AMSI VACATION RESEARCH SCHOLARSHIPS 2022–23

SET YOUR SIGHTS ON RESEARCH THIS SUMMER

Moving through Word Vector Space

Gabriel Schussler

Supervised by Stuart Fitzpatrick and Laurence Park

Western Sydney University

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

Contents

Abstract

This report aims to investigate and quantify the use of distance functions for measuring word embeddings generated by a Word2Vec model, to identify which function has the best performance for measuring semantic word similarity. Through a comparison of two word similarity datasets (Word353, SimLex999), the performance of Cosine similarity, and Minkowski distance with various p values was tested. The results indicate that cosine similarity is the best measurement of word similarity, followed by Minkowski distances with a p value greater than 4. Euclidean distance performed moderately well, while Manhattan and Chebyshev distance perform the worst.

1 Introduction

As the popularity of machine learning rapidly increases, so to does the need for easy-to-use techniques for converting non-numerical information to numerical information. Natural Language Processing is the branch of computer science concerned with teaching computers how to under words and text. There are many types of NLP techniques, such as traditional machine learning techniques, deep learning (neural networks) and statistical NLP. One such popular machine learning technique is Word2Vec. The Word2Vec algorithm was initially developed in 2013 (Mikolov, Chen, et al. 2013) and has been improved since (Mikolov, Grave, et al. 2017). Word2Vec allows for words and text to be converted to a "word vector space", where the words' location in this space relates to the semantic similarity of those words. This report explores different methods of measuring the similarity of words in this vector, to determine which technique offers the best results.

2 What is Word2Vec?

Word2Vec consists of a shallow neural network that is used to produce word embeddings for each word in the input vocabulary. These word embeddings can be used to relate such words based on the proximities within the space. As a simple example, a 2-dimensional word embedding for 4 words might look like figure 1.

It's easy to see in the figure 1 example that 'car' and 'tractor' are more related, and 'apple' and 'carrot' are more related. This highlights the usefulness of Word2Vec, it allows the numerical representation of semantic word similarity.

When words are embedded into a vector space we can also perform vector operations on these words, for example in figure 2, we can add the words apple and carrot to get a new vector.

These vector operations can be used to examine relationships between words. A commonly used example is:

- [King] - [Man] + [Woman] = [Queen]

This aligns well with our intuitions of how word similarity works, as (most of) the difference between a king and a queen is the one is a man and the other is a woman.

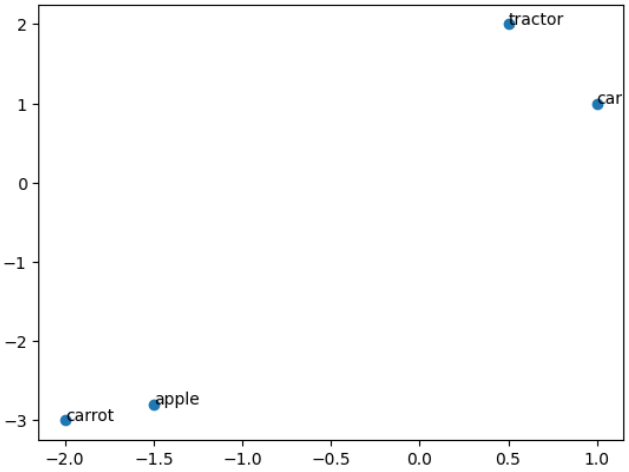

Figure 1: A simple example of word embeddings

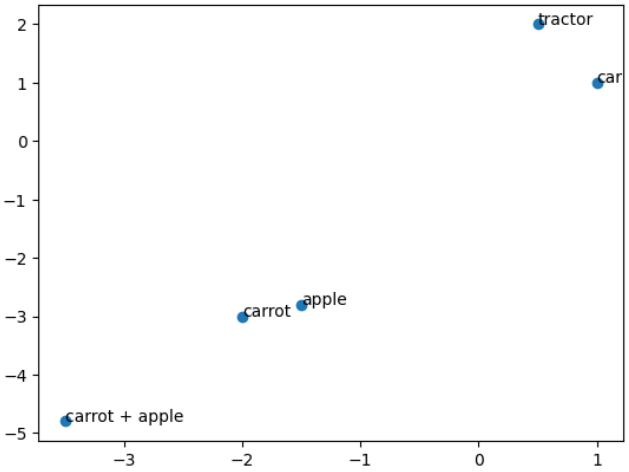

Figure 2: A simple example of vector operations on word embeddings

3 How does Word2Vec work?

Word2Vec uses a shallow neural network trained to predict an output based on a set of input context words (Continous bag-of-words model) or to predict output words based on a set an input words (Skip-gram model). However the neural network is not used in the final model, instead, the hidden layer is used as the word vector space.

When creating a Word2Vec model, a parameter that must be decided on is the context window. The context window is the number of words on either side of each word of the text. Another parameter is the number of dimensions for the word vectors, while any number can be chosen, typically 300 dimensions is consider to get the best performance.

Word2Vec uses two main models, *continuous bag-of-words* (CBOW) and *Skip-gram*. While the outputs of these models are fundamentally the same (the Skip-Gram has better performance), there are some minor differences in how they work.

3.1 Continous bag-of-words

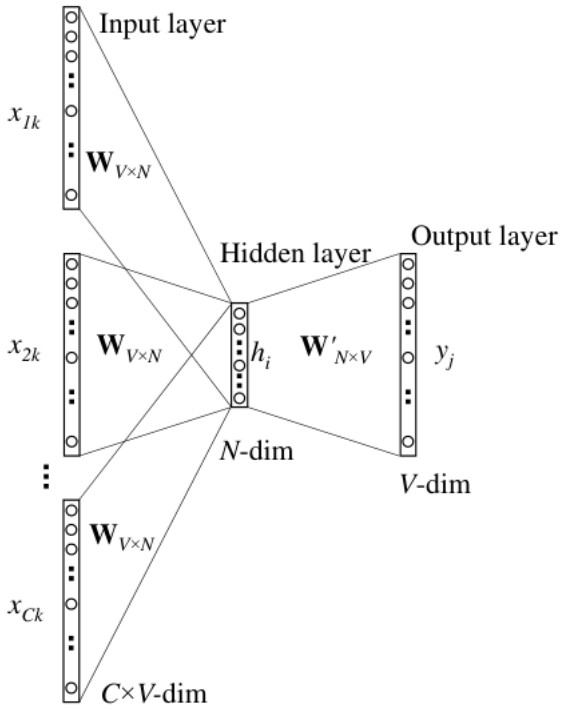

Figure 3: A simplified diagram of a CBOW neural network. Source: Rong 2016

As stated above the CBOW model for Word2Vec takes in a set of context words and trains a neural network to predict a target word as shown in figure 3. The vocabulary is converted to one hot encoded vector. There are two vector representations of each word, the vector representation between the input and hidden layer vector, and the vector representation between the hidden and output layer. These vectors are fed into the neural

network, where a score for each word is calculated. (Rong 2016)

$$\text{score}_{\text{word}} = \text{vector}^1_{\text{word}} \times \text{vector}^2_{\text{word}} = u_{\text{word}}$$

This score is then used in the softmax function to obtain a categorical distribution for each word.

$$p(\text{Output}_{\text{word}}|\text{Input}_{\text{word}}) = \frac{\exp(\text{score}_{\text{word}})}{\sum_{i=1}^{V} \exp(\text{score}_i)} = y_{\text{word}}$$

Where i is every word but the input word in the vocabulary. From this function, the loss function used for the neural network can be derived.

$$E = \max p(\text{Output}_{\text{word}}|\text{Input}_{\text{word}}) = \max y_{\text{word}} \tag{1}$$

$$= \max \log y_{\text{word}} \tag{2}$$

$$= u_{\text{word}} - \log \sum_{j'=1}^{V} \exp(\text{score}_{j'}) := -E \tag{3}$$

$$E = -\log p(\text{Output}_{\text{word}}|\text{Input}_{\text{word}}) = \text{Loss function} \tag{4}$$

Where E is equal to the loss.

3.2 Skip-gram model

The skip-gram model can be thought of as the inverse of the CBOW model. It uses an input word to predict a set of output words, this means that C multinomial distributions are outputted, where C is the size of the context window. The loss function changes to the following:

$$E = -\log p(O_{w1}, O_{w2}, ...O_{wC}|I_w) \tag{5}$$

$$= -\log \prod_{c=1}^{C} \frac{\exp(\text{score}_{\text{word}})}{\sum_{i=1}^{V} \exp(\text{score}_i)} \tag{6}$$

$$= -\sum_{c=1}^{C} \text{score}_{\text{context}} + C \times \log \sum_{i'=1}^{V} \exp(\text{score}'_i) \tag{7}$$

4 Applications of Word2Vec

Word2Vec has numerous applications, especially when it comes to machine learning. The clearest example is the ability to convert text to a numerical representation, which allows for much easier integration with machine learning methods. The algorithm used in Word2Vec has also been used to measure similarity across documents [1] and biological information Asgari and Mofrad 2015. A massive advantage of Word2Vec is that it can be used embedded words in any language or text as long as the quantity of the text is large enough.

[1] https://github.com/v1shwa/document-similarity

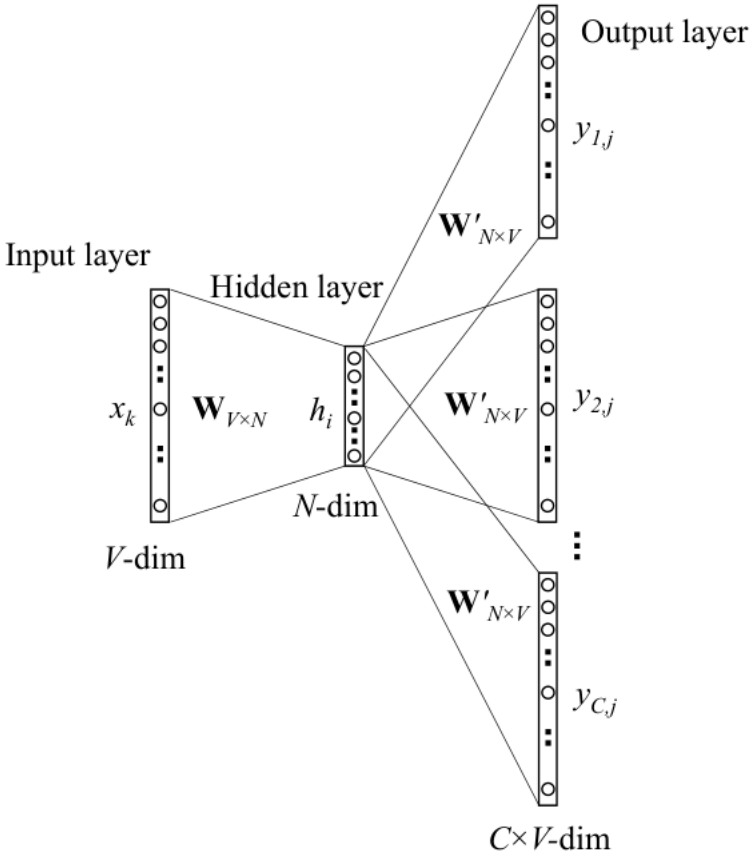

Figure 4: A simplified diagram of the Skip-gram neural network. Source: Rong 2016

5 How to train a Word2Vec Model

Creating a Word2Vec model on a custom dataset is easily enabled by the python library Gensim. The process is as simple as feeding a corpus of documents in to the function and getting output. While Word2Vec isn't an NLP task, it does benefit from text preprocessing. The "Gensim" library in python does include its own text preprocessing.

6 The Research

For this report, we examined the use of several distance functions to determine their performance in measuring word similarity. While it is possible and relatively easy to train a Word2Vec model, for the sake of consistency and reproducibility, we used the pre-trained Google News Word2Vec model. To determine the efficiency of each distance function, we used two word similarity datasets. The first dataset is Word353 (Agirre et al. 2009) containing 353 word pairings, the similarity between each pair was determined by an average score between 0 and 10, with 10 being the most related, from 13 - 16 human judgements for each word pair. The second dataset is simLex999 (Hill, Reichart, and Korhonen 2015), which contains 999 word pairs, along with a similarity score between 0-6 based on the rating of 500 US residents.

The distance functions used included Cosine Similarity and the Minkowski distance.

6.1 Cosine Similarity

Cosine Similarity is a result of cosine distance, whereas the cosine distance decreases the cosine similarity increases. Unlike euclidean and manhattan distance, which measure the distance between two points directly. Cosine similarity measures the angle between the two points. Cosine similarity is a very common technique used to measure the similarity between two vectors.

$$\text{Cosine Similarity} = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

6.2 Minkowski Distance

Minkowski Distance is a generalization of euclidean and manhattan distance in normed vector space.

$$D(X,Y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$$

The p value determines the distance being measured, when $p = 1$, the manhattan distance is measured when $p = 2$ the euclidean distance is measured. As $p \to \infty$, the Chebyshev distance is measured.

Note: cosine distance and euclidean distance are linearly related in a normed vector space.

$$D_{\text{euclidean}} = 2 - 2\cos(X,Y) \tag{8}$$

Metric	Spearman R correlation	P-Value
Cosine Similarity	0.772084	1.94901e-41
Manhattan Dist.	0.169496	0.0156253
Euclidean Dist.	0.562136	2.62299e-18
Mink. Dist. (p = 3)	0.318639	3.60032e-06
Mink. Dist. (p = 4)	0.74817	1.16159e-37
Mink. Dist. (p = 5)	0.471244	1.28481e-12
Mink. Dist. (p = 6)	0.709641	2.13132e-32
Mink. Dist. (p = 7)	0.496584	4.88427e-14
Mink. Dist. (p = 8)	0.666953	1.73069e-27
Mink. Dist. (p = 50)	0.589361	2.24426e-20
Chebyshev Dist. (p =∞)	0.231747	0.000878181

Table 1: Table showing Word353 spearman correlation across several distance functions

6.3 Method of Comparison

We used both of these data sets to compare different techniques of comparing similarity in the Word2Vec vector space. We measure the cosine similarity between each word, along with the Euclidean, Manhattan and various Minkowski distances. Once we had the measurements for each word pairing, we converted the measurements to ranks, so we could perform a Spearman Rank correlation, in which every measurement correlated strongest with the similarity ranks from the two data sets was considered to be the best measurement.

7 Results

7.1 Word353 Results

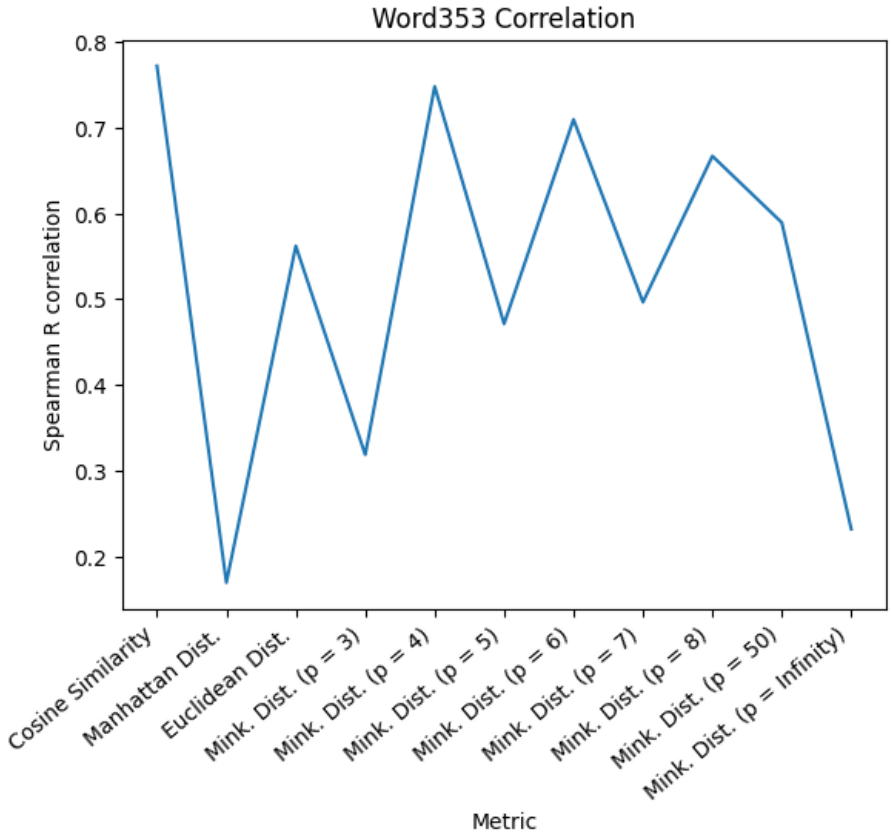

Figure 5: A plot of correlation across different metrics used to measure the Word353 database.

Metric	Spearman R correlation	P-Value
Cosine Similarity	0.44208	4.75799e-49
Manhattan Dist.	0.0938829	0.00297604
Euclidean Dist.	0.44208	4.75774e-49
Mink. Dist. (p = 3)	0.24203	8.77584e-15
Mink. Dist. (p = 4)	0.435195	2.01115e-47
Mink. Dist. (p = 5)	0.306126	4.06793e-23
Mink. Dist. (p = 6)	0.420068	5.58137e-44
Mink. Dist. (p = 7)	0.320897	2.32043e-25
Mink. Dist. (p = 8)	0.40502	1.00433e-40
Mink. Dist. (p = 50)	0.36727	2.92065e-33
Chebyshev Dist. (p = ∞)	0.182397	6.37648e-09

Table 2: Table showing Simlex999 spearman correlation across several distance functions

7.2 SimLex Results

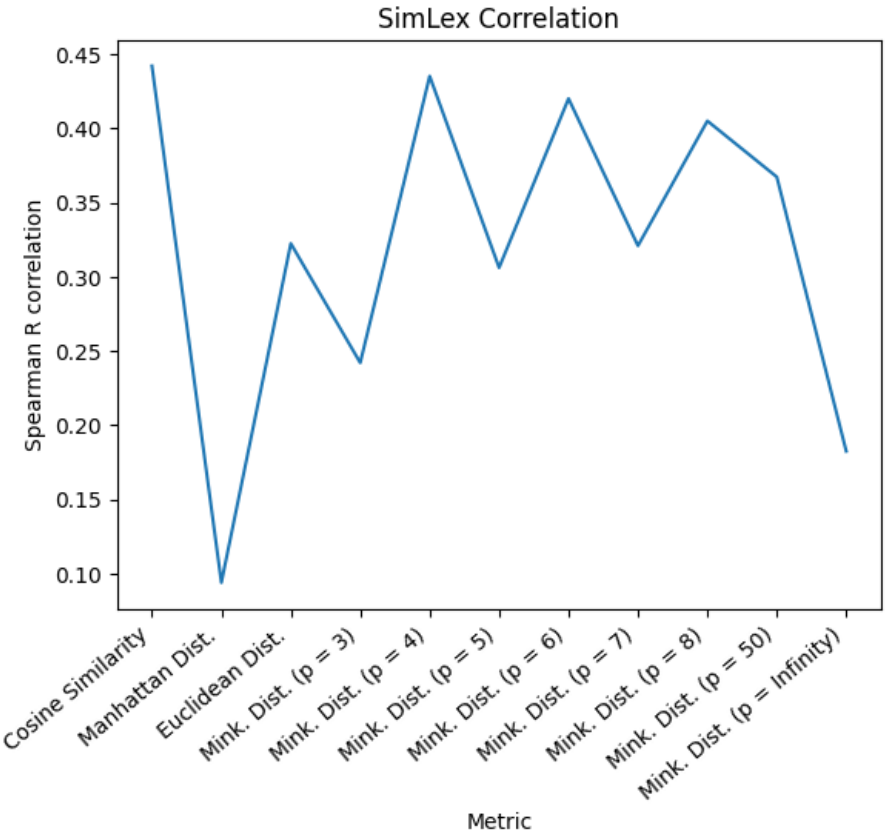

Figure 6: A plot of correlation across different metrics used to measure the SimLex word database.

7.3 Discussion

The results above indicate that cosine similarity is the best metric for measuring word similarity. As the p value for the Minkowski distance increase, the correlation between the similarity rankings calculated from the dataset and the similarity ranks for that distance decrease. The exception is the manhattan distance performing the worst of Minkowski variations.

There is also a clear difference in the overall strength of the correlation between the Word353 dataset and the SimLex dataset. This is most likely due to the difference in the method used by each dataset to determine which words are most similar. It can be argued that the SimLex dataset has a greater focus on word similarity when compared word353 which focuses more on association. For example, while coffee and cup are strongly associated, their respective meanings are very different.

This highlights the issue that while the Word2Vec model does a good job at finding word associations, its ability to find semantic similarity is not as good.

There is a strong pattern with odd p values performing significantly worse than even p values. The reason for this is not entirely clear and should be further researched.

8 Conclusion

- By comparing the two word similarity datasets (Word353, SimLex999), the performance of Cosine similarity, Euclidean distance, Manhattan Distance and Minkowski distances with various p values were examined.
- Th following results can be concluded, cosine similarity was found to be the best measurement of word similarity, followed by Minkowski distances with a p value greater than 4.
- Euclidean distance performed moderately well, while Manhattan and Chebyshev distance performed the worst.
- These results were unsurprising, and indicate that the current use of Cosine similarity is justified as the technique for measuring word embedding similarity.

References

Agirre, Eneko et al. (2009). "A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics on - NAACL '09*. Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics. Boulder, Colorado: Association for Computational Linguistics, p. 19. ISBN: 978-1-932432-41-1. DOI: 10.3115/1620754.1620758.

Asgari, Ehsaneddin and Mohammad R. K. Mofrad (Nov. 10, 2015). "Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics". In: *PLOS ONE* 10.11. Ed. by Firas H Kobeissy, e0141287. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0141287.

Hill, Felix, Roi Reichart, and Anna Korhonen (Dec. 2015). "SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation". In: *Computational Linguistics* 41.4, pp. 665–695. ISSN: 0891-2017, 1530-9312. DOI: 10.1162/COLI_a_00237.

Mikolov, Tomas, Kai Chen, et al. (Sept. 6, 2013). "Efficient Estimation of Word Representations in Vector Space". arXiv: 1301.3781 [cs].

Mikolov, Tomas, Edouard Grave, et al. (Dec. 26, 2017). *Advances in Pre-Training Distributed Word Representations*. arXiv: 1712.09405 [cs].

Rong, Xin (June 5, 2016). *Word2vec Parameter Learning Explained*. arXiv: 1411.2738 [cs].