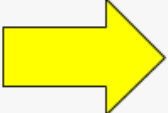


1. Explain One-Hot Encoding

One-Hot Encoding is a technique used to represent categorical variables as binary vectors. It converts categorical data into a binary representation where each category is mapped to a unique binary value. Each category is represented as a vector of zeros, except for the index that corresponds to the category, which is marked as 1.

One hot encoding creates new (binary) columns, indicating the presence of each possible value from the original data. Let's work through an example.



Color	Red	Yellow	Green
Red	1	0	0
Red	1	0	0
Yellow	0	1	0
Green	0	0	1
Yellow	0	0	1

The values in the original data are Red, Yellow and Green. We create a separate column for each possible value. Wherever the original value was Red, we put a 1 in the Red column.

2.Explain Bag of Words

Bag of Words is a commonly used technique in natural language processing (NLP) for representing text data. It involves treating each document or sentence as a "bag" of individual words, **disregarding grammar and word order**. The text is tokenized into words, and a vocabulary is created by considering all unique words in the corpus. The presence or absence of each word in a document is then used to create a numerical representation, typically a sparse vector, where each dimension corresponds to a word in the vocabulary. The frequency of occurrence of each word may also be considered. The resulting representation can be used as input for machine learning algorithms.

Suppose we wanted to vectorize the following:

the cat sat
the cat sat in the hat
the cat with the hat

We'll refer to each of these as a text document.

- We first define our vocabulary, which is the set of all words found in our document set. The only words that are found in the 3 documents above are: the, cat, sat, in, the, hat, and with.
- To vectorize our documents, all we have to do is count how many times each word appears:
- Now we have length-6 vectors for each document!
 - the cat sat: [1, 1, 1, 0, 0, 0]
 - the cat sat in the hat: [2, 1, 1, 1, 1, 0]
 - the cat with the hat: [2, 1, 0, 0, 1, 1]
- The Problem with BOW is, it does not preserve contextual information among the words. Notice that we lose contextual information, e.g. where in the document the word appeared, when we use BOW. It's like a literal bag-of-words: it only tells you what words occur in the document, not where they occurred.

3.Explain Bag of N-Grams.

Bag of N-Grams is an extension of the Bag of Words approach. Instead of considering individual words, **it takes into account sequences of N consecutive words**, known as N-grams. By including N-grams, **the model can capture some contextual information and word order to a certain extent**. For example, in the case of N=2 (bigrams), the phrases "good movie" and "great movie" would be represented differently, providing some context-specific information. The rest of the process, including creating a vocabulary and generating a numerical representation, follows a similar approach to Bag of Words.

4.Explain TF-IDF

TF-IDF stands for Term Frequency-Inverse Document Frequency. It is a **numerical statistic that reflects the importance of a word in a document** within a collection or corpus of documents.

TF-IDF is composed of two terms:

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i$$

Term Frequency (TF): The number of times a word appears in a document divided by the total number of words in that document.

$$tf_{i,j} = \frac{\text{Number of times term } i \text{ appears in document } j}{\text{Total number of terms in document } j}$$

Inverse Document Frequency (IDF): The logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

$$idf_i = \log \left(\frac{\text{Total number of documents}}{\text{Number of documents with term } i \text{ in it}} \right)$$

So, essentially, the TF-IDF value increases as the word's frequency in a document (TF) increases. However, this is offset by the number of times the word appears in the entire collection of documents or corpus (IDF).

Thus TF-IDF takes into account both the frequency of a word in a document (term frequency) and the rarity of the word across the entire corpus (inverse document frequency). A high TF-IDF score is obtained for words that appear frequently within a document but infrequently across the corpus, indicating that they are potentially more informative and relevant to that specific document. TF-IDF is commonly used for text feature extraction and document similarity calculations.

Applications of TF-IDF:

Determining how relevant a word is to a document, or TD-IDF, is useful in many ways, for example:

Information retrieval:

TF-IDF was invented for document search and can be used to deliver results that are most relevant to what you're searching for. Imagine you have a **search engine** and somebody looks for LeBron. The results will be displayed in order of relevance. That's to say the most relevant sports articles will be ranked higher because TF-IDF gives the word LeBron a higher score.

It's likely that every search engine you have ever encountered uses TF-IDF scores in its algorithm.

Keyword Extraction:

TF-IDF is also useful for extracting keywords from text. How? The highest scoring words of a document are the most relevant to that document, and therefore they can be considered keywords for that document. Pretty straightforward.

5.What is OOV problem?

The OOV (Out-Of-Vocabulary) problem refers to the challenge of **encountering words or tokens in a dataset that are not present in the vocabulary or dictionary used by a model**. When a model encounters an OOV word, it cannot effectively process or understand it, leading to difficulties in downstream tasks such as text classification or machine translation.

The OOV problem can arise when the training data does not cover all possible words or when the model is presented with new or unknown words during inference. Addressing the OOV problem typically involves techniques like handling unknown words with a special token or expanding the vocabulary through methods like subword tokenization.

How to fix OOV problem?

There are three main ways that often be used in AI application.

Ignoring them

Generally, words that are out of vocabulary often appear rarely, they will contribute less to our model. The performance of our model will drop scarcely, it means we can ignore them.

Replacing them using <UNK>

We can replace all words that are out of vocabulary by using word <UNK>.

Initializing them by a uniform distribution with range [-0.01, 0.01]

Out-Of-Vocabulary (OOV) words can be initialized from a uniform distribution with range [-0.01, 0.01]. We can use this uniform distribution to train our model.

6.What are word embeddings?

Word embeddings are **dense vector representations of words in a continuous vector space**.

They are created through the application of various techniques, such as neural networks, that learn the semantic relationships between words based on their co-occurrence patterns in large text corpora.

Word embeddings **encode semantic and syntactic information about words, allowing them to capture similarities and relationships between words.** These representations are useful in various NLP tasks, including sentiment analysis, language translation, and information retrieval.

7. Explain Continuous bag of words (CBOW)

Ans: The word2vec model has two different architectures to create the word embeddings. They are:

- Continuous bag of words(CBOW)
- Skip-gram model

Continuous bag of words(CBOW):

Imagine you have a big book and you want to teach a computer to understand the words in that book. CBOW is one way to do that.

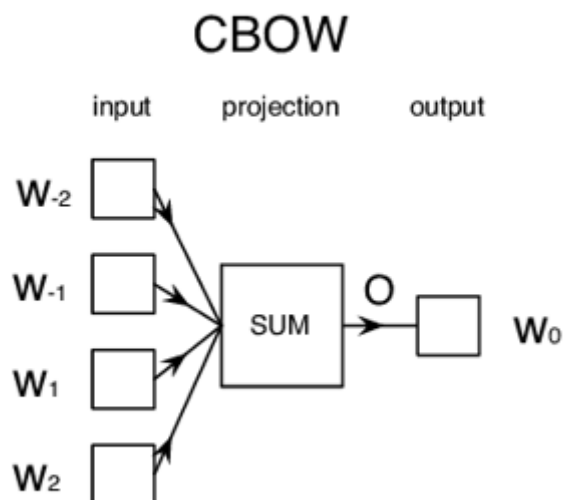
With CBOW, the computer learns by looking at the context words around a target word. The goal is to predict the target word based on its surrounding words. For example, if the context words are "I", "love", and "to", the computer will learn to predict the target word "read."

To train the CBOW model, the computer goes through the book multiple times. Each time it encounters a target word, it looks at the words around it within a certain window. It adds up the vectors of these context words and tries to predict the target word's vector. In this way, it learns to associate the context words with the target word.

Once the model is trained, each word in the book is represented by a unique numerical vector. These vectors capture the meaning and context of the words. Interestingly, these vectors can perform mathematical operations. For example, by adding the vectors for "king" and "woman" and subtracting the vector for "man," you can get a vector that is close to the word "queen." This shows how the CBOW model captures semantic relationships between words.

In simpler terms, CBOW helps the computer understand words by learning from the context in which they appear. It creates word representations that can be used to measure similarities between words, find related words, or even perform language tasks like text completion or sentiment analysis.

Overall, CBOW is a useful technique for building word embeddings that can enhance natural language processing tasks and enable computers to understand and work with words more effectively.



The Model Architecture:

The CBOW model tries to predict the target word by trying to understand the context of the surrounding words. Consider the same sentence as above, 'It is a pleasant day'. The model converts this sentence into word pairs in the form (context word, target word). The user will have to set the window size. If the window for the context word is 2 then the word pairs would look like this: ([it, a], is), ([is, pleasant], a), ([a, day], pleasant). With these word pairs, the model tries to predict the target word considered the context words.

If we have 4 context words used for predicting one target word the input layer will be in the form of four $1 \times W$ input vectors. These input vectors will be passed to the hidden layer where it is multiplied by a $W \times N$ matrix. Finally, the $1 \times N$ output from the hidden layer enters the sum layer where an element-wise summation is performed on the vectors before a final activation is performed and the output is obtained.

8. Explain SkipGram

Ans: Imagine you have a big book and you want to teach a computer to understand the words in that book. SkipGram is one way to do that.

With SkipGram, the computer learns by looking at each word in the book and trying to predict the words that are likely to appear around it. For example, if the word "cat" often

appears with "meow" and "purr," the computer will learn that there is a connection between these words.

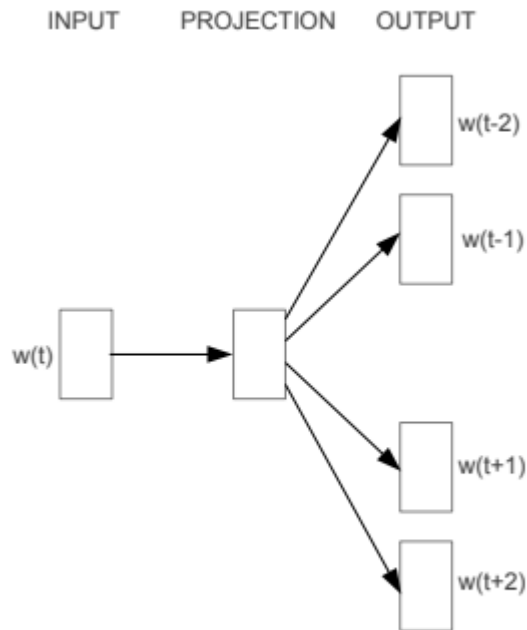
To train the SkipGram model, the computer goes through the book multiple times. Each time it sees a word, it tries to predict the words that come before and after it within a certain window of words. By doing this, it learns the relationships between words based on their co-occurrence patterns in the text.

Once the model is trained, each word in the book is represented by a unique numerical vector. These vectors capture the meaning and context of the words. Interestingly, these vectors can perform mathematical operations. For example, by adding the vector for "king" to the vector for "woman" and subtracting the vector for "man," you can get a vector that is close to the word "queen." This shows how the SkipGram model captures semantic relationships between words.

In simpler terms, SkipGram helps the computer understand words by learning from the context in which they appear.

It creates word representations that can be used to measure similarities between words, find related words, or even perform language tasks like text completion or sentiment analysis.

The Skip-gram Model: The Skip-gram model architecture usually tries to achieve the reverse of what the CBOW model does. It tries to **predict the source context words (surrounding words)** given a target word (the center word). Considering our simple sentence "the quick brown fox jumps over the lazy dog". If we used the CBOW model, we get pairs of (context_window, target_word) where if we consider a context window of size 2, we have examples like ([quick, fox], brown), ([the, brown], quick), ([the, dog], lazy) and so on. Now considering that the skip-gram model's aim is to predict the context from the target word, the model typically inverts the contexts and targets, and tries to predict each context word from its target word. Hence the task becomes to predict the context [quick, fox] given target word 'brown' or [the, brown] given target word 'quick' and so on. Thus the model tries to predict the context_window words based on the target_word.



Skip-gram

Just like we discussed in the CBOW model, we need to model this Skip-gram architecture now as a deep learning classification model such that we take in the target word as our input and try to predict the context words. This becomes slightly complex since we have multiple words in our context. We simplify this further by breaking down each (target, context_words) pair into (target, context) pairs such that each context consists of only one word. Hence our dataset from earlier gets transformed into pairs like (brown, quick), (brown, fox), (quick, the), (quick, brown) and so on. But how to supervise or train the model to know what is contextual and what is not?

For this, we feed our skip-gram model pairs of (X, Y) where X is our input and Y is our label. We do this by using [(target, context), 1] pairs as positive input samples where target is our word of interest and context is a context word occurring near the target word and the positive label 1 indicates this is a contextually relevant pair. We also feed in [(target, random), 0] pairs as negative input samples where target is again our word of interest but random is just a randomly selected word from our vocabulary which has no context or association with our target word. Hence the negative label 0 indicates this is a contextually irrelevant pair. We do this so that the model can then learn which pairs of words are contextually relevant and which are not and generate similar embeddings for semantically similar words.

9. Explain Glove Embeddings.

Ans: GloVe (Global Vectors for Word Representation) embeddings are a way to represent words in a numerical format. These embeddings capture the meaning and relationships between words.

GloVe embeddings are created by analyzing a large collection of text, like a big book or a vast amount of articles. The idea is to understand how often certain words appear near each other.

For example, if the word "cat" frequently appears near the word "meow," it suggests a relationship between these two words.

By analyzing the co-occurrence patterns of words, GloVe embeddings assign each word a unique numerical vector. These vectors encode information about the word's meaning and its relationship to other words in the text.

The interesting thing about GloVe embeddings is that they can perform mathematical operations on the word vectors. For example, by subtracting the vector for "king" from "queen" and adding the vector for "woman," you get a vector that is close to the word "man." This showcases how the embeddings capture semantic relationships between words.

Overall, GloVe embeddings are useful because they enable computers to understand and work with words in a more meaningful way. They are used in various tasks like language translation, sentiment analysis, and text classification to enhance the performance of natural language processing models.