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: TODO
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

Convolution Layer: The convolution layer will take two inputs that are batches of embedded vectors. 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 to 64, which allows our convolutional layer to be more expressive in terms of the information it is extracting. The output of the convolutional layer will see the out channels grow in size 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 Layer: Following convolution, the output is passed through a pooling layer. The pooling layer performs further feature dimensionality reduction, data compression and reduction of fitting degree of the sentence matrix [1]. There are multiple methods for pooling, however the original paper settled on using k-max pooling with a dynamic k-max value determine by equation (3)

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

K-max is a function of sentence length s and network depth of the l.

Fully Connected Layer: At this point, we have extracted the important features of the sentence and we can now flatten the tensor representation, pass it through the fully-connected linear layer to get a vector representation of the sentence, which can be used in similarity calculations.

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 1, which was based on the configuration information described in the original paper.

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

The parameters used for training are as follows:

3.4 Implementation

The 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 parts of Stanford Parser to extract the SPO parts of speech, using Word2Vec to create the vector representation and

[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-1
k_{top}	3

Table 1: Hyperparameters

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.

3.4.1 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

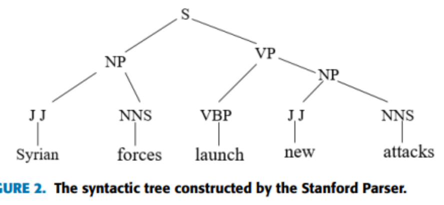
S
NP
VP
NP
JJ
NNS
VBP
JJ
NNS
Syrian
forces
launch
new
attacks
FIGURE 2. The syntactic tree constructed by the Stanford Parser.

Figure 1: Parsing Tree[1]

3.4.2 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 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}, S^{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 was a pre-trained Word2Vec model or their Word2Vec model 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.

3.4.3 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

[2]Code Repository https://github.com/mikepettenato/cs-598-dl4health-final-project

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.

3.4.4 CNN Model

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 [2].

Type	Activation	(in,out)	Parameters
Conv1D	ReLU	(50,64)	See table [1]
MaxPool	–	–	See table [1]
Linear	–	(,50)	See table [1]

Table 2: CNN Architecture

3.5 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 3: 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 4 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. ID 1 of table 4 shows the training performance that was achieved on a PC with specs listed in ID 2 of table 3.

ID	Epochs	Batch Size	Training Time (s)
1	100	64	31.79

Table 4: Performance

The trained model was then tested against the test dataset, which consisted of 1725 sentence pairs and achieved an *accuracy score of 0.670145.*

Our work, currently does not support all of the claims made by paper [1]. The authors documented that they were able to achieve 0an *accuracy score of about 0.75* when tested against the *MSRP* corpus.

The CNN model is very quick to train. The process of compressing the sentences using a parts-of-speech parser and training a CNN model, instead of an recurrent neural network (RNN) model, produces a system that can be trained quickly.

We were not able to determine that the parsing algorithm was capable of extracting enough important information from the sentences. Additionally, we were not able to determine that our CNN model was able to extract the important features from the sentence to make accurate predictions about similarity.

References

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