden layer and output layer the activation function is *softmax*.
Given a set of word-context pairs what this model learns about a word is the possible contexts this word can be used in. After learning process the weight matrix between the hidden layer and the output layer acts as a lookup table for vector representations of words. If we multiply the encoded one hot vector of a word with this weight matrix we would get the embedded word2vec representation for the word.
Lets take two words 'burger' and 'pizza'. Both of these words are example of fast foods. So, in our corpus these two words would be used in similar contexts. So the cosine distance between their vectors would be small. Whereas the words 'burger' and 'aeroplane' are two different words and they are used in different contexts. So the distance between their vectors would be large.