Reproducibility Project for CS598 DL4H in Spring 2023

Adam Michalsky and Michael Pettenato
{adamwm3, mp34}@illinois.edu

Group ID: 186
Paper ID: 151
Paper Title: Disease Prediction and Early Intervention System Based on Symptom Similarity Analysis
Paper Link: https://ieeexplore.ieee.org/document/8924757
Presentation link: https://youtu.be/mQxvnVAgSN8
Code link: https://github.com/mikepettenato/cs-598-dl4health-final-project

1 Introduction

In healthcare, timing is critical. A patient's description on symptoms will help guide a physician to a diagnosis. The challenge that physicians face when interfacing with a patient is that patients don't know terminology that the healthcare industry uses to catalog symptoms. It is up to the physician to understand the symptom statement, and then map it to a condition. When doing this, the physician is making a similarity assessment between the patient's statement and what is generally accepted as a symptom.

Sentence similarity is a task that has had significant research done on it already. Deep learning models were developed to perform this task in the healthcare industry already but each model has its pros and cons.

A common problem for the previous models is the training time required. This was part of the focus in *Disease Prediction and Early Intervention System Based on Symptom Similarity Analysis* [1] which aimed to reduce the training time while improving accuracy by using an approach consisting of a Stanford Parser, Word2Vec embedding, and a convolution neural network (CNN) model to produce a similarity analysis. We would like to reproduce the findings documented in their paper.

2 Scope of Reproducibility

A CNN-based model architecture for similarity analysis can perform as well or better than models proposed in similar research with less training complexity. The model leverages a Stanford Parser to perform part-of-speech data pre-processing to extract the sentence trunk of the sentence. The sentence trunk is defined, in this paper, as the subject, predicate, and object portions of a sentence. Word2Vec is used to replace each extracted word with an embedding vector.

2.1 Addressed claims from the original paper

Below are the claims from the original paper that we plan on testing.

- Claim 1: Pre-processing using a Stanford Parser implementation in conjunction with Word2Vec will reduce model training time since the parser can identify a compressed version of the sentence via the subject, predicate and object (SPO) and pass these sentence vectors as input into the CNN.
- Claim 2: Feeding the vectors produced from the pre-processing step to a CNN based architecture can offer strong performance in sentence similarity when compared to similar research.
- Claim 3: The convolution layers can extract the features of the sentence to produce accurate sentence similarity scores.

3 Methodology

We are attempting to reproduce findings for paper [1], which describes an approach to sentence similarity processing that requires less training complexity while still improving on sentence similarity predictions. We are attempting to validate that

- Parsing sentences, using the Stanford Parser, yields adequate sentence representation to detect similar sentences.
- Using Word2Vec in a bag-of-words fashion is a powerful enough embedding strategy
- Using a CNN based architecture for sentence similarity can offer strong performance in sentence similarity when compared to alternative approaches

3.1 Model descriptions

The methodology that we have adopted for this reproduction is to, (1) use the Stanford Parser to find the sentence trunk for each sentence in both the training and test datasets, (2) tokenize the sentence trunks, (3) use Word2Vec to replace each word with its associated vector representation (4) create a CNN model for determining similarity across two sentences, (5) train the CNN model with the training data, (6) test the CNN model with the test data.

3.1.1 Final CNN Model Description

This is a description of the final CNN model developed for the reproduction of the CNN described in paper [1]. Because the paper did not describe in detail all aspects of the model and there was no reference implementation associated, ours is a blend of concepts and ideas found in the paper as well as architectural and implementation decisions based on our own research and intuition.

Convolution Layer1: The convolution layer will take as input batches of sentences, represented as tensors with the following shape *(batch_size, max_sentence_length, embedding_size)*. Any sentence that is not of max_sentence_length will be padded at the end. X_{batch} and Y_{batch}, equations (1, 2), are sorted such that X_i and Y_i are the embedding vectors of the two sentences that are being compared for similarity.

$$X_{batch} = [X_1, X_2, ..., Xi] \tag{1}$$

$$Y_{batch} = [Y_1, Y_2, ..., Yi] \tag{2}$$

The convolution layer utilizes wide convolutions to capture edge features as described in the original paper. The intent of the convolutional layer is to extract important features from the word embeddings so we will convolve across the word dimension setting the input channels to the size of the word embedding vectors. Additionally, we will set the output channels larger, which allows our convolutional layers to be more expressive in terms of the information it is extracting. The output of the convolutional layers will see the out channels grow in size, as specified by a parameter passed into the model called *num_filters*, while the number of words will shrink. A ReLU function is used for activation due to its ability to introduce sparsity into the neural network.

Pooling Layer1: Following the convolution, the output is passed through a *dynamic k-max pooling* layer. The pooling layer performs further feature dimensionality reduction, data compression and reduction of fitting degree of the sentence matrix [1]. The dynamic pooling layer allows the CNN model to keep a number of high value embeddings. The *k* number of embeddings that get selected is dynamic because it is determined by both the sentence length and the network depth, shown in equation (3)

$$k = \max\left(k_{top}, \left\lceil \frac{L-l}{L} |s| \right\rceil\right) \tag{3}$$

Convolution Layer2: The tensor is then passed through another convolutional layer. This layer takes as input the output of the dynamic k-max pooling layer. It then outputs a tensor representation that has double the number of out channels of the first convolutional layer. Adding an additional layer gives us several advantages. It increases the non-linearity of the model, allowing it to learn more complex features and relationships. It also increases the number of learnable parameters of the model, allowing it to learn more complex representation of the input.

Pooling Layer2: The second pooling layer is k-max pooling. In paper [1] k is set to 3. K-max pooling is applied after the dynamic k-max pooling in order to obtain a fixed-length representation of the input sequence that captures the k most important features across all positions in the sequence.

Using k-max pooling as the final pooling layer after dynamic k-max pooling layers enables the network to capture the most salient features of the input sequence, regardless of their position in the sequence. This can be particularly useful for text classification tasks, where important features can occur at different positions in the input sequence. Additionally, k-max pooling can help to reduce overfitting by selecting only the k-most important features.

Fully Connected Layer: At this point, we have extracted the important features of the sentence and we can now flatten the tensor representation received from the k-max pooling layer and pass it through the fully-connected linear layer to get a vector representation of the sentence, which can be used in similarity calculations. Our CNN model takes as an input parameter the size of the output of the fully-connected layer. For our model, configured with a 50 dimension embedding vector, we found that the output size that produces optimal results is one with 300 dimensions.

Sentence Similarity: Sentence similarity is calculated using Manhattan distance as defined below in equation (4). In order to ensure the score will be between 0 and 1, equation (5) is used.

$$Man(\vec{V}_x, \vec{V}_y) = |x_1 - y_1| + |x_2 - y_2| + \ldots + |x_n - y_n| \tag{4}$$

$$score = e^{-Man(\vec{V}_x, \vec{V}_y)}, \quad score \in [0, 1] \tag{5}$$

Loss: The Mean Square Error loss is used to calculate the loss between the expected value of the similarity flag and the predicated value. The loss will be used to update the weights during the back-propagation phase.

Gradient Descent: Stochastic Gradient descent is an iterative optimization algorithm used to find optimal results by taking small steps in the direction of the gradient. The size of the step is defined by the learning rate. The learning rate used is defined in table 2, which was based on the optimal configuration parameters found while training and testing the model.

CNN Model in Summary: The overall topology of our CNN is depicted in Figure 1. This is a high level overview of the final CNN model we used for the reproduction of paper [1].

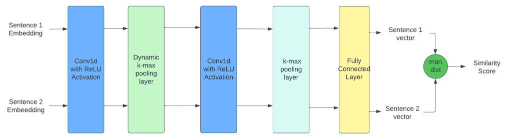

Figure 1: Final CNN Model

For further information about the input parameters to the CNN model, please refer to the the *SentenceSimilarityCNN2* class located in the *main.ipynb* of our github repository [2]

3.2 Data descriptions

The model was trained and tested using the Microsoft Research Paraphrase (MSRP) corpus. The MSRP corpus contains 5800 sentence pairs extracted from various media outlets. Each pair of sentences were annotated to indicate if they were paraphrases of one another where a value of 1 was they were and 0 was they were not. The data is available for download on the Microsoft website[1].

3.3 Hyperparameters Phase 1

The initial parameters that were used to train our first CNN, which used max pooling, are as follows:

Hyperparameter	Value
Batch Size	64
Kernel Size	3
Learning Rate	1e-1
Embedding Size	50
Max Pooling	3

Table 1: Hyperparameters

3.4 Hyperparameters Phase 2

Table 2 represent the final parameters used for training our second CNN that used dynamic k-max and k-max pooling. These parameters were found after doing a considerable amount of research and iterative training and testing runs.

3.5 Implementation

Paper [1] did not furnish a link to the existing code, so in order to reproduce the findings, we needed to implement all portions of this experiment ourselves. This included using the

[1]MSRP Data source https://www.microsoft.com/en-us/download/details.aspx?id=52398

Hyperparameter	Value
Batch Size	64
Kernel Size	3
Padding Size	1
Learning Rate	1e-2
Embedding Size	50
Num Filters	150
Sentence Vector Size	300
$dynamic - k_{max}$	len(sent)
k_{max}	3

Table 2: Final Hyperparameters

Stanford Parser to extract the SPO parts of speech, training and using Word2Vec to create the vector representation and implementing the CNN model to extract sentence features for the similarity comparison and benchmarks. Our code can be reviewed in the Github repository [2]

In many instances critical implementation details were missing from the paper. When faced with these situations we used our best judgement and have detailed the decisions made in the upcoming subsections as we tried to implement the algorithms and models described in this paper.

3.5.1 Parts-of-Speech Parser

The original paper used the Stanford Parser to find the parts of speech for each word in a sentence. This parser has been deprecated in favor of the *stanza* parser, so we used this as a replacement for the Stanford parser.

We found the parts-of-speech parsing to be slow. In order to speed it up we implemented a multi-threaded parsing class called *SentenceProcessingThread*. This class instantiated a stanza parser and took as parameters the sentences to process, the output list and the start and end indexes of the output list to store the results in. Additionally, we were able to detect if GPUs were available on the host machine and if they were we set the number of threads to a certain value and ran the stanza parser with GPUs enabled, adding additional performance improvements to the parsing performance. If GPU support was not enabled we changed the number of threads to use and ran in a slightly different configuration under CPU.

The original paper presents a pseudo-code algorithm, found in *Section III, subsection B SPO Kernel*. The paper had some discrepancies, where the pseudo-code did not match the textual description, found in *Algorithm 1, Trunk Construction*. We implemented our parsing algorithm according to the textual description, which seemed more elaborate than the pseudo-code, and from information gleaned from *Figure 3*, shown below for convenience

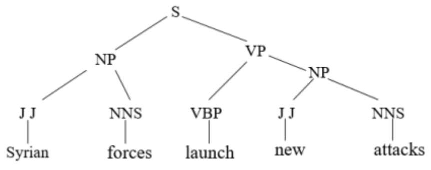

FIGURE 2. The syntactic tree constructed by the Stanford Parser.

Figure 2: Parsing Tree[1]

3.5.2 Embedding

Each word in the sentence needs to be replaced by an embedding vector. Consistent with the information described in the original paper, we used Word2Vec, with an embedding size of 50 to create sentence vectors. Equation (6) describes a word, W, in vector form:

$$W = (\mathbf{w}^1, \mathbf{w}^2, ..., \mathbf{w}^n) \tag{6}$$

where w represents an embedding vector and the superscript represents the sequence of words in a sentence.

Additionally, Equation (7) shows how the sentence is further organized

$$Sen = (\mathbf{S}^{\mathrm{T}}, \mathbf{P}^{\mathrm{T}}, \mathbf{O}^{\mathrm{T}}) \tag{7}$$

where S^{T}, P^{T}, O^{T} represent the subject, predicate, and object words, all with a similar structure to W.

The paper does not specify if their Word2Vec model used a pre-trained model or if it was trained on the MSRP corpus. For our reproduction, we choose to train Word2Vec on the MSRP corpus. The paper also does not go into detail about the shape of the sentence tensor. It is not clear if the subject, predicate and object words were each on their own axis or if the ordering of them was simply preserved on the same axis. In our implementation we preserved the subject, predicate object ordering on one axis and used it as input for the CNN.

3.5.3 Custom Dataset

Two instances of a PyTorch custom dataset, *MSPCDataset*, were instantiated, one that represented the training dataset and the other that represented the test dataset. The rational for using a custom dataset was to separate and encapsulate the data loading, pre-pocessing and manipulation that was required from the rest of the model. It also allowed us to easily use PyTorch's dataloader functionality, which has data batching facilities. The *MSPCDataset* produced an output format shown in equation (8).

$$Item = (Sent1, Sent2, SimFlag) \tag{8}$$

$Sent1$ and $Sent2$ have a format like Sen, shown in Equation (7), and $SimFlag$ is a true/false flag, 0 being not similar and 1 being similar.

3.5.4 CNN Model

Phase 1

Our initial approach to developing a CNN model that could learn sentence structures in a way that could predict sentence similarity was to use the details given in paper [1] and start with a simple version of the CNN. Specific implementation details were unavailable so we made certain assumptions about the model and selected things that made sense to us. The paper stated that the input was passed through two convolutions and two k-max pooling layers then to a fully connected layer. We started by building a simpler model defined in table [3].

Type	Activation	(in,out)
Conv1D	ReLU	(50,64)
MaxPool	–	–
Linear	–	(,50)

Table 3: Phase 1 CNN Architecture

Note: Please see table [1] for more information on the model parameters used for the phase 1 CNN architecture.

Using this simplified CNN we were able to achieve an *accuracy score of 0.670145*.

Phase 2

In phase 2 of the project we enhanced the CNN model by implementing two convolutional and pooling layers. The first convolutional layer used dynamic k-max pooling and the second used k-max pooling. This type of pooling would allow us to leverage a dynamic number of important features from the sentence embeddings. By doing this our sentence vector would still have the benefits of down-sampling, however, it would be more inclusive of the salient portions of the sentence than just using max pooling. As a result our sentence vector representation would be more robust and not as susceptible to over-fitting. The phase 2 model is shown in Table [4].

Type	Activation	(in,out)
Conv1D	ReLU	(50,150)
DKMaxPool	–	–
Conv1D	ReLU	(150,300)
KMaxPool	–	–
Linear	–	(900,300)

Table 4: Phase 2 CNN Architecture

Note: Please see table [2] for more information on the model parameters used for the phase 2 CNN architecture.

3.6 Computational requirements

As mentioned in the proposal, our intent is to be able to conduct our experiments on our personal PCs. The specifics for the machines used the experiments are below.

ID	Cores	Memory	GPU
1	8	8GB	–
2	6	16GB	8GB

Table 5: PC Specifications

The original paper mentioned a reduction in training time as a benefit of the model but never specifically offered empirical measurements of time for their model. Due to the lack of information, we did not have an estimate prior to implementation, however, based on the information in the Introduction section of paper [1], it can be inferred that other previous types of Neural Network models, used to perform similarity comparison, have been implemented using RNN models and, although they did yield good quality predictions, it is stated that the downsides of these models is the large amounts of time required for training. ID 2 of table 6 represents the training time for our CNN model

4 Results

The CNN model was trained on the *MSRP* training corpus. The model was trained on 4076 sentence pairs. Table 2 shows the hyperparameters used to train the model and table 6 shows the training performance that was achieved on a PC with specs listed in ID 2 of table 5.

The trained model was then tested against the test dataset. Table 7 shows the accuracy and F1 scores achieved.

Our work, shows that a parts-of-speech parser can extract enough important information from the sentences. Additionally, our CNN Model was able to use these parts-of-speech based sentences and achieve comparable sentence similarity

Epochs	Training Time (s)
80	104

Table 6: Performance

Accuracy Score	F1 Score
0.6945	0.8089

Table 7: Accuracy F1 Score

results. We were able to prove that, both, the parts of speech algorithm and the CNN model were able to extract the important pieces of information from the sentences to make accurate similarity predictions. We were able to achieve accuracy and F1 scores that approached those specified in the paper and we are confident that additional hyperparameter tuning can provide even closer results to the one achieved in the paper.

Additionally, it was shown through our training and testing iterations that using more parameters via larger sentence vector representations allowed the CNN model to be more expressive in describing the sentences and achieved better accuracy and F1 scores.

It is not clear, however, that the performance gains in training time, realized by the parts-of-speech parser, are truly worth it. Additionally, using the stanza parser is very slow, adding time and complexity to the pre-processing step.

Overall, the CNN model is quick to train. The process of training a CNN model for sentence similarity scoring in place of a more traditional recurrent neural network (RNN) model, produces a system that can be trained in a short amount of time.

5 Ablation Study

Planned Ablations vs. Actual Ablations

The ablations below were not the ones we had planned to do when selecting to reproduce the results of paper [1]. We felt the need to deviate from our planned ablations because our understanding of the subject matter changed during implementation. The actual ablations offered utility, better performance, and additional clarity for concepts we wanted to understand deeper. The actual ablations performed are as follows:

1. Conncurrent and GPU Parsing
2. Raw and SpaCy Sentence Pre-processing
3. Pretrained Word2Vec Embeddings

5.1 Concurrent and GPU Parsing

We found that the stanza parts-of-speech parser was very slow to use. This had a dramatic effect on pre-processing the data. We decided to develop a concurrent sentence processing subsystem that could use GPU capability if available. Table 8 shows the performance for different parsing strategies

Parsing Strategy	Threads	Time (s)
CPU	1	15193
GPU	1	1417
CPU	12	5940
GPU	2	983

Table 8: Stanza Sentence Parsing Times

5.2 Raw Sentence Pre-processing

In order to determine the value added by the parts-of-speech parsing we decided to process sentence with minimal pre-processing. We removed the stopwords and any word that was two characters or less. The goal of this ablation was to determine how valuable parts of speech parsing was to both the accuracy and training time performance of the model. Table 9

Parser	Accuracy	F1	Training Time (s)
SPO	0.6945	0.8089	104
RAW	0.6875	0.8088	105

Table 9: SPO Sentences vs Raw Sentences

We found that slightly better accuracy scores could be achieved when using the SPO based parser for sentence extraction. The overall training performance gains seemed to be nominal, however, further tests should be run on datasets with more complex sentences to see if this impacts the performance gains that can be achieved.

5.3 SpaCy Sentence Pre-processing

Our motivation for including the SpaCy parser was both functional and performance related. SpaCy's tagging strategy made SPO extraction easy and the SpaCy parser is known to be fast an efficient. Please refer to the *find_spacy_spo()* function in the *main.ipynb* file of our githb repository [2] for more details on SPO extraction using the SpaCy parser.

We wanted to see if we could achieve accuracy and F1 scores on par with that of the stanza sentence parsering using a SpaCy parser. Please refer to Table 10 for more details on the accuracy and F1 scores achieved using the SpaCy parser for SPO extraction.

Accuracy Score	F1 Score	Training Time (s)
0.6910	0.8092	109

Table 10: SpaCy Results

When comparing the accuracy and F1 scores acheived by SpaCy, in Table 10, and stanza, in Table 8, we found that both parsers had comparable SPO extraction capabilities. Performance, however, was vastly improved when using the SpaCy parser. This can be seen by comparing the SpaCy performance, shown in Table 11 with the stanza performance, shown in Table 8. The CPU based parsing speeds of the SpaCy parser vastly outperform the CPU based parsing of the stanza parser.

Parsing Strategy	Threads	Time (s)
CPU	1	690
CPU	12	622

Table 11: SpaCy Sentence Parsing Times

SpaCy also supports GPU based parsing, however, only CPU based performance was analyzed in this paper. Further investigation should be done on the performance gains that can be achieved using the GPU enabled SpaCy parser.

5.4 Pretrained Word2Vec Embeddings

Paper [1] called for using Word2Vec with a 50 dimensional word embedding. Based on the information in the paper, it was not clear if the authors used a pretrained Word2Vec model or a specifically trained model based on the MSRP corpus. We opted to train Word2Vec with the MSRP corpus and use this as way to create word embeddings.
As we worked through the implementation of our CNN model and got more familiar with Word2Vec, we questioned if training Word2Vec specifically on the MSRP corpus was the most effective way to use it. We hypothesised that using a specifically trained Word2Vec model on a small corpus of data would limit the ability for our model to derive similarity based on sentence meaning and using a pretrained Word2Vec model could improve the ability of our CNN to find similarities in sentences even when the exact same words were not used. Due to these questions and the general idea that we could improve our CNN model's ability to make sentence similarity predictions, we decided to use the *word2vec-google-news-300* pretrained Word2Vec model, which is trained on, approximately, 100 billion words. The vocabulary size of the model is around 3 million unique words and phrases and represents these words using vectors with 300 dimensions.
We were able to show that, when our model was trained with pretrained Word2Vec embeddings, the accuracy and F1 scores improved, as shown in 12

Accuracy Score	F1 Score	Training Time (s)
0.7119	0.818	118

Table 12: Pretrained Word2Vec CNN Results

6 Discussion

6.1 Claims

Claim 1: Pre-processing using a Stanford Parser implementation in conjunction with Word2Vec will reduce model training time since the parser can identify a compressed version of the sentence via the subject, predicate and object (SPO) and pass these sentence vectors as input into the CNN.

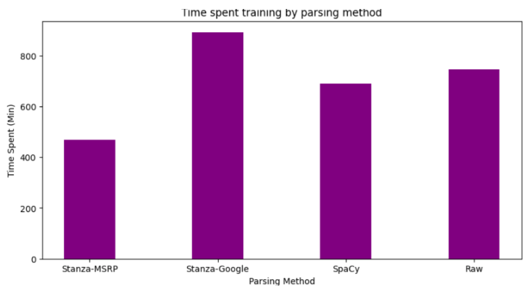

Figure 3: Training Times

As mentioned in the results section, the model can be trained in a short amount of time. The shortest time is 469 seconds for 80 epochs and the longest being 891 seconds for 20 epochs. Figure [3] is the comparison of training times for each model.

Claim 2: Feeding the vectors produced from the pre-processing step to a CNN based architecture can offer strong performance in sentence similarity when compared to similar research.

Due to time constraints, we couldn't validate all models using the data set we used for our experiment. Figure 4 is a model comparison offered in the original paper.

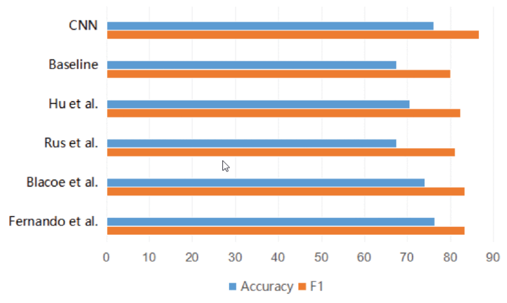

Figure 4: Original Paper Model Comparison

While our scores did not out perform the metrics shared for all models, our highest accuracy model was competitive with those shared. By being competitive and having similar evaluation metrics, we feel that our results support that CNN based architectures offer strong performance like other neural networks that perform similarity analysis.

Claim 3: The convolution layers can extract the features of the sentence to produce accurate sentence similarity scores.

Our accuracy metrics ranged from 0.68 to 0.712. We also performed an analysis using Word2Vec similarity function. See figure 5 for a comparison of scores.

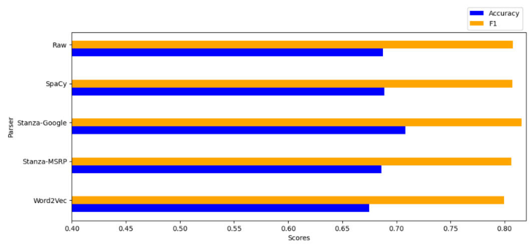

Figure 5: Original Paper Model Comparison

The CNN-based models were competitive with their accuracy metrics.

6.2 Reflection

Reproducing this experiment offered several challenges. While the paper covered a lot of content on how the model is constructed we found ourselves making educated guesses or experimenting through trial and error on decisions like using two-dimensional convolutions vs. one-dimensional convolutions, the number of out channels configured for the convolutions, how many dimensions the fully connected layer should use to represent the sentence vector.

As a way to empirically compare and contrast the model's hyperparameters, we developed a function called *find_training_parameters()*, which can be found in the *main.ipynb* file of our github project [2]. The function takes lists of hyperparameters as input and iterates through all of the combinations, training and evaluating the model. It stores the results of each combination in a pandas dataframe. It then outputs the result of the dataframe to a csv file for analysis. The results of some of the runs of this function can be seen in the *results/config-stanza-msrp.csv* and *results/config-stanza-google.csv* files of our github project [2]

Another challenge for us was understanding and then implementing dynamic k-max pooling into our network. While the formula makes sense, we found ourselves researching how to effectively use it in our model. Initially we used max pooling to produce our first results, but we modified our network based on the explanation provided in [3].

In general the lack of a code repository to reference offered the greatest challenge. A repository would have helped us validate some of our thought process, but we had to rely on other resources. One place where a reference would have helped was deciding how to pass input to the network. We eventually landed on passing each sentence as a parameter but we tried several other approaches before we got there. This would have been avoided if we had code to reference.

Although we experienced challenges while implementing this experiment, there were some aspects that made this paper easy to work with. The core concept of this paper is similarity analysis which is a natural language processing task. Our team was familiar with natural language processing so this aided in understanding the task and what needed to be done in the network. Additionally, the *Microsoft Research Paraphrase* dataset, used in paper [1], was readily available and easy to acquire. Once the core components of the experiment were developed, we found it easy to come up with useful and genuine ablations that we had questions on ourselves.

Future ablations that may be worth considering is to compare the CNN model for sentence accuracy described in this paper with the CNN model described in *Multi-Perspective Sentence Similarity Modeling with Convolutional Neural Network* by [4]. In this, the authors describes a process whereby sentences are passed through sentence convolutions with different sized kernels and dimensional convolutions to try and extract interesting features from the embedding dimensions themselves. This approach my offer an interesting comparison that compliments this work.

The primary feedback we would offer for the original authors would be to provide code to help reproducibility of the experiments conducted. The details provided are great, but a lack of reference code specific to what was done in the original experiment added a learning curve.

References

[1] Peiying Zhang, Xingzhe Huang, and Maozhen Li. Disease prediction and early intervention system based on symptom similarity analysis, 2019.

[2] Adam Michalsky Michael Pettenato. Reproducibility project for cs598 dl4h in spring 2023. `https://github.com/mikepettenato/cs-598-dl4health-final-project`, 2023. GitHub repository.

[3] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences, 2014.

[4] Kevin Gimpel Hua He and Jimmy Lin. Multi-perspective sentence similarity modeling with convolutional neural networks, 2015.