12-2019

Incorporating word order explicitly in GloVe word embedding

Brandon Cox

Incorporating word order explicitly in GloVe word embedding

Brandon S. Cox

An undergraduate honors thesis

Department of Computer Science and Computer Engineering

College of Engineering

University of Arkansas

November 22, 2019

This thesis is approved.

Thesis advisor: _____

Committee member: _____

Committee member: _____

Abstract

Word embedding is the process of representing words from a corpus of text as real number vectors. These vectors are often derived from frequency statistics from the source corpus. In the GloVe model as proposed by Pennington *et al.*, these vectors are generated using a word-word cooccurrence matrix. However, the GloVe model fails to explicitly take into account the order in which words appear within the contexts of other words. In this paper, multiple methods of incorporating word order in GloVe word embeddings are proposed. The most successful method involves directly concatenating several word vector matrices for each position in the context window. An improvement of 9.7% accuracy is achieved by using this explicit representation of word order with GloVe word embeddings.

Contents

1 Introduction

1.1 Background

Word embedding is the process of representing words as vectors of real numbers. These vectors can be used in many applications, such as search engines, information retrieval, document classification, and parsing. For example, word vectors may be used to find similar words in search engines to expand queries, similar to Gauch *et al.* [1]. In their paper, Gauch *et al.* start by building a list of vectors that represent words in a given corpus. They then take a series of search queries and essentially add words similar to those already in the query to the query. This resulted in slightly better search results when compared to previous methods [1].

Over time, word embedding has taken multiple forms. The form used by Gauch *et al.* is based on a word-word cooccurrence matrix where each individual entry is replaced by the mutual information values for the given word pair [1]. Recently, there has been a shift with a focus leaning more towards neural network-based models. For example, Mikolov *et al.* proposed the Skip-gram and continuous-bag-of-words models as discussed in the next section, which are primarily based on a neural network [2, 3]. Even more recently, Pennington *et al.* proposed the GloVe model, which is based on a least-squares problem as discussed in section 2.3 [4].

1.2 Objectives

Of the recent papers published in the field of word embedding, Pennington *et al.* produced the strongest results [4]. However, the GloVe model fails to explicitly take into account word position in its model. For example, all of the words in a given context window will be treated almost equally when computing the vector for a given word, no matter how far they are from the target word. This will become evident in section 2.3. The primary objective of this paper is to produce an approach to improve upon the GloVe model of word embedding by adding some explicit representation of word position in the final set of word vectors.

2 Related Work

2.1 Finding Synonyms

The process of finding synonyms computationally has had a long history. Initially, the distributional hypothesis was proposed by Harris, who claimed that words with similar meanings tend to appear in similar contexts [5]. As an example, consider the words "adversary" and "foe". As English speakers, we know that these words are synonyms. We can also note that these words are always used as the objects of verbs. For example, consider the sentence "I defeated my foe in battle". "foe" could be replaced by "adversary" directly without changing the validity or meaning of the sentence. Multiple authors have proposed means of computing distributionally similar words [6, 7, 8].

An issue with computational approaches as proposed by Lin *et al.* is that they fail to

distinguish between synonyms and antonyms. To address this, there are two proposed two solutions [9]. One solution is based on identify word patterns, e.g., if words X and Y that appear in the pattern "from X to Y" or "either X or Y" they likely to be antonyms [9]. Another solution involves using bilingual dictionaries to determine synonyms based on their translations [9]. Consider the case of a word that is distributionally similar to a set of words A. Now, translate the original word to French and back. Let the set of words given by the translated word be set B. $A \cap B$ would then represent the synonyms of the word, and $A - B$ would represent the antonyms of the word. The pattern-based approach proved to be more successful, with a precision of 86.4% [9].

Scheible *et al.* also attempted to solve the same problem as Lin *et al.* of distinguishing between synonyms and antonyms when using a distribution-based similarity approach. They hypothesize that the contexts of adjectival synonyms and antonyms are not distributionally similar and that not all word classes are useful for modelling the contextual differences between synonyms and antonyms [10]. The results generated by Scheible *et al.* support both of the hypotheses by a large margin [10].

2.2 Dimensionality Reduction

Dimensionality reduction is the process of reducing the dimension, or the number of columns, in a given matrix while losing as little data as possible. One method of dimensionality reduction is Principal Component Analysis (PCA). It treats each of the rows of the matrix as a vector of features in which each feature represents one dimension of a vector space.

These dimensions are assumed to have dependencies. It reduces the number of dimensions by calculating a smaller set of independent dimensions and mapping each original vector to this new, lower dimensional vector space [11].

Shlens derives the process of PCA in detail in [11]. Here is a brief summary of how to reduce a matrix using PCA: let $\mathbf{X}$ be a $m \times n$ matrix that we want to reduce to a columns. In other words, we want to find some $m \times a$ matrix $\mathbf{Y}$ such that $\mathbf{Y} = \mathbf{PX}$, where $\mathbf{P}$ is an othonormal matrix whose rows are the principal components of $\mathbf{X}$. The principal components of a matrix are the vectors along which the variance in $\mathbf{X}$ is maximized [11]. An additional constraint is that $\mathbf{C_Y} = \frac{1}{n}\mathbf{YY}^T$, where $\mathbf{C_Y}$ is a $m \times m$ diagonal matrix whose diagonal terms are the variance of particular columns in $\mathbf{Y}$. $\mathbf{C_Y}$ is also known as the covariance matrix of $\mathbf{Y}$.

We can find the principal components of $\mathbf{X}$ by computing $\mathbf{C_X}$, which can be found using the formula $\mathbf{C_X} = \frac{1}{n}\mathbf{XX}^T$. Once we have $\mathbf{C_X}$, we can find the eigenvectors of $\mathbf{C_X}$. These eigenvectors can be combined to form the principal component matrix $\mathbf{P}$ necessary to find $\mathbf{Y}$ [11]. If we want $\mathbf{Y}$ to have dimension a, then $\mathbf{P}$ should only be comprised of the eigenvectors corresponding to the a highest eigenvalues. Once we have $\mathbf{P}$, we can find $\mathbf{Y}$ by using the formula $\mathbf{Y} = \mathbf{PX}$. We can then perform the computations we need to do on $\mathbf{Y}$ instead of $\mathbf{X}$.

2.3 Word Embeddings

Word embedding is the process of representing words as real number vectors. These vectors are learned from a large text corpus and summarize the contexts in which the word has occurred within that corpus. Multiple methods of generating such vectors have been proposed in recent years. One such model proposed by Bengio *et al.* seeks to use a probabilistic feedforward neural network language model to generate word vectors. The model learned a distributed representation for each word along with the probability function for word sequences [12]. Bengio *et al.* were able to achieve significantly better results using the neural network when compared to previous works [12].

Mikolov *et al.* propose two different model architectures that seek to minimize the computational complexity. The first architecture, called the "Continuous Bag-of-Words Model" (CBOW), is similar to the feedforward neural network language model proposed by Bengio *et al.*, except a non-linear hidden layer is removed from the neural network, and the projection layer is shared for all words [2]. Mikolov *et al.* used a log-linear classifier with four history and four future words as the input with the training criterion to be to correctly classify the middle word [2]. Similarly, the second model proposed by Mikolov *et al.* is the continuous skip-gram model. They use a log-linear classifier to predict words within a certain range before and after the current word as the training criterion [2].

Mikolov *et al.* used an analogy model to verify that their architectures work as expected that will be used to evaluate the architecture proposed in this paper. This method of evaluation is discussed in section 4 of this paper. They found that their continuous skip-gram

model and CBOW model outperformed other neural network-based models by a significant margin when using the analogy test.

Later, Mikolov *et al.* expanded upon the Skip-Gram model by treating collocations as singular vectors rather than separate vectors [3]. For example, "Boston Globe" was treated as its own vector rather than the sum of the vectors for "Boston" and "Globe" because "Boston Globe" has a different meaning from the sum of its parts. They also found that summing together the vectors for two words yields words similar to the two original words. For example, the sum of the vectors corresponding to "Germany" and "capital" is similar to "Berlin" [3].

Pennington *et al.* propose the Global Vectors for Word Representation (GloVe) model. The GloVe model is trained using a weighted least squares problem defined in equation 1. In equation 1, w_i is the word vector corresponding to the i^{th} word, $\tilde{w}_j$ is the context word vector corresponding to the j^{th} word, b_i and $\tilde{b}_j$ are biases corresponding to w_i and $\tilde{w}_j$, respectively, and X_{ij} is the number of times that words i and j occur within the context of each other [4].

$$J = \Sigma_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{1}$$

The GloVe model was evaluated using a variety of tasks, including the analogy method used by Mikolov *et al.* as well as a word similarity task and a named entity recognition task. The GloVe model was able to outperform other methods of generating word vectors, including the CBOW model, the skip-gram model, and a singular value decomposition model [4].

3 Approach

3.1 Overview

GloVe is the most popular model used to generate word vectors. Internally, it is essentially a least-squares problem that is being used to evaluate each set of vectors and determine the strongest set [4]. The stated goal of this paper is to improve the GloVe algorithm. To do this, there are two options we have to improve upon GloVe: change the model used to generate the set of word vectors, or perform some post-processing operations on the set of word vectors itself. The solutions proposed in this paper mostly focus on post-processing, but they do utilize some internal changes. The GloVe implementation provided by the authors of the paper is broken up into four pieces as shown in the bulleted list below. This process is illustrated in Figure 1.

- `vocab_count` - generates a table of tokens that appear in the corpus sorted in descending order by their frequencies
- `cooccur` - generates the word-word cooccurrence matrix used in the model
- `shuffle` - shuffles the entries of the cooccurrence matrix so that `glove` does not read words in the same order they appeared in the corpus
- `glove` - generates the matrix of word vectors

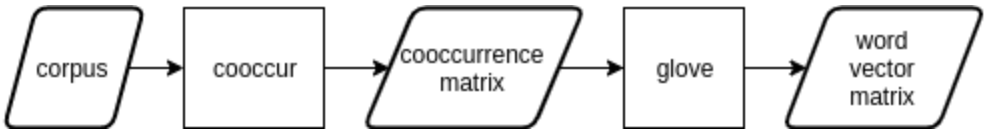

Figure 1: Diagram illustrating the GloVe process. Since `vocab_count` and `shuffle` are not modified in this paper, they are omitted from this diagram and all future diagrams.

3.2 Generating an Exclusive Matrix

GloVe does not explicitly model word order. However, it does weight the words in the cooccurrence matrix based on their distance from the target word. More specifically, this weight is $1/n$ if the words are n words apart. Our goal is to incorporate distance into GloVe explicitly. To do this, we focused on two modifications to GloVe. First, we focused on generating separate exclusive matrices for each position in a word's context. Then, we focused on post-processing multiple exclusive matrices to get a single result.

In order to get a finer control of how the matrices are generated, we needed exclusive matrices. By default, GloVe generates "inclusive" matrices, meaning that if the `cooccur` program is given a length of 5 and it is told to generate a symmetric matrix, it will count cooccurrences from the $i - 5^{th}$, $i - 4^{th}$, ..., $i + 4^{th}$, $i + 5^{th}$ positions relative to the i^{th} word. When given a length of 5, the `cooccur_exclusive` program will only take into account the $i - 5^{th}$ and the $i + 5^{th}$ words relative to the i^{th} word. A full description of these changes to the reference program can be found in Appendix A.

To give a clearer picture of how an inclusive matrix differs from an exclusive matrix, consider the following sentences:

"I went to the grocery store to buy some apples."

"Let's go to the store to return those clothes."

"What is the name of the store that sells phones?"

Say that we are interested in the cooccurrence values for "store". To keep the example simple, we can consider a symmetric context with a size of 2. So, to build the inclusive matrix, we can simply count the number of times each word within two words of "store" occurs. This means that we end up with the matrix shown in Table 1.

	the	grocery	store	to	buy	return	of	that	sells
store	3	1	0	3	1	1	1	1	1

Table 1: Partial inclusive coocurrence matrix with a range of 2 for the above sentences. Assume all words not represented by columns do not cooccur with "store".

Similarly, we can generate the exclusive matrix for the above sentences. The process is exactly the same as the process for generating the inclusive matrix, except the entries for "grocery" and "to" are not counted for the first sentence because they do not occur at the edge of the boundary. The exclusive matrix is shown in Table **??**.

We can also consider the exclusive matrix with boundaries at $i \pm 1$. Based on intuition, this matrix should be the difference between the inclusive $i \pm 2$ matrix and the exclusive $i \pm 2$, which are represented by Tables 1 and 2, respectively. This matrix can be seen in Table 3.

	the	grocery	store	to	buy	return	of	that	sells
store	1	0	0	1	1	1	1	0	1

Table 2: Partial exclusive coocurrence matrix with a range of 2 for the above sentences. Assume all words not represented by columns do not cooccur with "store".

	the	grocery	store	to	buy	return	of	that	sells
store	2	1	0	2	0	0	0	1	0

Table 3: Partial exclusive coocurrence matrix with a range of 1 for the above sentences. Assume all words not represented by columns do not cooccur with "store".

Once one exclusive matrix per position in the context is generated, we can consider using different methods of combining these matrices into a single matrix. This single matrix can then be passed to the same evaluation method used to evaluate the inclusive matrix as a baseline. In this paper, 4 methods of combining these exclusive matrices are defined and tested.

3.3 Method 1 - Direct Concatenation

The first method evaluated in this paper is the direct concatenation method. This process is depicted graphically in Figure 2. In this method, the exclusive word vector matrices are generated for each index in the window around a given word, i. They are then concatenated to generate the final word vector matrix.

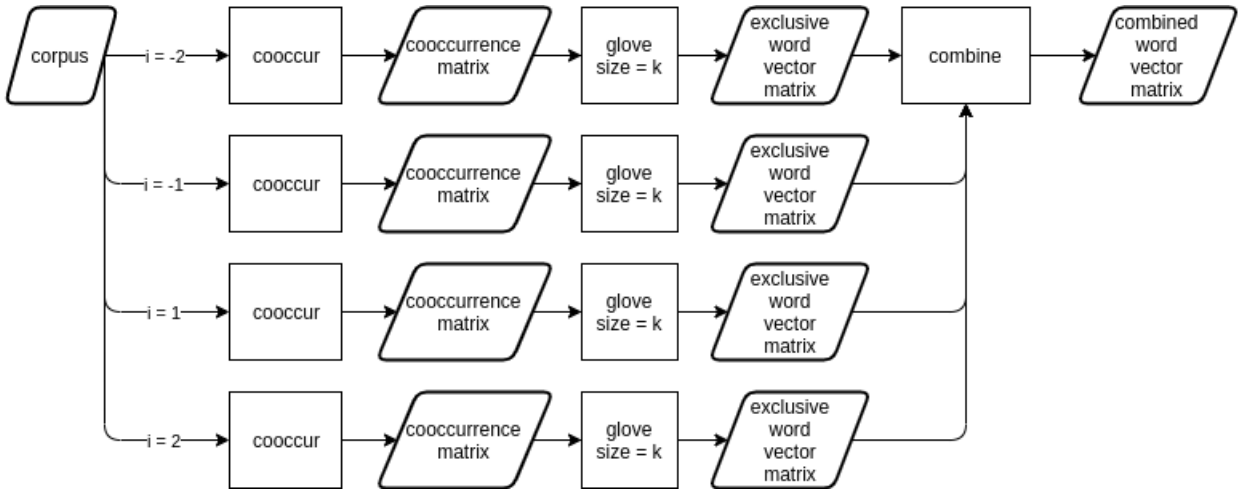

Figure 2: Diagram illustrating the GloVe process with direct concatenation with a range of $i \pm 2$. Note that the size of the vectors in the resulting matrix is $4k$.

To give an example, take the sentence used in the previous section as shown below. Say that the window size is from $i + 1$ to $i + 2$, and that we are interested in the word vector for "store". To build the overall word vector matrix, we must first generate the word vector matrix using solely the exclusive cooccurrence matrices at $i + 1$ and $i + 2$. Then, we can run the `glove` program to generate the word vector matrices from each of these exclusive cooccurrence matrices. Note that the dimension of the word vector matrices is a constant that is passed as a parameter to the `glove` program.

"I went to the grocery store to buy some apples."

Say that the i+1 word vector for "store" generated from the above sentence is $(1, 0.5, 0.75, 2.1)$ and that the $i + 2$ word vector for "store" is $(0.1, 3.8, 10, 1)$. Then, the overall word vector for "store" would become $(1, 0.5, 0.75, 2.1, 0.1, 3.8, 10, 1)$.

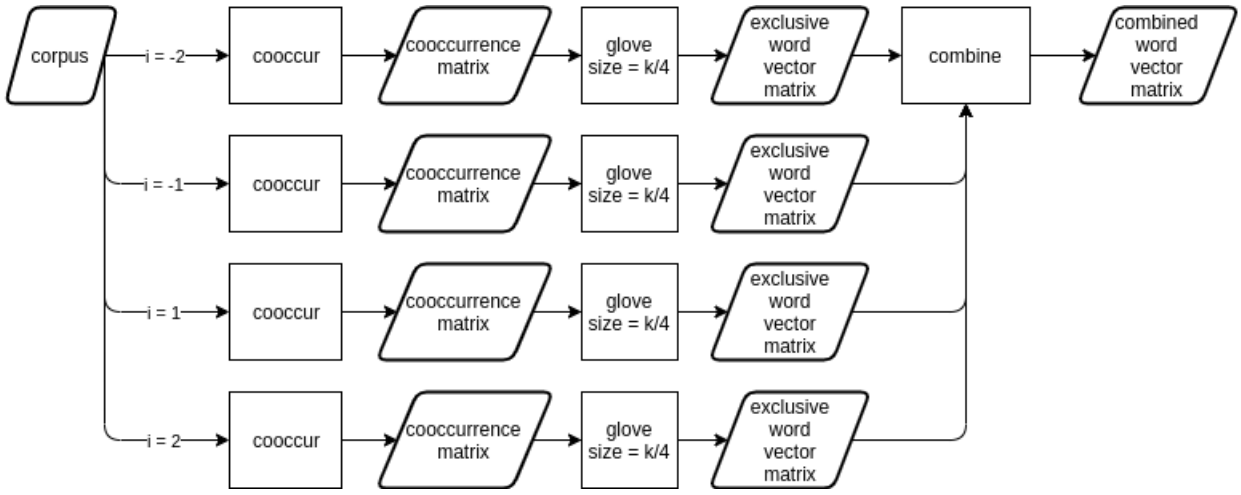

Figure 3: Diagram illustrating the GloVe process with reduced concatenation with a range of $i \pm 2$. Note that the size of the vectors in the resulting matrix is k.

3.4 Method 2 - Reduced Concatenation

The next method attempts to deal with an issue created in the first method: data size. The first method allows vectors to grow to a size that is a multiple of the size that `glove` generates. For example, if `glove` outputs vectors of length n and the window used to generate the word vector matrix is of size m, then the length of the vectors produced by method 1 is $m * n$. This is far too large, especially when dealing with a large corpus of text.

This method seeks to improve upon the data usage of the first method while still allowing the word vectors to retain information about where the words occurred relative to the i^{th} word. Here, the length of each vector is set to be the total desired length of the overall vector divided by the number of sub vectors. So, if the overall vector length was 4 and there were 2 subvectors, each subvector is of length 2. If the overall length does not divide evenly, then the remaining vector lengths are given to the vectors corresponding to the words closest to

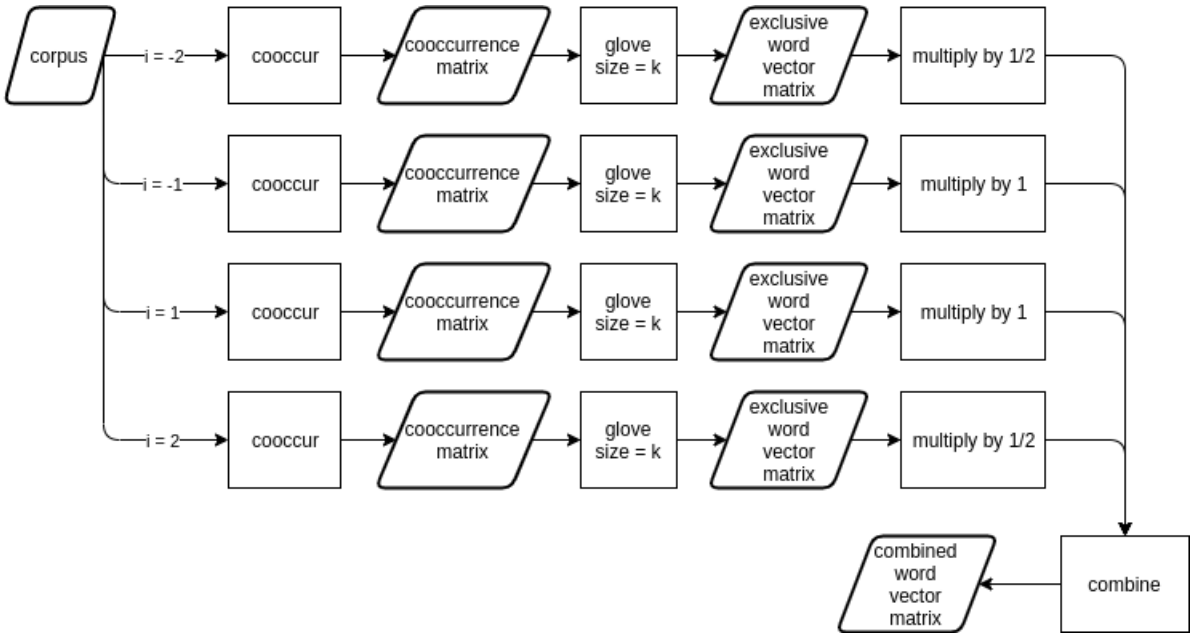

Figure 4: Diagram illustrating the GloVe process with weighted direct concatenation with a range of $i \pm 2$. Note that the size of the vectors in the resulting matrix is $4k$.

the i^{th} word. This method is illustrated in Figure 3.

3.5 Method 3 - Weighted Direct Concatenation

Direct concatenation also fails to account for the fact that words closer to the i^{th} word are more likely to bare more meaning to the context the word is used in. Using the same sentence used in section 3.3, the subvector related to "grocery" in the overall vector for "store" should be weighted more heavily than the subvector for "went". This method uses a proportional weighting scheme such that if a word is in the $i-2^{nd}$ position, then the values in its subvector are multiplied by 1/2. This process is illustrated in Figure 4.

4 Evaluation

The goal of these evaluations is to evaluate the overall quality of the word vectors generated by each of the proposed models against the word vectors generated by the original GloVe model. In the case of concepts so abstract as word vectors, "quality" is hard to define. This is especially true when considering that these word vectors are not necessarily purpose-built. They were constructed solely to better represent the relationships between words when represented as vectors.

As stated by Pennington *et al.* and Mikolov *et al.*, analogies provide a good evaluation metric for such vectors. This is because the word vectors maintain a linear relationship [2, 4]. As an example of this relationship, take the analogy "France is to Paris as Spain is to Madrid". As observed by Mikolov *et al.*, the vectors that correspond to France, Paris, Spain, and Madrid roughly exhibit the relationship shown in Equation 2.

$$vec(\text{“}Paris\text{”}) - vec(\text{“}France\text{”}) + vec(\text{“}Spain\text{”}) \approx vec(\text{“}Madrid\text{”}) \tag{2}$$

4.1 Dataset

The methods proposed in this paper were evaluated using the first 8.7 GB of text from a 2019 dump of Wikipedia text. The 8.7 GB cutoff was arbitrarily chosen due to time constraints. It was tokenized by removing XML tags, all punctuation, and all whitespace characters other than spaces between words. The corpus was then lowercased. Each article in the corpus was separated by a new line character as specified in the GloVe documentation.

4.2 Methods

For each method, the `cooccur` program was told to exclude words with a frequency less than 5. The `glove` program trained for a total of 50 iterations for each matrix it generated. The baseline GloVe method and reduced concatenation produced word vectors with 100 elements, and direct concatenation and weighted direct concatenation produced vectors that were $100w$ long, where w is the window size. The vectors were evaluated with varying window sizes as well to get an idea of how each method performs with more or less data. Each method was evaluated with window sizes from ± 1 word to ± 5 words from the target word.

A series of analogies were used to evaluate the set of vectors produced by a given method. Based on the description in section 4, these analogies can be effectively used to evaluate word vectors produced by GloVe. A total of 14 sets of analogies were used. These analogies were provided in the base GloVe implementation. They consist of a wide variety of analogies that should be answerable given such a large corpus of Wikipedia data. They contain analogies such as "Chicago is to Illinois as Dallas is to Texas", "Boy is to girl as father is to mother", and "Banana is to bananas as bird is to birds". These analogies evaluate both a more syntactic understanding of English as well as a semantic understanding of the words in the corpus.

For each analogy, the word vectors corresponding to the first three words in the analogy were fetched. Then, the vector sum and difference of the vectors were taken to get the "answer" vector as shown in Equation 3, where A, B, C, and D are the first, second, third, and fourth words in the analogy, respectively. The answer vector was then compared to all

of the vectors in the matrix. Whichever word corresponded to the most similar vector to the answer vector was said to be the predicted fourth word in the analogy. The predicted word was then compared to the given fourth word, and the correctness of the prediction was recorded. There were a total of 19544 analogies evaluated for each method.

$$A - B + C = D \tag{3}$$

4.3 Results

The results of the experiment described in section 4.2 are displayed below in Tables 4 and 5. Direct concatenation appears to outperform the GloVe baseline by roughly 10% accuracy across all tested window sizes. This is likely because direct concatenation has $2k$ times the number of elements in each vector when compared to GloVe, where k is the window size. However, this is a fair comparison because each element in the vectors produced by the GloVe baseline takes information from all word positions, whereas the direct concatenation scheme only takes information from one word position for each element.

	1	2	3	4	5
GloVe	2709	5314	6532	7096	7089
Direct concatenation	4273	7516	8517	8892	8891
Reduced concatenation	1559	577	103	22	3
Weighted direct concatenation	4336	6383	6273	6978	7238

Table 4: Number of correct analogies for each concatenation scheme by window size

	1	2	3	4	5
GloVe	13.9%	27.2%	33.4%	36.3%	36.3%
Direct concatenation	21.9%	38.5%	43.6%	45.5%	46.0%
Reduced concatenation	8.0%	3.0%	0.5%	0.1%	$< 0.1\%$
Weighted direct concatenation	22.2%	32.7%	32.1%	35.7%	37.0%

Table 5: Accuracy percentage for each concatenation scheme by window size

Reduced concatenation has extremely low results when compared to any other method, however. This is likely because each word position in the reduced concatenation method only has $\frac{50}{2k}$ elements to represent it, where k is the window size. Weighted direct concatenation performs better than the GloVe baseline for smaller window sizes, but it is relatively even with the baseline for larger window sizes.

5 Discussion

5.1 Conclusions

The main goal of this paper was to incorporate word order into GloVe word embedding. Three methods of doing so were investigated: direct concatenation, reduced concatenation, and weighted direct concatenation. Each of these methods can be found in section 3. As shown in the results section, direct concatenation outperforms the GloVe baseline as well as all other proposed methods. This is likely because direct concatenation has more elements in each vector which are used to give an explicit representation of word order.

5.2 Future Work

Future work on the topic of explicit inclusion of word order in GloVe word embedding could test how a more dynamic approach to expressing word order performs. For example, one may consider the direct concatenation approach with the output matrix reduced using dimensionality reduction techniques. This approach may eliminate the columns in the vectors that contain only noise. This would also reduce the increases in space complexity brought about by direct concatenation. One may also consider how modifications to the word-word cooccurrence matrices affects the results before it is processed by GloVe. For example, the exclusive matrices could be combined using functions other than inverse proportional weighting as used in the base GloVe algorithm.

References

[1] S. Gauch and M. K. Chong, "Automatic word similarity detection for trec 4 query expansion," in *TREC*, 1995.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *ICLR Workshop*, 2013.

[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 3111–3119, Curran Associates, Inc., 2013.

[4] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1532–1543, Association for Computational Linguistics, Oct. 2014.

[5] Z. Harris, "Mathematical structures of language," *Interscience tracts in pure and applied mathematics*, 1968.

[6] D. Hindle, "Noun classification from predicate-argument structures," in *28th Annual meeting of the Association for Computational Linguistics*, pp. 268–275, 1990.

[7] F. Pereira, N. Tishby, and L. Lee, "Distributional clustering of english words," in *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pp. 183–190, Association for Computational Linguistics, 1993.

[8] D. Lin, "Automatic retrieval and clustering of similar words," in *COLING 1998 Volume 2: The 17th International Conference on Computational Linguistics*, 1998.

[9] D. Lin, S. Zhao, L. Qin, and M. Zhou, "Identifying synonyms among distributionally similar words," in *International Joint Conferences on Artificial Intelligence*, vol. 3, pp. 1492–1493, 2003.

[10] S. Scheible, S. S. Im Walde, and S. Springorum, "Uncovering distributional differences between synonyms and antonyms in a word space model," in *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pp. 489–497, 2013.

[11] J. Shlens, "A tutorial on principal component analysis," *Google Research*, 2014.

[12] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

Appendices

A Code changes to GloVe

The only program modified for the purposes of this paper was the `cooccur` program. All original code was downloaded from the GloVe project website. On line 354 of the original implementation of `cooccur`, there is a for-loop that iterates over each element in the window size. This for-loop was replaced by a single statement that calculates the index of the element that is `window_size` elements away from the current element. In other words, the code in Figure 5 becomes the code in Figure 6.

```
for(k = j - 1; k >= ( (j > window_size) ? j - window_size : 0 ); k--) {
...
}
```

Figure 5: Original for-loop in the base GloVe `cooccur` implementation. `j` represents the index of the target word in the internal representation.

```
k = j - window_size;

if(k >= ((j > window_size) ? j - window_size : 0)) {

...

}
```

Figure 6: Code that replaced the for-loop in Figure 5