Word2Mat: Square matrix word embeddings to better represent grammar

Mikolaj Konrad Bochenski, Gilles Van De Vyver, Johan Luhr, and Max Junestrand

Abstract—**In this work, a new way of representing word embeddings is evaluated. Historically, word meaning has been represented using vectors, which provide no natural way of representing the meaning of a sentence given embeddings of the words in it. This is because word order isn't fully exploited in the vector arithmetic that goes into these models. In order to better capture the meaning of sentences, our approach represents the words as matrices, which provides an intuitive way of capturing word order through matrix multiplication. The embeddings are trained on the WikiAuto dataset using stochastic gradient descent on an auxiliary task of classifying sentences in order to encode sentence meaning. The results of this, when compared with common NLP models, such as Word2Vec and BERT, indicate that our model better encodes the meaning of sentences in a grammatical sense, yet somewhat fails to encode the semantic meaning of sentences.**

Index Terms—**Matrix word embeddings, Meaning of Sentences, NLP models, Deep learning, Stochastic gradient descent.**

I. Introduction

Word embedding models such as GloVe [1] or Word2Vec [2] only use neighbouring words to find the context each word appears in. For example, in the sentence "*Mary had a little lamb*", "*Mary had a little*" could be the context for "*lamb*". These models, however, do not make proper use of word order in that context. According to them, "*Little Mary had a*" is equivalent with "*Mary had a little*", which simply is not true.

The two reasons why this is harmful are that these models:

- Treat all context words equally
- Use a fixed-size context without regard for sentence boundaries

This project introduces a different kind of model that represents word embeddings as square matrices. This is different from the widely used approach of representing word embeddings as vectors. [1] [2] The model presented in this work operates on a very low level, meaning that it does not involve any recurrent neural networks, transformers, or the like. Due to the word embeddings being matrices, the model makes use of matrix multiplication to compound words to represent sentences. Because matrix multiplication is sensitive to order, the model factors in word order. The model introduced in this work is also able to process sequences of arbitrary length, meaning that it can be trained on entire sentences, and thus learn to understand the grammatical structures involved. The purpose of this model is therefore to train word embedding matrices which carry meaning.

Section two explores related work. Section three describes the data used. Section four provides information about the theory behind the model. Section fives gives an overview of the methods used. Section six reports the experiments. Section seven concludes the work.

II. Related Work

As of today, several ways of quantifying semantics (how words contribute to the meaning of a sentence) and syntax (how words are structured in a sentence) have been investigated. Most of the well-performing prior models utilize, in one way or another, pre-trained vector word embeddings [1] [2] [3]. A model is well-performing if it can account for semantics and syntax.

Let us now explain how state of the art NLP models work in terms of representing semantics and syntax.

Word2Vec and GloVe represent word meanings as vectors. They learn these vectors as part of an auxiliary task, which involves predicting the neighbors of each word. Because they only use neighborhood information, and ignore word order within the neighborhood, they are incapable of properly capturing the meaning of full sentences. [2] [1]

More recent models work differently in order to attribute each word a more fine-grained semantic and syntactic meaning. One example is presented in the work by [4] where a deep bidirectional model which uses the entire input sequence, i.e. a sentence, is used in order to quantify each word's meaning.

Another approach is presented in [5]. Here, the authors utilized several vectors to represent the same word, depending on which context the word is found in.

Another way to capture semantics and syntax was presented in [6]. The authors used the idea of sub-words to enrich the meaning of each individual word, depending on the context. The arguably best NLP model for common benchmark problems is Bidirectional Encoder Representations from Transformers (BERT), developed by Google [3]. Applications such as Google's Search, Docs, and Gmail all utilize BERT for text prediction. The reason behind BERT's success is the bidirectional encoding - meaning that it encodes the entire phrase at once. This differs from previous work that first encodes the phrase left to right, and then concatenates that with the encoding of right to left. [4].

The suggested approach in this project differs from all previous work in that it explores a new way of representing word meanings. In terms of complexity, our solution compares to Word2Vec, but it retains the order-sensitivity of more complicated methods. This approach provides us with a natural way of representing sentence meaning.

III. Data

The dataset **WikiAuto** created by [7] was used for the experiments. It is a dataset of sentences from English Wikipedia and Simple English Wikipedia. It was chosen because it was a straight-forward way to get hold of full sentences, which our model had to train on in order to learn sentence meaning. See appendix A for a more extensive explanation regarding the choice of dataset. We could of course have created our own dataset by extracting sentences from different resources, but that would likely have proven a tedious and unnecessary process.

The data contains normal sentences and corresponding simple sentences, and for most of the experiments carried out, the model was trained on the normal sentences. This was done because of the higher complexity of those sentences, in hopes that the model would learn more complex grammar. This is an example sentence from the dataset: *Use of English is growing country-by-country internally and for international communication.*

TABLE I
Examples of normal and simple sentences from WikiAuto dataset

Normal Sentence	Simple Sentence
A rocket (from Italian: rocchetto, lit. 'bobbin/spool') is a projectile that spacecraft, aircraft or other vehicles use to obtain thrust from a rocket engine.	A rocket may be a missile, spacecraft, aircraft or other vehicle which is pushed by a rocket engine.
It is the only domesticated species in the family Felidae and is often referred to as the domestic cat to distinguish it from the wild members of the family.	Domestic cats are often called 'house cats' when kept as indoor pets.
Death is the permanent, irreversible cessation of all biological functions that sustain a living organism.	Death is the end of a life in an organism.

The dataset consists of 373801 sentences. Some of these may be of no use for us for one of the following reasons:

- Containing rare words, i.e. words not present in the pre-trained Word2Vec embeddings we compared our model with
- Being a duplicate of another sentence in the dataset
- Being a tokenizer artifact, e.g. containing only whitespaces

After removing these sentences, there are 25589 sentences left in the dataset. They contain 35259 unique words, and 671414 words in total.

It is worth noting that we did many of our initial experiments without de-duplication (we didn't realise that sentences were duplicated). Because we thought we had many more sentences, we assumed that it's impossible to overfit given so many examples. We therefore calculated some metrics on the training set. However, the final AUC score and other metrics reported in the evaluation section are all calculated using a held-out test set. We used 10% of the data for the test set, and the rest for training.

We also needed negative examples, i.e. sentences with grammatical and semantic mistakes in them. These negative examples are generated by replacing a single word in the sentence by a randomly picked word. This means that the probability that a word will be chosen is proportional to its frequency in the corpus. This way of generating incorrect sentences worked great in practice, as we did not observe any correct sentences in the data, but could in theory of course still accidentally produce a correct sentence. Examples of the negative data are shown in table II

Table of incorrect sentences

TABLE II
Examples of correct and generated incorrect sentences

Correct Sentence	Incorrect Sentence
He ***was*** the center of the camp's activities that summer, the core of the vortex.	He ***challenger*** the center of the camp's activities that summer, the core of the vortex.
Puddings are thickened with starches such as corn starch or tapioca.	Puddings are thickened with starches such as corn starch or tapioca***$***
Saetas evoke strong emotion and are sung ***most*** often during public processions.	Saetas evoke strong emotion and are sung ***impact*** often during public processions.
They have a son ***born*** in may 2002, who was diagnosed with autism in may 2005.	They have a son, in may 2002, who was diagnosed with autism in may 2005.

IV. Theory

The meaning of each word is represented as a square matrix. To produce a matrix representing the meaning of an entire phrase or sentence, these matrices can be multiplied together, as shown in Fig. 1. This is our most significant contribution: our method naturally extends to embeddings of pieces of text, taking order into account.

Matrix multiplication is non-commutative, meaning that "*Mary had a little*" will indeed have a different embedding than that of "*Little Mary had a*". What's more, because the same type of mathematical object is used to represent words and phrases, one could hope for the embedding of the phrase "*ice cream*" to be fairly similar to that of "*sorbet*".

In-between the matrix multiplications, there is room for an activation function. As detailed in the experiments section, we investigated using multiple activation functions, including none at all.

$$\underset{\text{Python}}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}} \text{X} \underset{\text{Is}}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}} \text{X}$$

$$\text{X} \underset{\text{Cool}}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}} = \underset{\text{Meaning of Sentence}}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}}$$

Fig. 1. Compound sentence meaning through matrix multiplication

V. Methods

The purpose of our model is to better encode meaning of words compared with what previous work has been able to do. How one should go about doing this, however, is difficult. Our approach in this study has been to train the model to perform sentence classification, i.e. to differentiate between correct and incorrect sentences. The reason why this approach helps answer the research question is because it provides a way to train our matrix embeddings to capture meaning. This task also allows gradient descent to be applied, thus training the matrices to encode more meaning. The task of classifying sentences is therefore an auxiliary problem, in order to facilitate the higher purpose of encoding more meaning in the word embeddings.

Furthermore, finding good ways to implement and train our model has in nature been an exploratory process, and consisted of three main steps:

1) Evaluate many different parameters in shorter experiments
2) Evaluate a selection of the best parameters from previous steps in longer experiments
3) Evaluate the best trained model and compare it with other methods

Let us now explain the method of classifying sentences in further detail.

A. Sentence classification

In order to train our model, we needed to turn the matrix representation of a sentence into a probability. We do this by introducing to auxiliary vectors: a start-of-sentence and end-of-sentence vector. To find the probability of a sentence being correct, we perform matrix multiplication followed by the sigmoid function, as illustrated in Fig. 2.

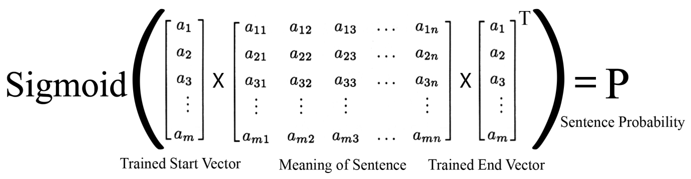
Sigmoid
$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_m \end{bmatrix}$ X $\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$ X $\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_m \end{bmatrix}^T$
= P
Sentence Probability
Trained Start Vector
Meaning of Sentence
Trained End Vector

Fig. 2. Calculate sentence probability

B. Training

The model learns to classify sentences by applying gradient descent. The training was done on the **WikiAuto** dataset. To create word encodings that capture meaning, we explored different methods of training the model. After which we evaluated the model on a number of NLP tasks in order to conclude if the matrices encoded any valuable information.

In order to achieve this, the following steps were completed to create a framework to train our model:

1) Read in the dataset
2) Clean the dataset
3) Generate incorrect sentences
4) Initialize the matrices for each word
5) Initialize sentence-start and sentence-end vectors
6) Train the matrix word embedding using gradient descent
7) Evaluate the network using Area Under the ROC Curve (AUC)

The training is done using stochastic gradient descent with a decaying learning rate. The loss function used was binary cross entropy. After some preliminary tests, we found that exploding gradients was a significant problem. Therefore, we used gradient clipping at a maximum absolute value of 1.0 in all of our experiments.

The library used to implement this was the Python-based deep learning framework PyTorch. This was chosen due to an author's previous experience with it, as well as the desire of the other authors to work with it. The framework comes with many perks such as the ability to compute gradients in a single function call, and the convenience of the tensor data structure. [8]

VI. Experiments

A. Activation functions & initialisation

We started by finding a suitable activation function and initialization. Our goal was for the distribution of values in the matrices to be similar after performing several matrix multiplications. We hoped this would stabilize training, because short sentences would have a similar distribution of values - and, hopefully, of gradients - as long sentences.

We investigated the following activation functions:

- No activation
- Signed square root: $sign(el) * \sqrt{|el|}$
- ReLU, see [9] for details

We investigated the following distributions from which to initially draw matrix elements:

- Normal distribution with $\sigma = 1/\sqrt{n}$
- Cosine distribution with $s = 1/\sqrt{n}$
- Signed square root of standard normal distribution
- He with $a = 0$ and $mode = fan\_in$, see [10] for details

where n refers to matrix size in the above equations.

It was found empirically that, when not using any activation function, every initialization results in a distribution with a peak around zero and heavy tails after a few matrix multiplications. This would be very harmful to gradient descent - as we would likely encounter exploding gradients.

To combat this, we used the signed square root activation function. The reason why we started investigating it in the first place is that taking the signed square root of every element makes small numbers bigger and big numbers smaller, towards one. When multiplying matrices with this activation function, the distribution of the elements converges to a stable distribution.

This stable distribution is the signed square root of the normal distribution. Initializing the matrices with this distribution did not, however, improve performance.

B. Training experiments

Our training-related experiments concerned:

- Matrix Size

- Number of Epochs
- Learning Rate
- Learning Rate Decay
- Initialization
- Activation function

The results from these experiments are shown in table VI. The best observed settings for the model were: matrix size = 16x16, epochs = 32, learning rate = 0.01, learning rate decay = 0.5 per epoch, initialization = cosine, and activation function = signed square root. With these settings, an AUC value of 0.9 was observed when classifying sentences as correct or incorrect. The matrix word embeddings learnt with these settings were thus used when comparing and evaluating our approach moving forward. The code for this can be found at [11].

C. Evaluation experiments

In order to understand what information the matrix word embeddings encoded, a number of experiments were carried out. The experiments investigated the following areas:

- Similarity of words
- Similarity of sentences
- Sentence probability
- Grammar in sentences
- Classification between correct and incorrect sentences

The code for this can be found at [12].

Using the matrix word embeddings achieved by training with the best settings, experiments concerning the similarity between phrases was investigated. The similarity score between two phrases ranges between -1 and 1, the closer a score is to 1, the more similar the phrases are. Firstly, experiments with individual word embeddings were conducted. Results of similarity scores between a few words are shown in table III. It is evident from the results that the matrix embeddings capture some of the meaning of the individual words, since words like ”Dog” and ”Cat” have a much higher similarity score than ”Dog” and ”Banana”.

TABLE III
EXPERIMENT WITH WORD SIMILARITY

Phrase 1	Phrase 2	Similarity Score
Dog	Dog	1
Dog	Cat	0.4054
Dog	Banana	-0.0445
Dog	America	-0.0414
America	Europe	0.4755

We also used Principal component analysis (PCA) to get a better overview of how the words were connected. To apply PCA, we first flattened the matrices, turning them into vectors. This made flaws in the results apparent. Fig. 3 shows a selection of a few words, and it is difficult to find any consistent pattern.

However, when comparing grammatically different words of classes such as nouns (”bird”, ”cat”, ”cow”), articles (”the”), copulas (”is”, ”are”) the PCA projection shows large differences between them, suggesting that the model has learnt at least some English grammar.

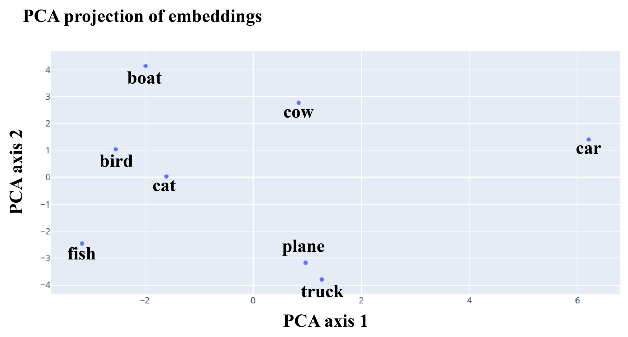

Fig. 3. PCA projection of noun word embeddings

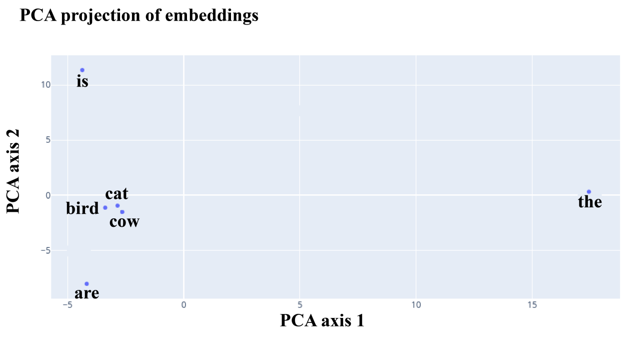

Fig. 4. PCA projection of words from different grammatical classes embeddings

We also calculated cosine similarity scores for entire sentences. Again, we flattened the matrices to turn them into vectors beforehand. Results from this are shown in table IV. It is not obvious that the model learnt the meaning of the sentences. Phrases such as ”I am called John” and ”My name is John” have very similar meaning, yet their similarity score is negative, indicating that the model does not consider them similar at all. However, the phrases ”I am called John” and ”I am a human”, two sentences that are grammatically alike, get a high similarity score, perhaps indicating that the model pays attention to that aspect. On the flip side, it is a bit underwhelming that ”My family is from Europe” and ”Love to banana chicken salad” get a similarity score of 0.3838.

TABLE IV
EXPERIMENT WITH SENTENCE SIMILARITY

Phrase 1	Phrase 2	Similarity Score
I am called John	My name is John	-0.4211
I am called John	I am a human	0.7379
My cat is black	My name is John	0.3194
My family is from Europe	Love to banana chicken salad	0.3838
My family is from Europe	My family is from China	0.8416

PCA projection of the sentence embeddings is shown in Fig. 5, and further illustrate the same point - the model has not learnt much about English semantics.

In order to test if the matrices capture grammar and meaning, we also evaluated our model on the auxiliary task of

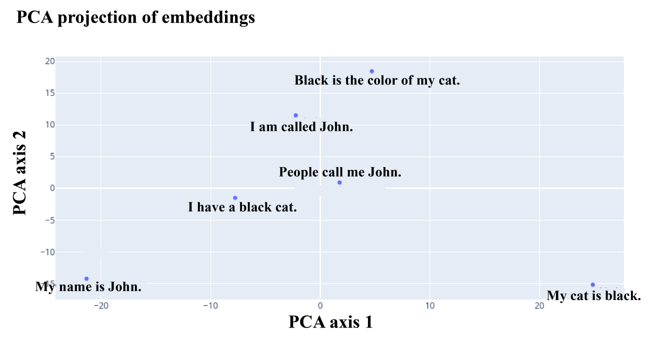

Fig. 5. PCA projection of sentences

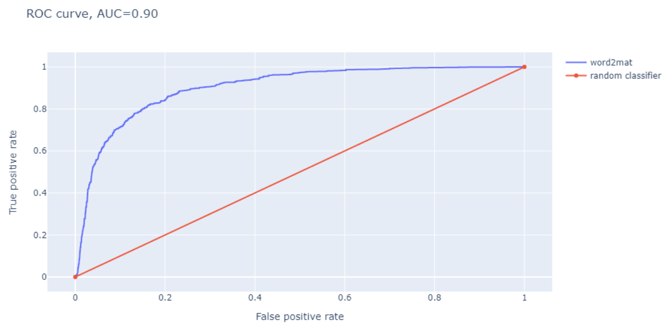

Fig. 6. Sentence classification using our proposed model, with the best settings

sentence classification and compared it with other models. Calculating the probability that a sentence is correct is done by applying the formula presented in the theory section in Fig. 2. Before applying the sigmoid function, logit values, as shown in table V are calculated. A positive value means that the sentence is correct, and a negative value indicates that the sentence is incorrect. The results from the experiment indicate for example that the model is capable of detecting plural and singular errors, as seen in the first and second row of the table. However, the model is not perfect, as the sentence ”The present coat of arms of Andalusia shows Hercules between two lions.” should have a positive score, yet does not. The reason for this misclassification most likely comes from the fact that ”present” is used as an adjective here, but usually it's a noun/verb, and removing ”present” does in fact give the sentence a positive score.

TABLE V
EXPERIMENT WITH SENTENCE MEANING VALUES

Sentence	Logit Value
The screenshots were subsequently taken down from both websites.	0.8042
The screenshots were subsequently taken down from both website.	-1.0518
The present coat of arms of Andalusia shows Hercules between two lions.	-0.9362
The coat of arms of Andalusia shows Hercules between two lions.	0.16112

This auxiliary task of sentence classification is the primary way to evaluate the matrix model. The results of training and classifying sentences are shown in Fig. 6. An AUC value of TODO is achieved. In the following two paragraphs, this model is compared with a classifier based on Word2Vec vectors and BERT embeddings.

First, the model is compared with a classifier based on Word2Vec. This model calculates the dot product of the vectors of consecutive words and turns it into a probability using the sigmoid function. It then takes the geometric mean of the probabilities of pairs of words to obtain a value for the whole sentence. We chose to take the geometric mean rather than multiplying the probabilities together to avoid longer sentences being deemed less likely. We're aware that this isn't the best way of using Word2Vec embeddings to find the probability of a sentence. When calculating the probability of a pair of words co-occurring, it would be better to use the context vector of one and the focus vector of the other. However, we settled on this approach because it only makes use of downloadable pre-trained word vectors, which is what would be used in a practical setting.

We're also aware that using a neural network on top of Word2Vec would lead to a better result. However, we wanted to make an apples-to-apples comparison here - we're not using a neural network on top of the matrices, either.

With all that said, the Word2Vec-based classifier yields a result of AUC = 0.51 on a the held-out test set, as shown in Fig. 7. This means that it is only 1 percentage point better than guessing. The code for this can be found at [2].

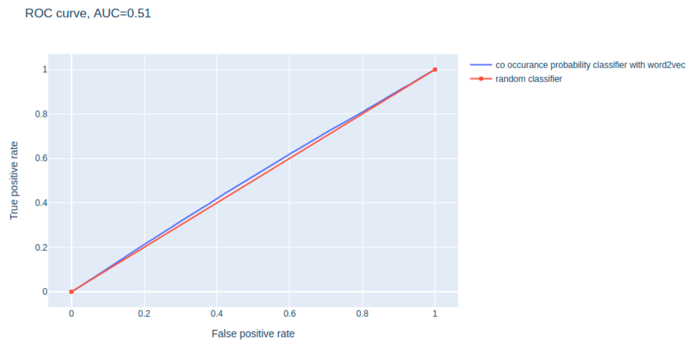

Fig. 7. Sentence classification using Word2vec, by calculating sentence probabilities using geometric average

We also trained and evaluated a classifier based on a pre-trained BERT model. This model produces one embedding vector per sentence. This means they take into account all words in the sentence. The final classification head for this experiment is a basic linear regression on these vectors. This got a result of 0.72 AUC on a held-out test set, as shown in Fig. 8. A high base line result, but still thirteen percent lower than what the matrix model achieved. It must be noted that this model is just a simple unoptimized baseline, i.e. BERT itself was not trained on our data. It is surely possible to get better results with a more elaborate classifier based on the BERT vectors. This could be the idea for a state-of-the-art solution for the task of classifying correct and incorrect sentences. The code for this can be found at [13].

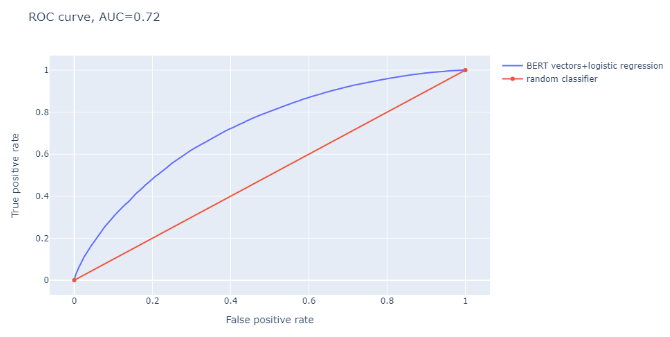

Fig. 8. Sentence classification using an unoptimized BERT classifier

Lastly, let us observe the models in the same plot to better highlight the differences, shown in Fig. 9. The code for this can be found at [14].

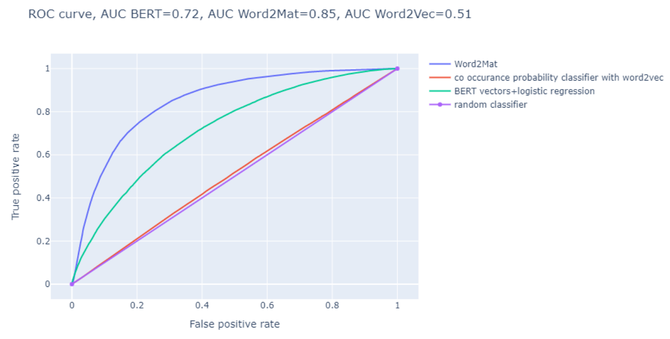

Fig. 9. Sentence classification all classifiers

VII. Conclusion

The matrix embeddings seem to primarily capture grammatical function as a consequence of how they are trained as compared with the semantic meaning of traditional word embeddings. The matrix model vaguely captures semantic meaning, but less than expected and definitely a lot less than Word2Vec or Glove. However these models are trained for longer and on a dataset that is bigger by several orders of magnitude.

Appendix A

An initial important consideration when choosing a dataset, was that the data had to be free from copyright. This was of importance because of our open source code. The size of the dataset was an important point of consideration. A dataset of less than 16 GB could be loaded into an average PC's working memory, thus facilitating experimentation. However, the dataset couldn't be too small, since that would limit conclusions the experiments too much.

It is not necessary to have an English dataset, however it is comfortable since the all the authors understand it. There is also more data available in English which is a clear advantage. English sentences are also grammatically easier to clean/prepare for experiments, since there are few special characters such as Swedish Å, Ä and Ö. It is also important that our model can be compared with existing ones, and English works for all these other models.

Initially we considered to use different books. We decided to not do this due to two problems. The first and most critical problem was that one book was too short and merging books was deemed problematic. The second and less important reason was copyright.

Acknowledgment

The authors would like to thank Josephine Sullivan and the TA team for the DD2424 course at KTH for their guidance.

References

[1] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[2] 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*, 2013, pp. 3111–3119.

[3] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[4] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *CoRR*, vol. abs/1802.05365, 2018. [Online]. Available: http://arxiv.org/abs/1802.05365

[5] A. Neelakantan, J. Shankar, A. Passos, and A. McCallum, "Efficient non-parametric estimation of multiple embeddings per word in vector space," *CoRR*, vol. abs/1504.06654, 2015. [Online]. Available: http://arxiv.org/abs/1504.06654

[6] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *CoRR*, vol. abs/1607.04606, 2016. [Online]. Available: http://arxiv.org/abs/1607.04606

[7] C. Jiang, M. Maddela, W. Lan, Y. Zhong, and W. Xu, "Neural CRF model for sentence alignment in text simplification," *CoRR*, vol. abs/2005.02324, 2020. [Online]. Available: https://arxiv.org/abs/2005.02324

[8] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[9] "Torch.nn.init," accessed: 2021-05-12. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html

[10] "Torch.nn.init," accessed: 2021-05-12. [Online]. Available: https://pytorch.org/docs/stable/nn.init.html#torch.nn.init.kaiming_uniform_

[11] J. L. Mikolaj Konrad Bochenski, Gilles Van De Vyver and M. Junestrand, "Training notebook." [Online]. Available: https://nbviewer.jupyter.org/github/malyvsen/word2mat/blob/a9ea1d177582a6776c644a7cee82a16ca26fd7a0/experiment.ipynb

[12] ——, "Qualitative evaluation notebook." [Online]. Available: https://nbviewer.jupyter.org/github/malyvsen/word2mat/blob/6e0dfc3b3396360e686378cc4bac9c56105b0899/evaluation.ipynb

[13] ——, "Bert evaluation notebook." [Online]. Available: https://nbviewer.jupyter.org/github/malyvsen/word2mat/blob/comparison_with_vectors/BERT_comparison.ipynb

[14] ——, "Comparison with bert and word2vec notebook." [Online]. Available: https://nbviewer.jupyter.org/github/malyvsen/word2mat/blob/comparison_with_vectors/Combined_plot.ipynb

TABLE VI
RESULTS FROM INITIAL EXPERIMENTS

Matrix Size	Initialization	Epochs	Learning rate	Learning rate decay	Activation	AUC	AUC
16	Cosine	32	1.00E-02	0.5	Signed square root	0.9	0.9
16	Cosine	4	1.00E-02	0.5	None	0.69	0.69
16	Cosine	4	1.00E-02	0.5	Signed square root	0.66	0.66
16	Cosine	32	1.00E-02	0.5	None	0.64	0.64
32	Cosine	4	1.00E-02	0.5	None	0.64	0.64
8	Cosine	4	1.00E-02	0.5	None	0.63	0.63
8	Cosine	4	1.00E-02	0.5	None	0.61	0.61
8	Cosine	1	1.00E-02	Irrelevant	None	0.61	0.61
8	Cosine	32	1.00E-02	0.9	None	0.6	0.6
8	Cosine	4	1.00E-02	0	None	0.6	0.6
16	He	4	1.00E-02	0.5	Signed square root	0.59	0.59
8	Cosine	4	1.00E-02	0.5	None	0.59	0.59
16	Signed Square	4	1.00E-03	0.5	Signed square root	0.59	0.59
16	Signed Square	4	1.00E-02	0.5	Signed square root	0.56	0.56
8	Cosine	4	1.00E-01	0.5	None	0.55	0.55
16	Cosine	4	1.00E-03	0.5	ReLU	0.51	0.51