Assignment No.5

Aim: Implement the Continuous Bag of Words (CBOW) Model. Stages can be:

a. Data preparation
b. Generate training data
c. Train model
d. Output

Objective: Student will learn:

- How Neural networks are useful for CBOW text analysis.
- what a CBOW model is and how it works

Methodology to be used

- word embedding technique

Theory :

What is NLP?

Natural language processing (NLP) is a subfield of computer science and artificial intelligence (AI) that uses machine learning to enable computers to understand and communicate with human language.
NLP enables computers and digital devices to recognize, understand and generate text and speech by combining computational linguistics—the rule-based modeling of human language—together with statistical modeling, machine learning (ML) and deep learning.

How NLP works?

NLP combines the power of computational linguistics together with machine learning algorithms and deep learning. Computational linguistics is a discipline of linguistics that uses data science to analyze language and speech. It includes two main types of analysis: syntactical analysis and semantical analysis. Syntactical analysis determines the meaning of a word, phrase or sentence by parsing the syntax of the words and applying preprogrammed rules of grammar. Semantical analysis uses the syntactic output to draw meaning from the words and interpret their meaning within the sentence structure.

The parsing of words can take one of two forms. *Dependency parsing* looks at the relationships between words, such as identifying nouns and verbs, while *constituency parsing* then builds a parse tree (or syntax tree): a rooted and ordered representation of the syntactic structure of the sentence or string of words. The resulting parse trees underly the functions of language translators and speech recognition. Ideally, this analysis makes the output—either text or speech—understandable to both NLP models and people.

What is Word embedding related to NLP ?

Word embeddings are a way of representing words as vectors in a multi-dimensional space, where the distance and direction between vectors reflect the similarity and relationships among the corresponding words.

The development of embedding to represent text has played a crucial role in advancing natural language processing (NLP) and machine learning (ML) applications. Word embeddings have become integral to tasks such as text classification, sentiment analysis, machine translation and more.

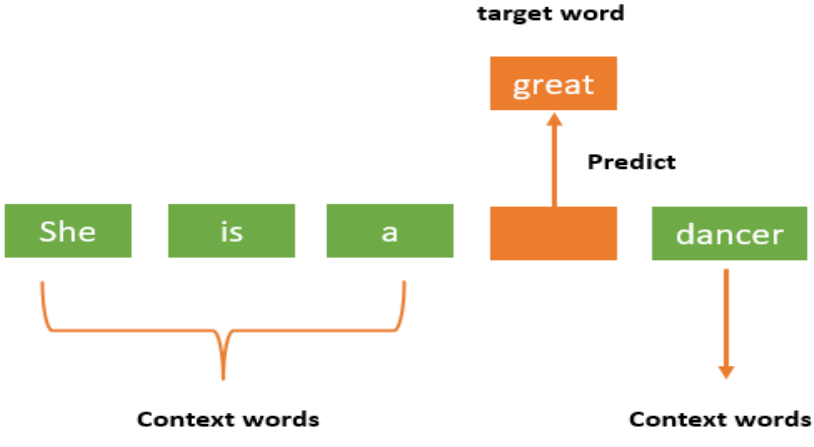
Word embeddings, on the other hand, are dense vectors with continuous values that are trained using machine learning techniques, often based on neural networks. The idea is to learn representations that encode semantic meaning and relationships between words. Word embeddings are trained by exposing a model to a

large amount of text data and adjusting the vector representations based on the context in which words appear.

One popular method for training word embeddings is Word2Vec, which uses a neural network to predict the surrounding words of a target word in a given context. Another widely used approach is GloVe (Global Vectors for Word Representation), which leverages global statistics to create embeddings.

What is a Continuous Bag of Words (CBOW)?

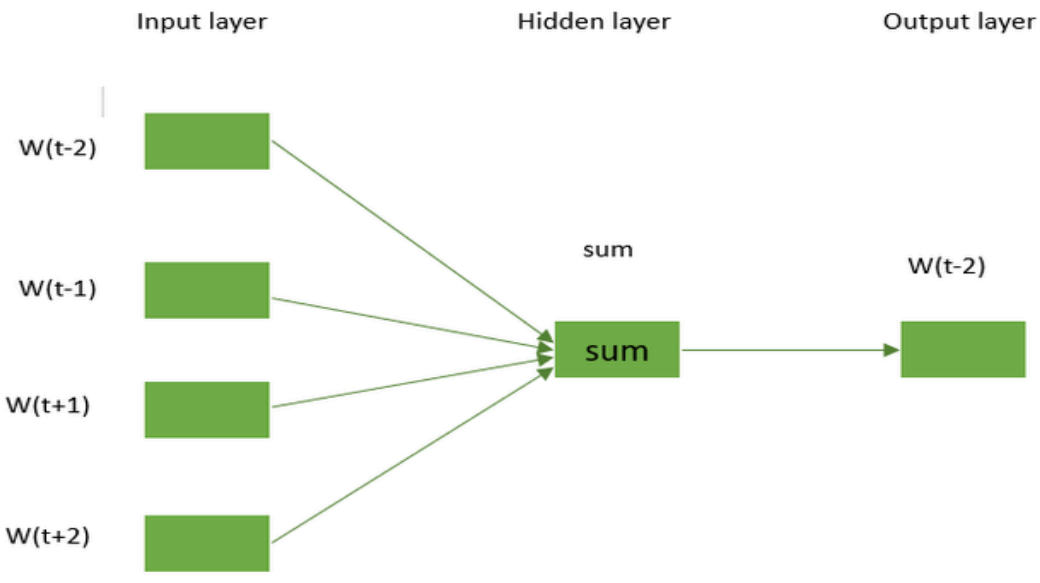
Continuous Bag of Words (CBOW) is a popular natural language processing technique used to generate word embeddings. Word embeddings are important for many NLP tasks because they capture semantic and syntactic relationships between words in a language. CBOW is a neural network-based algorithm that predicts a target word given its surrounding context words. It is a type of "unsupervised" learning, meaning that it can learn from unlabeled data, and it is often used to pre-train word embeddings that can be used for various NLP tasks such as sentiment analysis, text classification, and machine translation.



Explain CBOW architecture.?

The Continuous Bag of Words (CBOW) model is a key architecture in natural language processing used for learning word representations or embeddings. It's part of the family of Word2Vec models developed by Tomas Mikolov and his colleagues. Here's a simplified breakdown of how CBOW works:

The CBOW model uses the target word around the context word in order to predict it. Consider the above example "She is a great dancer." The CBOW model converts this phrase into pairs of context words and target words. The word pairings would appear like this ([she, a], is), ([is, great], a) ([a, dancer], great) having window size=2.



The model considers the context words and tries to predict the target term. The four 1*W input vectors will be passed to the input layer if have four words as context words are used to predict one target word. The hidden layer will receive the input vectors and then multiply them by a W*N matrix. The 1*N output from the hidden layer finally enters the sum layer, where the vectors are element-wise summed before a final activation is carried out and the output is obtained from the output layer.

1. **Objective**: CBOW is designed to predict a target word based on its surrounding context words. The primary goal is to learn word embeddings such that words with similar meanings have similar vector representations.
2. **Architecture**:
    - **Input Layer**: The input consists of context words (words surrounding the target word). These context words are represented as one-hot encoded vectors.
    - **Hidden Layer**: There is a hidden layer with a weight matrix that transforms the one-hot encoded vectors into a continuous vector space. This matrix is essentially the word embedding matrix.
    - **Output Layer**: The output is a probability distribution over all vocabulary words. It uses a softmax function to predict the probability of each word being the target word given the context.

Steps in CBOW

1. **Context Definition**: Define the context window size, which determines how many words on either side of the target word are considered as context. For example, if the context window size is 2, then the two words before and the two words after the target word are used as context.
2. **Training Data Preparation**: For each target word in a sentence, collect the context words according to the window size.
3. **Forward Pass**:
    - Convert context words into one-hot encoded vectors.
    - Multiply these vectors by the embedding matrix to obtain context word embeddings.
    - Average (or sum) these embeddings to get a single vector representing the context.
    - Pass this context vector through the output layer to predict the target word.
4. **Loss Function**: The model is trained using a loss function that measures the difference between the predicted probability distribution and the actual distribution (where the target word has a probability of 1 and all others 0). The common loss function used is the cross-entropy loss.
5. **Backpropagation**: Adjust the weights in the embedding matrix based on the gradient of the loss function with respect to these weights, using techniques like stochastic gradient descent.

What is Tokenizer ?

A tokenizer is a crucial component in natural language processing (NLP) and text analysis that breaks down text into smaller units, such as words or subwords, to facilitate further processing. These units are commonly referred to as tokens. Tokenization is the first step in many NLP tasks and models, including text classification, machine translation, and named entity recognition.

Types of Tokenizers:

1. Word Tokenizer:
    - Function: Splits text into individual words.
    - Example: The sentence "ChatGPT is amazing!" would be tokenized into ["ChatGPT", "is", "amazing", "!"].
2. Subword Tokenizer:
    - Function: Breaks words into smaller subword units or morphemes. This is especially useful for handling unknown words and morphologically rich languages.
    - Example: The word "tokenization" might be split into ["token", "##ization"] (where "##" denotes a subword prefix).
3. Character Tokenizer:
    - Function: Breaks text into individual characters.
    - Example: The word "ChatGPT" would be tokenized into ["C", "h", "a", "t", "G", "P", "T"].
4. Sentence Tokenizer:
    - Function: Divides text into sentences.
    - Example: The text "ChatGPT is amazing. It helps with various tasks." would be tokenized into ["ChatGPT is amazing.", "It helps with various tasks."].

Why is Tokenization Important ?

- Preparation for Modeling: Many NLP models require input data to be in a tokenized format. Tokenization converts raw text into a structured format that can be used by algorithms.
- Standardization: It helps in standardizing text data, making it easier to handle and analyze. For example, tokenization can address issues related to punctuation, capitalization, and special characters.
- Handling Variability: Tokenization helps in dealing with variability in text data, such as different spellings or word forms, by breaking down text into smaller units that are easier to manage.

Tokenization Techniques

- Rule-Based Tokenizers: Use predefined rules and patterns (e.g., regular expressions) to identify token boundaries.
- Statistical Tokenizers: Use statistical models to determine token boundaries based on learned patterns from data.
- Machine Learning-Based Tokenizers: Employ machine learning models to learn the best way to tokenize based on training data. For instance, tokenizers used in models like BERT or GPT use algorithms that have been trained on large corpora to handle tokenization in a more sophisticated manner.

Steps/ Algorithm :

1. Dataset link and libraries :

Create any English 5 to 10 sentence

paragraph as input

Import following data from keras :

keras.models import Sequential

keras.layers import Dense, Embedding, Lambda

keras.utils import np_utils

keras.preprocessing import sequence

keras.preprocessing.text import Tokenizer

Import Gensim for NLP operations : requirements :

Gensim runs on Linux, Windows and Mac OS X, and should run on any other platform that supports Python 3.6+ and NumPy. Gensim depends on the following software: Python, tested with versions 3.6, 3.7 and 3.8. NumPy for number crunching.

Ref:
https://analyticsindiamag.com/the-continuous-bag-of-words-cbow-model-in-nlp - hands-on-implementation-with-codes/

a) Import following libraries gemsim and numpy set i.e. text file created . It should be preprocessed.

b) Tokenize every word from the paragraph . You can call inbuilt tokenizer present inGensim

c) Fit the data to tokenizer

d) Find total no of words and total no of sentences.

e) Generate the pairs of Context words and target words :

e.g.

```
cbow_model(data,window_size,
   total_vocab):total_length =
   window_size*2
   for text in data:
      text_len = len(text)
      for i dx, word in
         enumerate(text):
         context_word = []
         target = []
         begin = idx - window_size
  end = idx + window_size + 1
        context_word.append([text[i] for i in range(begin, end) if 0 <= i < text_len and i
         != idx])

                  target.append(word)

                  contextual = sequence.pad_sequences(context_word, total_length=total_length)
```

```
final_target = np_utils.to_categorical(target, total_vocab)

yield(contextual, final_target)
```

f) Create a Neural Network model with following parameters . Model type : sequential

Layers : Dense , Lambda , embedding. Compile Options : (loss='categorical_crossentropy', optimizer='adam')

g) Create vector file of some

word for

testinge.g.:dimensions=100

```
vect_file = open('/content/gdrive/My Drive/vectors.txt' ,'w')

vect_file.write('{} {}\n'.format(total_vocab,dimensions)
```

h) Assign weights to your trained model

```
e.g. weights = model.get_weights()[0]

for text, i in vectorize.word_index.items():

final_vec = ' '.join(map(str, list(weights[i,

:])))  vect_file.write('{} {}\n'.format(text,

final_vec)

Close()
```

i) Use the vectors created in Gemsim :

e.g. cbow_output = gensim.models.KeyedVectors.load_word2vec_format('/content/gdrive/My Drive/vectors.txt', binary=False)

j) choose the word to get

similar type of words:

```
cbow_output.most_similar(positive=['Your

word'])
```

Conclusion:

In this experiment, we saw what a CBOW model is and how it works. We also implemented the model on a custom dataset and got good output. We learnt what word embeddings are and how CBOW is useful. These can be used for text recognition, speech to text conversion etc.