Word Embeddings and word2vec

Michael Gira, Andrew Janssen, Eric Jore

May 2020

Abstract

Word Embedding is the process of translating human readable text into a machine understandable format. Here, understandable is different from just readable. Understandable means that the meaning and relationships behind the text are also preserved.

In this activity, students will first explore the basics of word embeddings, progressing up to the word2vec algorithm. They will look at how one trains the model, and its various applications in machine learning problems. Problems will show students what the input and output of the word2vec algorithm is and allow students to play around with the relationships between similar and different words.

After getting a grasp of word2vec concepts, students will apply the word2vec algorithm on an email spam classification problem. They will also implement a bag-of-words binary linear classifier on the same dataset and compare the performance. Students will train their models on a set of training data, find the optimal lambda parameters from a tuning set, and evaluate the final performance of the model on a testing set. They will then compare the average error rate of both models and other metrics such as time required to train.
Students will:

- Learn about word embeddings and why they are useful
- Learn about word embedding using bag-of-words SVD
- Learn what the word2vec algorithm is and how it differs from bag-of-words
- Learn how to implement word2vec in a spam email classification problem
- Compare the tradeoffs and performance between bag-of-words and word2vec

1 Background

1.1 Representing Words - A Naïve Approach

One approach to feed words into a machine learning model is by simply representing each word in a set of N vocabulary with an n-dimensional vector. One could assign each word an index in the vector. The represented word would have a 1 in its respective index and 0 for all other locations. Take the sentence:

"I feel happy and great!"

Each of the unique words in this sentence can be represented as follows:

I = [1, 0, 0, 0, 0]
feel = [0, 1, 0, 0, 0]
happy = [0, 0, 1, 0, 0]
and = [0, 0, 0, 1, 0]
great = [0, 0, 0, 0, 1]

These vectors could then be the inputs to a classifier model. However, each of these vectors are orthogonal, which means that "happy" is as similar to "great" as it is to "and." There are no relationships represented between these words. Applying this method is problematic when you do not have enough training data for all words, and synonymous words are treated as completely separate. Take the example use case of sentiment analysis. A classifier can determine whether a sentence is happy or sad. However, "Have a good day!" and "Have a great day!" will be treated as completely different sentences because the vectors of "good" and "great" are orthogonal, and the encodings show no relationship between them. This problem reveals the motivation for creating word embeddings.

1.2 Introduction to Word Embeddings

Word embeddings are a class of algorithms that fall under the domain of Natural Language Processing (NLP). Word embeddings assign numerical vectors to words, which try to encode meaning and relationships behind these words. Similar-meaning words will have similar vectors. To form these vectors, word embeddings have to make some assumption about how they can find the meaning of a word.

1.3 Distributional Hypothesis

A popular hypothesis for developing word embeddings was popularized by Firth that stated "a word is characterized by the company it keeps" [Firth, change to number]. That is, words appearing in similar contexts will have similar meanings. For example:

"Have a **good** day!"
"Have a **great** day!"

The words "good" and "great" appear in similar contexts and will, therefore, have similar vectors. Word embeddings will scan through a large corpus of text to find patterns like these. When looking for context, a range is normally used to pick nearby words. If the range is too small, the context words will be flooded with modifying words such as "a" or "the." If the range is too big, words may lose meaning, as many different words will share a very similar context. The appropriate range to use may change based on the writing style of the author or based on the subject matter. Many models found online use a range of 10 words [1].

1.4 Matrix Representation

Many word embeddings start with a co-occurrence matrix. A co-occurrence matrix carries information about each word and their context, where a context is a collection of nearby words. A context of ± 2 might look like this (the orange represents the target word and the green represents its context):

The Cat In The Hat Is A Fine Cat

By looking at the context of words and applying the Distributional Hypothesis, some measure of similarity can be found. In this example, the pure number of times a word is seen will be used. Alternate approaches may perform some cleanup, such as punishing common words so the word "the" doesn't dominate the dataset. The co-occurrence matrix counting the pure number of times that words are observed in a context, looks like the following. Note that each row represents a target word and each column represents a word in that context.

	The	Cat	In	Hat	Is	A	Fine
The	0	2	2	1	1	0	0
Cat	2	0	1	0	0	1	1
In	2	1	0	1	0	0	0
Hat	1	0	1	0	1	1	0
Is	1	0	0	1	0	1	1
A	0	1	0	1	1	0	1
Fine	0	1	0	0	1	1	0

(Rows are words, columns are words in the surrounding context)

Note also that the sentence "The cat in the hoodie is a fine cat" would provide an identical matrix, just with the word "hat" replaced with the word "hoodie." Similarity between rows in a co-occurrence matrix suggests some similarity between words. Typically, the cosine similarity is used to evaluate similarity. Note that a cosine similarity of -1 means two vectors are complete opposites and a similarity of 1 means they are very similar. The cosine similarity is defined as:

$$\text{sim}(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|}$$

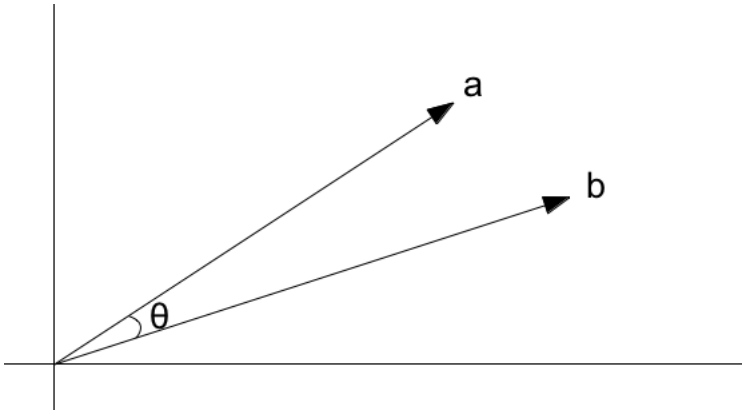

Figure 1: Graphical representation of cosine similarity [15]

Since working with a massive, raw co-occurrence matrix is expensive as well as prone to noise, some generalizing is needed. It is in this generalization step where a large amount of the differences between approaches appear.

1.5 SVD Implementation

One way of generalizing this data, extracting important features, is to take the SVD. By truncating the singular values (S) to a "good" size, each row of the matrix U can be used as an embedding of the word corresponding to that row [14]. These word vectors are then ready for the chosen application. Back to the example of sentiment analysis and classifying the emotion of a particular sentence, word embeddings enable accurate classification of words that were not in the training set. For example, because "good" and "great" should have similar contexts and therefore similar word embedding vectors, "Have a good day" and "Have a great day" should both be classified as happy, even if one of the words were not in the training set.

Although SVD is a quite accurate method of obtaining a model, it does suffer in being slow to train. One faster way to train a model is to use a neural network approach, like word2vec [6].

2 Introduction to word2vec

The word2vec algorithm, proposed in 2013, is one of the most well-known algorithms for producing word embeddings [9]. The word2vec algorithm marked a pivotal moment for word embeddings because machine learning techniques and an increase of computing capabilities enabled word embeddings to become high-quality and mainstream. [13] Since then, other neural-network based algorithms have been developed such as Global Vectors for Word Representation (GloVe) and FastText [11, 3]. Word2vec comes in two flavors, which mainly differ on how you want to think about the problem. Each variant has its benefits and tradeoffs (discussed below). These flavors are Continuous Bag-of-Words and Skip-gram.

2.1 Continuous Bag-of-Words (CBOW) Algorithm

The Continuous Bag-of-Words variant of word2vec approaches the problem as a "fill in the blank". By inputting the context, or surrounding words, of the target, we expect to output the target. This process is done with one hidden layer to create the decision boundary.

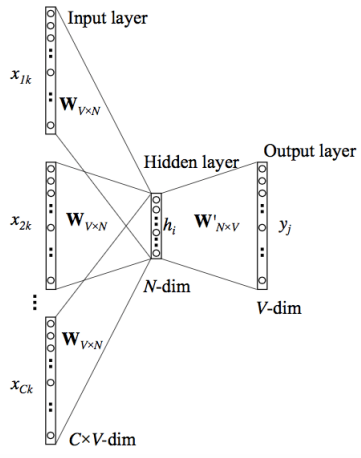

Figure 2: The 2-layer neural network architecture to create word embeddings with CBOW [7]

With our input of context word vectors, the neural network calculates the embedded word vectors and we then compute the average of the vectors to find a score vector, which is used to predict the context word. For each word input, the embedding for that word is the result it contributes to the central hidden layer. The number of hidden nodes is the dimensionality of the embedding. Increasing the dimensionality of the encoding may lead to better results if there is sufficient data and real differences to justify the change in dimension. While encodings are sometimes seen with as many as 1000 dimensions or as few as 25, between 100 and 300 dimensions is common. It should be noted that the dimension chosen has to be done before training takes place; the analyst has to know from prior knowledge what dimension would do a good job representing the data. In contrast, SVD allows for truncation of a custom size after having seen how the singular values look.

2.2 Skip-gram Algorithm

While continuous bag-of-words works by predicting a word from its surroundings, the skip-gram variant of word2vec works in the opposite direction: predicting the surroundings of a given word. To train this model, a neural network is set up where the input is our chosen word. Each context word is then guessed individually. This means that the output to be trained on in one trial would be exactly one of the context words [4]. Below is a diagram that shows the structure of the neural network.

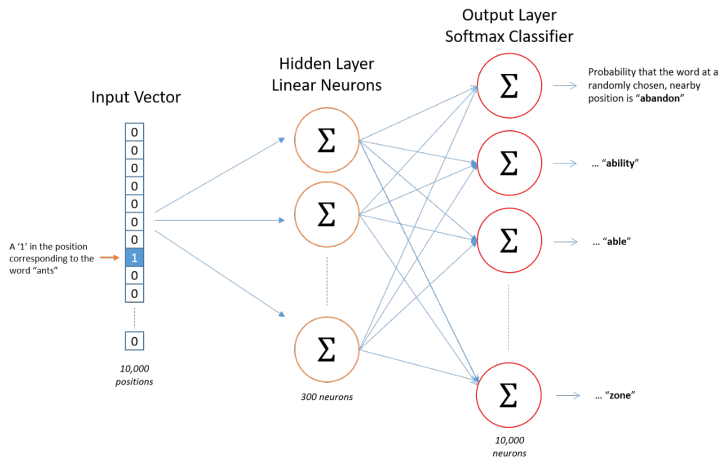

Figure 3: The 2-layer network architecture to create word embeddings with Skip-gram [2]

To get the vector values that we use as an encoding for a single word, we input the word and observe values on the center neuron layer. Here, the degree of the encoding is also the number of hidden neurons. Since synonymous words will have similar surroundings (according to the Distributional Hypothesis), this will lead to vector values close to one another. The distance between encodings of words embeds relationships between them.

2.3 CBOW vs Skip-gram Comparison

On average, while skip-gram has been found to find better representations for infrequent words, CBOW is faster. We will now dive into the math of the skip-gram variant.

2.4 The Softmax Function

The softmax function is a transformation of the final outputs of each algorithm that makes the resulting vector values always positive and add to one. As u_0 and v_c are more similar, the larger the numerator will be. The output then approximates a probability distribution of words. The probability of output "o" given context word "c" is given by the softmax function:

$$P(O = o \,|\, C = c) = \frac{\exp(u_o^T v_c)}{\sum_{w \epsilon Vocab} \exp(u_w^T v_c)}$$

In practice, the computation for this value for every node is very expensive. Approximations are commonly used to speed training time. More discussion on this topic is covered under the "Optimization" section.

2.5 Math Foundation for the Skip-gram Algorithm

The goal of the Skip-Gram algorithm is to predict the context word. To achieve this goal, we can maximize the probability that the correct word is chosen. For the purpose of this explanation, we will use probability and likelihood synonymously. At every step in training, the likelihood of a context word is computed. Since not just one word is the desired result of the whole model, the likelihood of getting everything right (the product of likelihoods) is used to gauge performance. This gives the likelihood that every output is guessed correctly. We try to maximize this value for the overall model. The likelihood function (the likelihood of getting everything correct) is given, where the capital pi symbol is similar to capital sigma symbol but denotes a product rather than a sum [10]:

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

Since the derivative of this function is complicated due to the product rule, an equivalent function is very useful. Since log(x) and x are both similar in being monotonically increasing, where x>0 (implied by the softmax function), the log-likelihood can be used. Since the product of terms is in the range (0,1) and the log of such a number is negative, minimizing the negative log-likelihood is often used. This simplifies the product to be a simple sum, which is easy to derive. The negative log-likelihood function is given:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

In short, the negative log-likelihood function J can be thought of as a tractable transformation of L. The Skip-Gram algorithm will try to minimize J, which equivalently maximizes L. From here, partial derivatives can be taken of this cost function to find the backpropagation formulas.

2.6 Optimization

One common optimization used to improve word2vec is the introduction of the hierarchical softmax. While the softmax function needs to compute the probabilities for every word, the hierarchical softmax instead calculates probabilities as a binary tree where each leaf of the tree represents a word and each edge contains a probability . In order to find the probability of a word, the algorithm takes a walk to the leaf, multiplying the probability of each edge on the path [12].

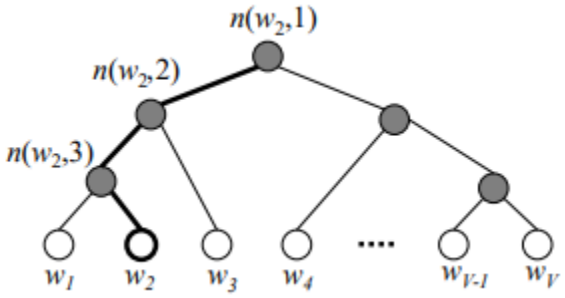

Figure 4: Huffman tree of probabilities [12]

This optimization reduces the time complexity from O(N) to O(logN). When implementing a smart reply feature in Allo, Google remarked on the improvement made from switching to hierarchical softmax as the time required to generate a response was lowered from over half a second to less than 200ms [8].

2.7 Ethical Considerations

Both a benefit and consequence of the word embeddings is that they model the relationships of words within text. However, these algorithms also model the implicit biases within their human-generated training text. Like all other machine learning systems, it is crucial to consider where the source of the training data is coming from and how to eliminate as much bias as possible.

For example, word embeddings can be used for sentiment analysis. Training data imperfections could flag specific types of words to be more of a certain emotion, disproportionally targeting certain groups. The stakes skyrocket as algorithms like word2vec become ingrained in everyday systems from grading essays, to reviewing job resumes, to government surveillance (ignoring the other ethical considerations of the latter).

3 Warm-up

1. According to the Distributional Hypothesis, which of the words is most similar to the word "**birds**" given the context: "I heard many **birds** in the forest."
   (a) **Fish** with the context "I saw lots of **fish** in the river."
   (b) **Bees** with the context "I heard many **bees** in the forest."
   (c) **Air** with the context "I felt the **air** on the palm of my hand."
   (d) **Rico** with the context "I caught the pass from Uncle **Rico**."
2. Fill in the blank: The (**continuous bag-of-words/skip-gram**) algorithm is faster to train, while the (**continuous bag-of-words/skip-gram**) algorithm tends to represent rarer words better.
3. Compute the value of the Softmax function given the following parameters:

$$U = \begin{bmatrix} 2 & 3 & 4 \\ 4 & 3 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \qquad V = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^T$$

4. Would you expect word2vec to create a meaningful representation of words with multiple meanings such as "light"?

4 Main Activity

Estimated Time: 15 minutes for P1, 15 minutes for P2, 15 minutes for P3

1. We will now find the co-occurrence matrix for the following sentence:

   "He ate his fruits and vegetables and was ready for dessert!"

   (a) Find the co-occurrence matrix with a context of ±2. You can assume that blank boxes represent 0 occurrences. (Hint: Remember that rows are target words, and columns are how many times that word appears in the surrounding context)

	he	ate	his	fruits	and	vegetables	was	ready	for	dessert
he										
ate										
his										
fruits										
and										
vegetables										
was										
ready										
for										
dessert										

(b) Find the pairwise cosine similarity between the row vectors of **fruits**, **vegetables**, and **dessert**.

(c) Of the three words chosen in part B, which two are most similar? Why?

2. The file 'Loading_Word_Vectors.ipynp' provides code to load a pre-trained continuous bag-of-words word2vec model. Passing in a word into the 'n_similar()' function will return its corresponding vector, if available.

   (a) Try passing "apple" into the 'n_similar()' function. What are the inputs and outputs of the word2vec algorithm?

   (b) Pass "apple", "fruit", and "ocean" into the algorithm. Which vectors have a greater cosine similarity: (apple & fruit) or (apple & ocean)?

   (c) Take the vector of "king", subtract the vector of "man", and add the vector of "woman". Use the 'n_similar()' function to find what word is closest to the resulting vector. What is this word?

   (d) Using the word2vec algorithm, write code to help you solve the analogy: China is to Beijing as Spain is to _____.

   (e) Finally, write down your own analogy that can be solved by word2vec.

3. Provided in 'Loading_Word_Vectors.ipynp' are two different truncated SVD classifiers for detecting text message spam. They use the classic bag-of-words method and the word2vec Skip-gram embeddings. To input the word embeddings into the model, we take the average of all the vectors in the text.

   (a) Run the **truncated SVD classifier** implemented with **bag-of-words**. What is its error rate?

   (b) Run the **truncated SVD classifier** implemented with **Skip-gram word2vec**. What is its error rate?

   (c) Which model would you use? What are the tradeoffs of using word2vec versus bag-of-words and vice versa? Note that the provided dataset is relatively small compared to ones used in-practice.

References

[1] 3Top. (2017, November 3). word2vec-api. Retrieved May 1, 2020, from https://github.com/3Top/word2vec-api#where-to-get-a-pretrained-models

[2] Barazza, L. (2017, February 18). How does Word2Vec's Skip-Gram work? Retrieved May 1, 2020, from https://becominghuman.ai/how-does-word2vecs-skip-gram-work-f92e0525def4

[3] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching word vectors with subword information. CoRR, abs/1607.04606. Retrieved from http://arxiv.org/abs/1607.04606

[4] Doshi, S. (2019, March 16). Skip-Gram: NLP context words prediction algorithm. Retrieved May 1, 2020, from https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c

[5] Firth, J.R. (1957). "A synopsis of linguistic theory 1930-1955". Studies in Linguistic Analysis: 1–32. Reprinted in F.R. Palmer, ed. (1968). Selected Papers of J.R. Firth 1952-1959. London: Longman.

[6] Jurafsky, D. (2015, July 10). Vector semantics [Lecture transcript]. Retrieved May 1, 2020, from https://web.stanford.edu/ jurafsky/li15/lec3.vector.pdf

[7] Karani, D. (2018, September 1). Introduction to word embedding and Word2Vec. Retrieved May 1, 2020, from https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

[8] Khaitan, P. (2016, May 18). Chat smarter with Allo. Retrieved May 1, 2020, from https://ai.googleblog.com/2016/05/chat-smarter-with-allo.html

[9] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv. Retrieved from https://arxiv.org/pdf/1301.3781.pdf

[10] Patil, T. (2017, October 11). What does capital pi mean in math? Retrieved May 1, 2020, from https://www.quora.com/What-does-capital-pi-mean-in-math

[11] Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). Retrieved from https://nlp.stanford.edu/pubs/glove.pdf

[12] Rong, X. (2014). word2vec parameter learning explained. CoRR, abs/1411.2738. Retrieved from https://arxiv.org/pdf/1411.2738.pdf

[13] Ruder, S. (2016, October 13). An overview of word embeddings and their connection to distributional semantic models. Retrieved May 1, 2020, from https://blog.aylien.com/overview-word-embeddings-history-word2vec-cbow-glove

[14] Sarwan, N. S. (2017, June 4). An intuitive understanding of word embeddings: From count vectors to Word2Vec. Retrieved May 1, 2020, from https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec

[15] Pragmatic Cosine Similarity. (n.d.). Retrieved from https://engineering.aweber.com/cosine-similarity/

Appendix

5 Warm-up Solutions

1. According to the Distributional Hypothesis, which of the words is most similar to the word "**birds**" given the context: "I heard many **birds** in the forest."

   (a) **Fish** with the context "I saw lots of **fish** in the river."

   (b) **Bees** with the context "I heard many **bees** in the forest."

   (c) **Air** with the context "I felt the **air** on the palm of my hand."

   (d) **Rico** with the context "I caught the pass from Uncle **Rico**."

   Explanation: Since answer b has the most similarity in context words, the words "birds" and "bees" would share the most similar word embeddings.

2. Fill in the blank: The (**continuous bag-of-words**/**skip-gram**) algorithm is faster to train, while the (**continuous bag-of-words**/**skip-gram**) algorithm tends to represent rarer words better.

3. Compute the value of the Softmax function given the following parameters:

$$U = \begin{bmatrix} 2 & 3 & 4 \\ 4 & 3 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \qquad V = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^T$$

   Answer: $U^T V = \begin{bmatrix} 11 & 9 & 7 \end{bmatrix}^T$ $\quad$ $\text{Softmax}(U^T V) = \begin{bmatrix} .8668 & .1173 & .0159 \end{bmatrix}^T$

4. Would you expect word2vec to create a meaningful representation of words with multiple meanings such as "light"?

   Explanation: Based solely on the spellings, word embeddings may not be useful for words with multiple meanings, since the contexts of each meaning will pull the vector in different directions. One way to combat this issue would be to create different words for each meaning, such as "light(photon)" and "light(weight)".

6 Main Activity

1. We will now find the co-occurrence matrix for the following sentence:

   "He ate his fruits and vegetables and was ready for dessert!"

   (a) Find the co-occurrence matrix with a context of ± 2. You can assume that blank boxes represent 0 occurrences. (Hint: Remember that rows are target words, and columns are how many times that word appears in the surrounding context)

	he	ate	his	fruits	and	vegetables	was	ready	for	dessert
he		1	1							
ate	1		1	1						
his	1	1		1	1					
fruits		1	1		1	1				
and			1	1	2	2	1	1		
vegetables				1	2		1			
was					1	1		1	1	
ready					1		1		1	1
for							1	1		1
dessert								1	1	

(b) Find the pairwise cosine similarity between the row vectors of **fruits**, **vegetables**, and **dessert**.
similarity(fruits, vegetables) = .4082
similarity(fruits, dessert) = 0
similarity(vegetables, dessert) = 0

(c) Of the three words chosen in part B, which two are most similar? Why?

2. The file 'Loading_Word_Vectors.ipynp' provides code to load a pre-trained continuous bag-of-words word2vec model. Passing in a word into the 'n_similar()' function will return its corresponding vector, if available.

(a) Try passing "apple" into the 'n_similar()' function. What are the inputs and outputs of the word2vec algorithm?
Input of the word "apple" results in outputs of words similar to "apple", like "apple", "apricot", "acorn", "blackberry", and "doritos".

(b) Pass "apple", "fruit", and "ocean" into the algorithm. Which vectors have a greater cosine similarity: (apple & fruit) or (apple & ocean)?
```
# code:
length = x['apple'].shape[0]
appleVecT = np.reshape(x['apple'],(1,length))
fruitVec = np.reshape(x['fruit'],(length,1))
oceanVec = np.reshape(x['ocean'],(length,1))
print(appleVecT @ fruitVec)
print(appleVecT @ oceanVec)
# apple is more similar to fruit than ocean, since it has a larger cosine similarity (.5428 vs .2532)
```

(c) Take the vector of "king", subtract the vector of "man", and add the vector of "woman". Use the 'n_similar()' function to find what word is closest to the resulting vector. What is this word?
Using the formula: x['king'] - x['man'] + x['woman'] Output from the small model is 'king', 'queen', 'monarch', 'princess', and 'norodom'. Disregarding the input of 'king', the best answer is 'queen'. Other answers show some understanding of words. 'Norodom' doesn't seem like a normal word. Look it up. What does this tell us about our training data? Is wikipedia a good source for every-day speech?

(d) Using the word2vec algorithm, write code to help you solve the analogy: China is to Beijing as Spain is to _____.
Using the formula: x['beijing'] - x['china'] + x['spain'] Output from the small model is

'spain', 'beijing', 'lisbon', 'madrid', and 'seoul'. Disregarding the input words, we are left with the capitals of Portugal, Spain, and South Korea. Although this language model did not produce the expected answer, the idea of asking for a capital was well established. Further questions might be asked if this was a correct formula or if there are relationships in the training set (wikipedia), which aren't fully incorporated into our formula.

(e) Finally, write down your own analogy that can be solved by word2vec.
Answers may vary

3. Provided in 'Loading_Word_Vectors.ipynp' are two different truncated SVD classifiers for detecting text message spam. They use the classic bag-of-words method and the word2vec Skip-gram embeddings. To input the word embeddings into the model, we take the average of all the vectors in the text.

   (a) Run the **truncated SVD classifier** implemented with **bag-of-words**. What is its error rate?
   Error rate: 15.495

   (b) Run the **truncated SVD classifier** implemented with **Skip-gram word2vec**. What is its error rate? Note that the provided dataset is relatively small compared to ones used in-practice.
   Error rate: 23.816

   (c) Which model would you use? What are the tradeoffs of using word2vec versus bag-of-words and vice versa? Note that the provided dataset is relatively small compared to ones used in-practice.
   As you can see, the relatively small word2vec dataset provided in this activity does not outperform the bag-of-words classifier. Getting more accurate word embeddings requires a lot more data and training time.
   If you want the most accuracy, you would probably use word2vec or another similar word embedding. However, this comes at the cost of increased complexity and increased time to obtain these word embeddings. The benefit of bag-of-words is that it's a simpler implementation.