`dish2vec`: A Comparison of Word Embedding Methods in an Unsupervised Setting

Guus Verstegen

Student Number: 481224

11th June 2019

Erasmus University Rotterdam

Erasmus School of Economics

Supervisor: Dr. M. van de Velden

Second Assessor: Dr. F. Frasincar

Master Thesis Econometrics and Management Science

Business Analytics and Quantitative Marketing

Abstract

While the popularity of continuous word embeddings has increased over the past years, detailed derivations and extensive explanations of these methods lack in the academic literature. This paper elaborates on three popular word embedding methods; `GloVe` and two versions of `word2vec`: continuous skip-gram and continuous bag-of-words. This research aims to enhance our understanding of both the foundation and extensions of these methods. In addition, this research addresses instability of the methods with respect to the hyperparameters. An example in a food menu context is used to illustrate the application of these word embedding techniques in quantitative marketing. For this specific case, the `GloVe` method performs best according to external expert evaluation. Nevertheless, the fact that `GloVe` performs best is not generalizable to other domains or data sets due to the instability of the models and the evaluation procedure.

Keywords: word embedding; word2vec; GloVe; quantitative marketing; food

Contents

1 Introduction

Natural language processing has gained popularity in both research and business over the past years (Nadkarni et al., 2011). The increase of computing power enables one to process text on a large scale, quantifying millions of words within hours. Language modeling by the quantification of text allows users to feed natural language as input to statistical models and machine learning techniques. A popular approach to quantify text is to represent each word in the vocabulary by a vector filled with real-valued numbers, called a word embedding. The numbers in these word embeddings represent scores of latent linguistic characteristics. The trained word embeddings capture similarities of words in text data. Derivations of the word embeddings and information on the algorithms designed to calculate them efficiently tend to be described rather concisely (e.g., Rong (2014)) or are even completely left out (e.g., Mikolov et al. (2013a), Mikolov et al. (2013b)). This can make it hard for people new to this field to grasp the word embedding methods.

The goal of this research is twofold. First, the main goal of this paper is to provide an extensive and structured description of word embedding techniques. Three major state-of-the-art word embedding techniques methods are considered: `GloVe` (Pennington et al., 2014), `word2vec` continuous skip-gram (SG), and `word2vec` continuous bag-of-words (CBOW) (Mikolov et al., 2013a). We provide an elaborate overview of the mathematical derivations and the computational procedures to generate the word embeddings. Rather than providing several steps of the algorithms, we explicitly derive the gradients used for the optimization of the algorithms. Second, this paper aims to apply and compare the three methods in an empirical unsupervised setting. We build on the foundation of knowledge on these methods laid to reach the main goal by applying the word embedding techniques in a food menu context. The menu context is constructed by using menu data scraped from the Internet as training data for the word embeddings. We evaluate the model stability and the performance of these methods in the task of creating word embeddings using external domain experts. Thereby, we try to answer the auxiliary research question: Do `GloVe`, `word2vec` SG and `word2vec` CBOW give different embeddings and if so, can we, based on expert opinions, determine which is best in capturing dish similarities using food menu data? The detailed description combined with the application provide researchers with a complete insight into the considered techniques.

This paper can be relevant to practitioners as well. In food marketing, amongst other applications, the word embeddings could function as predictors for buying behavior of caterers, be used to create personalized inspirational content for caterers, or be used to classify the businesses on cuisine. For example, the dish similarity based on the word embeddings can be used to recommend dishes as inspirational content to a business that is not served yet, but fits in according to the dishes they already serve. More generally, word embeddings can function as predictors in other domains as well.

In the next section, we provide a literature review on `word2vec` and `GloVe`. Furthermore, this part provides an overview of research on food and menu data. Section 3 provides insights into the computational procedure of both methods by describing the embedding algorithms and their mathematical derivations. Section 4 covers extensions on the basic, original word embedding algorithms used to speed the computations. Besides, this section discusses the link between the `word2vec` and `GloVe` model, the stability of the models, and how to implement the algorithms in practice. Section 5 contains an application in the food menu context. We describe the menu data and provide a model stability analysis based on the data set at hand. Besides, we compare the word embedding methods according to an argued hyperparameter setting, and we evaluate their performance using the expertise of chefs. Section 6 summarizes the paper and consequently, Section 7 concludes the paper a discussion.

2 Literature review

Language is a continuously transforming way of communication. Individuals use similar words differently and use different words in different contexts. Therefore it is difficult to model language with exact precision. Language models have been created to approximate language by capturing syntactic and semantic characteristics in text as well as possible. Examples of text modeling are manually created lexical databases (e.g., WordNet (Miller, 1995)). These nets store word definitions and semantic relations between words. The definitions of the words in these nets are subjective and limited in numbers. For example, related to our application, new cooking techniques and dishes will not be stored in such lexical databases. Besides, nuances in semantics are often lost using these fixed definitions. Therefore, it may be more appealing to derive these representations in a data-driven manner. Text and audio are examples of data that can be used to model language. This paper focuses on the prior.

Firth (1957) states that the distributional property of text is the property that 'a word is characterized by the company it keeps'. Put differently, the words in the vicinity of a word determine the meaning of that word. This definition finds its origin in the distributional hypothesis of word meaning introduced by Harris (1954). Within the domain of natural language processing (NLP), many word representations depend on this characteristic of text. Researchers have developed multiple variations of text representations over time.

2.1 Discrete distributional representation of text

Information retrieval from text started with discrete distributional representations based on word counts in the form of document-word and word-word co-occurrence matrices. The so-called one-hot encoding is at the basis of the document-word matrices. A one-hot encoding of a word is a vector where the size of the vector is equal to the number of unique words in the text data. Each element in the vector is associated with one of those unique words. The vector element has value one on the dimension corresponding to the dimension of the word the one-hot encoding is representing and zeros on all other positions in the vector. The bag-of-words (BOW) representation for documents combines the one-hot vectors. The BOW vector representation of a document contains the occurrence count for every word in the document and a zero on the position for the words that are not present in that file. The document-word co-occurrence matrix combines the BOW vectors of multiple documents into a matrix. These document-word matrices have a large number of dimensions as such a matrix has its number of columns equal to the number of unique words in the vocabulary. Furthermore, these matrices have relatively few non-zero entries, and hence, these are so-called sparse matrices. The sparsity of the matrix implies that combinations of words in text training data are not likely to occur in the same order in the test set and vice versa. Performing operations with sparse matrices may take a long time while many actions involve addition or multiplication of zeros. Besides, storing these large matrices in memory to perform any operations on them may be complex. Therefore, the sparse characteristic of the document-word matrix may complicate the use of these matrices in downstream analysis, such as a classification task. Moreover, the document-word matrices do not directly capture the syntactic and semantic characteristics of words since the counts are on document level.
The word-word co-occurrence matrix contains statistics on a more detailed level. The word-word matrix is constructed by counting the number of co-occurrences of two words in the data. The number of rows and columns of this data matrix equal the size of the vocabulary and therefore, as for the document-word matrix, this is a sparse matrix which may be hard to perform operations on.

2.2 Continuous distributed representation of text

The discrete distributional representations of the co-occurrence matrices can be used as input to some form of dimension reduction to transform the sparse matrix into a dense matrix. Such dense matrix, contrarily to a sparse matrix, contains mostly non-zero elements and is referred to as a continuous distributed representation of text. Using dimension reduction, (the words in) the text (are) is projected onto a low-dimensional latent space. Note that methods using the document-word matrix as input results in word relatedness if two words occur in the same documents. Using the word-word matrix as input results in word relatedness if two words are surrounded by similar words. Therefore, the prior might result in word relatedness such as ('sun' - 'hot') and ('sun' - 'star'). The latter may focus on the similarity subset of relatedness such as ('sun' - 'star'). The combination ('sun' - 'hot') might come out less related since the word hot is not a noun, and therefore, is less likely to be used similarly as 'sun' on a local level.

Latent semantic analysis (LSA) (Dumais et al., 1988) is a method to transform text data into vector representations of words and documents in lower dimensional space. The input for this global matrix factorization method is the document-word co-occurrence matrix. LSA applies singular value decomposition (SVD) to reduce the dimensions of this sparse matrix and generates word vectors based on the documents containing the words. Words are therefore related if they occur in the same documents. Note that the bag-of-words of the documents construct the document-word matrix and are therefore at the base of LSA. Hence, the order of word sequences has no impact on the vector representations. LSA fails to capture linguistic characteristic depending on local text structures such as grammar, syntax, and part-of-speech. The vector representation generated by LSA is called a distributed vector feature representation or real-value word feature vector representation. Lund and Burgess (1996) use the word-word co-occurrence matrix as input to vector representations of text introducing their hyperspace analogue to language (HAL) model. However, according to Pennington et al. (2014), the problem with methods related to HAL, based on the word-word co-occurrence matrix, is that 'the most frequent words contribute a disproportionate amount to the similarity measure'.

Bengio et al. (2003) provide a different real-valued vector representation of single words in the text data with their neural probabilistic language model (NPLM). NPLM generates word embeddings based on the local context window of a word without using the co-occurrence statistics, and hence, these representations can capture more complex syntactic and semantic characteristics of words. Collobert and Weston (2008) continue in the direction of neural language models and introduce the use of deep convolutional neural network language models (NNLM) to model language for multiple goals. An NNLM is a supervised training procedure where the word embeddings are a byproduct of this supervised task. The training times of these deep networks are in order of magnitude of weeks. The transformation of text to word embeddings is the first layers in their networks. After training, the embeddings capture syntactic and semantic word characteristics optimized to perform as predictions for the task of the network. Lebret and Collobert (2013) propose an alternative to the computational intensive approach of Collobert and Weston (2008) to generate word embeddings. They apply Hellinger principal component analysis (HPCA, Rao (1995)) on the word-word co-occurrence matrix to create the word embeddings. A major difference and disadvantage compared to the neural net approach is that the operations of this transformation are linear, while neural networks can capture highly nonlinear relations.

Continuing in the field of real-valued word feature vector creation and with the goal to capture more linguistic characteristic, `word2vec` SG, `word2vec` CBOW and `GloVe` find their origin. Currently, these three word embedding algorithms appear to be (at the base of) commonly used procedures in unsupervised text quantification. The methods are fast, can capture complex linguistic relations, and try to limit the disproportionate influence of frequent words on the word embeddings. All three methods start with a random initialized word vector for each word in the vocabulary and iteratively update these vectors by minimizing a loss function. This loss function, in turn, depends on a similarity measure between words. In the next three paragraphs, we will elaborate on characteristics of `word2vec`, characteristics of `GloVe`, and differences between the two, respectively.

2.3 `word2vec` and `GloVe`

The estimation procedure of `word2vec` is based on the prediction power of words to predict other words in the neighborhood, the so-called local context window, of those words in the text. The SG variant of `word2vec` uses word by word similarity comparisons and tries to predict a word in the local context window given the word in the middle of this context window. The CBOW variant, on the other hand, compares each word with the average representation of the surrounding words and the word in the center of the context window is predicted given its context. Due to the averaging of the representations of the words in the context, fewer comparisons are made in the CBOW algorithm, and therefore, CBOW is quicker compared to SG. SG, on the other hand, is preferred if the training set is small since the algorithm compares single word representations and thereby uses more information. Especially the quality of the word embeddings of infrequent words benefit from this use of extra information since these are updated the least in the iterative procedure.

`GloVe` is a count-based method and uses the word-word co-occurrence matrix to update the word vectors iteratively. Since `GloVe` is based on the co-occurrence counts, this method can make efficient use of the repetitive characteristic of text data. However, reducing text data to word-word co-occurrence counts removes information in the data (information that is used by `word2vec`). `GloVe` is related to global matrix factorization methods. It modifies the classical SVD approach by using iterative learning, as used in the local context window approach. `word2vec` fits more in the field of neural network approaches. Thereby, SG and CBOW appear to be better at capturing nonlinear relationships. Moreover, due to the efficient use of the co-occurrence statistics by `GloVe` and since `word2vec` can be represented as a shallow neural network, both methods are fast compared to deep neural networks. Since the distributional assumption is at the base of the local context window approach, and both methods are based on the local context windows, both methods make use of this assumption.

Menus of restaurants mainly exist out of enumerations of nouns. It is expected that these data contain little linguistic complexity. Hence, the relationships between words in the data are probably not highly nonlinear. In such setting, the use of a complex neural network estimation procedures such as `word2vec` might be redundant and we expect that `GloVe`, based on the word-word co-occurrence statistics, will outperform the two `word2vec` variants. Moreover, since many menu cards contain overlapping words such as 'rice', 'vegetable', and 'beef', CBOW may predict frequent words too often. Therefore, we hypothesize that CBOW performs worse compared to SG.

This paper theoretically and empirically compares `word2vec` SG, `word2vec` CBOW and `GloVe`. We provide a thorough overview of these algorithms including detailed explanations and full derivations, which we were not able to find in the literature. We attempt to provide the reader with a solid understanding of two prominent word embedding techniques, starting from scratch. Currently, such detailed explanations and derivations of the basic `word2vec` and `GloVe` procedures are not present in the literature. The hyperparameter issue becomes apparent when we derived these methods. We discuss the choice of hyperparameters and their impact on the stability of the models. Recent literature stresses the importance of hyperparameter tuning (Antoniak and Mimno (2018), Wendlandt et al. (2018), Caselles-Dupré et al. (2018)). Moreover, we touch upon research that has been carried out related to `word2vec` and `GloVe`. These papers focus on one or a few elements of the algorithms or focus on a single algorithm. We put the existing literature together and elaborate on these topics.

We train and compare word embeddings in an empirical food menu context. The performance of the methods considered in this paper (or a subset of them) is previously compared in different contexts (e.g., Tsvetkov et al. (2015) and Levy et al. (2015)). In our research, expert opinions are the base of the performance measure of the algorithms to generate the word vectors. The opinions of six professional chefs are used to evaluate the similarity of dishes based on the combination of the words in these dishes. We use this evaluation as a measure for the quality of the word embeddings. Schnabel et al. (2015) define this evaluation procedure as comparative intrinsic evaluation. The dish representations are generated by a weighted average of the word vectors as suggested by Arora et al. (2016b) and we use the cosine similarity as the similarity measure.

2.4 Food menu related literature

There is little research regarding food marketing with menu data. Recent research on menu data is mostly on the nutritional aspect of dishes on the menu, such as Hwang and Lorenzen (2008) and Kozup et al. (2003). Hwang and Lorenzen (2008) show that if nutritional information on each meal is displayed, customers tend to choose for the low-fat options first. However, no research uses menu text as input to create continuous feature vectors or uses menu data as input for downstream analysis.

3 Methodology: the base implementation

In this paper, the following notation is used. Vectors and matrices are denoted by bold lowercase and bold uppercase letters, respectively. Light uppercase letters denote sets and scalars are given as light lowercase letters. Table 1 in Section 3.5 provides a glossary on frequently used variables.

In this section `word2vec` and `GloVe` are described. Both methods transform text data into a set of real numbers. The goal of both methods is to capture linguistic properties of words into these sets of numbers such that similar words have a similar encoding and dissimilar words are encoded dissimilar. This quantified text data can, in turn, be used as input for statistical models or machine learning algorithms. Both methods start with a random initial state. The randomly initialized numbers are iteratively updated by optimization of a loss function. Minimizing the function value or loss value is the estimation part of the process and depends on how well similar words are similar to each other in the encoding. Each unique word in the data is encoded by one set of numbers, the word vector. Equivalently, each word in the vocabulary is linked to one word vector. The word vectors for all words have the same size, the so-called vector size or vector length. This vector size of a word vector d is the number of numbers in the vector. The ordering of these numbers in the word vector is important. The values at position $i \in \{1, 2, ..., d\}$ in these d-dimensional word vectors encapsulate the scores of the words on the i-th axis. This i-th axis represents latent linguistic property i. Each latent property can contain semantic information, syntactic information, or a combination of linguistic elements that are not captured by any terminology. The scores on the d dimensions together construct the word vector containing linguistic information of a word.
We illustrate the word vectors using a simple example. Suppose one of the methods is used to train three-dimensional word vectors ($d = 3$). The resulting word vectors of the

words ‘pasta’, ‘pizza’, and ‘poach’ and are as follows

$$\text{pasta: } \begin{bmatrix} 1 \\ 3 \\ -3 \end{bmatrix}, \qquad \text{pizza: } \begin{bmatrix} 1 \\ 2.5 \\ -5 \end{bmatrix}, \qquad \text{poach: } \begin{bmatrix} -2 \\ 1 \\ 4 \end{bmatrix}.$$

The vector notation shows that the words ‘pasta’ and ‘pizza’ score the same on the first dimension. Moreover, the scores on the second dimension are close for these two words. The scores on the third dimension are negative for both words, though not as close as for the first two dimensions. The scores in the vector of the word ‘poach’ show less similarity to the scores on the same dimensions of the other two word vectors. Figure 1 contains a visualization of the word vectors. One can observe from the figure that ‘pasta’ and ‘pizza’ are more similar than either ‘pasta’ or ‘pizza’ with respect to ‘poach’. The task of `word2vec` and `GloVe` is to map each word in the vocabulary to a word vector. Since the word vectors have non-zero elements on most or all dimensions, they are called dense vectors. Furthermore, the word vectors are an estimation of the true meaning of a word and reduce the complexity of linguistics to a limited set of characteristics, and thereby, reduce the highly complex information space of linguistics to a limited number of dimensions. The resulting real-valued dense word vectors are the so-called word embeddings. Two fundamental assumptions underlie these word embedding methods. The first assumption is that text has a distributional property. This property comes down to the general idea that the meaning of a word depends on the words surrounding that word, as discussed in section 1. The second assumption is that the word embeddings can capture the meaning of words. That is, words with similar meaning should be mapped to similar vectors in the embedding vector space, and dissimilar words should be assigned dissimilar vectors.

A word embedding is based on the words surrounding the word of interest. These surrounding words are the so-called context words. Based on the assumption of the distributional property of text, the word embedding captures linguistic properties corresponding to this word. The linguistic properties are extracted from a training text data set. This text data set is used to train the word embeddings and can consist of a single document or a combination of documents. Single or combined, this training set is called the corpus. `GloVe` and `word2vec` both bypass the sparsity problem of the word-word and document-word matrices. The sparsity problem entails that it may be complicated to perform mathematical operations on these matrices and to store them in memory. This is caused by the fact that there are mostly zero elements in these matrices and that their size is large. `word2vec` bypasses this problem by not using these matrices at all. This algorithm iterates through the text word by word and updates dense word vectors at each iteration based on the word vectors of the words surrounding at every word. `GloVe` iterates over the words in the vocabulary and updates the word vectors using a combination of the word-word co-occurrence matrix and the word vectors of all words. Thereby, `GloVe` reduces the dimensions of the sparse input matrix and transforms it into a dense output. The resulting vectors appear to be suitable for tasks such as text classification and part of speech tagging.

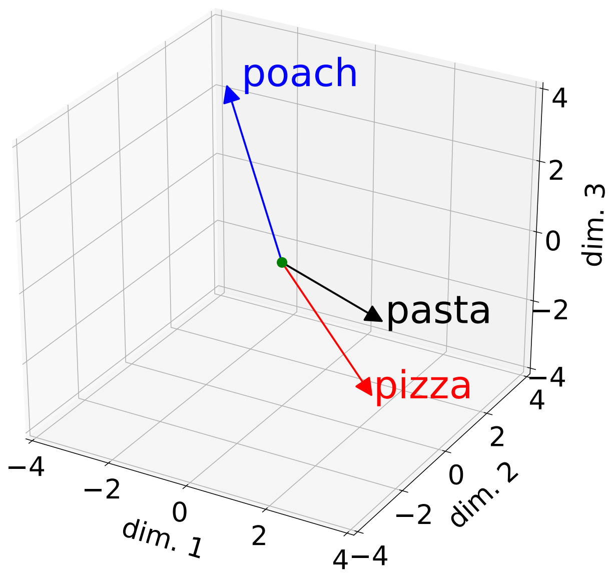


Figure 1: Three-dimensional word embedding space

`GloVe` and `word2vec` both find two word vectors for each word in the vocabulary. In `word2vec`, these vectors are referred to as the 'input' and 'output' representation. In `GloVe`, they are called an 'ordinary' and 'context' word vector. The algorithms both start with two randomly initialized vectors per word, the input and output vectors or the ordinary and context vectors. Iteratively updating the vectors results in two sets of trained word vectors, the word embeddings. The updating procedure is performed up to the point of a stopping criterion set by the user. This stopping criterion is usually one or multiple passes through the entire data, called an epoch. In a single epoch, the `word2vec` algorithm iterates over each word in the corpus and the objective function value may be updated with respect to the same word multiple times within one epoch. For `GloVe`, an epoch contains one iteration for each unique word in the text data and the objective function value is updated once for each word in the vocabulary. After training, we collect these two sets of vectors in two $d \times v$ matrices $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{d \times v}$, where d corresponds to the dimensionality of the embeddings set by the user, and v corresponds to the number of unique words in the corpus. Since the goal is to get one vector representation per word, two word vectors need to be combined or one should use one of the two representations. Since the authors of the `GloVe` paper (Pennington et al., 2014) argue that there is evidence that training a pair of vectors and combining these at the end improves general results and reduces over-fitting and noise, we choose to combine the vectors. There are multiple ways to combine the two word vectors, such as summation and row-wise concatenation. There is no evidence that summation of the vectors outperforms concatenation of the results, as investigated by Garten et al. (2015). Pennington et al. (2014) suggest to combine the resulting pairs of word vectors by summing $\mathbf{U}$ and $\mathbf{V}$ (simply $\mathbf{U} + \mathbf{V}$). Levy et al. (2015) as well show that addition is a suitable method to combine the pairs of word vectors. In this paper, we follow the suggestion of Pennington et al. (2014), and for consistency, we apply this summation of the two word vectors in the `word2vec` context as well.

3.1 Quantified word similarity: cosine similarity

The cosine similarity is a common choice to measure the similarity between word embeddings. The cosine similarity ignores the length of the vectors and measures similarity according to the angle between two vectors. In the `word2vec` context, the word vectors of frequent words are implicitly updated more often than the embeddings of infrequent words. The word vectors of frequent words are updated more often in the iterative algorithm and thereby, are more likely to have a larger norm. Since the cosine similarity ignores the vector length by normalization and only takes the angle between word vectors into account, frequent and infrequent words can still be very similar. Similarity measures such as the Manhattan and Euclidean norm do not have this property. The cosine similarity requires the two vectors of interest to be in the same space, and hence, they need to have the same dimensionality. The cosine similarity between vector $\mathbf{u}$ and $\mathbf{v}$ is given by

$$\text{cossim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\intercal \mathbf{v}}{||\mathbf{u}||\ ||\mathbf{v}||} = \frac{\mathbf{u}^\intercal \mathbf{v}}{\sqrt{\mathbf{u}^\intercal \mathbf{u}}\sqrt{\mathbf{v}^\intercal \mathbf{v}}},$$

where $\mathbf{u}^\intercal$ is defined as the transpose of vector $\mathbf{u}$. According to Levy et al. (2015) the cosine similarity measure 'states that words are similar if they tend to appear in similar contexts, *or* if they tend to appear in the contexts of each other (and preferably both)'. They argue that normalizing the word vectors to unit length using the L_2 normalization $||\cdot||$, as used in the cosine similarity measure, consistently outperforms other ways of normalization. Therefore, the cosine similarity is more suited than similarity measures obtained by other normalizations of the dot product (Levy et al. (2015) consider row-wise, column-wise and no normalization).

3.2 Model optimization: stochastic gradient descent

The solutions of `GloVe` and `word2vec` are found by optimization of the corresponding objective functions. Therefore, we shortly discuss gradient descent, one of the most intuitive and basic algorithms to minimize functions iteratively. The objective functions are derived one by one in the next sections and arranged together in Table 2. The minimum obtained by optimization of these objective functions by gradient descent can be local since these functions are not globally convex. Gradient descent uses the first order derivatives of a function to determine the gradient and thereby to obtain the direction in which the parameters of the function should be modified in order to decrease the value of objective function the fastest. This iterative procedure continues until a stopping condition is met. The stopping condition can, for example, be based on a maximum number of iterations or the size of the gradient. Each iteration the algorithm takes a step in the direction based on the gradient. The size of the steps $\alpha \in \mathbb{R}_+$ the algorithm takes, the so-called learning rate, needs to be fixed by the user. It should be small enough to avoid overshooting the minimum and large enough to make proper progression in each iteration. Let $\boldsymbol{\theta}^{(\text{old})} \in \mathbb{R}^n$ denote the vector containing the n values of the parameters at an iteration and $\boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \in \mathbb{R}^n$ denote the gradient of the objective function with respect to $\boldsymbol{\theta} \in \mathbb{R}^n$. The value of the parameter at the next iteration $\boldsymbol{\theta}^{(\text{new})} \in \mathbb{R}^n$ is given by

$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{old})} - \alpha \boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(\text{old})}).$$

An alternative to regular gradient descent is stochastic gradient descent (SGD). SGD randomly subsamples the data point(s) that is (are) used to update the parameter at each iteration of the optimization algorithm. Suppose $J(\boldsymbol{\theta}) := \sum_{t=1}^{s} J_t(\boldsymbol{\theta})$ for t data points, then due to linearity it holds that $\boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) := \sum_{t=1}^{s} \boldsymbol{\nabla}_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta})$. An SGD update according to one data point at t is given by

$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{old})} - \alpha \boldsymbol{\nabla}_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta}^{(\text{old})}).$$

The stochastic part of SGD is the fact that t is randomly sampled for each update. SGD may at some iterations increase the loss of the objective function, but the individual steps are less costly to compute due to limited data usage per iteration. Moreover, SGD may result in a better optimum compared to regular gradient descent if the problem is non-convex because it can move to a domain of the function with a larger loss value and thereby enter a valley different from the valley of the initialization point.

3.3 `word2vec`

In `word2vec`, the set of words surrounding and including a word define the local context window of a word in text data. This window can include words to the left and the right of this word. A local context window is symmetric if it contains as many words to the left as to the right of the word of interest. The symmetric local context window is the base of the `word2vec` word embeddings. Each word in the corpus has a local context window. Let S denote the corpus that is the sequence of input words. Let integer s denote the size of the corpus. $t \in \{1, ..., s\}$ corresponds to the place of a word in the corpus S. Consider an arbitrary word w_t in the corpus and consider a maximum context window size of c. Then C_{w_t} denotes the set of context words in the context window for when w_t is the center word. The words within c_{1,w_t} places to the left and c_{2,w_t} to the right of the center word w_t are called the context words and therefore C_{w_t} consists of $|C_{w_t}| = c_{1,w_t} + c_{2,w_t}$ words. Note that c_{1,w_t} and c_{2,w_t} are allowed to be dependent on the center word, but are strictly smaller than c. The context window includes the center word as well, and hence there are

$c_{1,w_t} + c_{2,w_t} + 1$ words in each context window. Each word with multiple appearances in the corpus has multiple context windows. It is inevitable that some local context windows are not symmetric due to words at the beginning or end of the document. There are two variants of `word2vec` model introduced by Mikolov et al. (2013a). Both algorithms start with randomly initialized word embeddings that are updated iteratively as the context window moves from word to word through the corpus. The word embeddings of the words in the vocabulary resulting from the training procedure are optimized such that either the center words predict the corresponding context window words as well as possible (skip-gram model) or such that the context window words predict the corresponding center words as well as possible (continuous bag-of-words model). Both variants optimize an objective function based on a conditional probability distribution using gradient descent. This probability depends on a similarity between words based on the dot product of the vector representations of these words. `word2vec` uses two representations for each word in the vocabulary, one vector for when a word is a center word and one vector for when a word is a context word. Let V be the set of all (unique) words w_j in the vocabulary with $j \in \{1, ..., v\}$, where v is the vocabulary size. Let $\mathbf{v}_{w_j}, \mathbf{u}_{w_j} \in \mathbb{R}^d$ be the vector representations of the word w_j when word w_j is the input or output word, respectively. The size d of these vectors is a hyperparameter, and the user chooses its value. Let w_o refer to a specific context word and w_i refer to a specific center word both from the vocabulary. Mikolov et al. (2013a) use the dot product of the word vectors as a similarity measure for the two words. Hence the similarity between context word w_o and center word w_i is denoted by

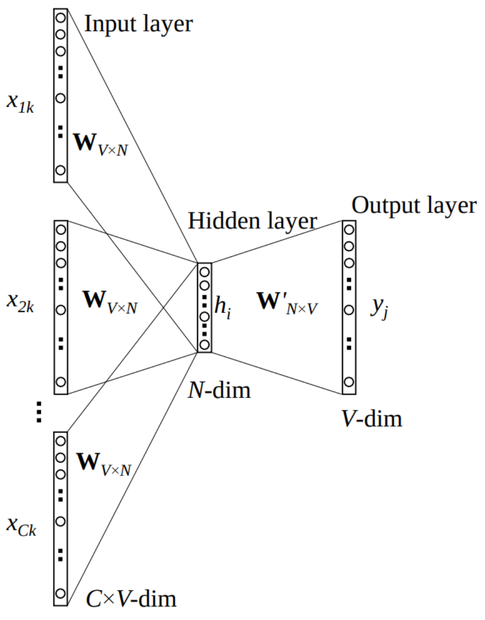
$$\mathbf{u}_{w_o}^{\intercal} \mathbf{v}_{w_i}, \qquad \forall\, i, o \in \{1, ..., v\}. \tag{1}$$

3.3.1 Hidden layer notation

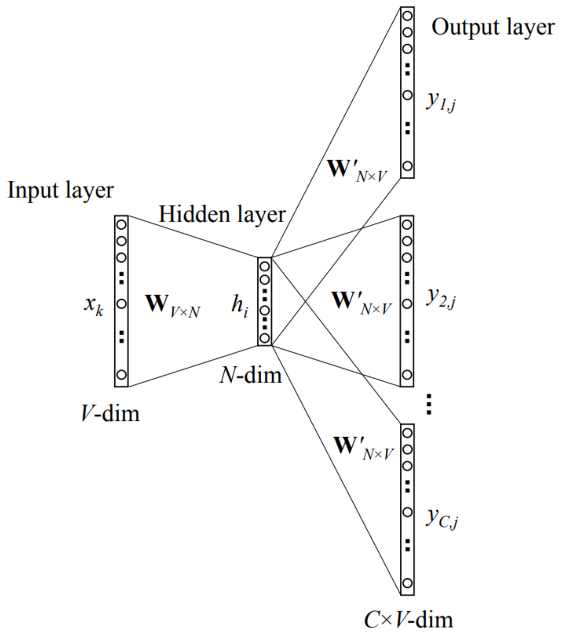
This section touches upon the visual representation that is often used to explain `word2vec`, the so-called hidden layer notation. Figure 2a and 2b[1] contain examples of these representations. In the other sections of this paper, this notation of the (rather flat) `word2vec` framework is not used to avoid unnecessary complexity. We provide only a summary of this representation and for further explanations of the hidden layer notation, we refer to Rong (2014).

`word2vec` originates from the field of Neural Net Language Models (NNLM) (Bengio et al., 2003). These networks contain four types of layers: input, projection, hidden, and output layers. Mikolov et al. (2013a) remove the nonlinear hidden layers to simplify the model. Multiplication of the input layer $\mathbf{x}$ (consisting of zeros and ones) by the input weighting matrix $\mathbf{W}$ (containing the input word vectors) maps the input word(s) to a continuous

[1]Figure 2a and 2b taken from Rong (2014)



(a) CBOW model



(b) SG model

Figure 2: `word2vec` hidden layer notation

vector space. The result is a single projection layer. Some call this projection layer a hidden layer (e.g., (Rong, 2014)), referring to the projection layer as it is defined in the initial paper of Mikolov et al. (2013a). Depending on the variant of `word2vec`, the input layer consists of multiple words or a single word. The prior corresponds to the continuous-bag-of-words (CBOW) variant (Section 3.3.3), the latter corresponds to the continuous skip-gram (SG) variant (Section 3.3.2). In `word2vec` CBOW, the resulting projection layer is the average of the context vectors if one updates the center word vector and is simply the center word vector if the context word vectors are updated. In `word2vec` SG, the projection layer contains the center word if the context word vectors are updated and contains a context word if the center word vector is updated.

The SG method processes each center word with each context word one by one. CBOW uses the average of context vectors to update the center word vector, and thereby requires fewer operations. Multiplication of the projection layer by the output weighting matrix $\mathbf{W}'$ (containing the output word vectors) followed by a soft-max transformation provides the output layer $\mathbf{y}$ containing values between 0 and 1. By this multiplication of the projection layer by $\mathbf{W}'$, the projection layer is joined with each output word vector for all words in the vocabulary. The soft-max transformation is used to convert the vector resulting from the multiplication in a vector with values between 0 and 1. These values are considered probabilities that a word occurs in the context given a word (SG) or that a word occurs given the context (CBOW). Comparing these probabilities with the true observed output word results in a prediction error. This prediction error is minimized by iteratively updating the weighting matrices alternately using backpropagation.

3.3.2 Continuous skip-gram (SG)

3.3.2.1 Objective function

The objective function of the SG model depends on the probability distribution of predicting a context word given the center word. This distribution contains the probabilities for each word to be located as a context word at a specific position with respect to a given center word. There are v possible nominal outcomes for this context word, where v is the vocabulary size. Therefore, a common way to model such a problem is utilizing a conditional multinomial logit model (equivalent to soft-max regression model). Mikolov et al. (2013a) use the estimates of the class label probabilities of the multinomial logit model, that is the soft-max function of the real-valued similarity measures, as measures for the conditional probabilities in the SG algorithm. The multinomial logit model (maximum entropy model) is introduced to an NLP setting by Berger et al. (1996). However, opposed to multinomial logistic regression, `word2vec` estimates the center and context word vector jointly and therefore, the problem is not the same as the multinomial logistic regression. Section A of the Appendix provides an overview of the assumptions for the multinomial logistic regression in a text context. Even though the assumptions might not hold, Mikolov et al. (2013a) transform the similarity measures in equation (1 into a probability distribution conditional on the center word w_i using the soft-max notation, 'building a log-linear classifier'. The corresponding conditional probability that word w_o is the true context word given center word w_i is denoted by

$$p(w_o|w_i) = \frac{\exp(\mathbf{u}_{w_o}^{\intercal}\mathbf{v}_{w_i})}{\sum_{j=1}^{v}\exp(\mathbf{u}_{w_j}^{\intercal}\mathbf{v}_{w_i})}, \qquad \forall\ i, o \in \{1, ..., v\}.$$

Next, only the words in the context window(s) of a center word are considered to have an impact on the representation of this center word. Therefore we want to maximize the joint likelihood of the context words given the center word. The corresponding joint probability distribution, under the assumption that context words are independent, is given by

$$\begin{aligned} p_{\text{SG}}(C_{w_t}|w_t) &= \prod_{-c_{1,w_t}\leq x\leq c_{2,w_t},\ x\neq 0} p(w_{t+x}|w_t) \\ &= \prod_{w_l\in C_{w_t}} p(w_l|w_t), \qquad \forall\ t \in \{1, ..., s\}. \end{aligned}$$

The assumption of independent context words is likely to be violated in practice. Such context words are words occurring close to each other and a core assumption to the

`word2vec` algorithm is that the meaning of a word is determined by its surrounding words.
Maximizing the likelihood function is equivalent to minimizing the negative log-likelihood function. We continue with the latter. The negative log-likelihood function (the objective function at position t) is given by

$$\begin{aligned}\mathcal{L}_{\text{SG},t}(\boldsymbol{\theta}) = J_{\text{SG},t}(\boldsymbol{\theta}) &= -\log \prod_{w_l \in C_{w_t}} p_{\text{SG}}(w_l|w_t) \\ &= -\sum_{w_l \in C_{w_t}} \log p_{\text{SG}}(w_l|w_t), \qquad \forall\, t \in \{1, ..., s\}. \end{aligned} \tag{2}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{2dv}$ contains the input and output word vector representations of all words in V. $\boldsymbol{\theta}$ is the only (non-hyper) parameter of the model. Minimizing the total sum of the negative log-likelihood for each position in the text results in the objective of the skip-gram model. The objective function $J(\boldsymbol{\theta})$ corresponding to this goal is given by

$$\begin{aligned} J_{\text{SG}}(\boldsymbol{\theta}) &= \sum_{t=1}^{s} \mathcal{L}_{\text{SG},t}(\boldsymbol{\theta}) \\ &= -\sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \log p_{\text{SG}}(w_l|w_t) \\ &= -\sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \log \frac{\exp(\mathbf{u}_{w_l}^{\intercal} \mathbf{v}_{w_t})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{w_t})}. \end{aligned}$$

3.3.2.2 Gradient derivation

To optimize the objective function with gradient descent, we need the gradient. The derivatives with respect to the center vectors are derived, followed by the derivation of the context vectors. Let $\mathbf{v}_{w_z}$ with $w_z \in V$ denote an arbitrary center word vector where $z \in \{1, ..., v\}$, then the partial derivative is denoted by the vector

$$\begin{aligned} \frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{SG}}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \mathbf{v}_{w_z}} \left[-\sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \log \frac{\exp(\mathbf{u}_{w_l}^{\intercal} \mathbf{v}_{w_t})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{w_t})} \right] \\ &= -\sum_{t=1}^{s} \frac{\partial}{\partial \mathbf{v}_{w_z}} \sum_{w_l \in C_{w_t}} \log \frac{\exp(\mathbf{u}_{w_l}^{\intercal} \mathbf{v}_{w_t})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{w_t})} && (3) \\ &= -\sum_{t \,\ni\, w_t = w_z} \frac{\partial}{\partial \mathbf{v}_{w_z}} \sum_{w_l \in C_{w_t}} \log \frac{\exp(\mathbf{u}_{w_l}^{\intercal} \mathbf{v}_{w_t})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{w_t})} && (4) \\ &= -\sum_{t \,\ni\, w_t = w_z} \sum_{w_l \in C_{w_t}} \frac{\partial}{\partial \mathbf{v}_{w_z}} \log \frac{\exp(\mathbf{u}_{w_l}^{\intercal} \mathbf{v}_{w_t})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{w_t})} \end{aligned}$$

$$= - \sum_{t \,\ni\, w_t = w_z} \sum_{w_l \in C_{w_t}} \frac{\partial}{\partial \mathbf{v}_{w_z}} \left[\log(\exp(\mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t})) - \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t)}) \right]$$

$$= - \sum_{t \,\ni\, w_t = w_z} \sum_{w_l \in C_{w_t}} \frac{\partial}{\partial \mathbf{v}_{w_z}} \left[\mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t} - \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t)}) \right]$$

$$= - \sum_{t \,\ni\, w_t = w_z} \sum_{w_l \in C_{w_t}} \left[\frac{\partial}{\partial \mathbf{v}_{w_z}} \mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t} - \frac{\partial}{\partial \mathbf{v}_{w_z}} \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t)}) \right],$$

where the symbol $\ni$ denotes 'such that'. From equation (3) to (4) we use that the objective function is only impacted by a change of $\mathbf{v}_{w_z}$ if the word in the text on place t corresponds to the word w_z. Under the condition that $t \ni w_t = w_z$ it holds that $\frac{\partial}{\partial \mathbf{v}_{w_t}} \mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t} = \mathbf{u}_{w_l}^\intercal$. Elaborating on the part after the minus sign using the chain rule and under the condition that $t \ni w_t = w_z$ gives

$$\frac{\partial}{\partial \mathbf{v}_{w_z}} \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t}) = \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \left[\frac{\partial}{\partial \mathbf{v}_{w_z}} \sum_{k=1}^{v} \exp(\mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_t}) \right] \tag{5}$$

$$= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \left[\sum_{k=1}^{v} \frac{\partial}{\partial \mathbf{v}_{w_z}} \exp(\mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_t}) \right]$$

$$= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \left[\sum_{k=1}^{v} \exp(\mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_t}) \frac{\partial}{\partial \mathbf{v}_{w_z}} \mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_t} \right] \tag{6}$$

$$= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \left[\sum_{k=1}^{v} \exp(\mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_t}) \mathbf{u}_{w_k}^\intercal \right]$$

$$= \sum_{k=1}^{v} \frac{\exp(\mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_t})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \mathbf{u}_{w_k}^\intercal$$

$$= \sum_{k=1}^{v} p(w_k | w_t) \mathbf{u}_{w_k}^\intercal.$$

Using the outcome of the prior intermezzo in the derivation of the partial derivative yields

$$\frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\mathrm{SG}}(\boldsymbol{\theta}) = - \sum_{t \,\ni\, w_t = w_z} \underbrace{\sum_{w_l \in C_{w_t}} \left[\mathbf{u}_{w_l}^\intercal - \sum_{k=1}^{v} p(w_k | w_t) \mathbf{u}_{w_k}^\intercal \right]}_{\text{(I)}} \tag{7}$$

$$= - \sum_{t \,\ni\, w_t = w_z} \sum_{w_l \in C_{w_t}} \sum_{k=1}^{v} \mathbb{1}_{\{w_k = w_l\}} \mathbf{u}_{w_k}^\intercal - p(w_k | w_t) \mathbf{u}_{w_k}^\intercal$$

$$= \sum_{t \,\ni\, w_t = w_z} \sum_{w_l \in C_{w_t}} \sum_{k=1}^{v} \left[p(w_k | w_t) - \mathbb{1}_{\{w_k = w_l\}} \right] \mathbf{u}_{w_k}^\intercal,$$

where $\mathbb{1}_{\{\cdot\}}$ denotes the indicator function with value one if the condition between brackets holds and zero otherwise. The part between square brackets in equation (7) is the difference between a single word in the context of center word w_t and the expectation of the context word vector given that the center word is w_t. Put differently, this is the prediction error of the output context word vector of w_l. Part (I) in equation (7) is the total prediction error of the context of the word w_t. Furthermore, summing this error for all occurrences of the word w_z in the corpus gives us the total prediction error of the contexts of the center word vector of w_z. Decreasing this difference implies optimization of the objective function. If all the derivatives are equal to zero, we reach an optimum.

We continue with the partial derivative derivation of the skip-gram objective function with respect to the context word vectors. Let $\mathbf{u}_{w_z} \in V$ denote an arbitrary context word vector with $z \in \{1, ..., v\}$, then

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{SG}}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \mathbf{u}_{w_z}} \left[-\sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \log \frac{\exp(\mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \right] \\
&= -\sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \frac{\partial}{\partial \mathbf{u}_{w_z}} \log \frac{\exp(\mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \\
&= -\sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \frac{\partial}{\partial \mathbf{u}_{w_z}} \left[\log(\exp(\mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t})) - \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t}) \right] \\
&= -\sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \frac{\partial}{\partial \mathbf{u}_{w_z}} \left[\mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t} - \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t}) \right] \\
&= -\sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \frac{\partial}{\partial \mathbf{u}_{w_z}} \mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t} - \frac{\partial}{\partial \mathbf{u}_{w_z}} \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t}).
\end{aligned}$$

The part before the minus sign can be written as

$$\frac{\partial}{\partial \mathbf{u}_{w_z}} \mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t} = \begin{cases} \mathbf{v}_{w_t}^\intercal, & \text{if } w_z = w_l \\ \mathbf{0}^\intercal, & \text{otherwise.} \end{cases}$$

Next, the part after the minus sign can be rewritten using the chain rule, resulting in

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{u}_{w_z}} \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t}) &= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \left[\frac{\partial}{\partial \mathbf{u}_{w_z}} \sum_{k=1}^{v} \exp(\mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_t}) \right] \qquad (8) \\
&= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \left[\frac{\partial}{\partial \mathbf{u}_{w_z}} \sum_{k=z} \exp(\mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_t}) \right] \\
&= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})} \left[\frac{\partial}{\partial \mathbf{u}_{w_z}} \exp(\mathbf{u}_{w_z}^\intercal \mathbf{v}_{w_t}) \right]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{w_t})} \left[\exp(\mathbf{u}_{w_z}^{\intercal} \mathbf{v}_{w_t}) \frac{\partial}{\partial \mathbf{u}_{w_z}} \mathbf{u}_{w_z}^{\intercal} \mathbf{v}_{w_t} \right] \\
&= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{w_t})} \left[\exp(\mathbf{u}_{w_z}^{\intercal} \mathbf{v}_{w_t}) \mathbf{v}_{w_t}^{\intercal} \right] \\
&= \frac{\exp(\mathbf{u}_{w_z}^{\intercal} \mathbf{v}_{w_t})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{w_t})} \mathbf{v}_{w_t}^{\intercal} \\
&= p(w_z|w_t) \mathbf{v}_{w_t}^{\intercal}.
\end{aligned}
$$

Substitution of the previous two results into the partial derivative of the skip-gram objective function with respect to $\mathbf{u}_{w_z}$ gives

$$
\begin{aligned}
\frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{SG}}(\boldsymbol{\theta}) &= -\sum_{t=1}^{s} \underbrace{\sum_{w_l \in C_{w_t}} \left[\mathbb{1}_{\{w_z = w_l\}} \mathbf{v}_{w_t}^{\intercal} - p(w_z|w_t) \mathbf{v}_{w_t}^{\intercal} \right]}_{\text{(II)}} \qquad (9) \\
&= \sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \left[p(w_z|w_t) - \mathbb{1}_{\{w_z = w_l\}} \right] \mathbf{v}_{w_t}^{\intercal}.
\end{aligned}
$$

The first part between square brackets in (II) is the input center word vector of word w_t if the word w_z occurs in the context window of center word w_t, and zero otherwise. The second part is the input center word vector weighted by the probability that the word w_z is in the context of the center word w_t. Combined, part (II) is the sum of the differences between these two parts over the positions in the context window of center word w_t. It is the prediction error of the input word vector of word w_t as a function of the word w_z. Summing these errors over all positions in the text gives the total error in the center word vectors based on the output word vector of word w_z, as given in equation (9).

3.3.3 Continuous bag-of-words (CBOW)

3.3.3.1 Objective function

The objective function of the CBOW model depends on the conditional probability distribution of the center word occurring given the context words. Using previous definitions, this conditional probability distribution is given by $p(w_t|C_{w_t})$. Let $\mathbf{v}_{C_{w_t}}$ be the input vector combining the input vectors of the words in the context of center word w_t. The most simple approach to construct $\mathbf{v}_{C_{w_t}}$ is to average the input vectors. A weighting depending on the distance to the center word could be an alternative. An argument in favor of such a weighting scheme is that in general words closer to another word have a stronger semantic and syntactical relation. Since in the SG model each word in the context window is treated equally, we continue with the prior definition, that is

$$\mathbf{v}_{C_{w_t}} = \frac{1}{c_{1,w_t} + c_{2,w_t}} \sum_{w_l \in C_{w_t}} \mathbf{v}_{w_l} = \frac{1}{|C_{w_t}|} \sum_{w_l \ \in \ C_{w_t}} \mathbf{v}_{w_l}$$

where $|C_{w_t}|$ denotes the number of context words in the window around w_t. As for the SG model the probability distribution is constructed using the soft-max function. The conditional probability distribution at place t is given by

$$p_{\text{CBOW}}(w_t|C_{w_t}) = \frac{\exp(\mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}})}, \qquad \forall\ i, o \in \{1, ..., v\}.$$

Maximizing this probability at place t is equivalent to minimizing the negative log-likelihood, given by

$$\mathcal{L}_{\text{CBOW},t}(\boldsymbol{\theta}) = J_{\text{CBOW},t}(\boldsymbol{\theta}) = -\log\, p_{\text{CBOW}}(w_t|C_{w_t}). \tag{10}$$

Taking the sum over all positions in the text, just as for the SG model, gives the objective function for the CBOW model

$$\begin{aligned} J_{\text{CBOW}}(\boldsymbol{\theta}) &= \sum_{t=1}^{s} \mathcal{L}_{\text{CBOW},t}(\boldsymbol{\theta}) \\ &= -\sum_{t=1}^{s} \log\, p_{\text{CBOW}}(w_t|C_{w_t}) \\ &= -\sum_{t=1}^{s} \log \frac{\exp(\mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}}} \\ &= -\sum_{t=1}^{s} \log \left[\frac{\exp\left(\mathbf{u}_{w_t}^\intercal \frac{1}{|C_{w_t}|} \sum_{w_l \in C_{w_t}} \mathbf{v}_{w_l}\right)}{\sum_{j=1}^{v} \exp\left(\mathbf{u}_{w_j}^\intercal \frac{1}{|C_{w_t}|} \sum_{w_l \in C_{w_t}} \mathbf{v}_{w_l}\right)} \right]. \end{aligned}$$

3.3.3.2 Gradient derivation

The derivative of the objective function with respect to the output vector of word w_z with $z \in \{1, ..., s\}$ can be written as

$$\begin{aligned} \frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{CBOW}}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \mathbf{u}_{w_z}} \left[-\sum_{t=1}^{s} \log \frac{\exp(\mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}})} \right] \\ &= -\sum_{t=1}^{s} \frac{\partial}{\partial \mathbf{u}_{w_z}} \log \frac{\exp(\mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}})} \end{aligned}$$

$$
\begin{aligned}
&= -\sum_{t=1}^{s} \frac{\partial}{\partial \mathbf{u}_{w_z}} \left[\log(\exp(\mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}})) - \log\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}}) \right] \\
&= -\sum_{t=1}^{s} \frac{\partial}{\partial \mathbf{u}_{w_z}} \left[\mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}} - \log\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}}) \right] \\
&= -\sum_{t=1}^{s} \frac{\partial}{\partial \mathbf{u}_{w_z}} \mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}} - \frac{\partial}{\partial \mathbf{u}_{w_z}} \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}}) \qquad (11)
\end{aligned}
$$

The part before the minus sign is simply

$$
\frac{\partial}{\partial \mathbf{u}_{w_z}} \mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}} = \begin{cases} \mathbf{v}_{C_{w_t}}^\intercal, & \text{if } w_z = w_t \\ \mathbf{0}^\intercal, & \text{otherwise.} \end{cases}
$$

The part after the minus sign is similar to equation (8), and hence we skip some in between steps of the derivation. It holds that

$$
\begin{aligned}
\frac{\partial}{\partial \mathbf{u}_{w_z}} \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}}) &= \frac{\exp(\mathbf{u}_{w_z}^\intercal \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}})} \mathbf{v}_{C_{w_t}}^\intercal \\
&= p(w_z | C_{w_t}) \mathbf{v}_{C_{w_t}}^\intercal .
\end{aligned}
$$

Substitution of the previous two results into equation (11) gives

$$
\begin{aligned}
\frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{CBOW}}(\boldsymbol{\theta}) &= -\sum_{t=1}^{s} \underbrace{\left[\mathbb{1}_{\{w_z = w_t\}} \mathbf{v}_{C_{w_t}}^\intercal - p(w_z | C_{w_t}) \mathbf{v}_{C_{w_t}}^\intercal \right]}_{\text{(III)}} \qquad (12) \\
&= \sum_{t=1}^{s} \left[p(w_z | C_{w_t}) - \mathbb{1}_{\{w_z = w_t\}} \right] \mathbf{v}_{C_{w_t}}^\intercal .
\end{aligned}
$$

If the word w_z is the center word, part (III) in equation (12) is the difference between the context word vector and the context word vector weighted by the probability that the word w_z is the center word given the context. If w_z is not the center word, (III) is the difference between the zero vector and the context word vector weighted by the probability that the word w_z is the center word vector. Put differently, part (III) is the prediction error of the combined input word vector at position t as a function of w_z. Summing this error over all positions in the corpus yields the total prediction error of $\mathbf{v}_{C_{w_t}}$ as a function of the w_z. This summation corresponds to the formula in equation (12).

Taking the derivative of the objective function with respect to an arbitrary input word vector $\mathbf{v}_{w_z}$ yields

$$\begin{aligned}\frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{CBOW}}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \mathbf{v}_{w_z}} \left[-\sum_{t=1}^{s} \log \frac{\exp(\mathbf{u}_{w_t}^{\intercal} \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}})} \right] \\ &= -\sum_{t=1}^{s} \frac{\partial}{\partial \mathbf{v}_{w_z}} \log \frac{\exp(\mathbf{u}_{w_t}^{\intercal} \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}})} \\ &= -\sum_{t \ \ni \ w_z \ \in \ C_{w_t}} \frac{\partial}{\partial \mathbf{v}_{w_z}} \log \frac{\exp(\mathbf{u}_{w_t}^{\intercal} \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}})} \\ &= -\sum_{t \ \ni \ w_z \ \in \ C_{w_t}} \frac{\partial}{\partial \mathbf{v}_{w_z}} \left[\log\ (\exp(\mathbf{u}_{w_t}^{\intercal} \mathbf{v}_{C_{w_t}})) - \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}}) \right] \\ &= -\sum_{t \ \ni \ w_z \ \in \ C_{w_t}} \frac{\partial}{\partial \mathbf{v}_{w_z}} \left[\mathbf{u}_{w_t}^{\intercal} \mathbf{v}_{C_{w_t}} - \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}}) \right] \\ &= -\sum_{t \ \ni \ w_z \ \in \ C_{w_t}} \frac{\partial}{\partial \mathbf{v}_{w_z}} \mathbf{u}_{w_t}^{\intercal} \mathbf{v}_{C_{w_t}} - \frac{\partial}{\partial \mathbf{v}_{w_z}} \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}}) \end{aligned} \tag{13}$$

Rewriting the part before the minus sign from equation (13) under the condition that $t \ \ni \ w_z \ \in \ C_{w_t}$ yields

$$\frac{\partial}{\partial \mathbf{v}_{w_z}} \mathbf{u}_{w_t}^{\intercal} \mathbf{v}_{C_{w_t}} = \frac{\partial}{\partial \mathbf{v}_{w_z}} \mathbf{u}_{w_t}^{\intercal} \frac{1}{|C_{w_t}|} \sum_{w_l \in C_{w_t}} \mathbf{v}_{w_l} = \frac{1}{|C_{w_t}|} \mathbf{u}_{w_t}^{\intercal} \tag{14}$$

where in the last step is used that $w_z = w_l$ for one word in the context of w_t due to the condition. Elaborating the part after the minus sign from equation (13) under the conditions that $t \ \ni \ w_z \ \in \ C_{w_t}$ and using the same rationale as in equation (5) - (6) and equation (14) gives

$$\begin{aligned}\frac{\partial}{\partial \mathbf{v}_{w_z}} \log \sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}}) &= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}})} \left[\sum_{k=1}^{v} \exp(\mathbf{u}_{w_k}^{\intercal} \mathbf{v}_{C_{w_t}}) \frac{\partial}{\partial \mathbf{v}_{w_z}} \mathbf{u}_{w_k}^{\intercal} \mathbf{v}_{C_{w_t}} \right] \\ &= \frac{1}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}})} \left[\sum_{k=1}^{v} \exp(\mathbf{u}_{w_k}^{\intercal} \mathbf{v}_{C_{w_t}}) \frac{1}{|C_{w_t}|} \mathbf{u}_{w_k}^{\intercal} \right] \\ &= \sum_{k=1}^{v} \frac{\exp(\mathbf{u}_{w_k}^{\intercal} \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}})} \frac{1}{|C_{w_t}|} \mathbf{u}_{w_k}^{\intercal} \\ &= \frac{1}{|C_{w_t}|} \sum_{k=1}^{v} p(w_k | C_{w_t}) \mathbf{u}_{w_k}^{\intercal}. \end{aligned}$$

Substitution of the previous two results into equation (13) gives

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{CBOW}}(\boldsymbol{\theta}) &= - \sum_{t \ \ni \ w_z \ \in \ C_{w_t}} \frac{1}{|C_{w_t}|}\mathbf{u}_{w_t}^{\mathsf{T}} - \frac{1}{|C_{w_t}|}\sum_{k=1}^{v} p(w_k|C_{w_t})\mathbf{u}_{w_k}^{\mathsf{T}} \\
&= - \sum_{t \ \ni \ w_z \ \in \ C_{w_t}} \frac{1}{|C_{w_t}|}\left[\underbrace{\mathbf{u}_{w_t}^{\mathsf{T}} - \sum_{k=1}^{v} p(w_k|C_{w_t})\mathbf{u}_{w_k}^{\mathsf{T}}}_{\text{(IV)}}\right] \qquad (15) \\
&= - \sum_{t \ \ni \ w_z \ \in \ C_{w_t}} \frac{1}{|C_{w_t}|}\sum_{k=1}^{v} \mathbb{1}_{\{w_t=w_k\}}\mathbf{u}_{w_k}^{\mathsf{T}} - p(w_k|C_{w_t})\mathbf{u}_{w_k}^{\mathsf{T}} \\
&= \sum_{t \ \ni \ w_z \ \in \ C_{w_t}} \frac{1}{|C_{w_t}|}\sum_{k=1}^{v} \left[p(w_k|C_{w_t}) - \mathbb{1}_{\{w_t=w_k\}}\right]\mathbf{u}_{w_k}^{\mathsf{T}}.
\end{aligned}$$

Part (IV) from equation (15) is the difference between the output vector of the word w_t and the expected value of that output vector given the context of the word at position t. Put differently, part (IV) is the total prediction error of the output word vector of word w_t given the context of word w_t.

3.3.4 Gradient update

The resulting first order derivatives to construct the full gradients of the two versions of the `word2vec` model are

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{SG}}(\boldsymbol{\theta}) &= \sum_{t \ \ni \ w_t=w_z} \sum_{w_l \in C_{w_t}} \sum_{k=1}^{v} \left[p(w_k|w_t) - \mathbb{1}_{\{w_k=w_l\}}\right]\mathbf{u}_{w_k}^{\mathsf{T}} \\
\frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{SG}}(\boldsymbol{\theta}) &= \sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \left[p(w_z|w_t) - \mathbb{1}_{\{w_z=w_l\}}\right]\mathbf{v}_{w_t}^{\mathsf{T}} \\
\frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{CBOW}}(\boldsymbol{\theta}) &= \sum_{t \ \ni \ w_z \ \in \ C_{w_t}} \frac{1}{|C_{w_t}|}\sum_{k=1}^{v} \left[p(w_k|C_{w_t}) - \mathbb{1}_{\{w_t=w_k\}}\right]\mathbf{u}_{w_k}^{\mathsf{T}} \\
\frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{CBOW}}(\boldsymbol{\theta}) &= \sum_{t=1}^{s} \left[p(w_z|C_{w_t}) - \mathbb{1}_{\{w_z=w_t\}}\right]\mathbf{v}_{C_{w_t}}^{\mathsf{T}}.
\end{aligned}$$

Computing the full gradient before updating the parameters is computational expensive and hence training the word vectors takes long. To improve the speed one can use an approximation of the gradient. As approximation one can use the partial derivative based on one window in the corpus. This approximation can then be used as input for the gradient descent. After updating the word vectors accordingly, a new gradient is calculated based on the next window. This results in optimization of the objective function at time t as provided in equation (2) and (10). This approach is also referred to as Stochastic

Gradient Descent (SGD) as discussed in Section 3.2. The approximate gradients for both models using updates for one window at a time are given by

$$\frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{SG},t}(\boldsymbol{\theta}) = \begin{cases} \sum_{w_l \in C_{w_t}} \sum_{k=1}^{v} \left[p(w_k|w_t) - \mathbb{1}_{\{w_k = w_l\}} \right] \mathbf{u}_{w_k}^{\intercal}, & \text{if } w_z \text{ is the center word at } t \\ \mathbf{0}^{\intercal}, & \text{otherwise} \end{cases} \tag{16}$$

$$\frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{SG},t}(\boldsymbol{\theta}) = \sum_{w_l \in C_{w_t}} \left[p(w_z|w_t) - \mathbb{1}_{\{w_z = w_l\}} \right] \mathbf{v}_{w_t}^{\intercal}$$

$$\frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{CBOW},t}(\boldsymbol{\theta}) = \begin{cases} \frac{1}{|C_{w_t}|} \sum_{k=1}^{v} \left[p(w_k|C_{w_t}) - \mathbb{1}_{\{w_k = w_t\}} \right] \mathbf{u}_{w_k}^{\intercal}, & \text{if } w_z \text{ is in the context of word at } t \\ \mathbf{0}^{\intercal}, & \text{otherwise} \end{cases}$$

$$\frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{CBOW},t}(\boldsymbol{\theta}) = \left[p(w_z|C_{w_t}) - \mathbb{1}_{\{w_z = w_t\}} \right] \mathbf{v}_{C_{w_t}}^{\intercal}.$$

The resulting derivatives are the same as provided by Rong (2014). The notation of Rong (2014) is slightly different; switching the summation signs in equation (16) results in the notation as given by Rong (2014). For a learning rate of η, the vector updates using gradient descent for any word w_z at position t in the text are obtained with the updating equations

$$\begin{aligned} \mathbf{v}_{w_z}^{(\text{new})} &= \mathbf{v}_{w_z}^{(\text{old})} - \eta \frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{SG},t}(\boldsymbol{\theta}) \\ \mathbf{u}_{w_z}^{(\text{new})} &= \mathbf{u}_{w_z}^{(\text{old})} - \eta \frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{SG},t}(\boldsymbol{\theta}) \\ \mathbf{v}_{w_z}^{(\text{new})} &= \mathbf{v}_{w_z}^{(\text{old})} - \eta \frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{CBOW},t}(\boldsymbol{\theta}) \\ \mathbf{u}_{w_z}^{(\text{new})} &= \mathbf{u}_{w_z}^{(\text{old})} - \eta \frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{CBOW},t}(\boldsymbol{\theta}). \end{aligned}$$

3.4 GloVe

The GloVe method introduced by Pennington et al. (2014) is based on a combination of the local context window (as is the word2vec approach) and the word-word co-occurrence matrix. The authors argue that the global vectors combine the best of both worlds, that is, making use of the co-occurrence statistics of the entire corpus while capturing complex linguistic relationships. Previous word count based methods such as LSA and HPCA apply dimension reduction on the word-word co-occurrence matrix. Using the count statistics in such a way is more efficient compared to word2vec. Namely, the number of iterations in GloVe is equal to the number of unique words while the word2vec algorithm iterates over all, possibly non-unique, words in the corpus. The word2vec algorithm evaluates the word co-occurrences separately in each window and updates the word vectors many times. The `GloVe` algorithm collects the counts only once and stores the word-word co-occurrence matrix, and subsequent dimension reduction is applied. In count based methods, frequent words tend to receive relatively high importance while these words often contain little semantic value (e.g., words like ‘the’ and ‘have’). The `GloVe` model tries to deal with this problem by using a count statistics based on combining all local context windows in the corpus instead of the normal counts, down-weighting the impact of frequent words. A disadvantage of `Glove` is that, since it is a count based method, the algorithm requires to store the word-word occurrence matrix in memory, which might cause problems if one is dealing with a large vocabulary. Besides, if new text data is added, one needs to reconstruct the word-word co-occurrence matrix.

Roughly following the notation introduced by Pennington et al. (2014), we define that $\mathbf{X} \in \mathbb{Z}_{\geq}^{v \times v}$ is the word-word co-occurrence matrix, where v is the size of the vocabulary V, $\mathbb{Z}_{\geq}$ is the set of non-negative whole numbers, and $X_{i,j}$ is the co-occurrence count of the word w_i and w_j. The matrix $\mathbf{X}$ is a symmetric matrix and constructed by counting the times two words co-occur within user defined context windows. The number of total occurrences of a word in the corpus is defined as $X_i = \sum_k X_{i,k} = \sum_k X_{k,i}$. Furthermore $P_{i,j} = P(j|i) = \frac{X_{i,j}}{X_i}$ is defined as the probability that word w_j occurs in the context of word w_i.

Pennington et al. (2014) argue that the ratio of co-occurrence probabilities $\frac{P_{i,k}}{P_{j,k}}$ is a better representation to be able to distinguish relevant and irrelevant words. This ratio is based on three words instead of two and therefore, captures better the interconnection between the meaning of words. Computing the ratio of the co-occurrence probabilities of two words w_i and w_j with multiple third words (w_k) provides the degree of relevance between these two words. If these ratios are close to one, w_i and w_j are both equally linked to w_k. It could be that w_i and w_j are either both relevant or irrelevant to w_k. If the value of the ratio is either small or large, one of the two is related to w_k, and the other one is not. Since the goal is to capture the information of the words in the vector space, Pennington et al. (2014) argue that the vector difference between the two words should capture this information and is ‘the most natural choice.’ Furthermore, for all word vectors, each dimension should be a measure of the information on that part of the vector space. Suppose there is a dimension that captures how likely it is that a word is a noun, then this dimension should be the same for all word vectors to be able to compare words linearly. Therefore, just as in the `word2vec` method, the dot product is used as a measure of similarity. The model of `GloVe` can be written as

$$\frac{P_{i,k}}{P_{j,k}} = F\left((\mathbf{u}_{w_i} - \mathbf{u}_{w_j})^\intercal \mathbf{v}_{w_k}\right) = F\left(\mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k} - \mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_k}\right), \tag{17}$$

where the function F is a model choice and a suitable option will be derived in the next steps. Pennington et al. (2014) choose F using the following reasoning. The definition of $\mathbf{u}$ being the input or ordinary word vector and $\mathbf{v}$ the output or context word vector is meaningless since the word vectors are based on the word count statistics. Therefore, these vectors should be interchangeable and the label of context word versus ordinary word should not have an impact. Put differently, changing $\mathbf{v}_{w_k}$ with either $\mathbf{u}_{w_i}$ or $\mathbf{u}_{w_j}$ should still hold with the same mapping F. The argument of F after the first equal sign in equation (17) is the additive group of real numbers $(\mathbb{R}, +)$, since both inner products are real-valued. The function F maps to a product of two real numbers ($P_{i,k}$ and $\frac{1}{P_{j,k}}$), the multiplicative group of real numbers $(\mathbb{R}_{>0}, \times)$. Therefore, if we require F to be a homomorphism between the groups, the ordinary and context labels are interchangeable. The condition of a homomorphism F between addition in the domain space $(\mathbb{R}, +)$ and multiplication in the positive target space $(\mathbb{R}_{>0}, \times)$ is given by

$$F(a+b) = F(a)F(b), \qquad \forall\ a, b \in \mathbb{R}. \tag{18}$$

The subtraction equivalent of equation (18) is given by

$$F(a-b) = \frac{F(a)}{F(b)}, \qquad \forall\ a, b \in \mathbb{R}.$$

Substitution of $a = \mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k}$ and $b = \mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_k}$ yields the condition for the mapping F that

$$F\left(\mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k} - \mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_k}\right) = \frac{F(\mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k})}{F(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_k})}, \qquad \forall\ \mathbf{u}_{w_i}, \mathbf{u}_{w_j}, \mathbf{v}_{w_k} \in \mathbb{R}^d. \tag{19}$$

A solution of equation (19) is $F(\cdot) = \exp(\cdot)$. Combining this solution with equation (17) yields

$$\begin{aligned}
\frac{F(\mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k})}{F(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_k})} = \frac{\exp(\mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k})}{\exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_k})} = \frac{P_{i,k}}{P_{j,k}} &\Rightarrow P_{i,k} = \exp(\mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k}) \\
&\Leftrightarrow \frac{X_{i,k}}{X_i} = \exp(\mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k}) \\
&\Leftrightarrow \log\left(\frac{X_{i,k}}{X_i}\right) = \mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k} \\
&\Leftrightarrow \mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_k} = \log\left(X_{i,k}\right) - \log\left(X_i\right), \qquad \forall\ i, k \in \{1, ..., v\}.
\end{aligned} \tag{20}$$

The solution (20) is not symmetric for i and k, changing i and k results in a different solution. Therefore, Pennington et al. (2014) replace the asymmetric part of (20), $\log\left(X_i\right)$, with a bias b_i related to the ordinary word vector of word w_i and they introduce an additional bias c_k related to the context word vector of word w_k to establish symmetry.

Furthermore, shifting the logarithm by adding 1 to $X_{i,k}$ for all i, k solves the issue that $\log(0)$ is not defined, resulting in the final `GloVe` model equation

$$\mathbf{u}_{w_i}^{\intercal}\mathbf{v}_{w_k} + b_i + c_k = \log\left(1 + X_{i,k}\right). \tag{21}$$

3.4.1 Objective function

The objective function introduced by Pennington et al. (2014) is a least squares problem and is constructed using the model defined in equation (21). Pennington et al. (2014) introduce an additional weighting function as a solution to the relatively large impact of frequent words. This weighting function down-weights the impact of frequent words in the objective function. Thereby, infrequent words have a relatively larger impact on the objective value. Besides introducing a weighting function, removing stop words from the corpus can deal with the problem for a few specific words. Let f be such a weighting function such that $f : \mathbb{Z}_{\geq} \mapsto \mathbb{R}_{+}$. Then the objective function is given by

$$J_{\text{GloVe}}(\boldsymbol{\theta}) = \sum_{i,j} f\left(X_{i,j}\right)\left[\mathbf{u}_{w_i}^{\intercal}\mathbf{v}_{w_j} + b_i + c_j - \log(1 + X_{i,j})\right]^2, \tag{22}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{2dv}$ contains the ordinary and context word vector of all words in the vocabulary. f should be non-decreasing and relatively small for larger values to still assigning more impact to frequent words and at the same time satisfy the goal of reducing the impact of frequent words on the objective value. Furthermore, if f is continuous, f should approach zero fast enough such that $f(x)\log^2 x$ has a finite limit, as $x \to 0$. If not, the parameters in the objective function will compensate for the log-term as $x \to 0$, and therefore, the magnitude of the vector elements and the biases might explode to minimize the value with the square brackets in equation (22). The last condition is that if x is 0, the value of $f(x)$ is equal to zero.

Pennington et al. (2014) suggest the following weighting function satisfying the conditions as mentioned earlier

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{otherwise,} \end{cases}$$

where x_{max} and α are hyperparameters of the model. The authors suggest fix $x_{max} = 100$ without motivation and they empirically motivate their choice for $\alpha = 0.75$ over $\alpha = 1$. The objective function in equation (22) might seem similar to the objective of SVD. However, it seems inappropriate to go into details on the (dis)similarities between the two here, and therefore, we have included more details on this link in section B of the Appendix.

3.4.2 Gradient derivation

We apply the chain rule to obtain partial derivatives. Note that the parameter $\boldsymbol{\theta}$ of the `GloVe` model consists of four parts: the two word vector representations, and the two corresponding bias terms, and thereby, we consider four partial derivatives. Let w_z denote an arbitrary word of the vocabulary, then the partial derivative of the objective function with respect to the ordinary word vector of w_z is as follows

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{GloVe}}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \mathbf{v}_{w_z}} \sum_{i,j} f\left(X_{i,j}\right) \left[\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_j} + b_i + c_j - \log(1 + X_{i,j})\right]^2 \\
&= \frac{\partial}{\partial \mathbf{v}_{w_z}} \sum_{i} \sum_{j=z} f\left(X_{i,j}\right) \left[\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_j} + b_i + c_j - \log(1 + X_{i,j})\right]^2 \\
&= \sum_{i} \sum_{j=z} \frac{\partial}{\partial \mathbf{v}_{w_z}} f\left(X_{i,j}\right) \left[\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_j} + b_i + c_j - \log(1 + X_{i,j})\right]^2 \\
&= \sum_{i} \frac{\partial}{\partial \mathbf{v}_{w_z}} f\left(X_{i,z}\right) \left[\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_z} + b_i + c_z - \log(1 + X_{i,z})\right]^2 \\
&= \sum_{i} f\left(X_{i,z}\right) \frac{\partial}{\partial \mathbf{v}_{w_z}} \left[\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_z} + b_i + c_z - \log(1 + X_{i,z})\right]^2 \\
&= \sum_{i} 2 f\left(X_{i,z}\right) \left(\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_z} + b_i + c_z - \log(1 + X_{i,z})\right) \frac{\partial}{\partial \mathbf{v}_{w_z}} \mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_z} \\
&= 2 \sum_{i} f\left(X_{i,z}\right) \left(\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_z} + b_i + c_z - \log(1 + X_{i,z})\right) \mathbf{u}_{w_i}^{\intercal}.
\end{aligned}$$

The same approach in the derivation can be used to obtain the partial derivative with respect to $\mathbf{u}_{w_z}$, resulting in

$$\frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{GloVe}}(\boldsymbol{\theta}) = 2 \sum_{j} f\left(X_{z,j}\right) \left(\mathbf{u}_{w_z}^{\intercal} \mathbf{v}_{w_j} + b_z + c_j - \log(1 + X_{z,j})\right) \mathbf{v}_{w_j}^{\intercal}.$$

Next, we consider the partial derivatives with respect to the biases. The approach is similar as for the partial derivative derivation with respect to the word vectors, so some intermediate steps are skipped, resulting in

$$\begin{aligned}
\frac{\partial}{\partial c_z} J_{\text{GloVe}}(\boldsymbol{\theta}) &= \frac{\partial}{\partial c_z} \sum_{i,j} f\left(X_{i,j}\right) \left[\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_j} + b_i + c_j - \log(1 + X_{i,j})\right]^2 \\
&= \sum_{i} \sum_{j=z} \frac{\partial}{\partial c_z} f\left(X_{i,j}\right) \left[\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_j} + b_i + c_j - \log(1 + X_{i,j})\right]^2 \\
&= \sum_{i} f\left(X_{i,z}\right) \frac{\partial}{\partial c_z} \left[\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_z} + b_i + c_z - \log(1 + X_{i,z})\right]^2 \\
&= 2 \sum_{i} f\left(X_{i,z}\right) \left(\mathbf{u}_{w_i}^{\intercal} \mathbf{v}_{w_z} + b_i + c_z - \log(1 + X_{i,z})\right).
\end{aligned}$$

A similar derivation with respect to b_z gives

$$\frac{\partial}{\partial b_z} J_{\text{GloVe}}(\boldsymbol{\theta}) = 2\sum_j f\left(X_{z,j}\right)\left(\mathbf{u}_{w_z}^{\intercal}\mathbf{v}_{w_j} + b_z + c_j - \log(1 + X_{z,j})\right).$$

3.4.3 Gradient update

SGD uses the partial derivatives based on a single observation. We obtained these partial derivatives by simply removing the summation and adding an index. The partial derivatives are

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{v}_{w_z}} J_{\text{GloVe},i}(\boldsymbol{\theta}) &= 2f\left(X_{i,z}\right)\left(\mathbf{u}_{w_i}^{\intercal}\mathbf{v}_{w_z} + b_i + c_z - \log(1 + X_{i,z})\right)\mathbf{u}_{w_i}^{\intercal} \\
\frac{\partial}{\partial \mathbf{u}_{w_z}} J_{\text{GloVe},j}(\boldsymbol{\theta}) &= 2f\left(X_{z,j}\right)\left(\mathbf{u}_{w_z}^{\intercal}\mathbf{v}_{w_j} + b_z + c_j - \log(1 + X_{z,j})\right)\mathbf{v}_{w_j}^{\intercal} \\
\frac{\partial}{\partial c_z} J_{\text{GloVe},i}(\boldsymbol{\theta}) &= 2f\left(X_{i,z}\right)\left(\mathbf{u}_{w_i}^{\intercal}\mathbf{v}_{w_z} + b_i + c_z - \log(1 + X_{i,z})\right) \\
\frac{\partial}{\partial b_z} J_{\text{GloVe},j}(\boldsymbol{\theta}) &= 2f\left(X_{z,j}\right)\left(\mathbf{u}_{w_z}^{\intercal}\mathbf{v}_{w_j} + b_z + c_j - \log(1 + X_{z,j})\right).
\end{aligned}$$

The parameter updates, considering a learning rate η are similar to the updates for `word2vec` in Section 3.3.4 and not provided explicitly for brevity.

3.5 Glossary

Variable	Definition
d	Dimension of a word vector
$\boldsymbol{\theta}$	Parameter vector (containing all input/ordinary and output/context word vectors)
S	Set of words constructing the corpus, a sequence of words
s	Size of the corpus, number of words in the corpus
V	Vocabulary, set of unique words in the corpus
v	Size of the vocabulary, number of unique words in the corpus
t	Position of a word in the text data where $t \in \{1, ..., s\}$
w_t with $t \in \{1, ..., s\}$	Word at position t in the text data
w_i with $i \in \{1, ..., v\}$	Word i from the vocabulary
C_{w_t}	Set of context words of the word at position t in the text
c_{1,w_t}	Number of words to the left of w_t that are included in the context of word w_t
c_{2,w_t}	Number of words to the right of w_t that are included in the context of w_t
$\mathbf{u}_{w_z}$	Input/ordinary word vector of word w_z
$\mathbf{v}_{w_z}$	Output/context word vector of word w_z
$\mathbf{U}$	Matrix of input/ordinary word vectors
$\mathbf{V}$	Matrix of output/context word vectors
$\mathbf{X}$	Matrix of word-word co-occurrence counts

Table 1: Glossary with definitions of frequently used variables

4 Methodology: beyond the base implementation

This section covers extensions of the basic ideas of `word2vec` and `GloVe` to speed the optimization algorithms and to decrease the computational intensity of the methods (Section 4.1). The extension provided in Section 4.1.2.2 is used for both `word2vec` implementations in the application of this paper to compare the methods and therefore, slightly more detailed. Next, we discuss the link between `word2vec` and `GloVe` (Section 4.2). Besides, this part of the methodology covers the model stability and the factors impacting this stability (Section 4.3). Furthermore, we discuss how these methods can be applied in practice in a food menu context (Section 4.4). This part of the methodology includes intuitive explanations, and it does not provide mathematical derivations and proofs. However, we provide pointers to research containing these proofs.

4.1 Estimation speed improvement

4.1.1 Function optimization algorithm

The problem of function optimization is well-known in the literature. Therefore, classical gradient descent can be changed to another optimization algorithm to improve the convergence speed or to find a better optimum. Ruder (2016) provides an overview of such procedures. The algorithms of both `GloVe` and `word2vec` use a variation of SGD to optimize the non-convex objective function. Since the objective functions are non-convex, the solutions are likely to be local minima. We make use of the optimizations used in the Python implementations of the word embedding techniques. The optimizations used in this paper are AdaGrad (Duchi et al., 2011) and SGD with linearly decaying learning rate for `GloVe` and `word2vec`, respectively. Both methods decrease the learning rate over iterations. Moreover, AdaGrad adjusts the learning rate for different dimensions. The gradients used in previous updates combined influence the next adjustment. SGD with linearly decaying learning rate decays the learning rate as the algorithm iterates over the text. In the algorithm used for this paper, the linear decrease continues to a small value such that it will never reach zero. This lower bound ensures that all training data is used to train the word embeddings. For the gradient update formulas see Ruder (2016).

4.1.2 `word2vec`

Updating the full gradient of the CBOW and SG model at each position in the text is a costly operation. An option to solve this is to update the gradient using limited data, based on one or a few positions in the text. Nevertheless, computing the probability p_{SG} and p_{CBOW} in the objective functions of `word2vec` with respect to all words in the context of a center word is still computationally expensive. Suppose the vocabulary size is 10.000, for each context word the evaluation of $p(w_l|w_t)$ in the SG objective is required, including calculating the denominator $\sum_{j=1}^{v} \exp(\mathbf{u}_{w_j}^{\intercal}\mathbf{v}_{w_i})$. This denominator part includes a dot-product of the center word with all 10.000 words in the vocabulary. Doing this at each position of the text is a computationally expensive operation. The gradients of the `word2vec` model include this factor as well (it does not cancel in the gradient derivation) and hence evaluating these gradients is expensive as well. As Mikolov et al. (2013b) mention, the cost of computing the gradient is proportional to the vocabulary size, which is often large ($10^5 - 10^7$ terms).

There are two approaches to solving this problem. The first approach is to speed the calculation of the soft-max while keeping the soft-max form in the objective function. The second approach abandons the soft-max form and uses a sampling-based procedure. Mikolov et al. (2013a) and Mikolov et al. (2013b) suggest one of each, hierarchical soft-max and negative sampling. Rong (2014) discusses both approaches in more details; here, only the basic intuition of the approaches is provided.

4.1.2.1 Hierarchical soft-max

The hierarchical soft-max approach uses a binary tree structure to represent all words in the vocabulary. Given the input, one walks down the tree going either left or right at each node. The probability of going left or right is the value of the sigmoid function, evaluated at the value resulting from the inner product of the node vector with the center word vector. The probability of ending at a context word (that is at a given leaf) is, instead of the full soft-max form, simply the multiplication of the probabilities going left and right, following the path such that one ends at the word observed as the context word.

4.1.2.2 Negative sampling

Mikolov et al. (2013b) present negative sampling as an alternative to hierarchical soft-max. Since their results show that negative sampling performs superior in their application, we elaborate on this method, and we use this approach in our application. The principle of comparing the center word with the positive word (the true context word) and with a set of negative words (words from the vocabulary that are not in the context) is at the core of negative sampling. Thereby, a method with negative sampling makes a limited amount of comparisons, only with the positive word and negative samples. Note that the full soft-max form compares the center word with all words in the vocabulary. The negative samples are a representation of the words that are not in the context. Noise-Contrastive Estimation (NCE) introduced by Gutmann and Hyvärinen (2010) precedes negative sampling. NCE, in short, modifies the form of the objective function such that the optimum is reached with the same parameter values and similarly simplifies the objective function. Negative sampling simplifies the objective function of NCE by replacing the most computationally intensive part with a constant and thereby lets go of the property of NCE that the objective is in the limit the initial skip-gram objective (Dyer, 2014). The SG and CBOW model with negative sampling are respectively referred to as SGNS and CBOWNS. The corresponding objective functions are

$$J_{SGNS}(\boldsymbol{\theta}) = -\sum_{t=1}^{s} \sum_{w_l \in C_{w_t}} \log \underbrace{\sigma(\mathbf{u}_{w_l}^{\intercal} \mathbf{v}_{w_t})}_{\text{(XIII)}} + \sum_{j=1}^{k} \mathop{\mathbb{E}}_{w_j \sim P_D(w)} \left[\log \underbrace{\sigma(-\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{w_t})}_{\text{(XIV)}} \right] \tag{23}$$

$$J_{CBOWNS}(\boldsymbol{\theta}) = -\sum_{t=1}^{s} \log \sigma(\mathbf{u}_{w_t}^{\intercal} \mathbf{v}_{C_{w_t}}) + \sum_{j=1}^{k} \mathop{\mathbb{E}}_{w_j \sim P_D(w)} \left[\log \sigma(-\mathbf{u}_{w_j}^{\intercal} \mathbf{v}_{C_{w_t}}) \right],$$

where $P_D(w)$ is the noise distribution and $\sigma(\cdot)$ the sigmoid function. A detailed derivation of the `word2vec` SGNS is provided by Goldberg and Levy (2014). There are k words w_j sampled from this distribution, these are the negative samples. According to Levy et al. (2015), the number of negative samples k has two functions. First, larger k implies more data used to train the word embeddings and thereby more accurate word embeddings. Second, words that do not occur together are scattered more in the embedding space because of an increasing proportion of negative versus positive samples. Levy et al. (2015) mention that with `word2vec` SGNS many negative samples are preferred. The disadvantage of a large k is the increase in training time. Since it outperforms the regular unigram and the uniform distributions according to their analysis, Mikolov et al. (2013b) propose to use the unigram distribution raised to the power 0.75. Consider that D is the collection of all observed word-context pairs and let $|D|$ be the number of pairs in D. Furthermore, let $\#(w_i, c_j)$, $\#(w_i)$, and $\#(c_j)$ be the count of co-occurrences and single occurrences of the center word w_i and context word c_j in D respectively. Then the proposed noise distribution is

$$P_D(w_j) = \left(\frac{\#(w_j)}{|D|} \right)^{0.75}, \qquad \forall j \in \{1, ..., v\}. \tag{24}$$

As explained by Levy and Goldberg (2014), the skip-gram negative sampling objective is to maximize the sigmoid probability of an observed pair (w_i, w_l), given by part (XIII) in equation (23). Simultaneously, `word2vec` SGNS maximizes probabilities that w_i does not co-occur with the negative samples. The probability that w_i does not co-occur with one of the negative samples is given by part (XIV) in equation (23). Note that simple algebra shows that $\sigma(-a) = 1 - \sigma(a)$. Similar reasoning applies for CBOWNS, but there the input vectors are the combined context vectors as discussed in Section 3.3. The updating equations for the negative sampling are not provided in this paper, but can be derived in like manner as the derivation of the updating equation of the basic `word2vec`

updating equations. Rong (2014) provides a short derivation of these equations. Table 2 summarizes the objective functions discussed in this paper.
Researchers tried to rewrite the `word2vec` into a matrix factorization problem. Levy and Goldberg (2014) succeed in this by showing that `word2vec` SGNS is implicitly a factorization of the word-context matrix. Since this relation goes beyond the scope of our research we have included a summary of this link in section C of the Appendix.

Method	Objective Function
word2vec SG	$\sum_{t=1}^{s}\sum_{w_l\in C_{w_t}} \log \frac{\exp(\mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t})}{\sum_{j=1}^{v}\exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})}$
word2vec CBOW	$\sum_{t=1}^{s}\log \frac{\exp(\mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}})}{\sum_{j=1}^{v}\exp(\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}}}$
word2vec SGNS	$\sum_{t=1}^{s}\sum_{w_l\in C_{w_t}} \log \sigma(\mathbf{u}_{w_l}^\intercal \mathbf{v}_{w_t}) + \sum_{j=1}^{k}\mathbb{E}_{w_j\sim P_D(w)}\left[\log \sigma(-\mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_t})\right]$
word2vec CBOWNS	$\sum_{t=1}^{s}\log \sigma(\mathbf{u}_{w_t}^\intercal \mathbf{v}_{C_{w_t}}) + \sum_{j=1}^{k}\mathbb{E}_{w_j\sim P_D(w)}\left[\log \sigma(-\mathbf{u}_{w_j}^\intercal \mathbf{v}_{C_{w_t}})\right]$
GloVe	$\sum_{i,j} f\left(X_{i,j}\right)\left[\mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_j} + b_i + c_j - \log(1+X_{i,j})\right]^2$

Table 2: The objective functions of the skip-gram (negative sampling) SG(NS), continuous bag-of-words (negative sampling) CBOW(NS) and `GloVe`

4.1.2.3 Subsampling frequent words

Another way to speed up the word embedding generation is to skip frequent words in the corpus. Frequent words are updated more often compared to less frequent words, and therefore, one can argue that it is not needed to update the word vector of frequent words for each time it occurs. Subsampling the frequent words implies the deletion of frequent words from the corpus before training of the word embeddings. Besides speeding training time due to a smaller corpus, more distant words could occur within each others context window in the new corpus, while in the original corpus this was not the case. Thereby, subsampling of frequent words may increase the capture of relevant information and similarly obscures the actual context window used. Levy et al. (2015) refer to this way of subsampling as 'dirty' subsampling, where 'clean' subsampling does not influence the range of the context window. Mikolov et al. (2013b) argue that subsampling common words not only speeds the algorithm but also results in significantly more accurate word embeddings in some cases. These significant increases in performance are mainly in combination with hierarchical soft-max. Since we do not use hierarchical soft-max, we do not subsample frequent terms, but we choose to delete stopwords instead. Removing stopwords before training can be easily applied to the `GloVe` scenario as well and thereby does not create differences in the context windows used for `GloVe` and `word2vec`.

4.2 Link between `word2vec` and `GloVe`

In the initial `GloVe` paper, Pennington et al. (2014) are the first to compare `word2vec` and `GloVe`. They do this based on performance. Shi and Liu (2014) take it one step further, based of the results of Levy and Goldberg (2014) presented in the previous section. Shi and Liu (2014) rewrite the solutions of `GloVe` and `word2vec` SGNS respectively as

$$\texttt{GloVe:} \qquad \mathbf{u}_{w_i}^{\intercal}\mathbf{v}_{c_j} = \log\#(w_i, c_j) - b_{w_i} - b_{c_j}, \tag{25}$$

$$SGNS: \qquad \mathbf{u}_{w_i}^{\intercal}\mathbf{v}_{c_j} = \log\#(w_i, c_j) - \log\#(w_i) - \log\#(c_j) + \log\sum_{i=1}^{v}\#(w_i) - \log k, \tag{26}$$

where the b's denote the biases to be learned by the model. The first difference to note is that this notation is not completely thorough, but used for the sake of showing the similarity. This notation, although not noted by Shi and Liu (2014), lacks to address the fact that `GloVe` and `word2vec` SGNS solve the fact that $\log\#(w_i, c_j) = -\infty$ for non co-occurring words differently. As mentioned in the previous section, the authors of `GloVe` add a constant to the co-occurrence matrix and Levy and Goldberg (2014) use the SPPMI. Shi and Liu (2014) mention two other main difference between the two. First, the cost functions are differently defined, even though they share the same objective. Second, the weighting strategy is different. `GloVe` down weights the impact of infrequent co-occurrences to the cost function and ignores non co-occurring words. Levy and Goldberg (2014) show that `word2vec` SGNS down weights only frequent words, and due to the negative samples, `word2vec` SGNS learns by looking at word-context pairs that do not co-occur in the text. The similarity between `GloVe` and `word2vec` SGNS can be observed by comparing equation (25) and (26). The former contains biases that are trained by the model. The latter contains constants that could be considered as values for the biases in the former. Thereby, the solution of `word2vec` SGNS could be a solution of the `GloVe` model. The biases of the `GloVe` model could converge to the values of the SGNS solution. Thereby, Shi and Liu (2014) argue that the `GloVe` model is 'more general and has a wider domain for optimization'.

4.3 Model stability

The word embedding methods discussed in this research require the input of hyperparameters and training data. Both impact the solution of the word embedding models. This section discusses how to measure the stability of the models and provides an overview of the hyperparameters of `word2vec` and `GloVe`. To measure the stability and speed of the algorithms and to obtain the embeddings, we made use of the Python implementations of `word2vec` and `GloVe` using Cynthon (Behnel et al., 2011) to use C in the back-end for speed. Some hyperparameters are not tunable without entering the C source code of these packages. Out of convenience, we chose not to dive into this, and we made use of the off-the-shelf implementation of the methods. We used the *word2vec* function out of the `gensim` package created by Řehůřek and Sojka (2010) for `word2vec` and the `glove-python` package for `GloVe`. Both python applications use the default settings as set by the original C implementations of Mikolov et al. (2013a) and Pennington et al. (2014).

4.3.1 Stability Measure

In this research, we use the stability definition introduced by Antoniak and Mimno (2018) since this similarity measure is intuitive and easy to implement. Suppose we have two arbitrary hyperparameter settings A and B. According to Antoniak and Mimno (2018) the stability is measured by comparing the top 10 similar words of all words in the vocabulary according to hyperparameter setting A to the top 10 similar words of all words in the vocabulary according to hyperparameter setting B. If, for a given word, the top 10 similar words are equal, the stability scores 1 for this word. If none are similar, this score equals 0. Hence, the stability score is always a number between 0 and 1. The average of these scores for all words is the value of the stability score between run A and B of the algorithm.

4.3.2 Training data

As mentioned above, the data play a large role in the outcome of the word embeddings. Antoniak and Mimno (2018) and Wendlandt et al. (2018) show that amongst others the length of the text, the length of the individual documents, and the POS influence the outcome significantly. Furthermore, Antoniak and Mimno (2018) show that words with a high frequency do not need to be more stable in the solution. Another observation is that the `GloVe` model is in general more stable than `word2vec`. Moreover, the data influences the optimal dimensionalities of the word embedding, as shown by Yin and Shen (2018). Optimal dimensionalities range from 56 to 860 for the data sets in their research.

4.3.3 Hyperparameters

Two obvious examples of hyperparameters of `word2vec` and `GloVe` are the context window size and the dimensionality of the word vectors. However, several, less obvious parameters need to be defined by practitioners as well. Levy et al. (2015) extensively discuss all the hyperparameters used in `word2vec` and `GloVe` and divide them into three types of hyperparameters: pre-processing hyperparameters, association metric hyperparameters, and post-processing hyperparameters. This section covers the hyperparameters according to the structure of Levy et al. (2015) and discusses common choices for these hyperparameters. Note that the word embedding methods are unsupervised learning mechanisms, and therefore, the motivations are not always backed by theoretical evidence. Often, empirical evidence using word embeddings in combination with a down-stream task such as word analogy classification or part-of-speech-tagging (POS-tagging) are used to motivate the choice of the hyperparameters in the literature. These motivations are dependent on the input data and the task, and hence, can only be considered as best practices. Another motivation for choosing certain hyperparameters can be the stability of the solutions and the speed of the algorithm.

Stability of the word embeddings can provide information on the importance of choosing appropriate hyperparameter values. Suppose, as an example, that the stability analysis of training the word embeddings twice with hyperparameter settings C and D yields a high stability score. This would inform us that setting C and setting D are likely to result in similar word embeddings. Note again that this comparison of the stability of the solutions is not an indication of the quality of the solutions. In the application, we use the stability definition provided by Antoniak and Mimno (2018) and we compare the stability by adjusting the following five hyperparameters: the window size, the embedding size, the `GloVe` weighting function hyperparameters (α and x_{max}), and the number of negative samples. Besides these variables, there are choices to be made regarding other hyperparameters as discussed in the next paragraphs.

Pre-processing hyperparameter: dynamic context window

Words in text data are, in general, more related when they occur close to each other in the data. Therefore, the implementations of both `GloVe` and `word2vec` include a procedure to increase the impact of words closer to words within a context window. The `GloVe` implementation makes use of a harmonic weighting function to weight the impact of a co-occurrence of two terms by $\frac{1}{p}$, where p is the number of tokens that the context word is distant from the center word, $p = 1$ if the word is next to the center word. The `word2vec` implementation uses sampling the window size uniformly (dynamic context window) from $\mathcal{U}\{1, w\}$, where w is the window size set by the user. The expected overall impact of tokens on p positions from the center word is $\frac{w+1-p}{w}$. In our application, we make use of these default procedures.

Pre-processing hyperparameter: subsampling and deleting rare words

As motivated in Section 4.1.2.3 we do not subsample frequent words, but we choose to remove stopwords. Furthermore, rare words are removed based on a threshold on the number of occurrences. Words below this threshold are considered to be not frequent enough to train meaningful word vectors for. Deleting these words is done before training and thereby increases the range of some context windows.

Association metric hyperparameter: context distribution smoothing
Levy et al. (2015) pose that the smoothing constant used in the noise distribution of equation (24) with a value of 0.75 is 'the only parameter that can be blindly applied in any situation'. We follow this advice in our application.

Post-processing hyperparameters: adding context vectors
Levy et al. (2015) state that adding word and context vectors can be trivially applied to methods as `GloVe` and `word2vec`. Therefore, we use this way to generate our word embeddings.

Post-processing hyperparameters: vector normalization
Since, as motivated in Section 3.1, we use the cosine similarity measure. It follows that by using the same reasoning, the $L2$-normalization is the most appropriate choice to use in post-processing the word embeddings.

4.4 Practical implementation

4.4.1 N-grams

So far, we have referred to a word embedding as an embedding of a single word. However, in language, there are combinations of words which are inseparable or often occur in a fixed combination. These combinations of terms, the so-called collocations, lose their meaning if used individually. For example, 'New' and 'York' are stand-alone words, but generating the word vectors, we want to obtain a single word vector of 'New York', since this multiple word unit has one meaning. An N-gram is a collocation of N subsequent words observed in the corpus (e.g., New York is a 2-gram). By replacing frequent multiple word units by their N-gram representation, one can extract word vectors for each linguistic entity of interest. Moreover, this replacement improves the word embeddings of the single words due to less influence of irrelevant words on the word of interest. As an example, using N-grams, 'York' will not influence the vector representation 'New' anymore. An association measure represents to what degree word pairs are associated with each other. Collocations are word pairs with a score above a threshold. One needs to make three choices to extract N-grams: the association measure, the threshold, and the maximum size of N. In this paper we use the normalized point-wise mutual information (NPMI) measure introduced by Bouma (2009). The core of the NPMI is the ratio of the probability of two words occurring together (the joint distribution) versus the probability that each of these two words occurs in the corpus at a given place (marginal distributions) under the assumption of independence of word occurrence. The log of this ratio is the PMI. Normalization of the MPI with the negative log of the co-occurrence probability results in

$$NPMI(x,y) = \left(\log\frac{p(x,y)}{p(x)p(y)}\right) \Big/ -\log p(x,y).$$

For a complete derivation and motivation of the normalization, we refer to Bouma (2009). Since the NPMI is a normalized measure, the interpretation is fixed. Therefore, it is possible to use the same threshold for different data sets. The NPMI score is a value between -1 and 1, where the NPMI has a score of 1 for word pairs that occur solely together, 0 in case two words occur independently of each other and -1 when two words never occur together. The normalized nature of the NPMI provides a data-driven solution for the maximum choice of N. Given a fixed threshold one can create 2-grams in the data iteratively until no pair of words yields an association measure larger than the threshold. Note that a collocation of a 2-gram with a word results in a 3-gram. Therefore, a threshold between 0 and 1 is appropriate. However, there is no literature on which specific value to use in a situation. Besides, the NPMI is not the focus of this paper. Therefore an arbitrary threshold value of 0.7 is used in the application. Note that the three word embedding methods of interest use the same training data based on the same N-gram threshold value.

4.4.2 `sentence2vec` / `dish2vec`

To compare not just single terms, but a sequence of terms (such as in a sentence, paragraph, document or in our research, dish) using the cosine similarity one requires two equally sized vectors. Le and Mikolov (2014) propose `doc2vec`, an extension of `word2vec` training document vectors in parallel with the word vectors. However, since the goal of this paper is to compare the results of `GloVe` and `word2vec` after being trained, and `doc2vec` is a different training procedure, we choose for a word vector combination as dish representation. Averaging word vectors, as opposed to vector concatenation, results in equally sized dish vectors. However, in general, less frequent words are more discriminate for dishes. Consider, for example, the words ‘tomato’, ‘rice’, and ‘soup’. These words are less likely to contain dish unique information compared to ‘alligator’, ‘seaweed’ or ‘caviar’. Therefore assigning a larger weight to these words seems reasonable and hence, infrequent words have a larger influence in the dish similarity. In our application, we use the approach suggested by Arora et al. (2016b), taking the weighted average of word vectors in a dish and subsequently remove the first principal component obtained by Principal Component Analysis (PCA) from the matrix $\mathbf{Y} \in \mathbb{R}^{q\times d}$. Here, q is the number of dishes and d is the dimension of the word vectors. The columns of $\mathbf{Y}$ are the weighted averages of the word vectors of the dishes. These weights are based on the word frequency and hyperparameter a. For word w, the weight is defined as $\frac{a}{a+p(w)}$, where $p(w)$ is the frequency of word w. The motivation for the removal of the first principal component is based on empirical observations. Arora et al. (2016b) state that 'most word embedding methods, since they seek to capture word co-occurrence probabilities using vector inner product, end up giving large vectors to frequent words, as well as giving unnecessarily large inner product to word pairs, simply to fit the empirical observation that words sometimes occur out of context'. The large components have a disproportionately substantial impact on the resulting (weighted) average embedding of a sentence without capturing meaningful semantic characteristics of the words. Removing the first component smooths the resulting sentence embedding and, according to Arora et al. (2016b), this method outperforms the unweighted average and even beats more sophisticated methods. Another motivation to use this approach is that it is intuitive and easy to implement. See Arora et al. (2016b) for a detailed description of the weighting procedure and the algorithm of how to construct the dish embeddings. In the application we use a weighting parameter value of $a = 10^{-3}$ as empirically motivated by Arora et al. (2016b).

5 Application: empirical comparison of `GloVe` and `word2vec`

In this section, we apply the word embedding methods in a real-world setting using text data form the catering industry. In the catering industry, businesses (e.g., restaurants, hotels, and food delivery services) use text in the form of a list of dishes on their menu to inform customers about the food they offer. This information is publicly available on the Internet and provides one with a rich open source of text data. These data can be used to personalize marketing strategies and are, therefore, a potentially valuable resource for businesses. Embeddings of dishes and menus can be used to feed the modeling techniques used for such tasks if one wants to use text data in an automated way. Applying the different word embedding methods results in different solutions, where each solution captures a different set of linguistic elements. It is difficult to determine the quality of the word embeddings by inspection of the word embeddings. Therefore, we will create dish embeddings out of the word embeddings, and we let experts evaluate the quality of these dish embeddings indirectly. This evaluation can provide insights in which method is the best. Since there is a limited amount of chefs available to evaluate the models, we choose to use one variant of each model. Moreover, for the final hyperparameter setting, we fix the values across algorithms for the hyperparameters that are required in multiple algorithms. The application is performed on the menu data of restaurants only. The menus of restaurants are more readily found online compared to menus of hotels and other caterers. Moreover, the restaurant sector is large and therefore, enables us to collect a large amount of data required to train the word embeddings.

5.1 Data

To train the word embeddings, we use text data scraped from the Internet. Menu cards of more than 20,000 restaurants from the United Kingdom are collected, cleaned, and processed. After removal of all diacritics, punctuation and freestanding, single characters, words are removed that do not occur on less than five menu cards. The last step removes, besides infrequent words, strings that are not part of any language but generated by the scraping procedure. Next, the words are tokenized and stemmed using the Porter stemming algorithm (Porter, 1980). Stemming removes information in the data, such as conjugations and plural forms. Besides, stemming the words concentrates the information, since fewer word embeddings have to be learned and each term occurs more frequently in the stemmed text data. The motivation to choose for stemming versus no stemming is that we do not regard conjugations as important on menus of restaurants, and moreover, removing plural forms is desired under the assumption that singular or plural form does not impact the meaning of a dish. Furthermore, fewer words have to be trained using stemming, speeding the embedding methods. Consequently, we insert N-grams according to Section 4.4.1. The vocabulary of the resulting processed data contains 25944 unique tokens. The processed data consists of 23,140 unique businesses with on average 95.3 menu items. The average length of a dish is 3.66 tokens. Note that each menu item does not need to contain a full dish. The scraping procedure maps each line on the menu card to a menu item. Therefore, a menu item could consist of the name of a dish, the main ingredient(s), additional dish information, additional ingredients, or a combination of the former. Furthermore, if a menu contains information on the corresponding business, this information is not excluded and may result in noise in the menu data.

5.2 Methods, model stability, and hyperparameters

In the application `word2vec` SG and CBOW, both with negative sampling, and `GloVe` are estimated using the menu data. The negative sampling extensions of `word2vec` are used to reduce the training time of the algorithm. The off-the-shelf python implementations of `word2vec` from the `gensim` package and `GloVe` from the `glove-python` package are used. The hyperparameter settings discussed in Section 4.3 and 4.4 are used, except for the window size, embedding size, the number of negative samples (`word2vec`), α (`GloVe`), and x_{max} (`GloVe`) since these will be tuned using a stability analysis. While performing the stability analysis, we keep track of the computation times. All results are based on ten epochs. This number is rather arbitrarily chosen. Using multiple epochs increases the amount of information that is extracted from the training data and simultaneously extends the training duration. The training time in our context with ten epochs was still within reasonable bounds. The stability analysis is based on a default hyperparameter setting based on the papers introducing `word2vec` and `GloVe`. These default hyperparameters are modified one at a time. The corresponding default values are given by

- windows size $w = 5$,
- number of dimensions $d = 300$,
- number of negative samples $k = 10$,
- power hyperparameter $\alpha = 0.75$,
- maximum function value $x_{max} = 100$.

The stability analysis is based on the stability measure, as discussed in Section 4.3.1. Figure 3 visualizes the results of the stability analysis. Each column contains the results for a different method, and each row contains information concerning one hyperparameter. Recall that the stability ranges from 0 to 1, where 0 is unstable, and 1 is stable. The heat maps in Figure 3 show that the word embeddings are highly unstable for some hyperparameter settings. If the dimensionality of the word vectors is 10, the stability score is approximately 0.1 for the `GloVe` model, even when we use the same hyperparameter settings for the two runs. A dimensionality of 10 is in practice not common and results in information loss. In general, the `GloVe` model is the most stable one, followed by the `word2vec` SGNS model. The `word2vec` CBOWNS is the most unstable. However, the off-diagonal entries of the `GloVe` are smaller than those of `word2vec` SGNS and `word2vec` CBOWNS in the window size graphs in Figure 3. Thereby, `word2vec` SGNS and `word2vec` CBOWNS are more stable if one compares vectors based on different window sizes. Note that the number of negative samples has no impact on the stability and computing time of `GloVe` since this hyperparameter is not used in that algorithm. Similarly, this hold for α and $x_{\max}$ in the case of `word2vec` SGNS and CBOWNS.

5.2.1 Final hyperparameter setting

The window size determines the reach of words influencing the embedding of each other. Since the text corresponding to a dish can be spread over multiple menu items, we choose the window size such that words of the surrounding menu items have an impact on the words of a menu item. Note that for larger window size, the word embedding of a word located in a menu item can be influenced by words of other dishes as well and therefore the window size should not be chosen too large. On the other hand, being too strict using a smaller window size might cause a loss in information captured. In our application, we set the window size such that, on average, the words in three menu items influence each word (the menu item the word is in and the menu item above and below the menu item the word is in). Since the average menu item size is 3.66 we set the window size to $w = \frac{3\times 3.66-1}{2} \approx 5$. Since the data is scraped and since we do therefore not know which menu items compose a single dish, this window size is likely to be too large in some cases and too small in some other cases.

The dimensionality of the vectors is set to $d = 300$. This vector size is of the same order of magnitude as the optimal vector sizes found by Yin and Shen (2018). The stability analysis (Figure 3) shows that an increase of the vector size to a size larger than 300 results in relatively little stability gain, while Figure 4 indicates that the computation time increases linearly for all three methods. Therefore, in this trade-off between computation time and stability, we set the window size to $d = 300$.

A negative sample parameter of one or one of larger than 200 results in relatively unstable model results for both SGNS and CBOWNS. Since the models are most stable for $5 \leq k \leq 100$, as one can see on the diagonal axis of figure 3, and the number of negative samples improves the word embeddings according to Levy et al. (2015), yet has a significant impact on the speed of these two methods; we set $k = 10$.

First, Pennington et al. (2014) find empirical evidence that $\alpha = 0.75$ slightly improves the resulting word embeddings over $\alpha = 1$. Second, the stability analysis on α provides little insights besides the fact that $\alpha \in \{0.4, 0.6, 0.75, 0.85, 0.95, 1\}$ provide the most stable results (Figure 3). Combining the prior two arguments, we use the proposed value of Pennington et al. (2014), $\alpha = 0.75$.

x_{max} is the upper boundary of what is considered to be an infrequent term. Given the frequency distribution of the words in our text presented in Figure 6 of section D of the Appendix, it seems reasonable to use the definition that a word frequently occurs if it occurs more than 100 times in the text. The stability results for x_{max} indicate that the `GloVe` model is increasingly unstable as the upper boundary x_{max} is increased. Moreover, $x_{max} = \{10, 50, 100\}$ provides the most stable results in our stability analysis (Figure 3). Therefore, we set $x_{max} = 100$.

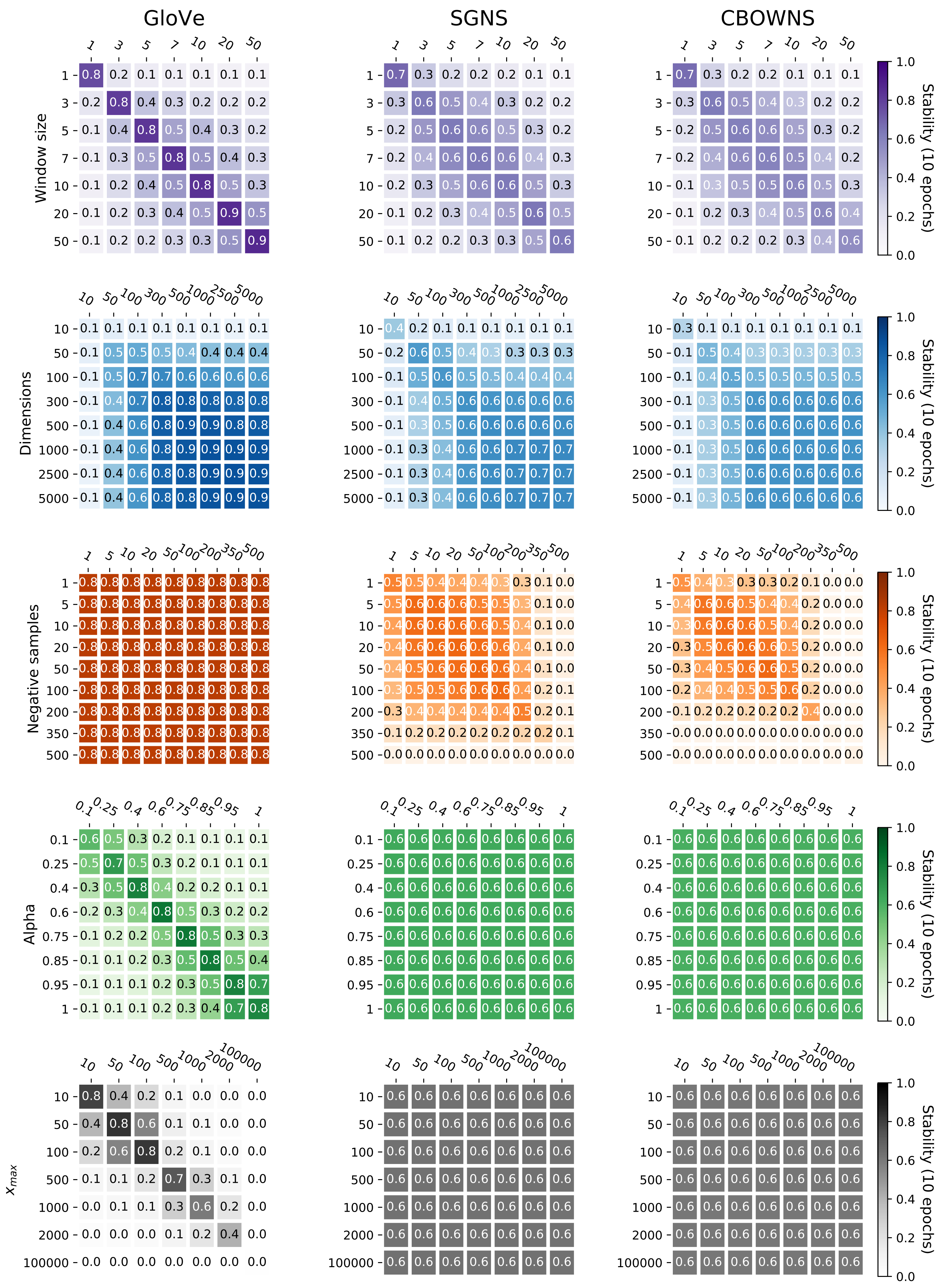


Figure 3: Stability analysis for 10 epochs

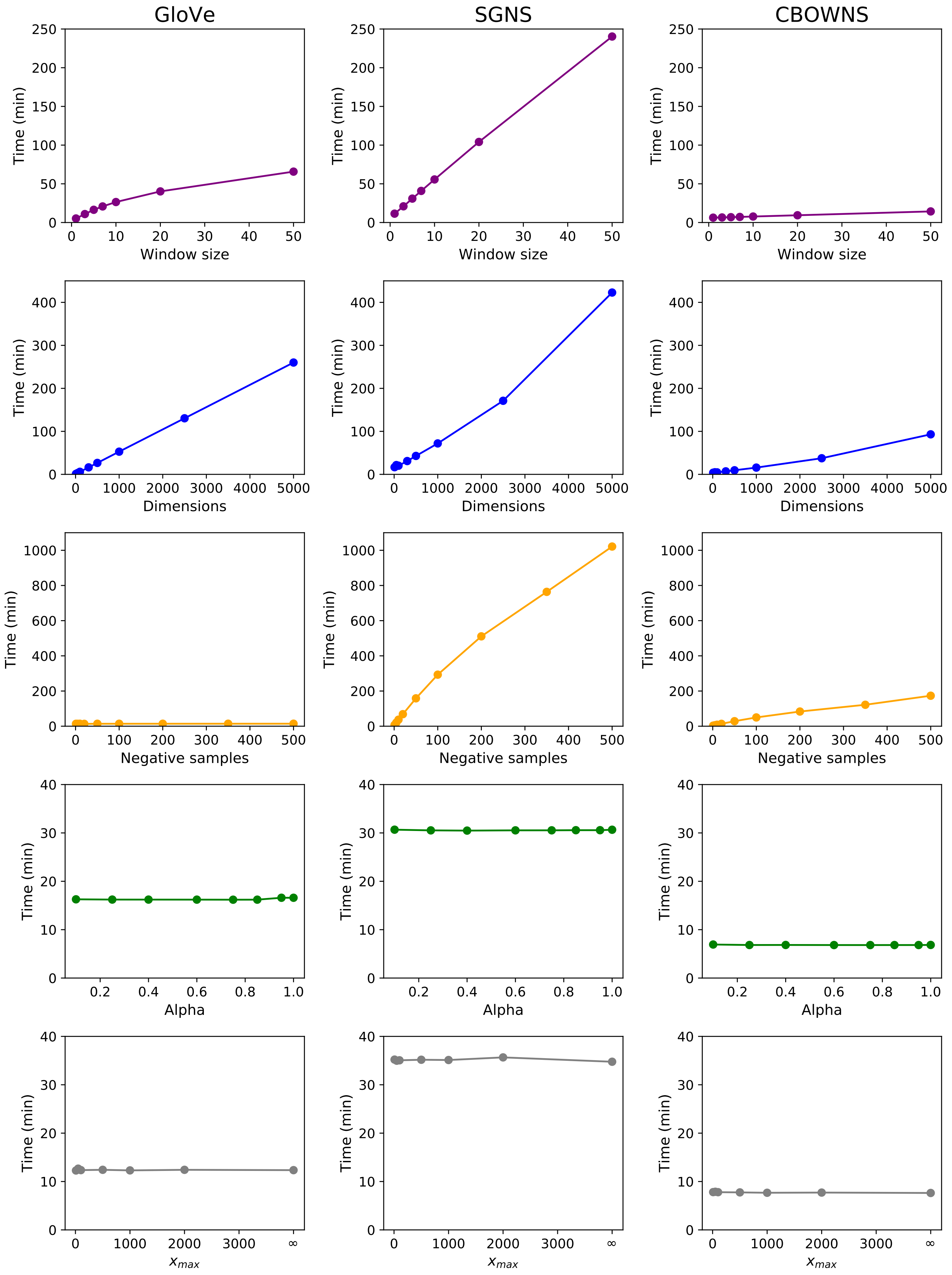


Figure 4: Computation times for the three methods. The time is in minutes. All calculations are performed single threaded on a Microsoft Azure Standard_DS3_v2 Cloud Node.

5.3 Comparison

We use the argued final hyperparameter setting of the previous section to evaluate the algorithms empirically. In this application, the evaluation of the performance of the methods is done using comparative intrinsic evaluation, as suggested by Schnabel et al. (2015). In this evaluation procedure, people are asked to provide feedback on the word embeddings directly. Only the input embedding, the embedding representing the query dish, needs to be defined by the researcher. The output for each method is the most similar embedding, representing the most similar dish according to that method. To perform a comparative intrinsic evaluation of `GloVe`, `word2vec` SGNS, and `word2vec` CBOWNS, we submit a set of popular (query) dishes with their most similar dishes according to each of the three methods to professional chefs. The chefs provide feedback by choosing from a set of three options. The chefs are instructed to choose the most similar dish according to their expertise. Each option is a dish that is the closest, and thereby, most similar to the query dish according to one of the three methods in question. A method receives a vote if a chef chooses a dish, related to that method, that is the most similar to the query dish according to him/her. In case two methods result in the same most similar dish, and a chef chose this option, the vote is assigned to both methods. The method with the most votes is considered to be the most appropriate method in a menu data context. The popular query dishes are a subset of the CCN's top 50 of the world's best foods.[2] This subset is generated by leaving out the dishes which are not present in the menu data at hand. The total number of dishes that remain in the application is 43. These dishes cover a wide variety of cuisines and contain low and high-frequency words according to the data at hand and therefore, provide relatively generic results. The professional chefs are found through the transnational network of a consumer goods company. These chefs are part of a network to spot worldwide food trends and are, therefore, each informed of multiple cuisines. Thereby, we assume that these chefs can provide proper feedback even though they are not all from the UK.

Each chef submits their feedback by completing a multiple choice survey choosing the most similar dishes. We have included a 'no choice' option to provide the chefs with the opportunity to be indecisive if all answers are equally (dis)similar to the dish of interest. An example of a survey question is given in the square block below (question 1). Since the answers are different according to the three methods, we can conclude that each algorithm has generated different word embeddings.

[2]https://edition.cnn.com/travel/article/world-best-food-dishes/index.html

1. *Query dish: Neapolitan stone baked pizza*
    (a) *Napoli sauce, cream, italian, sausage, friarielli, procini mushroom*
    (b) *Boscaiola, salsiccia e friarielli, pancetta a ricotta*
    (c) *Formaggi 10 pizzebase*
    (d) *No choice*

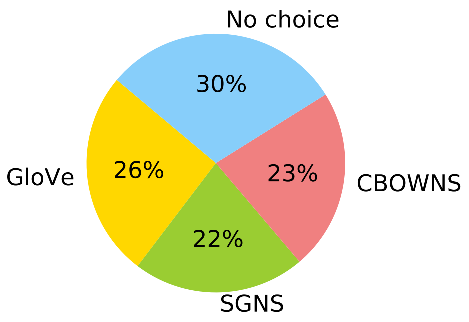


Figure 5: Pie chart of the responses

The quality of the word embedding is derived from the number of times a chef has voted for a method indirectly. The results are the percentages of votes the methods received with respect to the total number of votes. Figure 5 shows a pie chart of the vote percentages. Thereby, we conclude that in this specific example with this particular set of hyperparameters, the `GloVe` method performs best according to the experts. More than 30 percent of the cases the options were unrelated or equally similar to the word of interest. Important to note is that these results are based on a small subset of the total dish universe. Besides, the stability analysis shows that the solutions of all three methods are not stable. Therefore, we consider these results as not generalizable. However, our results contribute as an example case to the empirical literature of word embedding method comparisons. Marketers in the food industry should at least, considering the performance and the speed of the `GloVe` method in our application, consider this method to create word embeddings. Moreover, our research informs marketers which factors need to be taken into account in the process of word vector generation. Evaluation of the performance of more hyperparameter settings can provide more insights into the overall performance of the methods in a specific context and left for further research.

6 Summary

This paper has analyzed and applied three prominent word embedding methods; `GloVe`, and two versions of `word2vec`: continuous skip-gram (SGNS) and continuous bag-of-words (CBOW). In the application, the negative sampling extensions of the `word2vec` algorithms are used, `word2vec` SGNS and `word2vec` CBOWNS. We touched upon the relevance of the use of word embedding in practice. Besides, we provided an intuitive explanation of how these training algorithms capture linguistic elements of text data. The provided derivations of the loss functions and corresponding gradients combined with explanations on the optimization procedures of these methods contribute to a complete understanding of these word embedding algorithms. Consequently, extensions to accelerate the algorithms, the link between the algorithms, the stability of the models, and suggestions to improve the quality of the embeddings in practice have been discussed. The three word embedding methods (`GloVe`, `word2vec` SGNS, and `word2vec` CBOWNS) were applied to a food menu context, each resulting in a different set of word and dish embeddings. We evaluated the performance of these methods using an unsupervised framework combined with opinions of experts in the field where the data are from, the catering industry. In this specific context, using argued hyperparameters, the `GloVe` method performed best over `word2vec` CBOWNS and `word2vec` SGNS. Unfortunately, due to the instability of the methods, a large number of hyperparameters to tune, and the limited evaluation resources, we cannot answer the auxiliary research question with the performed application. However, we did achieve the goal of filling the gap regarding the lack of detailed information on word embedding techniques.

7 Discussion

'Argued hyperparameters', as stated in the summary of Section 6, might raise a concern of ambiguity to the reader, and that is related to the first result of this paper. Evaluation of plain word embeddings — a set of vectors consisting of one vector per word, the form `GloVe` and `word2vec` produce them — is difficult. We consider the word vectors 'good' if they capture linguistic elements from the text data train set. However, the extent to which the vectors capture linguistic properties cannot be derived directly by inspection of these vectors. Using the vectors as predictors of linguistic properties has been done in other papers. This approach adds an extra model layer to transform the vectors into a prediction. Thereby, the quality of the vectors is dependent on the task of the downstream analysis and the choice of model. In our research, we tried to stay closer to the core of the embeddings by evaluating the similarity of embeddings and compare these with the similarity expectations of experts. Even though this way of evaluation compares the raw word embeddings, it provides substantial limitations. First, the experts' opinions are exhaustible. Namely, there was a limited number of chefs available, and the chefs were willing to spend a limited amount of 15 minutes of their time to the research. Therefore, we were not able to compare multiple versions of the same model to the experts. Second, the word embedding algorithms do include many hyperparameters. So far, in the literature, there is no conformity on what hyperparameter settings to use in each situation. Hence, even though best practices, academic reasoning, and rationale back the choices of the hyperparameters, the similarities presented to the experts based on the resulting final models include many researcher degrees of freedom. Therefore, based on the results of our application, we cannot generalize the fact that the `GloVe` method performs best in the task of word embedding generation in the food menu context. Moreover, we cannot conclude that the `GloVe` method is the best method to generate word embeddings for the data set used in this paper.

Another conclusion from our research is that solutions of the word embedding methods can be unstable. If one uses the same hyperparameter setting and trains the word embeddings twice on the same data, one will obtain two different word embeddings. Random initialization, random negative sampling, and stochastic optimization are factors that can have an impact on this stability. In this paper, the `GloVe` model is the most stable of the three, followed by `word2vec` SG with negative sampling. The stability analysis is carried out by changing one hyperparameter at a time. A full grid search would have given even more insight into the behavior of stability. However, such a grid search is computationally expensive and therefore, left for further research. Furthermore, evaluation of what factors play an important role in the instability of the models is a topic yet to be investigated.

Since the solution of the word embeddings methods is highly dependent on the hyperparameter setting and the input data for the learning algorithm, we do not consider the results of the application in this research to be generalizable to other contexts. The main contribution of this paper lays in the explanation of the methods and in the description of the approach of how to use text data in quantitative marketing.

References

Aldrich, J. H., Nelson, F. D., and Adler, E. S. (1984). *Linear Probability, Logit, and Probit Models*, volume 45. Sage.

Antoniak, M. and Mimno, D. (2018). Evaluating the stability of embedding-based word similarities. *Transactions of the Association for Computational Linguistics*, 6:107–119.

Arora, S., Li, Y., Liang, Y., Ma, T., and Risteski, A. (2016a). A latent variable model approach to pmi-based word embeddings. *Transactions of the Association for Computational Linguistics*, 4:385–399.

Arora, S., Liang, Y., and Ma, T. (2016b). A simple but tough-to-beat baseline for sentence embeddings. In *International Conference on Learning Representations, 2017.*

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. (2011). Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155.

Berger, A. L., Pietra, V. J. D., and Pietra, S. A. D. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Bouma, G. (2009). Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL*, pages 31–40.

Caselles-Dupré, H., Lesaint, F., and Royo-Letelier, J. (2018). word2vec applied to recommendation: Hyperparameters matter. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 352–356. ACM.

Church, K. W. and Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167. ACM.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Dumais, S. T., Furnas, G. W., Landauer, T. K., Deerwester, S., and Harshman, R. (1988). Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 281–285. ACM.

Dyer, C. (2014). Notes on noise contrastive estimation and negative sampling. *arXiv Preprint arXiv:1410.8251*.

Eckart, C. and Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218.

Faruqui, M. and Dyer, C. (2014). Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 462–471.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in Linguistic Analysis*.

Garten, J., Sagae, K., Ustun, V., and Dehghani, M. (2015). Combining distributed vector representations for words. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 95–101.

Goldberg, Y. and Levy, O. (2014). word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv Preprint arXiv:1402.3722.*

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Hwang, J. and Lorenzen, C. L. (2008). Effective nutrition labeling of restaurant menu and pricing of healthy menu. *Journal of Foodservice*, 19(5):270–276.

Keerthi, S. S., Schnabel, T., and Khanna, R. (2015). Towards a better understanding of predict and count models. *arXiv Preprint arXiv:1511.02024.*

Kozup, J. C., Creyer, E. H., and Burton, S. (2003). Making healthful food choices: The influence of health claims and nutrition information on consumers evaluations of packaged food products and restaurant menu items. *Journal of Marketing*, 67(2):19–34.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196.

Lebret, R. and Collobert, R. (2013). Word emdeddings through hellinger pca. *arXiv Preprint arXiv:1312.5542.*

Levy, O. and Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185.

Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Lund, K. and Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Miller, G. A. (1995). WordNet: A lexical database for english. *Communications of the ACM*, 38(11):39–41.

Nadkarni, P. M., Ohno-Machado, L., and Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

Rao, C. R. (1995). A review of canonical coordinates and an alternative to correspondence analysis using hellinger distance. *Qüestiió: Quaderns d'Estadística i Investigació Operativa*, 19(1).

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. `http://is.muni.cz/publication/884893/en`.

Rong, X. (2014). word2vec parameter learning explained. *arXiv Preprint arXiv:1411.2738.*

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv Preprint arXiv:1609.04747.*

Schnabel, T., Labutov, I., Mimno, D., and Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307.

Shi, T. and Liu, Z. (2014). Linking glove with word2vec. *arXiv Preprint arXiv:1411.5595.*

Tsvetkov, Y., Faruqui, M., Ling, W., Lample, G., and Dyer, C. (2015). Evaluation of word vector representations by subspace alignment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2049–2054.

Wendlandt, L., Kummerfeld, J. K., and Mihalcea, R. (2018). Factors influencing the surprising instability of word embeddings. *arXiv Preprint arXiv:1804.09692.*

Yin, Z. and Shen, Y. (2018). On the dimensionality of word embedding. In *Advances in Neural Information Processing Systems*, pages 895–906.

Appendices

A Multinomial logit model assumptions

According to Aldrich et al. (1984), four assumptions are underlying the multinomial logit model (soft-max regression model, maximum entropy model). Below, these assumptions are provided, translated to a word embedding context under the condition that either the context or center words are fixed. Furthermore, we include notes on whether the assumption is likely to hold at the end of the assumptions. The assumptions are:

1. *The dimensions of the word embeddings are linearly independent.* — The skip-gram method uses random initialization as a starting point for the exogenous variables (the features of the center word vector) and hence the dimensions are likely to be linearly independent. Since iterating over the text causes the exogenous features to enclose more and more linguistic information, which is full of highly nonlinear characteristics, the features will probably not become linear dependent. Therefore, this property is likely to hold for later stages of the algorithm as well.

2. *The events of being or not being the context word at any given position in the text does not influence the probabilities for any other combination of center and context word in the text, this is the independence assumption.* — In the case of text data, the first assumption will never be perfectly met since the word occurrences influence the probability of the words occurring in their surround. For example, if the word 'strawberry' occurs in the text, the probability that this word will occur in the next place as well will be smaller.

3. *Classes are not perfectly separated by independent variables.* — This assumption is likely to be satisfied because words can occur in various contexts.

4. *Given an observation (center word), the probability of the context word is given by the soft-max function.*

B Link between `GloVe` and SVD

The authors of the `GloVe` paper mention that they combine the best of two worlds by updating the objective function by iterating over the text using the word co-occurrence statistics. SVD based methods such as LSA only take the co-occurrence statistics into account and therefore are suitable to capture patterns from the text that occur on a large scale. These methods, however, fail to take the more subtle linguistic elements that are captured by the context window approach. Let $\mathbf{Y_1}$ be the $v \times v$ co-occurrence count statistic matrix of the text data. Furthermore, suppose we want to generate word vectors of dimension d. Let $\mathbf{A_1^*}$ be matrix with rank d obtained by the truncated SVD of $\mathbf{Y_1}$. Then $\mathbf{A_1^*}$ can be denoted as

$$\mathbf{A_1^*} = \mathbf{B_1}_d \mathbf{\Sigma_1}_d \mathbf{C_1}_d^\intercal. \tag{27}$$

According to the Eckart-Young-Mirsky theorem (Eckart and Young, 1936), $\mathbf{A_1^*}$ is the optimal analytical solution to

$$\min_{\mathbf{A_1}} ||\mathbf{A_1} - \mathbf{Y_1}||_2, \qquad \text{under the condition that } Rank(\mathbf{A_1}) \leq d.$$

This minimization has similarities with the `GloVe` objective. Writing the objective of the `GloVe` method as defined in equation (22) as a minimization function yields the following objective

$$\min_{\mathbf{U},\mathbf{V},\mathbf{b},\mathbf{c}} \sum_{i,j} f\left(X_{i,j}\right) \left[\mathbf{u}_{w_i}^\intercal \mathbf{v}_{w_j} + b_i + c_j - \log(1 + X_{i,j})\right]^2.$$

In order to show the similarity we fix the value of the weighting function in equation $f(X_{i,j}) = 1$ and we assume that $\mathbf{b}$ and $\mathbf{c}$ are fixed. Furthermore, let $\mathbf{Y_2}$ be defined as $\mathbf{Y_2} = \mathbf{b}\mathbf{1}^\intercal + \mathbf{1}\mathbf{c}^\intercal - \log(\mathbf{J}_v + \mathbf{X})$, where $\mathbf{J}_v$ is a v dimensional square matrix of ones. Let $\mathbf{A_2^*}$ be the truncated SVD of $\mathbf{Y_2}$ similar to the definition in equation (27). The minimization problem then becomes

$$\begin{aligned} &\min_{\mathbf{U},\mathbf{V}} \left[\mathbf{U}\mathbf{V}^\intercal + \mathbf{b}\mathbf{1}^\intercal + \mathbf{1}\mathbf{c}^\intercal - \log(\mathbf{J}_v + \mathbf{X})\right]^2 \\ =&\min_{\mathbf{U},\mathbf{V}} \left[\mathbf{U}\mathbf{V}^\intercal - \mathbf{Y_2}\right]^2 \\ =&\min_{\mathbf{U},\mathbf{V}} ||\mathbf{U}\mathbf{V}^\intercal - \mathbf{Y_2}||_2, \end{aligned}$$

which has, according to the Eckart-Young-Mirsky theorem the optimal analytical solution $\mathbf{A_2^*}$. Defining $\mathbf{W} = \mathbf{B_2}_d \mathbf{\Sigma_2}_d$ and $\mathbf{Z} = \mathbf{C_2}_d$ gives the desired word and context matrices respectively based on the co-occurrence statistic with use of SVD, that is $\mathbf{U}^* = \mathbf{W}$ and $\mathbf{V}^* = \mathbf{Z}$. Note that there are multiple solutions possible by weighing the eigenvalue matrix $\mathbf{\Sigma_2}_d$ differently. Levy et al. (2015) empirically show by varying $p \in [0, 1]$ in $\mathbf{U}^* = \mathbf{B_2}_d \mathbf{\Sigma_2}_d^p$ and $\mathbf{V}^* = \mathbf{\Sigma_2}_d^{1-p} \mathbf{C_2}_d$ that using $\mathbf{U}^*$ as the word embedding matrices with a value of $p = 0$ or $p = 0.5$ outperforms the frequently used (e.g., Faruqui and Dyer (2014)) choice of $p = 1$ in practice.

Despite the similarity of SVD and `GloVe` there are major differences. First, the `GloVe` method is a non-convex optimization problem, while SVD guarantees a globally optimal solution. Second, the solution of SVD can be obtained analytically without any hyperparameter tuning while on the contrary, `GloVe` requires a gradient descent type of optimization procedure and hyperparameter choices. Last, SVD does not down weight the relatively large impact of co-occurrences of infrequent terms.

C Link between `word2vec` SGNS and matrix factorization

Since the embedding methods have the same goal, researchers have tried to find the similarity between multiple methods. Levy and Goldberg (2014) show that SGNS is implicitly a factorization of the word-context matrix, where the entries of this matrix are based on the point-wise mutual information (PMI). As pointed out by Arora et al. (2016a), the reasoning provided by Levy and Goldberg (2014) only applies to word embeddings with a large dimensionality, while the goal application of the word embeddings is to generate word embedding with a small dimensionality. The PMI, introduced by Church and Hanks (1990), is an association measure between two words and resembles the joint probability of two words with respect to the marginal probability of both words under the assumption that their appearance is independent. The formula of the PMI is

$$PMI(x,y) = \log\frac{p(x,y)}{p(x)p(y)}.$$

Levy and Goldberg (2014) show that the objective function of SGNS is equivalent to factorizing $v \times v$ matrix $\mathbf{M}$, where

$$\mathbf{u}_{w_i}^\intercal \mathbf{v}_{c_j} = M_{i,j}^{SGNS} = PMI(w_i, c_j) - \log k, \qquad \forall i,j \in \{1, ..., v\}.$$

Consider that the definitions of D, $\#(w_i, c_j)$, $\#(w_i)$, and $\#(c_j)$ provided in Section 4.1.2.2. Then, Levy and Goldberg (2014) show that $\mathbf{M}$ can be empirically constructed by

$$\begin{aligned} M_{i,j}^{SGNS} &= \log\frac{\#(w_i,c_j)\cdot|D|}{\#(w_i)\cdot\#(c_j)} - \log k \\ &= \log\#(w_i,c_j) - \log\#(w_i) - \log\#(c_j) + \log|D| - \log k \\ &= \log\#(w_i,c_j) - \log\#(w_i) - \log\#(c_j) + \log\sum_{i=1}^{v}\#(w_i) - \log k. \end{aligned} \tag{28}$$

Two words that never co-occur will result in a value of $-\infty$, since $\log\#(w_i, c_j) = \log(0) = -\infty$. As in equation (21), a small constant can be added to prevent this issue. However, Levy and Goldberg (2014) introduce the shifted positive PMI (SPPMI), where $M_{i,j}^{SSPMI} = 0$ not only if $\#(w_i, c_j) = 0$, but more strict, if $PMI(w_i, c_j) - \log k < 0$. If $PMI(w_i, c_j) - \log k \geq 0$, $M^{SSPMI} = M^{SGNS}$ as defined in equation (28). For the interested reader, Levy and Goldberg (2014) show a full derivation of this outcome, together with improvements and a comparison of SGNS solved using the stochastic gradient descent approach and SGNS solved with SVD. Though outside of the scope of this paper, Keerthi et al. (2015) elaborate further on the link between PMI and `word2vec` SGNS.

D Menu data word count

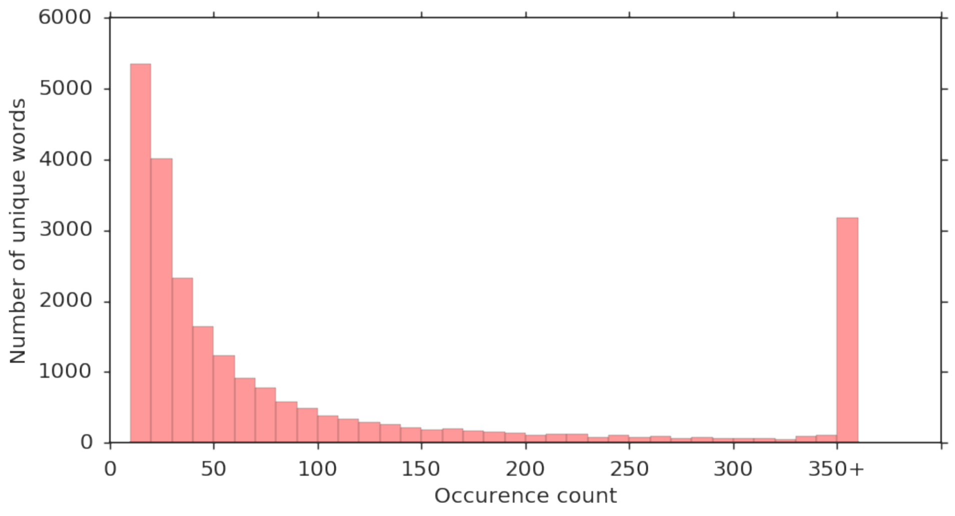


Figure 6: Histogram of the word counts in the processed menu data