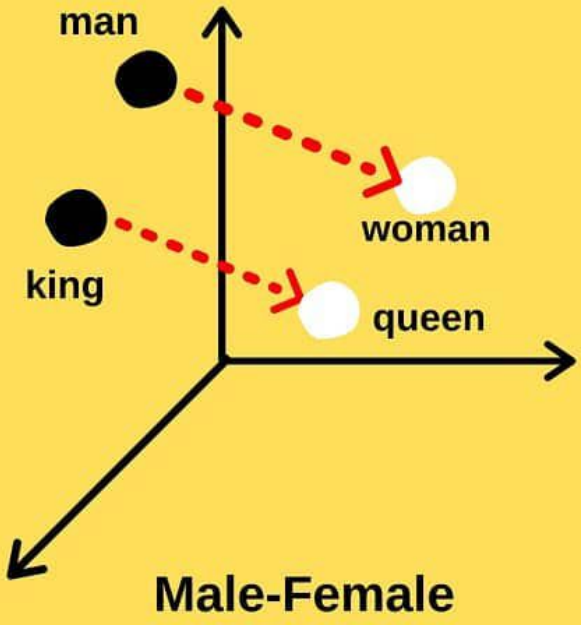
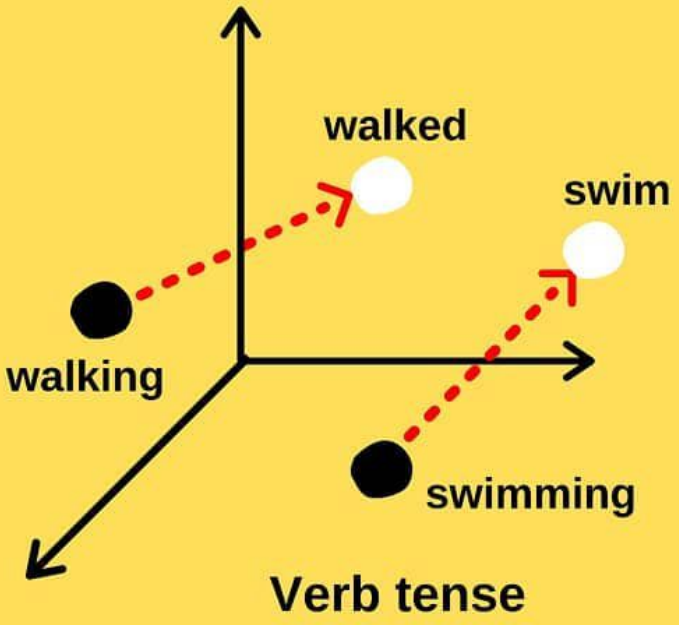
WORD2VEC





BUILD YOUR OWN WORD2VEC EMBEDDINGS

WORD EMBEDDINGS

- It is technique which provides a dense vector representation of words/sentences.
- Word embeddings are an improvement over simpler bag-of-word model word encoding schemes like word counts and frequencies that result in large and sparse vectors

- We train an algorithm with a set of fixed length dense and continious valued vectors to generate word embeddings
- It is defining a word by the company that it keeps that allows the word embedding to learn something about the meaning of words. The vector space representation of the words provides a projection where words with similar meanings are locally clustered within the space.

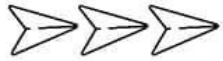
WORD2VEC

- Word embeddings are a modern approach for representing text in natural language processing.
- Word2vec is a type of word embedding algorithm.
- We will be training our own word embedding model using Gensim(python).
- It is one of the most popular technique to learn word embeddings using a two-layer neural network. Its input is a text corpus and its output is a set of

WORD2VEC

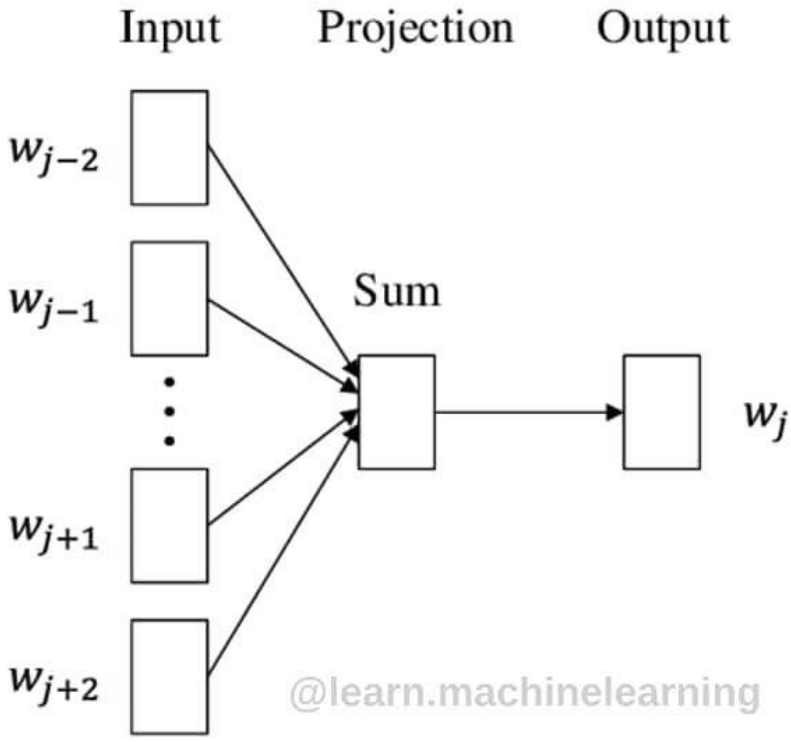
- This is developed by Tomas Mikolov, et al. at Google in 2013. To know more about this you need to have knowledge of neural networks.
- We find that these representations are surprisingly good at capturing syntactic and semantic regularities.
- This allows vector-oriented reasoning based on the offsets between words.

- For example, the male/female relationship is automatically learned, and with the induced vector representations, "King — Man + Woman" results in a vector very close to "Queen".
- This method, we can also achieve state of art tasks like word analogies and word similarities.
- We can build this in 2 ways
  - CBOW (Continuous bag of words)
  - Continuous Skip-Gram

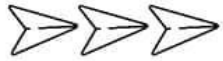


CBOW (CONTINUOUS BAG OF WORDS)

- Here we predict the current word based on neighbouring words (context). The context is defined by the window of neighbouring words. Here Window is a hyperparameter. This Window size plays a major role in defining the vectors.

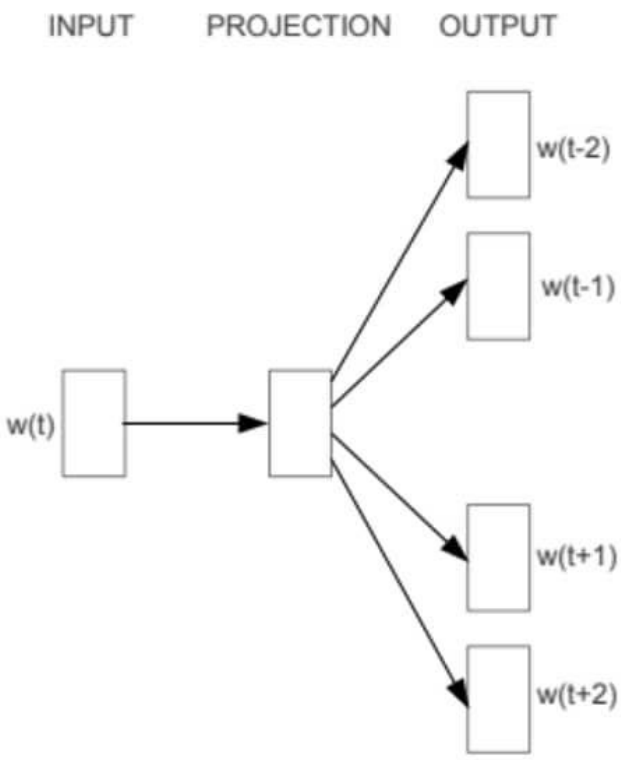


- With small size, we can get more functional and syntactic. If the window size is more we get more topical relations.
- Each input word is represented as one hot encoder vector of total vocabulary size. The output is the predicted word. The weights between the output and the hidden layer are taken as the word vector representation after training.



SKIP-GRAM

- It is a reverse of CBOW. Here we use target word as input to find the context words. The main aim is to predict the whole context when given a word. The weights between the input and the hidden layer are taken as the word vector representation after training. The loss function or the objective is of the same type as of the CBOW model.



Skip-gram

- In these two methods, we get an N dimension vector for each word. N here is the hidden layer units.

ADVANTAGES AND DISADVANTAGES

- Advantages of CBOW
  - Required less memory compared to co-occurrence matrix(will discuss later in this blog)
  - CBOW is faster than skip-gram.
  - It has an advantage of better representations of frequent words.
- Disadvantages of CBOW
  - It cannot perform well to capture the rare words.
  - It takes forever to train if we don't care about the optimization.
  - Here word order doesn't have much influence because of the nature of the algorithm
- Advantages of Skip-Gram
  - Using negative sampling it makes the algorithm work faster.
  - It works well with the small amount of data.
  - It can find representations of rare words.
- Disadvantages of Skip-Gram
  - Runs slower if we don't care about optimization techniques.
  - Can't capture the polysemy.

ENOUGH THEORY LETS WRITE CODE

- Install gensim library

```
pip install --upgrade gensim
```

- I will be using a twitter data of an airline company which i downloaded.
- And i did few data preprocessing steps to clean the data.
- Convert the preprocessed data into word tokens.

  - Example: "i love machine learning"
  - output: [i, love, machine, learning]

ENOUGH THEORY LETS WRITE CODE

- Train model

```
from gensim.models import Word2Vec
data = #list of list of word tokens
model = Word2Vec(data)
```

- summarize vocabulary

```
words = list(model.wv.vocab)
```

- Access vector for one word

```
vector = model['word']
```

- save model

```
model.save('name.bin')
```

FOR MORE CHECK THE LINK IN BIO
FOR COMPLETE CODE AND MORE.....