Natural Language Processing

One-Hot encoding:

It means assigning a unique binary vector to each word, where only one element is one (indicating the presence of the word) and all other elements are 0. It does not capture the semantic relationship between the words.

Ex:

["my","name","is","Daniel"]

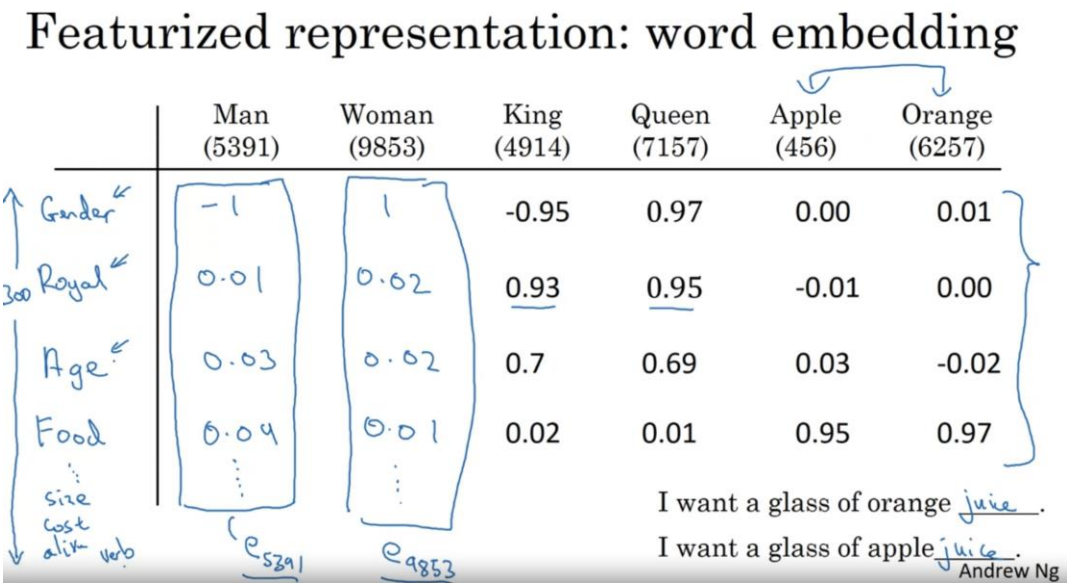
my = [1,0,0,0]

name = [0,1,0,0]

is = [0,0,1,0]

Daniel = [0,0,0,1]

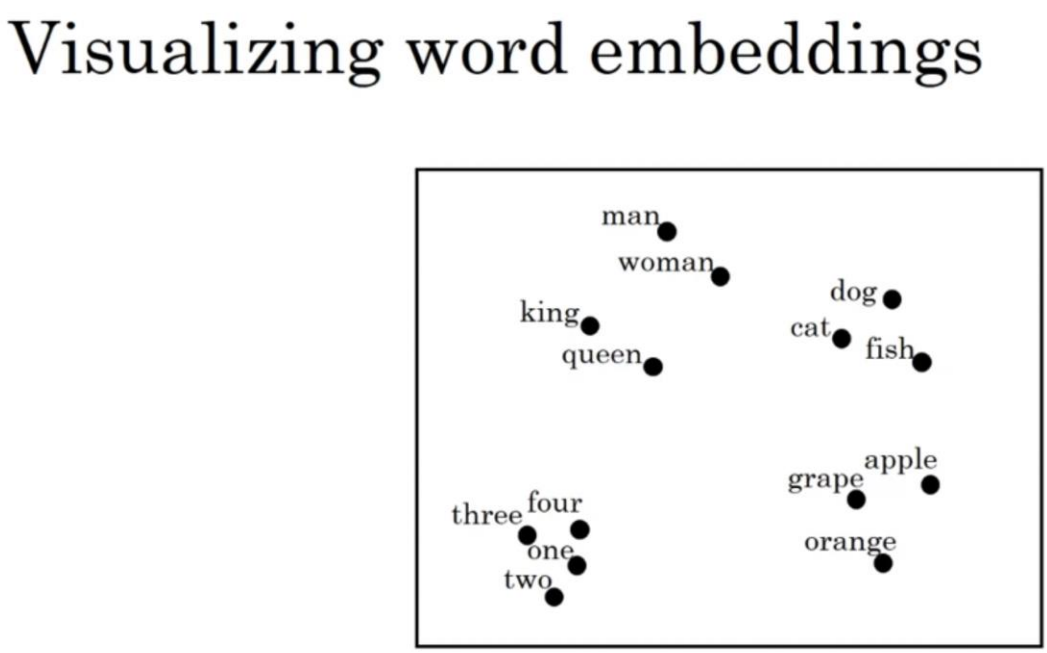
Featurized representation: Word Embedding



In the above table, the man is represented as the vector of all these feature values. Suppose let's say we have taken 300 features into consideration, then the man vector will be the values of all those features represented in a single vector.

Here, we have taken different features like gender, royal, age, food, size, cost, etc... we can take many more features like that (for example, we can take 300 features). We will rate each word with respect to those features and based on that we will group the similar words together.

Since we have taken 300 features, it will give us the 300-Dimensional vector space in which the words are grouped together based on the similarity. We need to convert them to 2D for visual representation of grouped words.



Transferring the learning and word embeddings:

1. Step 1 is to learn word embeddings from a large text corpus, a very large text corpus or you can also download pre-trained word embeddings online.
2. Step2 is that you can then take these word embeddings and transfer the embedding to new task, where you have a much smaller labeled training sets.
3. Can continue to finetune the word embeddings with new data.

We can now use relatively lower dimensional feature vectors. So rather than using a 10,000 dimensional one-hot vector, you can now instead use maybe a 300-dimensional dense vector. Although the one-hot vector is fast and the 300-dimensional vector that you might learn for your embedding will be a dense vector.

Analogies using Word Embeddings:

Used for finding exploring semantic relationship between words.

Classic Examples:

King = Man, woman = ?

In this, we will find out by

King – Man + woman = Queen

Here, the operation involves subtracting the vector (Featurized vector: word embedding) for “Man” for the vector for “King” to remove the male gender component and adding the vector for “Woman” to represent the female gender component, resulting in a vector that closely corresponds to “Queen”.

Similar way,

- Cats – cat + dog = dogs (plurals and singulars)
- running – run + walk = walking (word tenses)
- Paris – France + Italy = Rome (country and capital)
- hot – cold + warm = warm (opposites)

Embedding Matrix:

Let's say, as usual we're using our 10,000-word vocabulary. So, the vocabulary has A, Aaron, Orange, Zulu, maybe also unknown word as a token. We're going to learn embedding matrix E, which will be a 300-dimensional by 10,000-dimensional matrix.

King, queen, princess, boy are the words of the 10,000-word vocabulary

Royal, Male, Female, Age are the features that we use in 300-Dimensional vector.

Let the word Boy be 6257th word of the vocabulary and E be the embedding matrix

Then,

For E6257 we're going to use to represent the embedding vector that 300 by one dimensional vector for the word Boy.

Bag-of-Words (BOW):

Bag-of-Words is a simple and commonly used technique for converting text data into numerical feature vectors. The basic idea behind BOW is to represent a document as an unordered collection of words, ignoring grammar and word order. Here's how it works:

Tokenization: First, the text is tokenized, breaking it into individual words or tokens.

Vocabulary Creation: Next, a fixed-size vocabulary is created from all unique words in the corpus. Each word is assigned a unique index.

Vectorization: For each document, a feature vector is created where each element represents the frequency of a word from the vocabulary in that document. This can be done by counting the occurrences of each word in the document.

Example: Suppose we have a corpus consisting of three documents: "I love coding," "Coding is fun," and "I code every day." The vocabulary might be ["I", "love", "coding", "is", "fun", "every". "day"]. The BOW representation of the first document would be [1, 1, 1, 0, 0, 0, 0), indicating the counts of each word in the document.

Normalization: Optionally, the counts can be normalized by dividing by the total number of words in the document or using techniques like JF to adjust for the document length and word frequency.

BOW representations are simple and easy to compute, but they do not capture the semantic relationships between words and may not perform well with large vocabularies or sparse data.

Term Frequency-Inverse Document Frequency (TF-IDF):

TF-IDF is a more advanced technique that aims to address some of the limitations of BOW, particularly regarding word importance. It measures the importance of a word in a document relative to a corpus of documents. Here's how it works:

Term Frequency (TF): This measures the frequency of a term (word) in a document. It's similar to BOW, where each element in the feature vector represents the frequency of a word in the document.

Inverse Document Frequency (IDF): This measures how important a term is across the entire corpus. Words that occur frequently in a document but rarely in the corpus are given higher weights.

The formula for IDF is:

IDF(t) = log (N/df(t))

where N is the total number of documents in the corpus, and df(t) is the number of documents containing term f Words that occur in many documents have lower IDF scores because they are less discriminative or informative.

TF-IDF Calculation: TF-IDF is calculated as the product of TF and IDF. This results in a numerical representation of each term in the document that reflects both its frequency in the document and its importance in the corpus.

Example: Using the same example corpus as before, the TF-IDF representation of the first document might be [0.477, 0.477, 0.477, 0, 0, 0, 01, with values calculated based on both the term frequency and inverse document frequency.

TF-IDF representations tend to give more weight to important words while downplaying common words that appear in many documents. This can help capture the essence of a document more effectively than simple BOW representations.

In summary, BOW and TF-IDF are both techniques for representing text data numerically, with TF- IDF offering a more sophisticated approach that considers the importance of words in the context of a document corpus. They are widely used in NLP tasks, depending on the task's specific requirements and the data's characteristics.

let's illustrate this with a simple example using a small corpus consisting of three documents:

1. Document 1: "The cat sat on the mat."

2. Document 2: "The dog chased the cat."

3. Document 3: "The dog and the cat are friends."

Now, let's calculate the TF-IDF scores for the terms "cat" and "dog" in each document:

Term Frequency (TF):

In Document 1, "cat" appears once.

In Document 2, "cat" appears once.

In Document 3, "cat" appears once.

In Document 1, "dog" appears zero times.

In Document 2, "dog" appears once.

In Document 3, "dog" appears twice.

Inverse Document Frequency (IDF):

* Since we have three documents in the corpus, IDF for both "cat" and "dog" is log (3/df(t))

TF-IDF Score:

• For "cat" in Document 1: TF=1, IDF = log (3/3) = log (1) =0, so TF-IDF=0.

For "cat" in Document 2: TF=1, IDF = log (3/3) = log (1) = 0 so TF-IDF=0

• For "cat" in Document 3: TF=1, IDF=log| (3/3) =log (1) =0, so TF-IDF=0

• For "dog" in Document 1:TF=0, so ~ TF-IDF=0.

• For "dog" in Document 2:TF=1, IDF =log (3/1) =log (3), so TF-IDF=1*log (3)

• For "dog" in Document 3:TF=2, IDF =log (3/1) = log (3) so TF-IDF=2 * log (3)

So, in this example, the term "dog" has higher TF-IDF scores in Documents 2 and 3 because it appears frequently within those documents but rarely in the corpus overall. Conversely, the term "cat" has TF-IDF scores of 0 in all documents because it appears with the same frequency in all documents and thus doesn't contribute to document distinctiveness.

Problems with BOW and TF-IDF :

- In both BOW and TF-IDF approach semantic relationship is not stored.TF-IDF gives importance to uncommon words.
- There is a chance of overfitting.

Word2Vec:

In this specific model, each word is basically represented as a vector of 32 or more dimensions instead of a single number.

Here, the semantic information and relationship between different words are also reserved.

Word2Vec is a popular technique in natural language processing (NLP) for generating distributed representations of words in a continuous vector space. It was introduced by researchers at Google in 2013, notably Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. The key idea behind Word2Vec is to represent words as dense vectors, where words with similar meanings or contexts are mapped to nearby points in the vector space.

There are two main algorithms for training Word2Vec models: Continuous Bag of Words (CBOW) and Skip-gram.

- Continuous Bag of Words (CBOW): In this approach, the model predicts the current word based on the context words within a fixed-size window. The input to the model is a set of context words, and the output is the target word.

- Skip-gram: This is the opposite of CBOW. Given a target word, the model aims to predict the surrounding context words. It takes the target word as input and tries to predict the context words.

Word2Vec models are trained on large corpora of text using neural networks. Once trained, these models can be used to compute similarity between words, find words with similar meanings, or even perform tasks such as word analogy.

Word2Vec has been widely used in various NLP applications, including sentiment analysis, named entity recognition, machine translation, document clustering, and recommendation systems. It has proven to be effective in capturing semantic relationships between words and is a fundamental component in many modern NLP pipelines.