Saarland University

Chair for Clinical Bioinformatics

Spoken Language Systems Group

FINAL REPORT

Word2Mat: A New Type of Word Representation

Project By

Ayan Majumdar* (2571656), Mossad Helali* (2571699)
Shahzain Mehboob (2571564), Ehtisham Ali (2567631)

Supervised By

Prof. Dietrich Klakow

Table of Contents

INTRODUCTION AND PROBLEM STATEMENT

This report describes the model, implementation and results for word2mat, a new type of word representation. The project was conducted over 3 months as the submission for the machine learning seminar in Saarland University for the Summer Semester 2018.

Word2mat is a novel idea for word representation. In natural language processing (NLP) tasks, words are represented as vectors, or more specific embeddings. Representing words as real-valued vectors helps in reducing the size of the representation (as opposed to one-hot vectors) and makes it more compact and dense. In addition, word embeddings have proven to capture word semantics far better than the naïve approach of one-hot.

The standard embedding approach in the literature is word2vec. It works by going over a text corpus and trying to predict the context words in a specified window size. This is achieved using a one-layer neural network whose objective is to learn the word embeddings. Figure 1 shows the skip-gram model, the architecture suggested by the word2vec authors in [1]. The model takes a one-hot vector as input and multiplies it by the projection matrix to get the corresponding word embedding. Then, the embedding is fed to the noise contrastive estimation (NCE) layer, which predict context words using the embedding. Finally, the embedding is updated using negative sampling.

The word2vec model had much success in the past years and it was shown to be among the best word representations for modern NLP tasks. However, it has its shortcomings. Consider the task of having one embedding for a complete sentence. The current word2vec models in the literature simply add the embeddings of the individual words to get the sentence embedding. This embedding can then be used in the NLP task in hand. However, using this approach results in the same embedding even if the word order (and therefore, the semantics) of the sentence changes. Clearly this approach is not context-sensitive. Therefore, we try in this project to attack the problem of having word-order-sensitive embeddings.

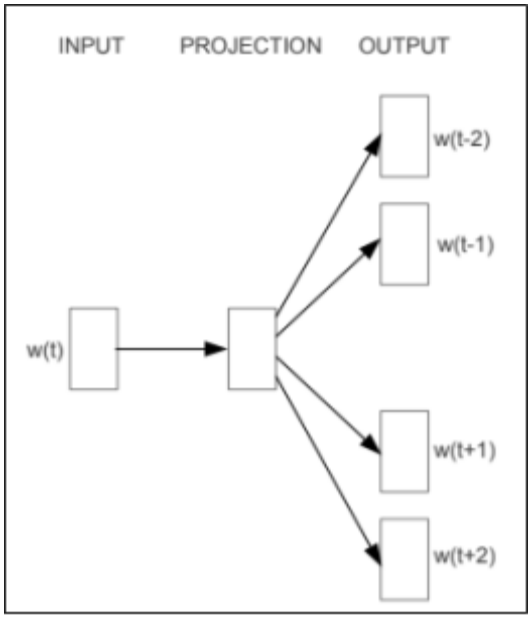


*Figure SEQ Figure * ARABIC 1: The Skip Gram Model*

THE WORD2MAT MODEL

In this project, we propose the idea of word2mat which aims to produce embeddings that are sensitive to word order of the sentence. The word2mat model differs from word2vec in the following aspects:

1. word embeddings are matrices:
   Instead of being 1-D vectors, embeddings are implemented as 2-D square matrices. The embedding size can be the same or different to the original size for word2vec. This is a hyper parameter that is tuned in later steps.
2. matrix multiplication is used to combine embeddings:
   Instead of using vector addition to combing embeddings, matrix-matrix multiplication is used. This is because matrix-matrix multiplication is not commutative. Therefore, the combined embedding of a sentence is different from the combined embedding of the same sentence but with different word order.

By making these changes, the embeddings are inherently sensitive to word order since the representation is different for different word orders. Figure 2 shows the new skip gram model.

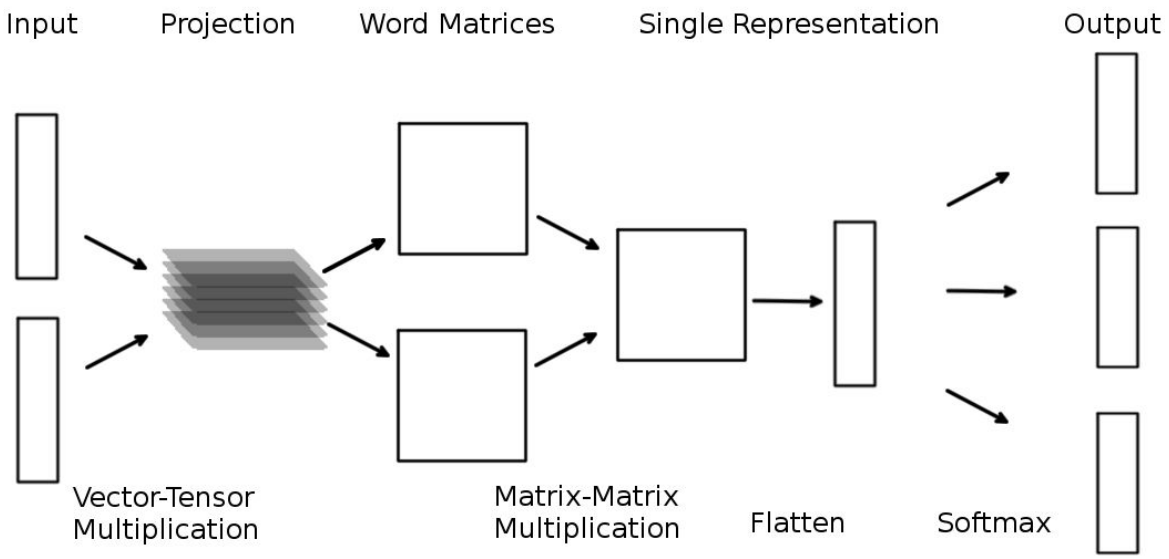


Figure 2: The modified skip gram model for word2mat

As an input, the model takes two one-hot vectors, gets their embeddings through the projection tensor, multiplies them using matrix-matrix multiplication and finally predicts the context using the NCE layer. Clearly, if the two words are flipped in order, we end up with different intermediate representation and consequently, different embedding.

It is to be noted that this model operates on bigrams (two words at a time). Even though this is restrictive, we show that it achieves high quality results. A future direction of this work is to operate on n-grams and observe the behavior of the model.

EXPERIMENTAL RESULTS

In this section, we provide a set of plots and results obtained by carrying out various experiments on our word2mat model. The metrics measured are:

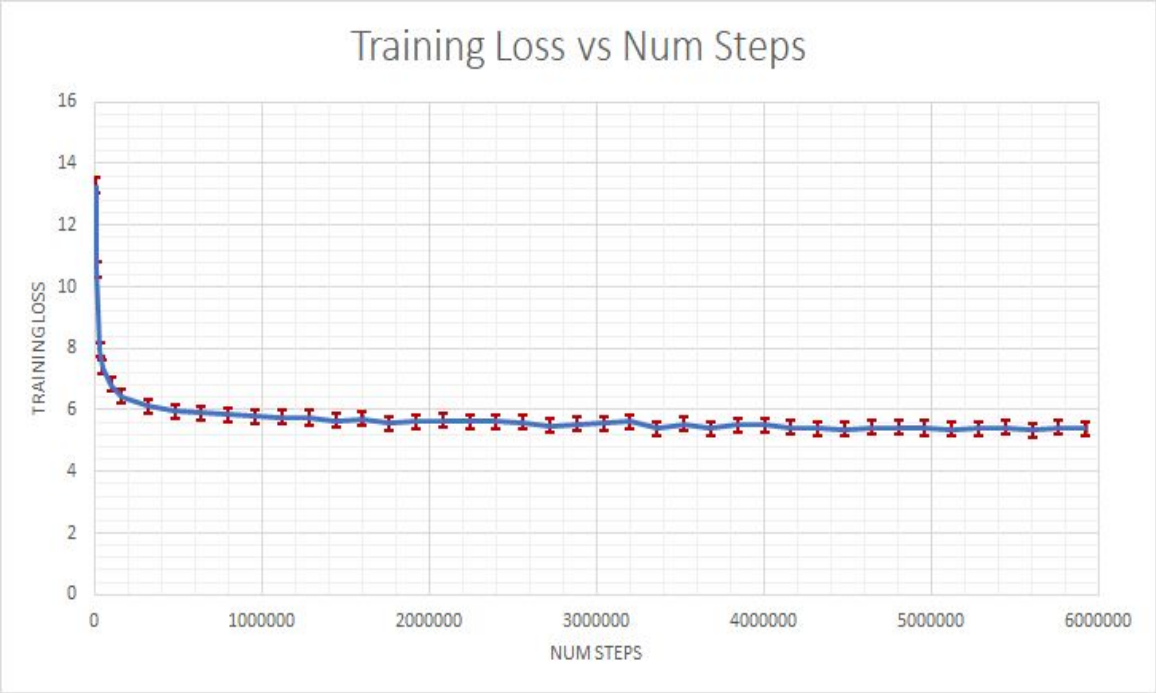
a. Training Loss: estimated as Noise Contrastive Estimation (NCE) loss.
b. Validation prediction accuracy and reversed prediction accuracy: prediction accuracy is estimated by taking a pair of words (bigram) from the validation dataset and predicting words from the context. This is done in the normal word order as they appeared in the sentence as well as reversing (flipping) the bigram to test the sensitivity to context.
c. The effect of varying embedding size on the prediction accuracy.
d. The effect of varying the number of negative samples on the prediction accuracy.
e. The prediction accuracy of word2vec against word2mat.
f. A comparison between the learned embeddings of word2vec against that of word2mat.

For our experiments, the hyperparameter values that we selected as a baseline are stated below**.** Throughout the report, we refer to this set of values as the **baseline system**:

- Number of Steps: 6000000 (37 epochs)
- Learning Rate: 0.06 (with decaying)
- Batch Size: 400
- Embedding Size: 400
- Skip Window: 2
- Number of Skips: 4
- Number of Negative Samples: 300

For experiments a, b and c, we use these set of values. For experiments d and e, we fix all hyperparameters except for embedding size in d and number of negative samples in e.
Although we try to measure the prediction accuracy of context words using a pair of words, it is important to note that this is not a proper text prediction task from NLP perspective. This is because there is no use of a proper sequential model (e.g. an RNN system). Instead, we take a pair of words and try to predict a single word that could occur in its context at any point of the text. A future extension of this project would be to use the learned embeddings in a proper NLP sequencing task. However, this simple prediction task is sufficient to show the difference in how matrix embeddings and vector embeddings work. In Addition, the prediction accuracy gives a numeric metric to compare between different settings, as it is not possible to measure how “good” a word embedding is, other than going through the closest words and seeing if they make sense or not. We also show the results of this experiment later in this report.
Nonetheless, our main goal is to get proper embeddings and, as we will show later, the embeddings learned do show relatedness amongst different words quite nicely in the matrix format.

A. Training Loss vs. Number of Steps



As one of the evaluation measures, we plot the variation of training loss with respect to the number of training steps (each step corresponds to one batch). We train the model for 6 million steps (roughly 37 epochs) on the training data. As shown in Figure 3, the training loss reduces from higher values and settles down at values around 5.4 to 5.6. The training loss is measured through Noise Contrastive Estimation (NCE) loss. This shows that the network can learn the embeddings through matrix multiplication by trying to predict words in the context in a skip-gram fashion. Also, the training loss keeps on going down, albeit more slowly, as we go through more and more steps.

For better interpretation, we ran the experiment multiple times, and took the mean of the observed output loss values. The vertical bars show the standard deviation. Note the values are recorded after each epoch.

B. Validation Accuracy vs. Number of Steps

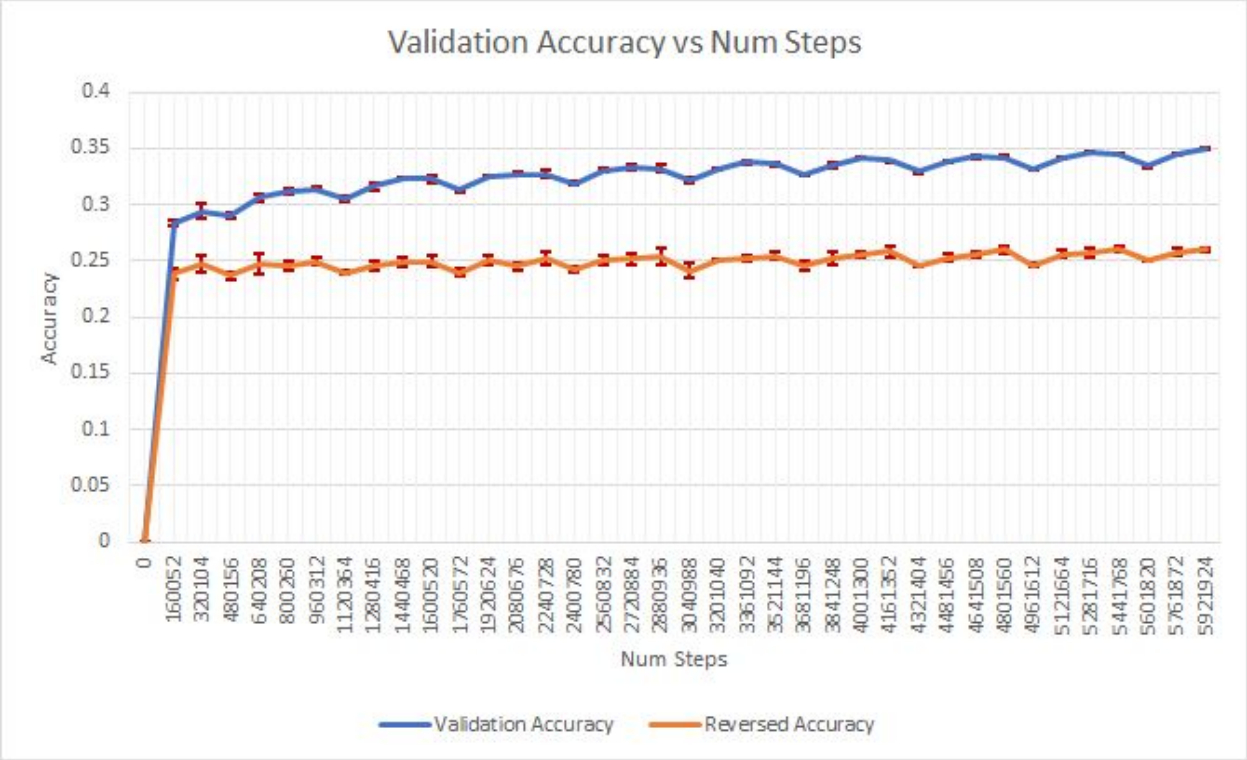


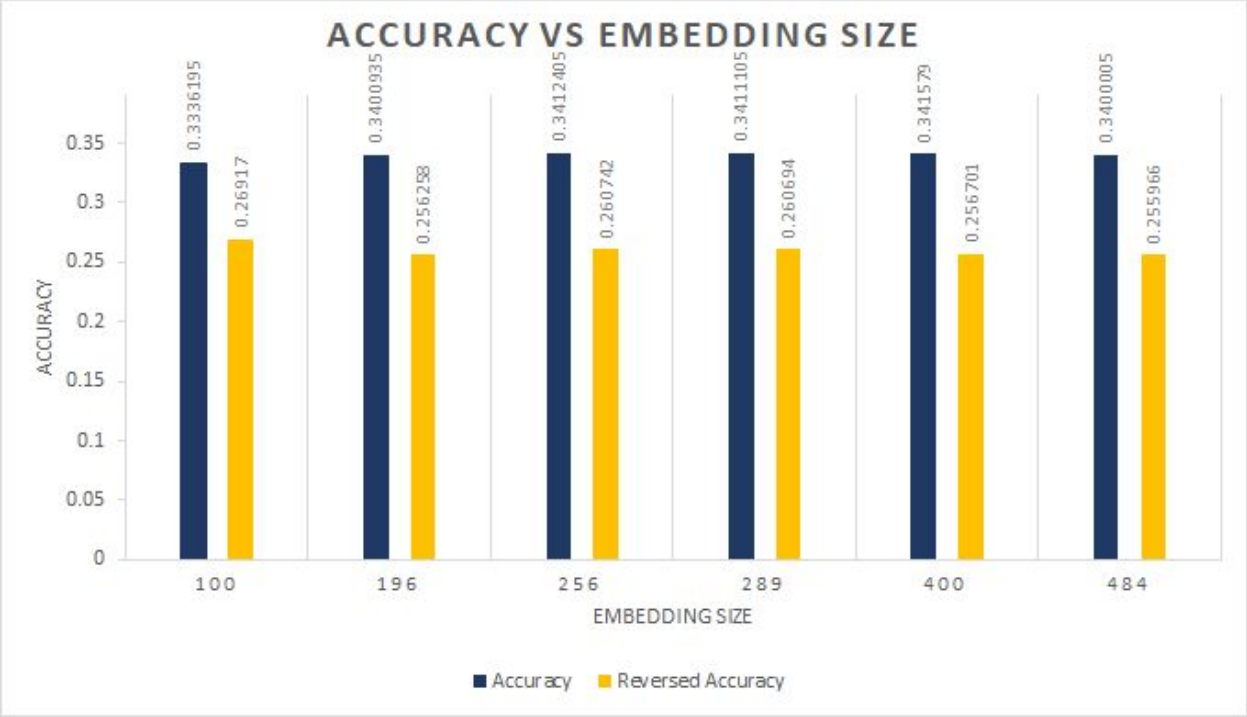
Figure 4: Validation Accuracy vs. Number of Steps

As part of the analysis, we observe the prediction accuracy on the validation set at each epoch. We show the difference between two settings: one in which the order of the two words in the bigram is the same as they occurred in the sentence, and the other is the same bigram with the two words flipped (reversed). The task is to predict a word that can occur in the context around the bigram. We define the context the words appeared in the skip window, i.e. 2 words left and right of the bigram (4 words in total). As with the previous experiment, we run it for multiple times and record the mean and standard deviation.

As seen in Figure 4, the accuracy increases with the number of epochs, and the average value we get eventually is around 35%. When we reverse the order of the words and measure accuracy, we get a value of around 25%. As mentioned above, this prediction task is not entirely a proper NLP task to evaluate a metric on, but we still clearly see a difference when we reverse the order of words, even with such a shallow and simple prediction system. The accuracy clearly drops by 10% when we reverse the word ordering, showing that the matrix embeddings are indeed able to show some contextual structure, as opposed to vectors. It is to be noted that in the case of vectors, we get the same value when we flip the bigram, since vector addition is carried out, which is commutative.

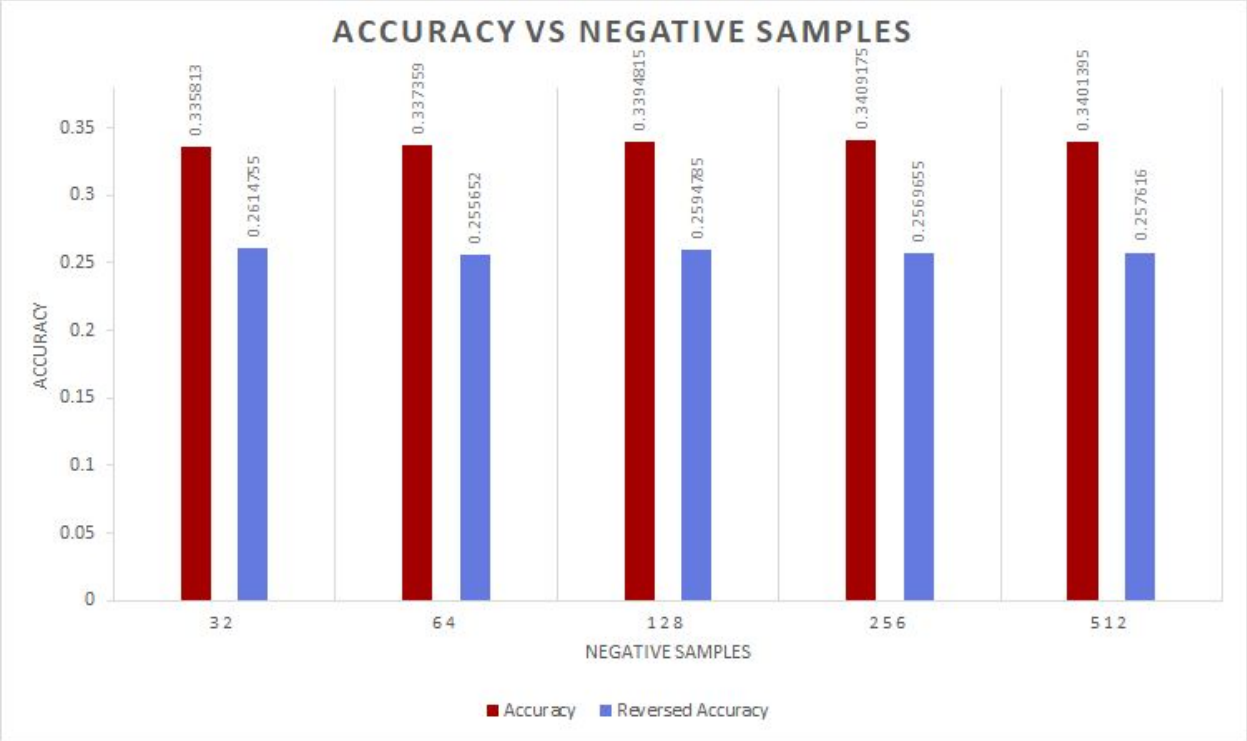
C. Validation Accuracy vs Embedding Size

In the next experiment, we try to find out the effect of the word embedding size on the prediction task. As our embeddings are square matrices, we choose embedding size to be a perfect square. We vary the size as shown in Figure 5. As we see the in-order accuracy varies slightly, and so does the reverse-ordered accuracy. For the embedding of size around 400, we saw an acceptable increase in the in-order accuracy for prediction, and an acceptable reduction in the reverse-order accuracy. Of course, the embedding size should be chosen according to the NLP task in hand. The embedding size may also be dependent on the amount of training data we train on.



D. Validation Accuracy vs Negative Samples

Next, the variation of the prediction accuracy with the number of negative samples is observed. Negative are the samples that NCE loss would pick for the loss computation during training. NCE loss is used to speed up the learning process and give better embeddings. As we see in Figure 6, increasing the number of negative samples increases the prediction accuracy slightly for both in-order and reversed-order bigrams. The number of negative samples in the baseline system is 300, because increasing it to 512 incurs a compactional cost without a noticeable increase in prediction accuracy.



E. Accuracy Comparison Between Different Models

Here we compare the prediction accuracy of the word2mat model against the word2vec model.

1. In the case of word2mat, we have two parts:
    a. Predict the context words based on a bigram from the validation dataset.
    b. Predict the context words based on the flipped bigram to demonstrate context sensitivity.
2. In the case of word2vec, we calculate the accuracy as follows:
    a. Obtain word embeddings using the original word2vec model.
    b. Use an intermediate model to learn the NCE layer weights and biases. This model uses vector addition of the embeddings of the bigrams to predict context words. The embeddings are fixed and only the NCE layer is trained. The model is trained using the word2mat training data.
    c. Use this learned network with the obtained embeddings to measure the prediction. The model is evaluated on the validation set of word2mat.

The process of evaluating word2vec is done to have a fair comparison, since without performing these steps, the accuracy of the word2vec model is much worse than that of word2mat.

It is to be noted that in the case of word2vec, we vector addition to predict the context, which is a commutative operation. Therefore, reversing the bigram order has no effect on the prediction of word2vec. However, this should not be the case. For example, “Wall Street” has a different meaning and context from “street wall”. In the case of matrices, flipping the order should give a lower accuracy, and that is exactly what we see, as shown in Table 1.

word2vec Prediction Accuracy	**word2vec (Reversed) Prediction Accuracy**	**word2mat Prediction Accuracy**	**word2mat (Reversed) Prediction Accuracy**
42.5%	42.5%	35%	26%

Table 1: Prediction Accuracy of word2vec vs. word2mat vs word2mat (reversed)

As seen, word2vec gives better accuracy than word2mat. This could be because of multiple reasons, such as the need for more training epochs, more training data or the need for a proper NLP prediction task.

However, as mentioned above, our main goal is to show that matrices indeed can help represent the contextual structure better than vectors, and this is clearly shown from the values. Word2mat predicts the context of in-order bigrams with 35% accuracy. For the flipped bigrams, the accuracy drops to around 26%. This shows that word2mat is indeed sensitive to word order in the sentence.

F. Word Embeddings

In this experiment, we observe the word embeddings obtained from the two trained models. For both the word2vec and word2mat, we use the same value of embedding size and skip window.

The embeddings are compared using a similarity measure, and for each embedding, we get the top-k closest ones. Ideally, if the embeddings have been learned properly, with a proper similarity measure, we should be able to see semantically similar words appear close to one another.

Similarity Measure:

The similarity measure used for word2vec is cosine similarity. For word2mat, we tried two different measures: First, we experimented with Frobenius norm of matrices, but it didn't work well in embedding comparison. Next, we tried a similar idea to the cosine similarity, where we normalize the embeddings and do the Frobenius inner product between the two matrices.

For this task, we choose words with different semantic meanings, like a number, a pronoun, a color, a country name, a sport and a personality name.

For word2vec, the nearest embeddings to the test words are as follows:

Nearest to **six**: seven, eight, five, four, three, nine, zero, two,
Nearest to **he**: she, it, they, there, mjw, who, microsite, joram,
Nearest to **computer**: computers, pc, video, mjw, application, television, digital,
Nearest to **france**: italy, spain, germany, russia, europe, ireland, french, britain,
Nearest to **company**: companies, corporation, industry, club, government, altenberg,
Nearest to **book**: books, novel, story, chapter, work, stories, paper, mjw,
Nearest to **red**: blue, white, black, yellow, green, clo, mjt, mjw,
Nearest to **car**: cars, motorcycle, aircraft, horse, driver, diesel, hiddenstructure, mjw,
Nearest to **football**: baseball, hockey, basketball, team, league, sports, soccer, nfl,
Nearest to **winter**: summer, spring, cold, temperatures, season, humidity, occur
Nearest to **apple**: ibm, microsoft, amiga, macintosh, windows, early, industry

For word2mat, the nearest embeddings are as follows:

Nearest to **five**: four, three, six, seven, eight, two, one, zero,
Nearest to **he**: she, it, they, but, which, who, while, however,
Nearest to **computer**: software, business, computers, program, modern, video, systems, technology,
Nearest to **france**: italy, germany, spain, russia, canada, england, europe, australia,
Nearest to **company**: industry, corporation, business, team, then, system, UNK,
Nearest to **oxygen**: water, carbon, hydrogen, energy, gas, heat, chemical, potential,
Nearest to **tree**: trees, family, meaning, UNK, cycle, near, root, common,
Nearest to **man**: person, men, young, woman, and, society, UNK, god,
Nearest to **book**: books, novel, biography, version, work, works, stories,
Nearest to **car**: company, aircraft, sound, program, single, engine, material, built,
Nearest to **football**: baseball, basketball, hockey, professional, league, sports, team
Nearest to **orange**: white, green, dark, blue, light, little, yellow, black,
Nearest to **winter**: summer, spring, season, during, cold, however, near, while,
Nearest to **apple**: microsoft, software, computer, pc, windows, ibm, standard,

As seen from the embeddings learned using a matrix model, word2mat is indeed able to learn the semantic differences and relatedness amongst different words. E.g. “football” shows related sports and sports related terms as it’s nearby neighbors. When we query for “clinton”, we get the names of former presidents of the United States as the nearby neighbors. Note that the term **UNK** is an **Out of Vocabulary word** that has been marked as UNK because its frequency is below a certain threshold. This is a typical NLP preprocessing technique

G. The Effect of Reversal in Matrix Embeddings

Lastly, we perform an experiment where, given a pair of words, we predict the top-k words that can come in the surrounding context. We compare this with the predictions when the bigram is flipped. As mentioned, although the prediction task is not as sound as a proper NLP task should be, it still can be used for a preliminary insight into the learned embeddings and the effect that word orderings can have on the matrix embeddings. For our experiment, we decided on some bigrams of choice: (‘wall’, ‘street’), (‘los’, ‘angeles’), (‘university’, ‘of’). The predictions are as follows; we bold the words in the top-k outputs that are distinguishing from common English words (of, to, a, an, the, etc.).

a. predictions of wall - street : ['to', 'of', 'a', **'journal'**]
   predictions of street - wall : ['from', 'to', 'at', 'and']
b. predictions of los - angeles : [**'dodgers'**, 'a', 'to', 'UNK']
   predictions of angeles - los : ['at', 'of', 'one', 'and']
c. predictions of san - francisco : ['s', **'bay'**, 'to', **'california'**]
   predictions of francisco - san : ['and', 'the', 'one', 'jose']

By having a closer look at the predictions, we see the following:

a. For this bigram, when we consider the correct order “*wall street*”, we see a familiar term, “journal” in the top-4, suggesting the familiar trigram “*Wall Street Journal*”. But if we reverse the order, we see that all the predictions are mostly safe words, i.e. words that are common in English, we do not get “journal”.
b. For this bigram, again we see a distinguishing word for “*los angeles”*, i.e. “dodgers a distinct word for the *Los Angeles Dodgers* baseball team. But, in the flipped ordering, we have, as in bigram a, all the safe words.
c. For this bigram, we get predicted context words like “bay” and “california”, both can come in text as (“*San Francisco Bay*”, or “*San Francisco California*”). When we reverse the order, other than the common words, we do not get these words.

These results show the effect of word orders matter in matrix embeddings. This will never be the case in vector embeddings as we do vector additions there, and even if we reverse the word orders, we will still get the same outputs. Naturally, the predictions can differ from one bigram to the other, but at least for the task in hand, we show the context sensitivity of word2mat.

CONCLUSION

In conclusion, in this project we implemented a new type of word representation, namely, word2mat. It is a model based on word2vec but uses matrix embeddings and matrix multiplication in the learning process. Qualitative and quantitative results show that the word2mat model is indeed able to learn the contextual and the semantic relationships of words and represent them properly in a matrix format. For vector embeddings, we would never see a representation change as vectors are combined by vector addition. Future directions of this project include comparing word2mat and word2vec in a real-world NLP task using a proper sequencing architecture. In addition, the model behavior is to be observed when modified to operate on n-grams instead of bigrams.

TASK DISTRIBUTION

As a 4-member team, it wasn't very easy to distribute the tasks evenly and keep all members busy at all times. At some points of the project, some members were busier with course load than others. At other points, there were not enough tasks to be distributed. Overall however, we believe that we achieved this to some extent. Table 2 shows the task distribution.

Tasks	Members			
	Ayan	Ehtisham	Mossad	Shahzain
Understand word2vec model and existing code	X	X	X	X
Implement proof-of-concept (intermediate) model	X		X	
Optimize hyper parameters for proof of concept model and word2vec		X		X
Implement word2mat model	X		X	
Modify similarity measure and learning rate decay			X	
Implement side experiments to analyze common bigrams		X		
Implement context prediction task	X		X	
Implement validation tests	X		X	
Optimize word2mat hyper parameters, record experimental results				X
Conduct final analysis of the results	X			X

Table 2: Task Distribution

REFERENCES

1. “Efficient Estimation of Word Representations in Vector Space” – Mikolov et.al. 2013
2. “Distributed Representations of Words and Phrases and their Compositionality” - Mikolov et.al. 2013
3. https://www.tensorflow.org/tutorials/representation/word2vec
4. http://mattmahoney.net/dc/textdata.html