**Embedding**

**What is Embedding ?**

An **embedding** is a relatively low-dimensional space into which you can translate high-dimensional vectors. Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. Ideally, an embedding capture some of the semantics of the input by placing semantically similar inputs close together in the embedding space. An embedding can be learned and reused across models.

Example: when applied to a text mining project, embeddings can assist us in learning the semantic meaning of a word by studying what other words it often appears next to. Then we can produce a list of embeddings, which can be treated as **a task-specific dictionary**. If you want to learn more about a particular word in your corpus, go to the “dictionary” and look it up. However, instead of providing you with the human language definition, it will return **a vector of numerical values to reflect its semantic meaning**. Furthermore, the distance between those vectors measures the similarity and relationship between the terms in the project.

As the name of word2vec - “word to vector,” which turns a word into a vector of numbers. In other words, **embedding is a string of numbers that serves as a unique identifier**. We can use the embedding technique to assign a unique numerical ID to a word, an individual, a voice sound, an image, etc. in your research. Scientists, using this idea, have created many fascinating 2vec-style models to facilitate the machine learning process.

***Why We Want Embeddings***

First, current **machine learning models continue to favor numerical values as inputs**. They are similar to math nerds in that when fed numbers, they can quickly capture vital information but are slow with discrete, categorical variables. However, when researching computer vision or voice recognition and the like, we are unlikely to be able to collect or only collect numerical data for our targets/dependent variables.**Converting such discrete, categorical variables to numbers can help with model fitting**.

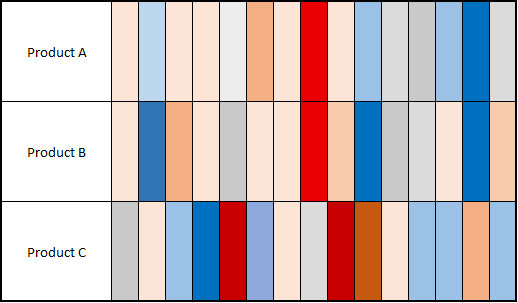


Second, it helps **reduce dimensions**. Someone may argue that the one-hot-encoding technique is how we handle categorical variables. However, in today’s data science world, it has proven to be much less effective.

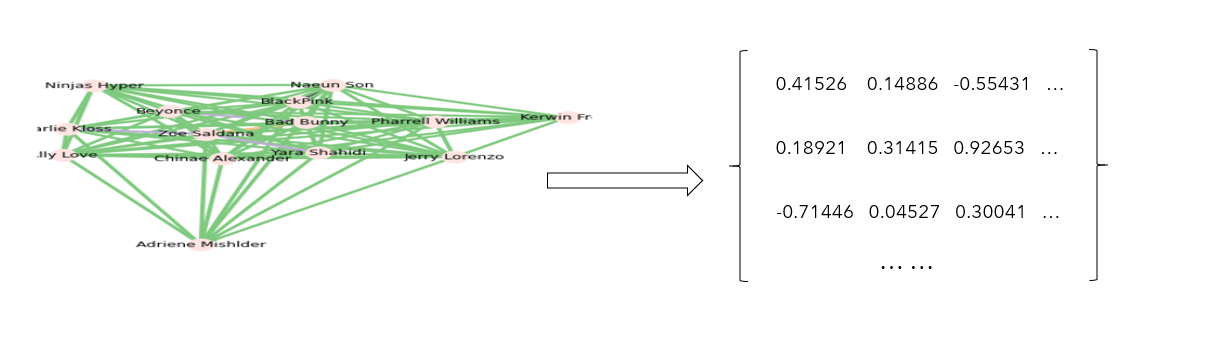
When dealing with a variable with four distinct types, we will usually create four new dummy variables to cope with it. Moreover, it has previously worked well.

Yet, consider the following scenario: we are researching consumer feedback for three products. We would only have one variable for each observation — the review content. We can build a term-document matrix and then put it into a classifier or some other types of models. However, let’s suppose we have 50 thousand reviews for each product, and the number of total unique words in the corpus is one million. Then we will end up with a matrix whose shape is (150K x 1M). This is a ridiculously large input for any model. And that is when we need to bring in the idea of embedding.

Assume we reduce the dimensions to 15 (a 15 digits ID for each product), take the average of the embeddings of each product, and then colorize them based on the numerical values, this is what we got:



The third reason is to **reduce complexity**. It is kind of like the extension of the second reason. Embedding also can help translate the very complex information into a vector of numbers. Here is an example in social network analysis:



Initially, we collected the data from social media and converted it into a social network analysis graph. In the graph, we can use the distance between the nodes and the color of the ties to interpret the similarities between the nodes. However, it is complicated and hard to read. Right now, we only have 14 nodes in the graph, and it is already a mess. Can you imagine what happens if we were to investigate 100 nodes? This is referred to as complex (high-dimensional) data. However, by using certain techniques to aid in dimensionality reduction, we can transform the graph into a list of embeddings. As a result, instead of the jumbled graph, we now have a new, clean “dictionary” for the nodes. We can use the “dictionary” to make a human-readable visualization.

**Word embedding techniques in NLP**

## **1. TF-IDF (Term Frequency-Inverse Document Frequency)**

TF-IDF is a machine learning (ML) algorithm based on a statistical measure of finding the relevance of words in the text. The text can be in the form of a document or various documents (corpus). It is a combination of two metrics: Term Frequency (TF) and Inverse Document Frequency (IDF).

TF score is based on the frequency of words in a document. Words are counted for their number of occurrences in the documents. TF is calculated by dividing the number of occurrences of a word (i) by the total number (N) of words in the document (j).

TF (i) = log (frequency (i,j)) / log (N (j))

IDF score calculates the rarity of the words. It is important because TF gives more weightage to words that occur more frequently. However, words that are rarely used in the corpus may hold significant information. IDF captures this information. It can be calculated by dividing the total number (N) of documents (d) by the number of documents containing the word (i).

IDF (i) = log (N (d) / frequency (d,i))

The log is taken in the above formulas to dampen the effect of large scores for TF and IDF. The final TF-IDF score is calculated by multiplying TF and IDF scores.

TF-IDF algorithm is used in solving simpler ML and NLP problems. It is better used for information retrieval, keyword extraction, stop words (like ‘a’, ‘the’, ‘are’, ‘is’) removal, and basic [text analysis](https://algoscale.com/data-analytics/text-analytics/). It cannot capture the semantic meaning of words in a sequence efficiently.

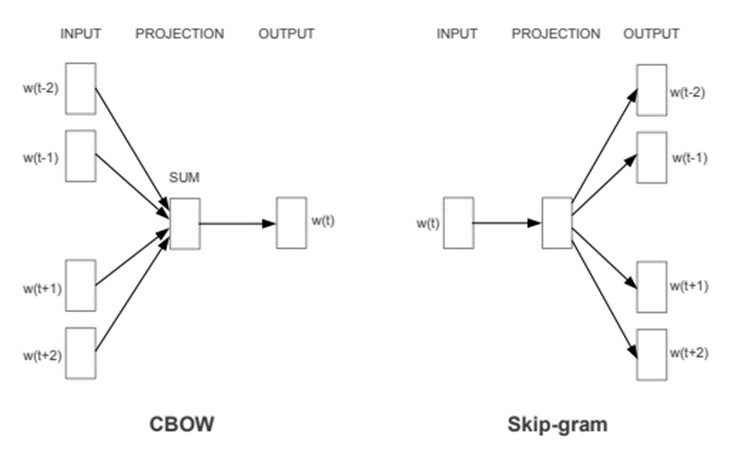
## **2.**[**Word2Vec**](https://arxiv.org/abs/1301.3781)

Word2Vec is a word embedding technique for solving advanced NLP problems. It can iterate over a large corpus of text to learn associations or dependencies among words.

Word2Vec finds similarities among words by using the [cosine similarity](https://www.machinelearningplus.com/nlp/cosine-similarity/) metric. If the cosine angle is 1, that means words are overlapping. If the cosine angle is 90, that means words are independent or hold no contextual similarity. It assigns similar vector representations to similar words.

Word2Vec offers two [neural network](https://www.ibm.com/cloud/learn/neural-networks)-based variants: Continuous Bag of Words (CBOW) and Skip-gram. In CBOW, the neural network model takes various words as input and predicts the target word that is closely related to the context of the input words. On the other hand, the Skip-gram architecture takes one word as input and predicts its closely related context words.

CBOW is quick and finds better numerical representations for frequent words, while Skip Gram can efficiently represent rare words. Word2Vec models are good at capturing semantic relationships among words. For example, the relationship between a country and its capital, like Paris is the capital of France and Berlin is the capital of Germany. It is best suited for performing [semantic analysis](https://www.expert.ai/blog/natural-language-process-semantic-analysis-definition/), which has application in recommendation systems and knowledge discovery.



CBOW & Skip-gram architectures.

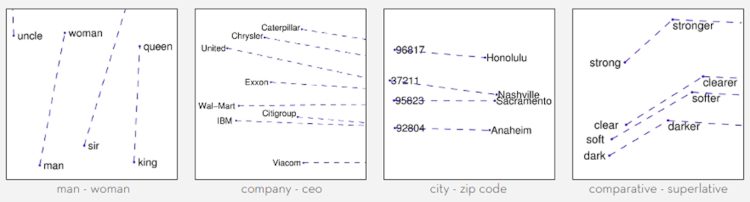
## **3.**[**GloVe**](https://nlp.stanford.edu/pubs/glove.pdf)**(Global Vectors for Word Representation)**

GloVe extends the work of Word2Vec to capture global contextual information in a text corpus by calculating a global [word-word co-occurrence matrix](https://medium.com/swlh/co-occurrence-matrix-9cacc5dd396e).

Word2Vec only captures the local context of words. During training, it only considers neighboring words to capture the context. GloVe considers the entire corpus and creates a large matrix that can capture the co-occurrence of words within the corpus.

GloVe combines the advantages of two-word vector learning methods: matrix factorization like [latent semantic analysis](https://blog.marketmuse.com/glossary/latent-semantic-analysis-definition/) (LSA) and local context window method like Skip-gram. The GloVe technique has a simpler [least square](https://www.investopedia.com/terms/l/least-squares-method.asp) cost or error function that reduces the computational cost of training the model. The resulting word embeddings are different and improved.

GloVe performs significantly better in word analogy and [named entity recognition](https://www.expert.ai/blog/entity-extraction-work/) problems. It is better than Word2Vec in some tasks and competes in others. However, both techniques are good at capturing semantic information within a corpus.



GloVe word vectors capturing words with similar semantics.

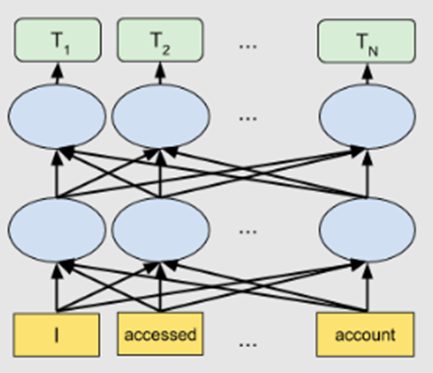
## **4.**[**BERT**](https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html)**— (Bidirectional Encoder Representations from Transformers)**

BERT belongs to a class of NLP-based language algorithms known as [transformers](https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html). BERT is a massive pre-trained deeply bidirectional encoder-based transformer model that comes in two variants. BERT-Base has 110 million parameters, and BERT-Large has 340 million parameters.

For generating word embeddings, BERT relies on an [attention mechanism](https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/). It generates high-quality context-aware or contextualized word embeddings. During the training process, embeddings are refined by passing through each BERT encoder layer. For each word, the attention mechanism captures word associations based on the words on the left and the words on the right. Word embeddings are also positionally encoded to keep track of the pattern or position of each word in a sentence.

BERT is more advanced than any of the techniques discussed above. It creates better word embeddings as the model is pre-trained on massive word corpus and Wikipedia datasets. BERT can be improved by fine-tuning the embeddings on task-specific datasets.

Though, BERT is most suited for language translation tasks. It has been optimized for many other applications and domains.



Bidirectional BERT architecture

**Concluding Thoughts**

With advancements in NLP, word embedding techniques are also improving. There are many NLP tasks that don’t require advanced embedding techniques. Many can perform equally well with simple word embedding techniques. The selection of a word embedding technique must be based on careful experimentations and task-specific requirements. Fine-tuning the word embedding models can improve the accuracy significantly.

In this article, we have given a high-level overview of various word embedding algorithms. Let’s summarize them below:

|  |  |  |
| --- | --- | --- |
| **Word Embedding Technique** | **Main Characteristics** | **Use cases** |
| TF-IDF | Statistical method to capture the relevance of words w.r.t the corpus of text. It does not capture semantic word associations. | Better for information retrieval and keyword extraction in documents. |
| Word2Vec | Neural network-based CBOW and Skip-gram architectures, better at capturing semantic information. | Useful in semantic analysis task. |
| GloVe | Matrix factorization based on global word-word co-occurrence. It solves the local context limitations of Word2Vec. | Better at word analogy and named-entity recognition tasks. Comparable results with Word2Vec in some semantic analysis tasks while better in others. |
| BERT | Transformer-based attention mechanism to capture high-quality contextual information. | Language translation, question-answering system. Deployed in Google Search engine to understand search queries. |