Understanding Word Embeddings

Minor in AI - IIT ROPAR

18th March, 2025

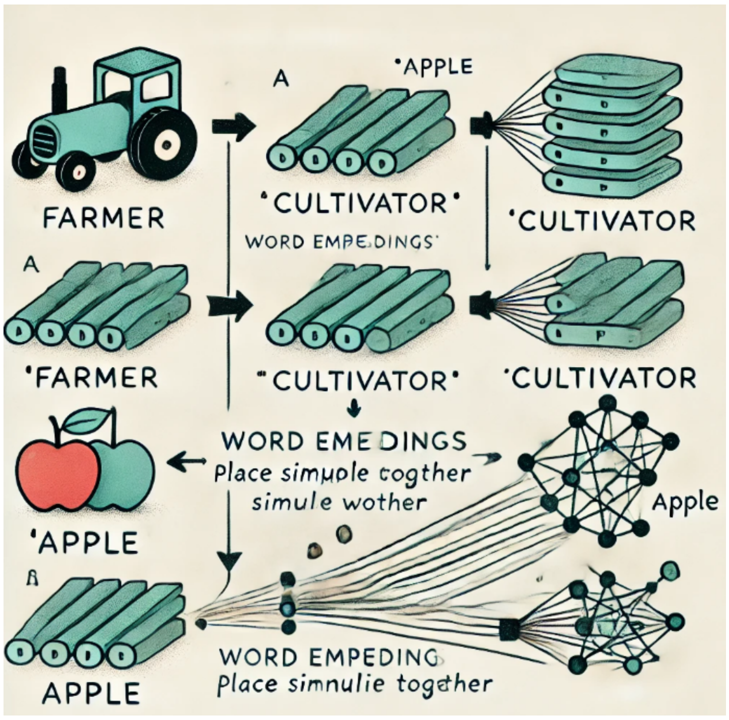
The Journey of Understanding Word Embeddings

Imagine you're in a small village, and the language spoken in this village is very simplistic. At first, there are only a few words that the villagers know, and each word has a distinct meaning. Over time, more complex concepts emerge, and people start talking about objects and actions that relate to one another. For example, "farmer" and "cultivator" mean nearly the same thing, but "farmer" and "apple" are two completely different things. At first, the villagers can only represent these words as unique symbols or codes, which is like the process of one-hot encoding, where each word has a unique identifier.

However, over time, the villagers realize that "farmer" and "cultivator" are very much related, while "farmer" and "apple" are not. This observation leads them to develop a better way of representing words: a representation that shows how words are related based on their meaning.

This breakthrough leads to a new representation system called **word embeddings**. With word embeddings, words with similar meanings, like "farmer" and "cultivator," will be closer in the vector space, while unrelated words, like "farmer" and "apple," will be farther apart. So instead of just assigning random codes to words, this system represents words as vectors that capture their semantic meaning based on context.

As more and more words are processed, and relationships between them are captured, the villagers develop a much more powerful system of understanding language, where machines can not only represent words numerically but also understand the relationships and meanings behind them.



What Are Word Embeddings?

Word embeddings are a technique in Natural Language Processing (NLP) where words or phrases are mapped to vectors of real numbers in a continuous vector space. Unlike traditional methods like one-hot encoding, which represent words as binary vectors, word embeddings capture the meaning of words by positioning them in a vector space where semantically similar words are located closer together.

For example:

- "King" and "queen" are related in terms of royalty, so their vectors would be closer in space.
- "King" and "apple" are unrelated, so their vectors would be farther apart.

Word embeddings are crucial because they allow machines to understand and perform tasks like sentiment analysis, machine translation, and question answering with greater accuracy.

Why Not Just Use One-Hot Encoding?

One-hot encoding was the simplest method used for word representation. In one-hot encoding, each word in a vocabulary is represented as a vector where one element is 1 (the index of the word) and all other elements are 0. For example, if we have a vocabulary size of 5 and the word "apple" is the third word, its one-hot encoding might look like:

$$[\text{"apple"}] \rightarrow [0, 0, 1, 0, 0]$$

This seems simple enough, but there's a major flaw. One-hot vectors don't capture any semantic relationships between words. The one-hot vector for "apple" is the same as for "banana" or "dog," even though "apple" and "banana" are closely related. The one-hot encoding for both words is orthogonal and does not reflect any real-world similarity.

For instance:

- "Farmer" and "Cultivator": Both are related terms, but their one-hot encodings will look completely different, leading to the conclusion that there is no similarity between them.
- "Farmer" and "Apple": These two words are semantically unrelated, but one-hot encoding will also fail to capture this.

So, we needed something better—a technique that could capture word relationships in a more meaningful way. This is where word embeddings come in.

Word Embeddings: Technical Breakdown

1. Word Embeddings as Dense Vectors

Word embeddings represent words as dense vectors in a continuous vector space. Instead of using a sparse vector with just one "1" and the rest "0"s (like one-hot encoding), we represent words with a dense vector where every element holds a real-valued number.

Let's imagine that the word "king" is represented by a 3-dimensional vector, like so:

$$\text{"king"} \rightarrow [0.5, 0.2, -0.3]$$

This vector represents ”king” in a 3-dimensional space. However, in real-world applications, word embeddings typically use vectors of 100-1000 dimensions. These vectors capture semantic properties of words.

2. The Power of Similarity: Cosine Similarity and Dot Product

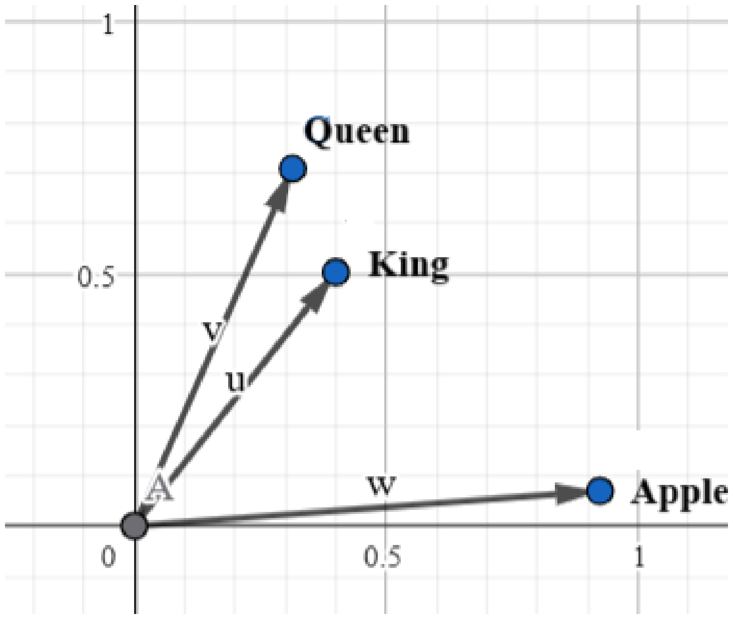
One of the fundamental ideas in word embeddings is that words that are close in meaning should have similar vector representations. To quantify the relationship between two vectors, we use the dot product or cosine similarity.

For example, if we have the following vectors:

$$\text{"king"} \rightarrow [0.4, 0.5, -0.1]$$
$$\text{"queen"} \rightarrow [0.3, 0.7, 0.2]$$

The dot product (or cosine similarity) of these two vectors will yield a non-zero value, indicating that ”king” and ”queen” are related. The closer the vectors are, the higher the cosine similarity.

For vectors that are farther apart (e.g., ”king” and ”apple”), the cosine similarity will be close to zero.



3. Skip-Gram and CBOW (Continuous Bag of Words)

Two common models used for learning word embeddings are Skip-Gram and CBOW (Continuous Bag of Words).

- **Skip-Gram Model**: This model takes a word as input and tries to predict the surrounding context words. For example, in the sentence “I want to drink a cup of coffee,” the word “drink” might be used to predict words like “I,” “want,” “to,” “a,” “cup,” “coffee.”
- **CBOW Model**: In contrast, CBOW takes a context of words as input and tries to predict the target word. For example, for the context {“I,” “want,” “to,” “drink”}, CBOW will predict the word ”coffee”.

Both models work by mapping words to a dense vector and then optimizing the embeddings based on the prediction task using backpropagation.

4. Mathematical Formulation

Given the simplicity of the Skip-Gram or CBOW architecture, the goal is to predict the surrounding words in the context from a given word. This is typically formulated as a softmax function, where we compute the probability of a target word given a context:

$$P(w_{target}|w_{context}) = \text{softmax}(w_{target} \cdot w_{context})$$

Where:

- w_{target} is the vector representing the target word.
- $w_{context}$ is the vector representing the context.

The dot product of these vectors is passed through a softmax function to get a probability distribution. The model is then trained to minimize the difference between the predicted word and the actual word using a loss function like cross-entropy loss.

5. Hierarchical Softmax and Negative Sampling

- **Hierarchical Softmax**: The softmax function becomes computationally expensive as the vocabulary size grows, because it requires summing over all the words in the vocabulary. To overcome this, hierarchical softmax organizes the words into a binary tree and reduces the number of calculations needed.
- **Negative Sampling**: Negative sampling is another technique to optimize training. Instead of updating the weights for every word in the vocabulary, the model only updates the weights for the target word and a few random words (negative examples), making training much faster.

6. Visualizing Word Embeddings

To understand the relationships between words, we can visualize word embeddings in a 2D or 3D space. t-SNE (t-distributed Stochastic Neighbor Embedding) is often used to reduce the dimensions of the word vectors so that we can plot them.

For example, when visualizing a large set of word embeddings, we might see that words related to royalty like ”king” and ”queen” are placed close together, while words related to animals like ”dog” and ”cat” form another cluster.

7. Applications of Word Embeddings

- **Sentiment Analysis**: Word embeddings help capture the sentiment of text by understanding the meaning of words in the context of sentences.
- **Machine Translation**: In machine translation, word embeddings can map words from one language to another by representing them in a shared vector space.
- **Question Answering**: Word embeddings can help machines understand the relationships between words in questions and answers, enabling better responses.
- **Named Entity Recognition (NER)**: Word embeddings can assist in recognizing entities like names, places, or organizations in text, as they understand the context in which these words appear.

8. Transfer Learning and Fine-Tuning Word Embeddings

Word embeddings can be trained on large corpora (like Wikipedia or Project Gutenberg) and then fine-tuned for specific tasks. This is called transfer learning. You can take pre-trained embeddings (like Word2Vec or GloVe) and apply them to new tasks with your own data without needing to train them from scratch.

For example, fine-tuning a word embedding trained on general text can make it more specific to a domain, like medical or legal language, by training it further on domain-specific text.

Key Takeaways

- **Word embeddings** are a technique in NLP where words are mapped to vectors of real numbers in a continuous vector space.
- Unlike **one-hot encoding**, word embeddings capture the meaning of words by positioning similar words closer together in the vector space.
- Word embeddings enable tasks like **sentiment analysis**, **machine translation**, and **question answering** with greater accuracy.
- **One-hot encoding** assigns each word a unique index in a vector, but it does not capture semantic relationships between words.
- **Word embeddings** represent words as **dense vectors** in a continuous vector space, where each element is a real-valued number.
- **Cosine similarity** and **dot product** are used to measure the relationship between word vectors.
- **Skip-Gram** and **CBOW (Continuous Bag of Words)** are common models used to learn word embeddings.
- Word embeddings are typically learned using a **softmax function** to predict a target word given a context.
- **Hierarchical softmax** and **negative sampling** optimize training by reducing the computational cost and focusing on relevant words.
- Word embeddings can be visualized using techniques like **t-SNE** to reveal clusters of related words.
- Applications of word embeddings include **sentiment analysis**, **machine translation**, **question answering**, and **named entity recognition (NER)**.
- **Transfer learning** allows pre-trained word embeddings to be fine-tuned for specific tasks, making them adaptable to different domains.