Search Medium Write

Member-only story

Word2Vec, GloVe, and FastText, Explained

How computers understand words

Ajay Halthor · Follow
Published in Towards Data Science · 10 min read · Jun 20

188 4

Photo by Growtika on Unsplash

Computers don’t understand words like we do. They prefer to work with numbers. So, to help computers understand words and their meanings, we use something called embeddings. These embeddings numerically represent words as mathematical vectors.

The cool thing about these embeddings is that if we learn them properly, words that have similar meanings will have similar numeric values. In other words, their numbers will be closer to each other. This allows computers to grasp the connections and similarities between different words based on their numeric representations.

One prominent method for learning word embeddings is Word2Vec. In this article, we will delve into the intricacies of Word2Vec and explore its various architectures and variants.

Word2Vec

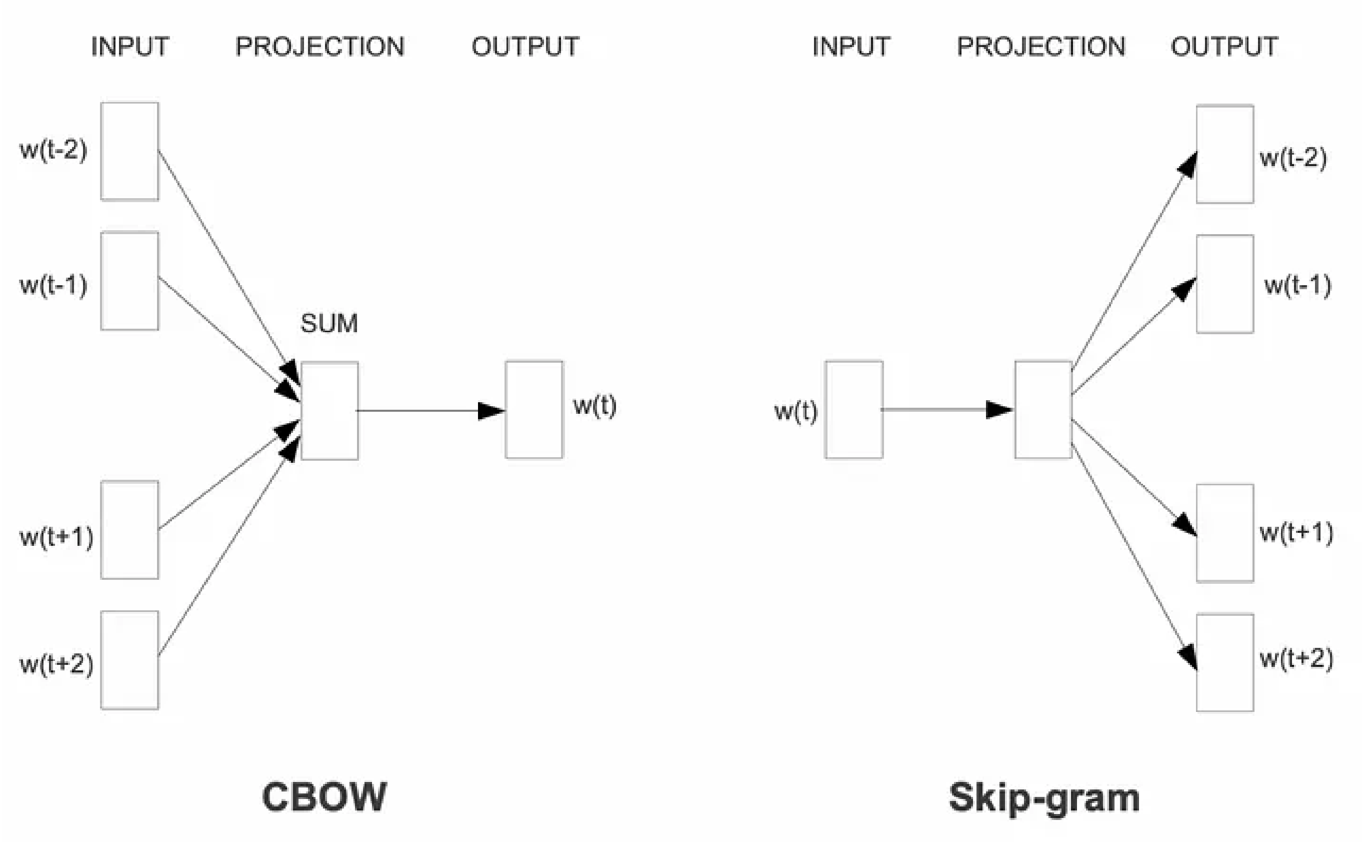


Figure 1: Word2Vec architectures (Source)

In the early days, sentences were represented with n-gram vectors. These vectors aimed to capture the essence of a sentence by considering sequences of words. However, they had some limitations. N-gram vectors were often large and sparse, which made them computationally challenging to create. This created a problem known as the curse of dimensionality. Essentially, it meant that in high-dimensional spaces, the vectors representing words were so far apart that it became difficult to determine which words were truly similar.

Then, in 2003, a remarkable breakthrough occurred with the introduction of a neural probabilistic language model. This model completely changed how we represent words by using something called continuous dense vectors. Unlike n-gram vectors, which were discrete and sparse, these dense vectors offered a continuous representation. Even small changes to these vectors resulted in meaningful representations, although they might not directly correspond to specific English words.

Building upon this exciting progress, the Word2Vec framework emerged in 2013. It presented a powerful method for encoding word meanings into continuous dense vectors. Within Word2Vec, two primary architectures were introduced: Continuous Bag of Words (CBoW) and Skip-gram.

These architectures opened doors to efficient training models capable of generating high-quality word embeddings. By leveraging vast amounts of text data, Word2Vec brought words to life in the numeric world. This enabled computers to understand the contextual meanings and relationships between words, offering a transformative approach to natural language processing.

Continuous Bag-of-Words (CBoW)

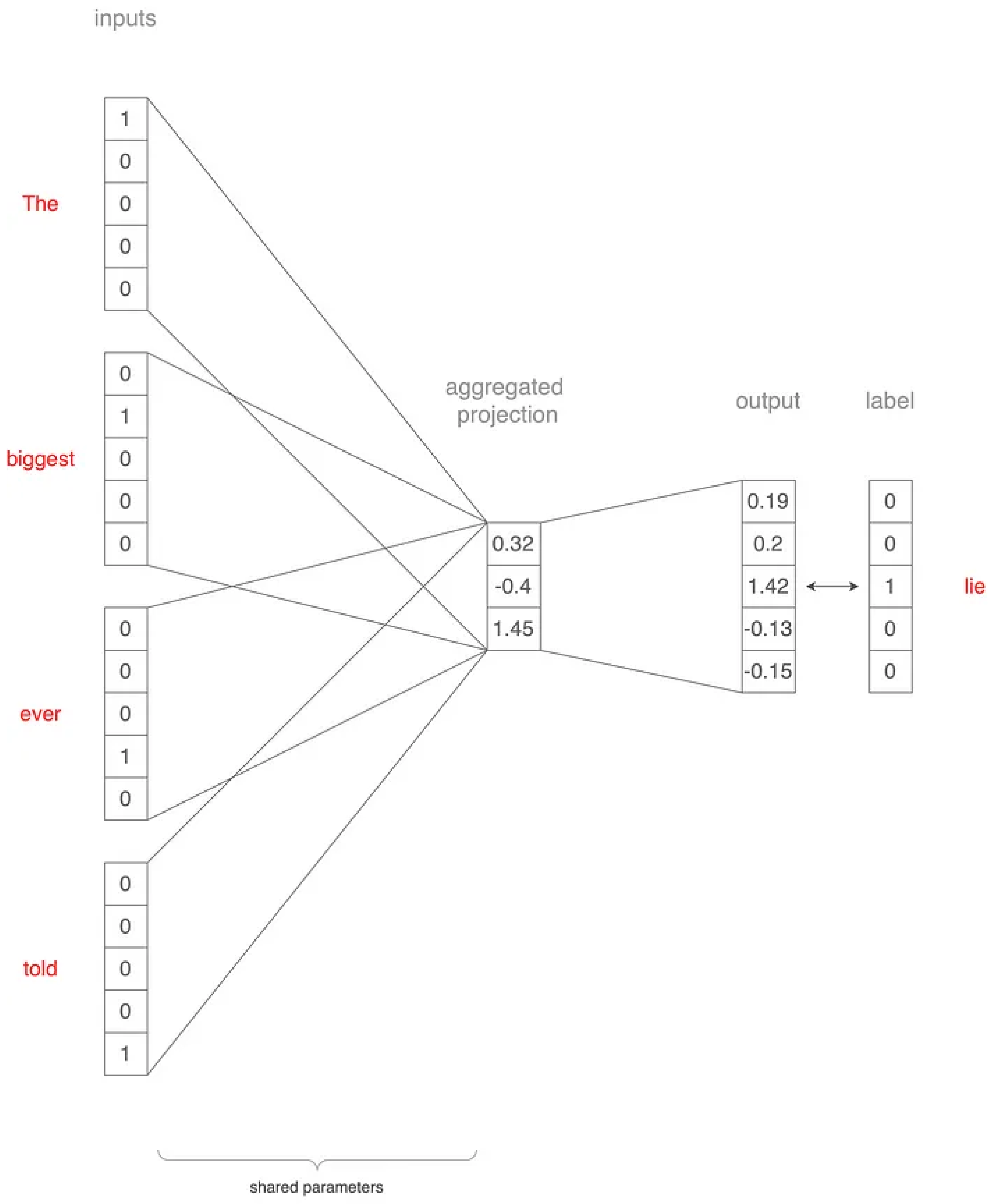


Figure 2: CBoW training illustration (image by author)

In this section and the next, let's understand how CBoW and skip-gram models are trained using a small vocabulary of five words: biggest, ever, lie, told, and the. And we have an example sentence “The biggest lie ever told”. How would we pass this into the CBoW architecture? This is shown in *Figure 2* above, but we will describe the process as well.

Suppose we set the context window size to 2. We take the words “The,” “biggest,” “ever,” and “told” and convert them into 5x1 one-hot vectors.

These vectors are then passed as input to the model and mapped to a projection layer. Let’s say this projection layer has a size of 3. Each word’s vector is multiplied by a 5x3 weight matrix (shared across inputs), resulting in four 3x1 vectors. Taking the average of these vectors gives us a single 3x1 vector. This vector is then projected back to a 5x1 vector using another 3x5 weight matrix.

This final vector represents the middle word “lie.” By calculating the true one hot vector and the actual output vector, we get a loss that is used to update the network’s weights through backpropagation.

We repeat this process by sliding the context window and then applying it to thousands of sentences. After training is complete, the first layer of the model, with dimensions 5x3 (vocabulary size x projection size), contains the learned parameters. These parameters are used as a lookup table to map each word to its corresponding vector representation.

Skip-gram

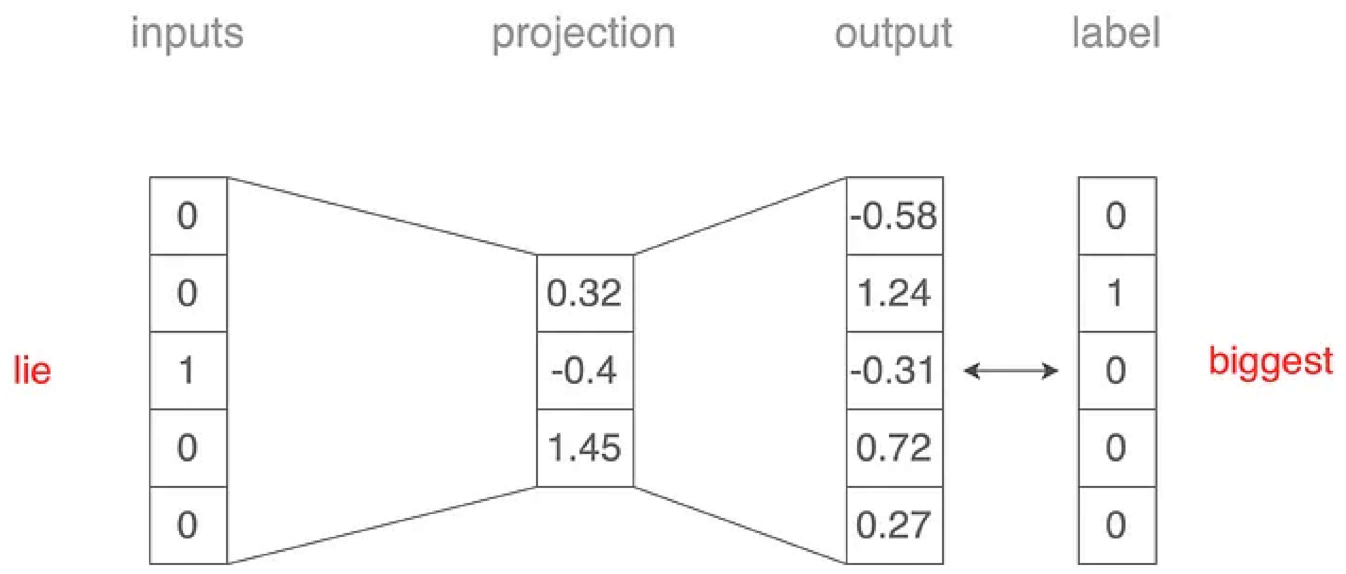


Figure 3: Skip-gram training illustration (image by author)

In the skip-gram model, we use a similar architecture as the continuous bag-of-words (CBoW) case. However, instead of predicting the target word based on its surrounding words, we flip the scenario as shwon in *Figure 3*. Now, the word "lie" becomes the input, and we aim to predict its context words. The name "skip-gram" reflects this approach, as we predict context words that may "skip" over a few words.

To illustrate this, let's consider some examples:

- The input word "lie" is paired with the output word "the."
- The input word "lie" is paired with the output word "biggest."
- The input word "lie" is paired with the output word "ever."
- The input word "lie" is paired with the output word "told."

We repeat this process for all the words in the training data. Once the training is complete, the parameters of the first layer, with dimensions of vocabulary size x projection size, capture the relationships between input words and their corresponding vector representations. These learned parameters allow us to map an input word to its respective vector representation in the skip-gram model.

Pros

1. **Overcomes the curse of dimensionality with simplicity**: Word2Vec provides a straightforward and efficient solution to the curse of dimensionality. By representing words as dense vectors, it reduces the sparsity and computational complexity associated with traditional methods like n-gram vectors.
2. **Generates vectors such that words closer in meaning have closer vector values:** Word2Vec's embeddings exhibit a valuable property where words with similar meanings are represented by vectors that are closer in numerical value. This allows for capturing semantic relationships and performing tasks like word similarity and analogy detection.
3. **Pretrained embeddings for various NLP applications:** Word2Vec's pretrained embeddings are widely available and can be utilized in a range of natural language processing (NLP) applications. These embeddings, trained on large corpora, provide a valuable resource for tasks like sentiment analysis, named entity recognition, machine translation, and more.
4. **Self-supervised framework for data augmentation and training**: Word2Vec operates in a self-supervised manner, leveraging the existing data to learn word representations. This makes it easy to gather more data and train the model, as it does not require extensive labeled datasets. The framework can be applied to large amounts of unlabeled text, enhancing the training process.

Cons

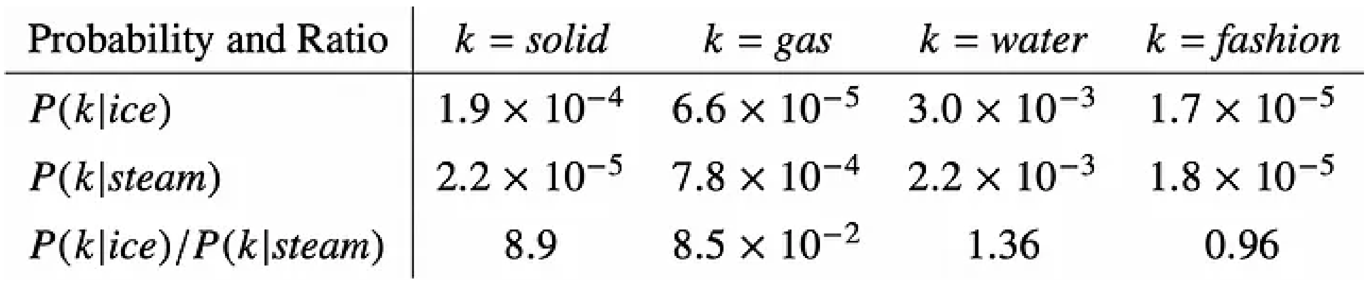
1. **Limited preservation of global information**: Word2Vec's embeddings focus primarily on capturing local context information and may not preserve global relationships between words. This limitation can impact tasks that require a broader understanding of text, such as document classification or sentiment analysis at the document level.
2. **Less suitable for morphologically rich languages**: Morphologically rich languages, characterized by complex word forms and inflections, may pose challenges for Word2Vec. Since Word2Vec treats each word as an atomic unit, it may struggle to capture the rich morphology and semantic nuances present in such languages.
3. **Lack of broad context awareness**: Word2Vec models consider only a local context window of words surrounding the target word during training. This limited context awareness may result in incomplete understanding of word meanings in certain contexts. It may struggle to capture long-range dependencies and intricate semantic relationships present in certain language phenomena.

In the following sections, we will see some word embedding architectures that help address these cons.

GloVe: Global Vectors

Word2Vec methods have been successful in capturing local context to a certain extent, but they do not take full advantage of the global context available in the corpus. Global context refers to using multiple sentences across the corpus to gather information. This is where GloVe comes in, as it leverages word-word co-occurrence for learning word embeddings.

The concept of a word-word co-occurrence matrix is key to Glove. It is a matrix that captures the occurrences of each word in the context of every other word in the corpus. Each cell in the matrix represents the count of occurrences of one word in the context of another word.



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 4: Example of word co-occurrence probability ratio (Source)

Instead of working directly with the probabilities of co-occurrence as in Word2Vec, Glove starts with the ratios of co-occurrence probabilities. In the context of *Figure 4*, P(*k* | *ice*) represents the probability of word k occurring in the context of the word "ice," and P(*k* | *steam*) represents the probability of word k occurring in the context of the word "steam." By comparing the ratio P(*k* | *ice*) / P(*k* | *steam*), we can determine the association of word k with either ice or steam. If the ratio is much greater than 1, it indicates a stronger association with ice. Conversely, if it is closer to 0, it suggests a stronger association with steam. A ratio closer to 1 implies no clear association with either ice or steam.

For example, when k = "solid," the probability ratio is much greater than 1, indicating a strong association with ice. On the other hand, when k = "gas," the probability ratio is much closer to 0, suggesting a stronger association with steam. As for the words "water" and "fashion," they do not exhibit a clear association with either ice or steam.

This association of words based on probability ratios is precisely what we aim to achieve. And this is optimized when learning embeddings with GloVe.

FastText

The traditional word2vec architectures, besides lacking the utilization of global information, do not effectively handle languages that are morphologically rich.

So, what does it mean for a language to be morphologically rich? In such languages, a word can change its form based on the context in which it is used. Let's take the example of a South Indian language called "Kannada."

In Kannada, the word for "house" is written as ಮನೆ (mane). However, when we say "in the house," it becomes ಮನೆಯಲ್ಲಿ (maneyalli), and when we say "from the house," it changes to ಮನೆಯಿಂದ (maneyinda). As you can see, only the preposition changes, but the translated words have different forms. In English, they are all simply "house." Consequently, traditional word2vec architectures would map all of these variations to the same vector. However, if we were to create a word2vec model for Kannada, which is morphologically rich, each of these three cases would be assigned different vectors. Moreover, the word "house" in Kannada can take on many more forms than just these three examples. Since our corpus may not contain all of these variations, the traditional word2vec training might not capture all the diverse word representations.

To address this issue, FastText introduces a solution by considering subword information when generating word vectors. Instead of treating each word as a whole, FastText breaks down words into character n-grams, ranging from tri-grams to 6-grams. These n-grams are then mapped to vectors, which are subsequently aggregated to represent the entire word. These aggregated vectors are then fed into a skip-gram architecture.

This approach allows for the recognition of shared characteristics among different word forms within a language. Even though we may not have seen every single form of a word in the corpus, the learned vectors capture the commonalities and similarities among these forms. Morphologically rich languages, such as Arabic, Turkish, Finnish, and various Indian languages, can benefit from FastText's ability to generate word vectors that account for different forms and variations.

Context Awareness

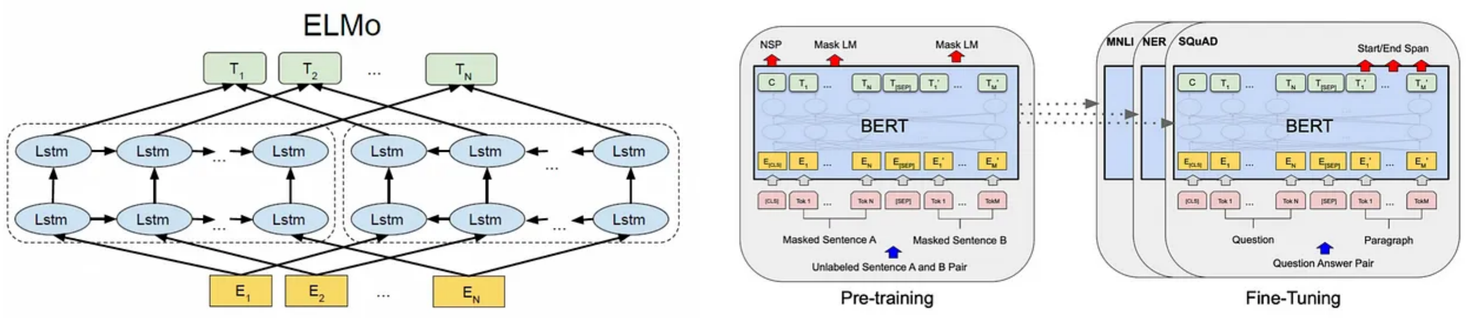


Figure 5: ELMo and BERT (Source)

Despite their advantages, the aforementioned word2vec architectures suffer from a limitation: they generate the same vector representation for a given word, regardless of its context.

To illustrate this point, let's consider the following two sentences:

1. "That drag queen slays."
2. "She has an ace and queen for a perfect hand."

In these sentences, the word “queen” has different meanings. However, in word2vec architectures, the vectors for “queen” in both cases would be the same. This is not ideal because we want word vectors to capture and represent different meanings based on their contexts.

To address this issue, more advanced architectures such as LSTM cells were introduced. These architectures were designed to incorporate contextual information into word representations. Over time, transformer-based models like BERT and GPT emerged, leading to the development of large-scale language models we see today. These models excel at considering context and generating word representations that are sensitive to the surrounding words and sentences.

By taking context into account, these advanced architectures enable the creation of more nuanced and meaningful word vectors, ensuring that the same word can have different vector representations depending on its specific context.

Conclusion

In conclusion, this blog post provided insights into the word2vec architecture and its ability to represent words using continuous dense vectors. Subsequent implementations such as GloVe capitalized on global context, while FastText enabled efficient learning of vectors for morphologically rich languages like Arabic, Finnish, and various Indian languages. However, a common drawback among these approaches is that they assign the same vector to a word regardless of its context during inference, which can hinder accurate representation of words with multiple meanings.

To address this limitation, subsequent advancements in NLP introduced LSTM cells and transformer architectures, which excel at capturing specific context and have become the foundation for modern large language models. These models have the capability to understand and generate word representations that vary based on their surrounding context, accommodating the nuanced meanings of words in different scenarios.

Nevertheless, it is important to acknowledge that the word2vec framework remains significant, as it continues to power numerous applications in the field of natural language processing. Its simplicity and ability to generate meaningful word embeddings have proven valuable, despite the challenges posed by contextual variations in word meanings.

For more information on language models, check out this YouTube playlist.

Happy Learning!

NLP Machine Learning Deep Learning Language Model Getting Started

Characteristics of Word Embeddings
✦ · 11 min read · Sep 4, 2021

The Word2vec Hyperparameters
✦ · 6 min read · Sep 3, 2021

The Word2ve
✦ · 15 min rea

View list

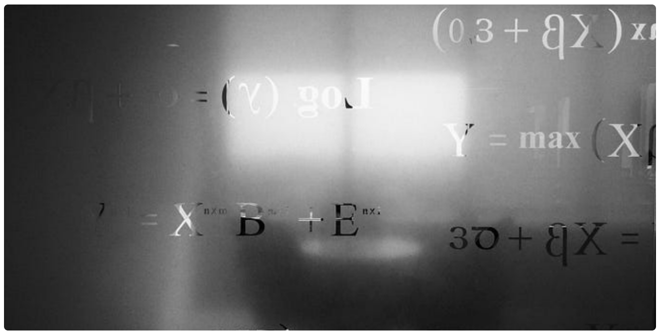
Written by Ajay Halthor

Follow

850 Followers · Writer for Towards Data Science

Machine Learning Engineer and educator

More from Ajay Halthor and Towards Data Science



Ajay Halthor in Towards Data Science

Likelihood, Probability, and the Math You Should Know

What role does likelihood play in machine learning?

✦ · 10 min read · Jul 26, 2022

Antonis Makropoulos in Towards Data Science

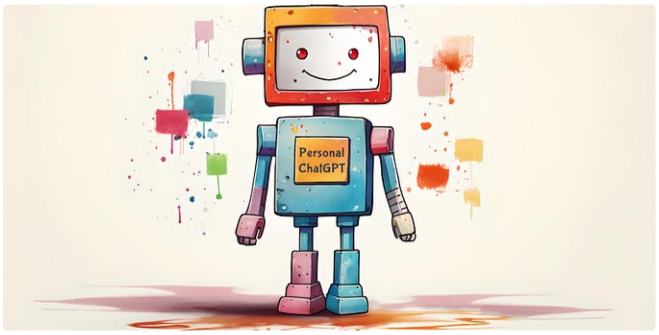
How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and…

10 min read · Sep 17

326 7

549 11



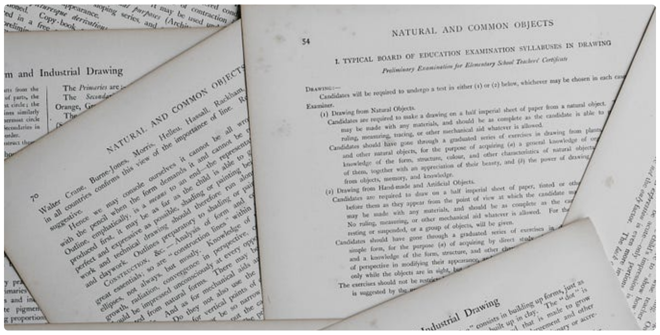
Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you...

· 15 min read · Sep 8

595 7



Ajay Halthor

Read these 20 papers to master language modeling

Understand the landscape of NLP in minutes

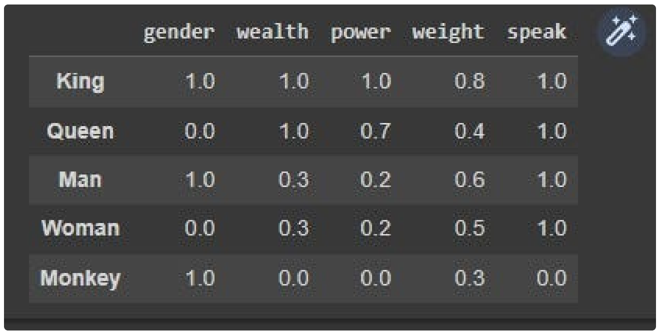
· 13 min read · Sep 9

2

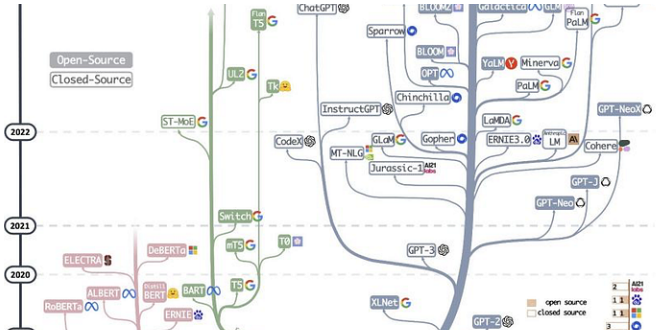
See all from Ajay Halthor

See all from Towards Data Science

Recommended from Medium



	gender	wealth	power	weight	speak
King	1.0	1.0	1.0	0.8	1.0
Queen	0.0	1.0	0.7	0.4	1.0
Man	1.0	0.3	0.2	0.6	1.0
Woman	0.0	0.3	0.2	0.5	1.0
Monkey	1.0	0.0	0.0	0.3	0.0



Md. Ishtiuk Ahammed

Word2Vec

Word2Vec text vectorization technique explanation.

2 min read · May 20

26

Haifeng Li

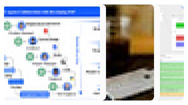
A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

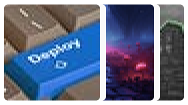
372

Lists



Natural Language Processing

669 stories · 283 saves



Predictive Modeling w/ Python

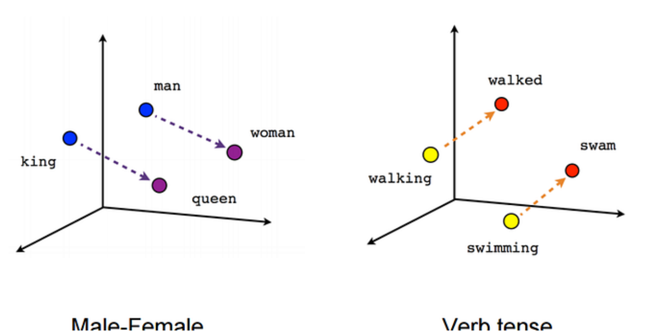
20 stories · 452 saves

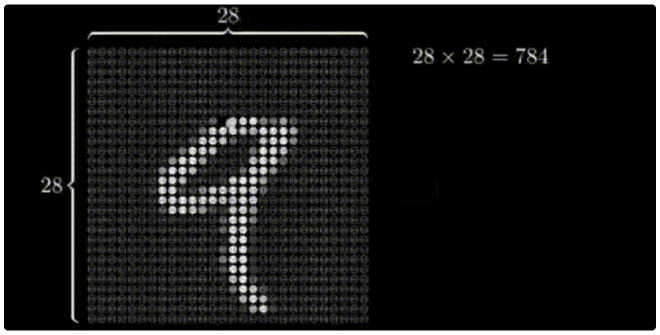
Practical Guides to Machine Learning

10 stories · 519 saves

The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 133 saves





Maninder Singh

Accelerate Your Text Data Analysis with Custom BERT Word...

One thing is for sure the way humans interact with each other naturally is one of the most...

4 min read · Apr 24

156

Sadaf Saleem

Neural Networks in 10mins. Simply Explained!

What are Neural Networks?

9 min read · May 15

252 2

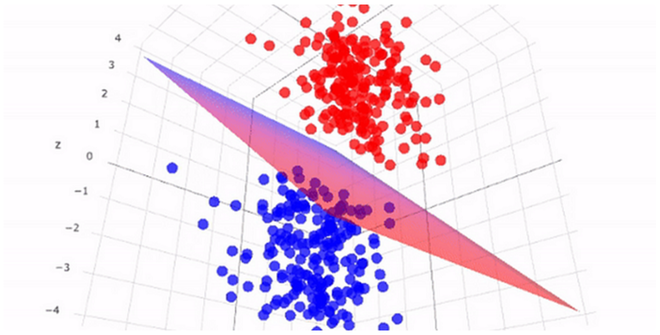
David Shapiro

A Pro's Guide to Finetuning LLMs

Large language models (LLMs) like GPT-3 and Llama have shown immense promise for...

12 min read · Sep 23

283 6



Tasmay Pankaj Tibre... in Low Code for Data Scie...

Support Vector Machines (SVM): An Intuitive Explanation

Everything you always wanted to know about this powerful supervised ML algorithm

17 min read · Jul 1

732 4

See more recommendations