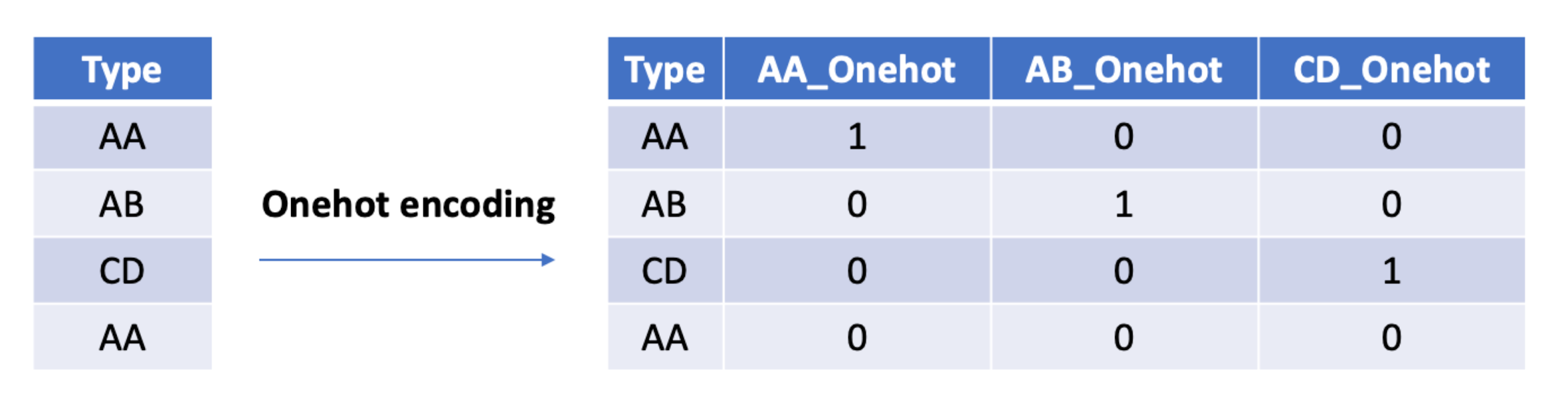
1. Explain One-Hot Encoding

One hot encoding is a process of **converting categorical data variables** so they can be provided to machine learning algorithms to improve predictions. *One hot encoding is a crucial part of feature engineering for machine learning.*

One hot encoding is one method of **converting data** to prepare it for an algorithm and get a better prediction. With one-hot, we convert each categorical value into a new categorical column and assign a binary value of 1 or 0 to those columns. Each integer value is represented as a binary vector. All the values are zero, and the index is marked with a 1.

Take a look at this chart for a better understanding:

**

Let’s apply this to an example. Say we have the values red and blue. With one-hot, we would assign red with a numeric value of 0 and blue with a numeric value of 1.

It’s crucial to be **consistent** when we use these values. This makes it possible to invert our encoding at a later point to get our original categorical back.

Once we assign numeric values, we create a **binary vector** that represents our numerical values. In this case, our vector will have 2 as its length since we have 2 values. Thus, the red value can be represented with the binary vector [1,0], and the blue value will be represented as [0,1].

### Why use one hot encoding?

One hot encoding is useful for data that has no relationship to each other. Machine learning algorithms treat the order of numbers as an *attribute of significance*. In other words, they will read a higher number as better or more important than a lower number.

While this is helpful for some ordinal situations, some input data **does not have any ranking** for category values, and this can lead to issues with predictions and poor performance. That’s when one hot encoding saves the day.

One hot encoding makes our training data more useful and expressive, and it can be rescaled easily. By using numeric values, we more easily determine a probability for our values. In particular, one hot encoding is used for our output values, since it provides more nuanced predictions than single labels.

1. Explain Bag of Words

Ans: The **bag-of-words model** is a simplifying representation used in [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing) and [information retrieval](https://en.wikipedia.org/wiki/Information_retrieval) (IR). In this model, a text (such as a sentence or a document) is represented as the [bag (multiset)](https://en.wikipedia.org/wiki/Multiset) of its words, disregarding grammar and even word order but keeping [multiplicity](https://en.wikipedia.org/wiki/Multiplicity_(mathematics)).

Bag of words is a [Natural Language Processing](https://www.mygreatlearning.com/blog/natural-language-processing-tutorial/) technique of text modelling. In technical terms, we can say that it is a method of feature extraction with text data.

One of the biggest problems with text is that it is messy and unstructured, and [machine learning](https://www.mygreatlearning.com/blog/what-is-machine-learning/?highlight=what%20is%20machine%20learning) algorithms prefer structured, well defined fixed-length inputs and by using the Bag-of-Words technique we can convert variable-length texts into a fixed-length **vector.**

## Why is the Bag-of-Words algorithm used?

So, why bag-of-words, what is wrong with the simple and easy text?

One of the biggest problems with text is that it is messy and unstructured, and [machine learning](https://www.mygreatlearning.com/blog/what-is-machine-learning/?highlight=what%20is%20machine%20learning) algorithms prefer structured, well defined fixed-length inputs and by using the Bag-of-Words technique we can convert variable-length texts into a fixed-length vector.

Also, at a much granular level, the machine learning models work with numerical data rather than textual data. So to be more specific, by using the bag-of-words (BoW) technique, we convert a text into its equivalent vector of numbers.

### Example with preprocessing:

Sentence 1: ”Welcome to Great Learning, Now start learning”

Sentence 2: “Learning is a good practice”

Step 1: Convert the above sentences in lower case as the case of the word does not hold any information.

Step 2: Remove special characters and stopwords from the text. Stopwords are the words that do not contain much information about text like ‘is’, ‘a’,’the and many more’.

After applying the above steps, the sentences are changed to

Sentence 1: ”welcome great learning now start learning”

Sentence 2: “learning good practice”

Although the above sentences do not make much sense the maximum information is contained in these words only.

Step 3: Go through all the words in the above text and make a list of all of the words in our model vocabulary.

* welcome
* great
* learning
* now
* start
* good
* practice

Now as the vocabulary has only 7 words, we can use a fixed-length document-representation of 7, with one position in the vector to score each word.

The scoring method we use here is the same as used in the previous example. For sentence 1, the count of words is as follow:

| Word | Frequency |
| --- | --- |
| welcome | 1 |
| great | 1 |
| learning | 2 |
| now | 1 |
| start | 1 |
| good | 0 |
| practice | 0 |

Writing the above frequencies in the vector

Sentence 1 ➝ [ 1,1,2,1,1,0,0 ]

Now for sentence 2, the scoring would be like

| Word | Frequency |
| --- | --- |
| welcome | 0 |
| great | 0 |
| learning | 1 |
| now | 0 |
| start | 0 |
| good | 1 |
| practice | 1 |

Similarly, writing the above frequencies in the vector form

Sentence 2 ➝ [ 0,0,1,0,0,1,1 ]

| Sentence | welcome | great | learning | now | start | good | practice |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sentence1 | 1 | 1 | 2 | 1 | 1 | 0 | 0 |
| Sentence2 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

The approach used in example two is the one that is generally used in the Bag-of-Words technique, the reason being that the datasets used in Machine learning are tremendously large and can contain vocabulary of a few thousand or even millions of words. Hence, preprocessing the text before using bag-of-words is a better way to go.

There are various preprocessing steps that can increase the performance of Bag-of-Words.

In the examples above we use all the words from vocabulary to form a vector, which is neither a practical way nor the best way to implement the BoW model. In practice, only a few words from the vocabulary, more preferably most common words are used to form the vector.

Also, at a much granular level, the machine learning models work with numerical data rather than textual data. So to be more specific, by using the bag-of-words (BoW) technique, we convert a text into its equivalent vector of numbers

1. Explain Bag of N-Grams

Ans:

An N-gram is an N-token sequence of words: a 2-gram (more commonly called a bigram) is a two-word sequence of words like “really good”, “not good”, or “your homework”, and a 3-gram (more commonly called a trigram) is a three-word sequence of words like “not at all”, or “turn off light”.

For example, the bigrams in the first line of text in the previous section: “This is not good at all” are as follows:

* “This is”
* “is not”
* “not good”
* “good at”
* “at all”

Now instead of using just words in the above example, we use bigrams (Bag-of-bigrams) as shown above. The model can differentiate between sentence 1 and sentence 2. So, using bi-grams makes tokens more understandable (for example, “HSR Layout”, in Bengaluru, is more informative than “HSR” and “layout”)

So we can conclude that a bag-of-bigrams representation is much more powerful than bag-of-words, and in many cases proves very hard to beat.

1. Explain TF-IDF

Ans:

The scoring method being used above takes the count of each word and represents the word in the vector by the number of counts of that particular word. What does a word having high word count signify?

Does this mean that the word is important in retrieving information about documents? The answer is NO. Let me explain, if a word occurs many times in a document but also along with many other documents in our dataset, maybe it is because this word is just a frequent word; not because it is relevant or meaningful.

One approach is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like “the” that are also frequent across all documents are penalized. This approach is called term frequency-inverse document frequency or shortly known as Tf-Idf approach of scoring.TF-IDF is intended to reflect how relevant a term is in a given document. So how is Tf-Idf of a document in a dataset calculated?

TF-IDF for a word in a document is calculated by multiplying two different metrics:

The **term frequency (TF)** of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are other ways to adjust the frequency. For example, by dividing the raw count of instances of a word by either length of the document, or by the raw frequency of the most frequent word in the document. The formula to calculate Term-Frequency is

TF(i,j)=n(i,j)/Σ n(i,j)

Where,

n(i,j )= number of times nth word occurred in a document

Σn(i,j) = total number of words in a document.

The **inverse document frequency(IDF)** of the word across a set of documents. This suggests how common or rare a word is in the entire document set. The closer it is to 0, the more common is the word. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating theSo, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.

Multiplying these two numbers results in the TF-IDF score of a word in a document. The higher the score, the more relevant that word is in that particular document.

To put it in mathematical terms, the TF-IDF score is calculated as follows:

IDF=1+log(N/dN)

Where

N=Total number of documents in the dataset

dN=total number of documents in which nth word occur

Also, note that the 1 added in the above formula is so that terms with zero IDF don’t get suppressed entirely. This process is known as IDF smoothing.

The TF-IDF is obtained by

TF-IDF=TF\*IDF

1. What is OOV problem?

Ans:

Out-of-vocabulary (OOV) are terms that are not part of the normal lexicon found in a natural language processing environment.

In speech recognition, it’s the audio signal that contains these terms. Word vectors are the mathematical equivalent of word meaning. But the limitation of word embeddings is that the words need to have been seen before in the training data.

When a word that’s not in the training set occurs in real data, this causes a problem. There are various techniques to avoid a zero-probability occurrence including smoothing and replacing the word as a synonym.

Handling OOV using a model:

Overview of Model

The model is built to produce embeddings for Out-of-Vocabulary(OOV) words depending on the OOV word’s context. This is done using language model built using a Bi-directional Recurrent Neural Network with a Long-Short Term Memory cell. This language model is used for predicting the most probable word embedding for the OOV word based on its context, by predicting words in place of the OOV word and then taking a weighted average of their mapped word embeddings. This gives a word embedding for the OOV word which is reliable in terms of usability for entity recognition tasks and a meaningful representation for it in the vector space.

The Model for predicting OOV word embeddings consist of two segments: The first step is where the Model is prepared by tokenising, training and saving a model based on a training corpus, and the second step, which consists of the Prepared model being used to predict embeddings. The first step is called the Preparation Step and the second step is called the Embedding Prediction step.

1. What are word embeddings?

Ans: Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation.

They are a distributed representation for text that is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on challenging [natural language processing problems](https://machinelearningmastery.com/natural-language-processing/).

Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning.

Key to the approach is the idea of using a dense distributed representation for each word.

Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding.

The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning. This can be contrasted with the crisp but fragile representation in a bag of words model where, unless explicitly managed, different words have different representations, regardless of how they are used.

There is deeper linguistic theory behind the approach, namely the “*distributional hypothesis*” by Zellig Harris that could be summarized as: words that have similar context will have similar meanings. For more depth see Harris’ 1956 paper “[Distributional structure](http://www.tandfonline.com/doi/pdf/10.1080/00437956.1954.11659520)“.

# What Are Word Embeddings for Text?

by [**Jason Brownlee**](https://machinelearningmastery.com/author/jasonb/) on October 11, 2017 in [**Deep Learning for Natural Language Processing**](https://machinelearningmastery.com/category/natural-language-processing/)

Tweet Tweet **Share**

**Share**

Last Updated on August 7, 2019

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation.

They are a distributed representation for text that is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on challenging [natural language processing problems](https://machinelearningmastery.com/natural-language-processing/).

In this post, you will discover the word embedding approach for representing text data.

After completing this post, you will know:

* What the word embedding approach for representing text is and how it differs from other feature extraction methods.
* That there are 3 main algorithms for learning a word embedding from text data.
* That you can either train a new embedding or use a pre-trained embedding on your natural language processing task.

**Kick-start your project** with my new book [Deep Learning for Natural Language Processing](https://machinelearningmastery.com/deep-learning-for-nlp/), including *step-by-step tutorials* and the *Python source code* files for all examples.

Let’s get started.



What Are Word Embeddings for Text?

Photo by [Heather](https://www.flickr.com/photos/woolgenie/32248092754/), some rights reserved.

## Overview

This post is divided into 3 parts; they are:

1. What Are Word Embeddings?
2. Word Embedding Algorithms
3. Using Word Embeddings

### Need help with Deep Learning for Text Data?

Take my free 7-day email crash course now (with code).

Click to sign-up and also get a free PDF Ebook version of the course.

Start Your FREE Crash-Course Now

## What Are Word Embeddings?

A word embedding is a learned representation for text where words that have the same meaning have a similar representation.

It is this approach to representing words and documents that may be considered one of the key breakthroughs of deep learning on challenging natural language processing problems.

One of the benefits of using dense and low-dimensional vectors is computational: the majority of neural network toolkits do not play well with very high-dimensional, sparse vectors. … The main benefit of the dense representations is generalization power: if we believe some features may provide similar clues, it is worthwhile to provide a representation that is able to capture these similarities.

Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning.

Key to the approach is the idea of using a dense distributed representation for each word.

Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding.

associate with each word in the vocabulary a distributed word feature vector … The feature vector represents different aspects of the word: each word is associated with a point in a vector space. The number of features … is much smaller than the size of the vocabulary

The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning. This can be contrasted with the crisp but fragile representation in a bag of words model where, unless explicitly managed, different words have different representations, regardless of how they are used.

There is deeper linguistic theory behind the approach, namely the “distributional hypothesis” by Zellig Harris that could be summarized as: words that have similar context will have similar meanings. For more depth see Harris’ 1956 paper “[Distributional structure](http://www.tandfonline.com/doi/pdf/10.1080/00437956.1954.11659520)“.

## Word Embedding Algorithms

Word embedding methods learn a real-valued vector representation for a predefined fixed sized vocabulary from a corpus of text.

The learning process is either joint with the neural network model on some task, such as document classification, or is an unsupervised process, using document statistics.

This section reviews three techniques that can be used to learn a word embedding from text data.

### 1. Embedding Layer

An embedding layer, for lack of a better name, is a word embedding that is learned jointly with a neural network model on a specific natural language processing task, such as [language modeling](https://machinelearningmastery.com/statistical-language-modeling-and-neural-language-models/) or document classification.

It requires that document text be cleaned and prepared such that each word is one-hot encoded. The size of the vector space is specified as part of the model, such as 50, 100, or 300 dimensions. The vectors are initialized with small random numbers. The embedding layer is used on the front end of a neural network and is fit in a supervised way using the Backpropagation algorithm.

… when the input to a neural network contains symbolic categorical features (e.g. features that take one of k distinct symbols, such as words from a closed vocabulary), it is common to associate each possible feature value (i.e., each word in the vocabulary) with a d-dimensional vector for some d. These vectors are then considered parameters of the model, and are trained jointly with the other parameters.

The one-hot encoded words are mapped to the word vectors. If a multilayer Perceptron model is used, then the word vectors are concatenated before being fed as input to the model. If a recurrent neural network is used, then each word may be taken as one input in a sequence.

This approach of learning an embedding layer requires a lot of training data and can be slow, but will learn an embedding both targeted to the specific text data and the NLP task.

### 2. Word2Vec

Word2Vec is a statistical method for efficiently learning a standalone word embedding from a text corpus.

It was developed by Tomas Mikolov, et al. at Google in 2013 as a response to make the neural-network-based training of the embedding more efficient and since then has become the de facto standard for developing pre-trained word embedding.

Additionally, the work involved analysis of the learned vectors and the exploration of vector math on the representations of words. For example, that subtracting the “man-ness” from “King” and adding “women-ness” results in the word “Queen“, capturing the analogy “king is to queen as man is to woman“.

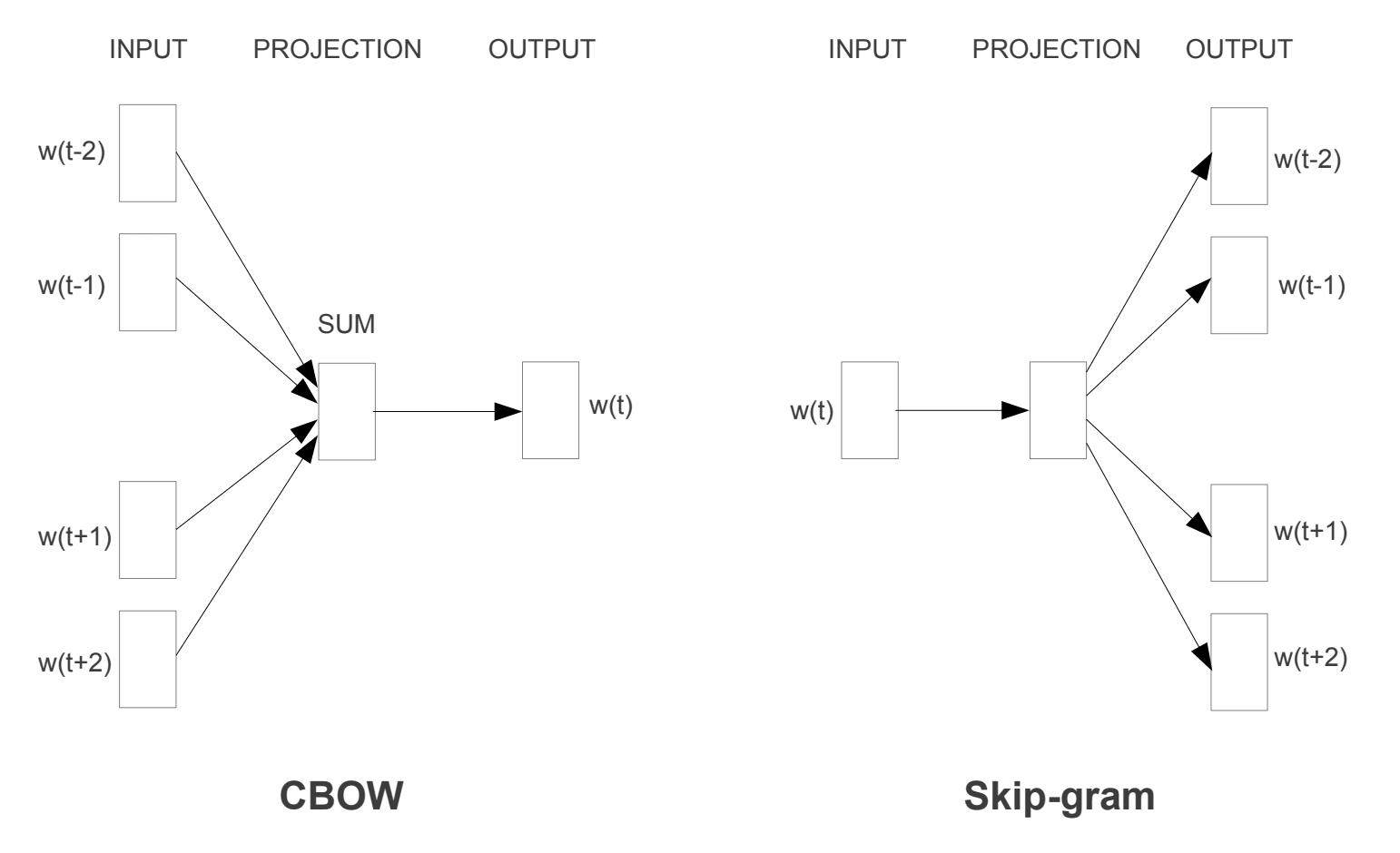
We find that these representations are surprisingly good at capturing syntactic and semantic regularities in language, and that each relationship is characterized by a relation-specific vector offset. This allows vector-oriented reasoning based on the offsets between words. For example, the male/female relationship is automatically learned, and with the induced vector representations, “King – Man + Woman” results in a vector very close to “Queen.”

Two different learning models were introduced that can be used as part of the word2vec approach to learn the word embedding; they are:

* Continuous Bag-of-Words, or CBOW model.
* Continuous Skip-Gram Model.

The CBOW model learns the embedding by predicting the current word based on its context. The continuous skip-gram model learns by predicting the surrounding words given a current word.

The continuous skip-gram model learns by predicting the surrounding words given a current word.



Word2Vec Training Models

Taken from “Efficient Estimation of Word Representations in Vector Space”, 2013

Both models are focused on learning about words given their local usage context, where the context is defined by a window of neighboring words. This window is a configurable parameter of the model.

The size of the sliding window has a strong effect on the resulting vector similarities. Large windows tend to produce more topical similarities […], while smaller windows tend to produce more functional and syntactic similarities.

The key benefit of the approach is that high-quality word embeddings can be learned efficiently (low space and time complexity), allowing larger embeddings to be learned (more dimensions) from much larger corpora of text (billions of words).

### 3. GloVe

The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford.

Classical vector space model representations of words were developed using matrix factorization techniques such as Latent Semantic Analysis (LSA) that do a good job of using global text statistics but are not as good as the learned methods like word2vec at capturing meaning and demonstrating it on tasks like calculating analogies (e.g. the King and Queen example above).

GloVe is an approach to marry both the global statistics of matrix factorization techniques like LSA with the local context-based learning in word2vec.

Rather than using a window to define local context, GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. The result is a learning model that may result in generally better word embeddings.

1. Explain Continuous bag of words (CBOW)

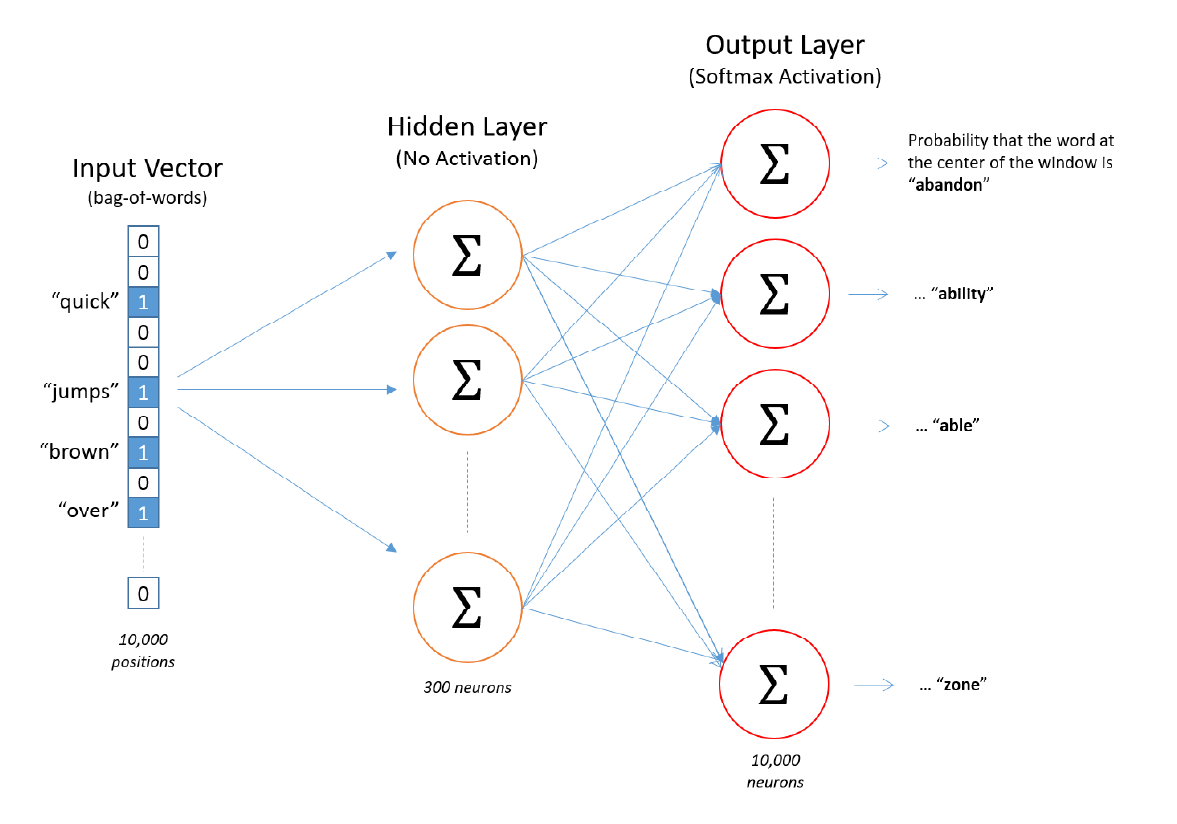
Ans:

The CBOW model learns the embedding by predicting the current word based on its context

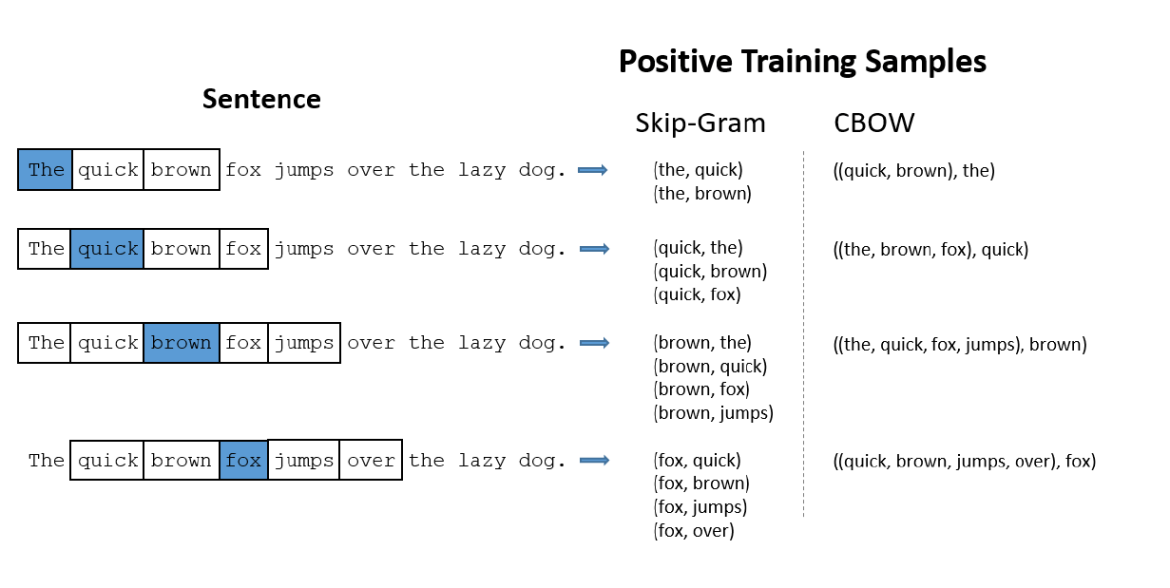
CBOW is a variant of the word2vec model predicts the center word from (bag of) context words. So given all the words in the context window (excluding the middle one), CBOW would tell us the most likely the word at the center.

For example, say we have a window size of 2 on the following sentence. Given the words (“PM”, “American”, “and”), we want the network to predict “Modi”

CBOW Model Architecture- The CBOW architecture then looks like the following:

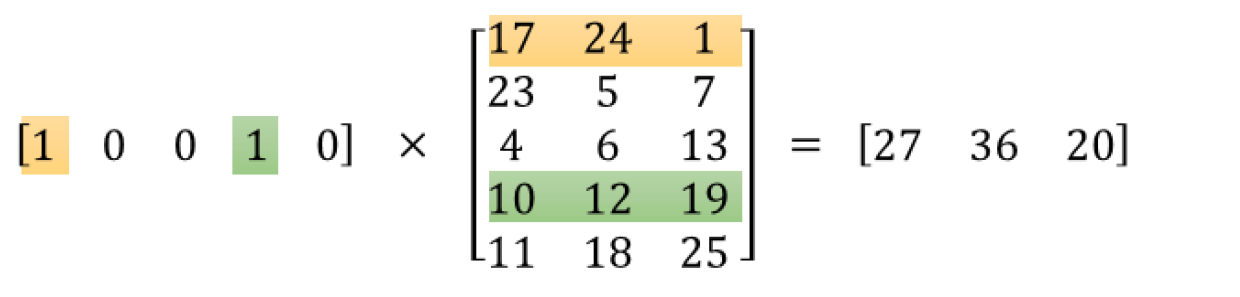
.

The training samples for CBOW look different than those generated for skip-gram.

Fig: Training Samples examples for CBOW

With a window size of 2, skip-gram will generate (up to) four training samples per center word, whereas CBOW only generates one. With skip-gram, we saw that multiplying with a one-hot vector just selects a row from the hidden layer weight matrix. What happens when you multiply with a bag-of-words vector instead? The result is that it

selects the corresponding rows and sums them together.

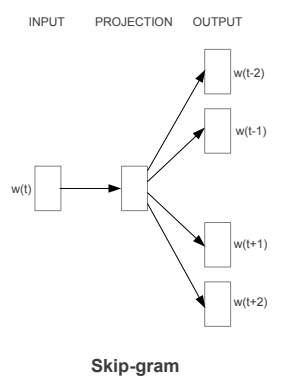
For the CBOW architecture, we also divide this sum by the number of context words to calculate their average word vector. So the output of the hidden layer in the CBOW architecture is the average of all the context word vectors. From there, the output layer is identical to the one in skip-gram.

1. Explain SkipGram

Ans:

The continuous skip-gram model learns by predicting the surrounding words given a current word.

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 from earlier, “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.



The Skip-gram model architecture (Source: <https://arxiv.org/pdf/1301.3781.pdf> Mikolov el al.)

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 0indicates 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.

1. Explain Glove Embeddings.

Ans:

The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford.

Classical vector space model representations of words were developed using matrix factorization techniques such as Latent Semantic Analysis (LSA) that do a good job of using global text statistics but are not as good as the learned methods like word2vec at capturing meaning and demonstrating it on tasks like calculating analogies (e.g. the King and Queen example above).

GloVe is an approach to marry both the global statistics of matrix factorization techniques like LSA with the local context-based learning in word2vec.

Rather than using a window to define local context, GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. The result is a learning model that may result in generally better word embeddings.