Reproducibility Report for “Comparing deep learning and concept extraction based methods for patient phenotyping from clinical narratives”

Jeremy Bao and Luke Freitag
{jbao8, lukegf2}@illinois.edu

Group ID: 50
Paper ID: 163
Presentation link: https://www.youtube.com/watch?v=Vn7eNdW615g
Code link: https://github.com/jbao8899/DLH-project

1 Introduction

We worked to replicate the paper “Comparing deep learning and concept extraction based methods for patient phenotyping from clinical narratives” [1]. It is about using CNNs to determine if patients have or lack various characteristics, based on discharge summaries. The paper’s authors said that creating automated methods for determining the characteristics of patients based on clinical texts is an important task, as these texts contain a lot of information not found in medical codes, but extracting it manually is time-consuming and error prone. Past attempts to use machine learning to accomplish this task were unwieldy, as they required experts to spend a lot of time creating dictionaries of terms relevant to the condition of interest and often did not generalize between hospitals.

This paper’s authors then claimed that advances in deep learning techniques enabled them to use neural networks to extract patient characteristics from natural language texts. They also described how their models were interpretable and introduced a method for figuring out which words and phrases contributed to their models’ decisions (we did not implement this, as the authors did not provide the dictionaries used to evaluate this functionality). Their models investigated a subset of the discharge notes found in MIMIC-III, and were designed to determine if patients described by those notes had advanced or metastatic cancer, advanced heart disease, advanced lung disease, chronic neurologic dystrophies, chronic pain, alcohol abuse, substance abuse, obesity, psychiatric disorders, or depression.

2 Scope of reproducibility

The first claim we tested is that CNNs following the architecture proposed in this paper can achieve higher performance (F1-score and area under the ROC curve (AUC)) than past methods, such as logistic regression on documents represented as bags-of-words or n-grams and random forest, naive Bayes, and logistic regression models run on counts of various medical terms extracted from documents using the cTAKES library. We tested this claim by constructing the CNNs described in the paper and comparing their performance to that of the logistic regression models we created, as well as to the results of the other baseline models described in the paper. This claim is interesting to investigate, as it would help us figure out if the CNN architecture proposed by this paper was actually useful.

The second claim we tested is that CNNs which consider larger n-grams of words will tend to yield better results than those which consider smaller n-grams, though diminishing returns will occur as larger n-grams are considered. We tested this claim by constructing CNNs considering different combinations of n-gram sizes and seeing how varying the size of the n-grams considered affected performance. Investigating this claim is important, as doing so would help us determine the optimal configuration for CNNs built according to the general architecture proposed in this paper.

3 Methodology

3.1 Model descriptions

The models discussed by this paper were CNNs operating on documents represented as stacks of embedding vectors created using Word2Vec. Each CNN was responsible for taking in discharge summaries and predicting if the patients described in them had or lacked one of the ten conditions under investigation. In our implementation, each document was stored as a PyTorch tensor with a height equal to the number of words in the longest document and a width equal to the embedding vector size chosen (with rows of 0s used to pad all documents to the same length). The j-th row of such a tensor contained the embedding vector for the j-th word in the document represented by that tensor.

As was done in the paper, for each of the ten conditions, we made 6 CNNs, one of which considered unigrams, a second that considered unigrams and bigrams, a third which considered unigrams, bigrams, and trigrams, a fourth which considered everything from unigrams to 4-grams, a fifth which considered everything from unigrams to 5-grams, and a sixth which considered everything from bigrams to 5-grams. For each n-gram size, 100 filters with a height equal to the length of that n-gram and a width equal to the embedding vector size were applied to the document, sliding 1 row down at a time. Each filter generated a columnar feature map with a size equal to the length of the longest document, plus one, minus the the size of the n-gram being investigated. As an example, a CNN considering unigrams and bigrams would have 100 filters with heights of 1 and 100 filters with heights of 2, all with a width of the embedding vector size. If the document size was 10, each unigram filter would return 10 elements, and each bigram filter would return 9. Implicitly, each filter detected occurrences of a certain phrase of a certain length.

Then, the ReLU activation function was applied to the feature maps, followed by a global max pooling operation, thus yielding a number of outputs equal to the total number of filters. For example, in a CNN considering unigrams, bigrams, and trigrams, the pooling operation would yield 300 outputs. After that, a dropout was applied to reduce overfitting, and the output of the dropout operation was fed into a fully connected layer to produce two outputs. These two outputs were passed through LogSoftmax to produce the final results. If the first output was greater, then the CNN classified the patient as not having the condition it was designed to detect. Otherwise, it classified that patient as having the given condition. Both we and this paper's authors used the Adadelta optimizer and negative log-likelihood loss function for training.

The models from this study were fairly small. For each kernel of each n-gram size in the convolutional layer, there is a single bias term and a number of weights equal to the size of the n-gram times the embedding vector size. The fully-connected layer has two bias terms and a number of weight terms equal to double the number of n-gram sizes times the number of filters per n-gram size. For a model considering unigrams and bigrams, with 100 filters per n-gram size and an embedding size of 100, the convolutional layer would have $2 * 100 = 200$ biases and $100 * 1 * 100 + 100 * 2 * 100 = 30000$ weights, and the fully-connected layer would have 2 biases and $2 * 2 * 100 = 400$ weights, for a total of 30602 trainable parameters.

We also created some baseline logistic regression models operating on bag-of-words and n-gram representations of the documents. CountVectorizer from scikit-learn was used to create bag-of-words and n-gram representations of the documents (cleaned and tokenized using the previously-mentioned function provided by the study's authors), and scikit-learn was used to train and test logistic regression models for each of the ten conditions under investigation.

3.2 Data descriptions

The data used in this model came from NOTEEVENTS.csv in MIMIC-III. We were able to download this from PhysioNet after completing a training course and signing a usage agreement. NOTEEVENTS.csv contained 2,083,180 clinical notes of various types, including 59,652 discharge summaries. For each note, NOTEEVENTS.csv recorded its type, the ID of the visit and patient described, its creation time, its text contents, and some information irrelevant to us. Of these discharge summaries, the study's authors focused on 1,610, having domain experts label each for the presence or absence of the ten conditions under investigation. These annotations were stored in data/annotations.csv, within the GitHub repository linked to the paper. However, the paper's authors improperly labeled their annotations, preventing us from matching labels to documents whenever a patient/visit combination had more than one discharge summary (they essentially stored either the visit ID or 999999 as the creation time of each annotation). We therefore used only the 1,341 discharge summaries which were both labelled and could be uniquely identified.

After cleaning and tokenizing all discharge summaries in MIMIC-III (using code provided with the paper), we used Word2Vec to create word embeddings for the words present in them (as done in the paper) and represented each of the 1,341 discharge summaries to be used as described in section 3.1.

3.3 Hyperparameters

For Word2Vec, we used the continuous bag of words model, a window size of 10, negative sampling of 10, a word dropping threshold of 5 appearances, and trained for 15 iterations. All of these hyperparameter values were taken from a PDF attached to the paper. The size of the embeddings was not mentioned in the provided materials, so we used the default embedding vector size of 100. We used the gensim library to carry out Word2Vec.

For the CNNs, we used 100 filters per n-gram size, a dropout probability of 0.5, an Adadelta optimizer with $\rho = 0.95$ and $\epsilon = 1e - 6$, and 20 training epochs, as was done in the paper. We used a batch size of 32 because that is what we did in the homework. 80% of the data was used for training and 20% for testing, because the authors of the study used 70% for training, 10% for validation, and 20% for testing, and we did not need a validation set because we copied their hyperparameters.

The authors of the study said "all parameters were initialized using a uniform distribution from -0.05 to 0.05" and that "after every parameter update, the parameters of the feature maps were normalized to a norm of 3" in a PDF attached to their paper, but did not elaborate further on this. Based on inspection of their CNN code (which was written in Lua and hard to understand), they actually initialized the weights of the convolutional layers using a uniform distribution between -0.01 and 0.01, the weights of the fully connected layer using a normal distribution with a mean of 0 and a standard deviation of 0.01, and set all biases to 0. We decided to initialize the convolutional weights and biases as was done in the provided code, but allowed PyTorch to initialize the parameters for the fully-connected layer, because that seemed to yield the best results (according to some unsystematic trial-and-error). In their normalization step, they normalized the two vectors of weights going into the two output nodes of the fully connected layer to have L2 norms of 3, which we also did.

We largely copied the hyperparameters from the paper instead of optimizing them ourselves due to a variety of reasons. Firstly, we were investigating 60 different models, so we did not know what should be used as the criteria for determining hyperparameter quality during optimization. Secondly, there were a large number of hyperparameters, many of which we did not fully understand, and while each model was quick to train, training all of them took over an hour, so the optimization process might have taken too long and consumed too many computational resources. Thirdly, the provided hyperparameters yielded acceptable results for most of the models under investigation, so we were more concerned with trying to figure out what was causing the total failure of some of the models than with trying to see if we could squeeze a bit more performance out by tinkering with hyperparameters.

For the logistic regression models, we just used regular logistic regression, without adding any penalty terms to make it use Ridge, LASSO, or elastic net. 75% of the data was used for training and 25% for testing, as that is the default for sklearn.model_selection.train_test_split().

3.4 Implementation

The authors of this study provided some code at https://github.com/sebastianGehrmann /phenotyping/. There, they included code for preprocessing the data, implementing baseline models, and training and testing their CNNs. However, this code requires you to input some Word2Vec data in an unknown format (which was not provided), so we were unable to run it. Also, we found it difficult to understand their CNN code, as it was written in Lua and not very well documented. We therefore mostly used the provided code for inspiration and wrote most of our code ourselves.

Our code is at https:// github.com/jbao8899 /DLH-project. It should be uploaded to and run on Google Drive. The actual data used is not included within the repository due to privacy concerns.

3.5 Computational requirements

It was difficult for us to predict the computational requirements ahead of time. The paper did not discuss this, beyond mentioning in the attached GitHub repository that training might be slow without GPU acceleration. We predicted that it would not take extensive computational resources or that much time to train their models, because less than 2,000 documents were involved in the study, and those documents were not very long.

Our code was run on Google Colab. The code for creating word embeddings using Word2Vec was run using a high-RAM setup (25.5 GB of RAM, no GPU), and took around 35.77 minutes to complete. The code for training CNNs was run on a setup with normal RAM and normal GPU acceleration, (12.7 GB of RAM, 15 GB of GPU RAM). With this setup, each of our models could be trained fairly quickly, with the unigram-only models taking around 10 seconds to train and the largest models taking around 2 minutes. Total training time for all

60 CNNs was around 1 hour and 10 minutes. The logistic regression code was run with a basic setup (12.7 GB of RAM, no GPU), with training time varying between about 1 second for 1-gram models to 24.3 seconds for models considering 1-grams to 5-grams, with a total training time of about 11 minutes for all 50 logistic regression models.

4 Results

All in all, our results were poor. Due to the number (60) of models created, we will not display all of our data here (data and visualizations can be found in "Display Notebook.ipynb" in our GitHub repository). Some of the CNNs we created performed fairly well, achieving results similar to those described in the paper. For instance, our CNN using unigrams, bigrams, and trigrams to detect advanced cancer obtained precision, recall, F1-score, and AUC values of 0.79, 0.58, 0.67, and 0.93, compared to 0.74, 0.65, 0.69, and 0.93 for the same model from the paper. Others performed worse than the corresponding CNNs from the paper, but did not totally fail. For instance, our model for predicting advanced heart disease using unigrams, bigrams, trigrams, and 4-grams obtained precision, recall, F1-score, and AUC values of 0.75, 0.24, 0.37, and 0.93, compared to 0.74, 0.63, 0.68, and 0.91 for the corresponding model from the paper.

However, some of the models classified every or nearly every testing example as not having the relevant condition, resulting F1-scores and recalls of (or near) 0 (0 for the F1-scores indicated an undefined score). We tried many ways of dealing with this, but were unable to resolve the issue. Setting different seeds would cause different models to fail, though we noticed that the models designed to predict obesity tended to fail especially often, for unknown reasons. Below, you can see a table displaying the performance of models designed to predict if patients were obese. Each column contains the values of the relevant statistic from our implementation (first) and from the paper (second).

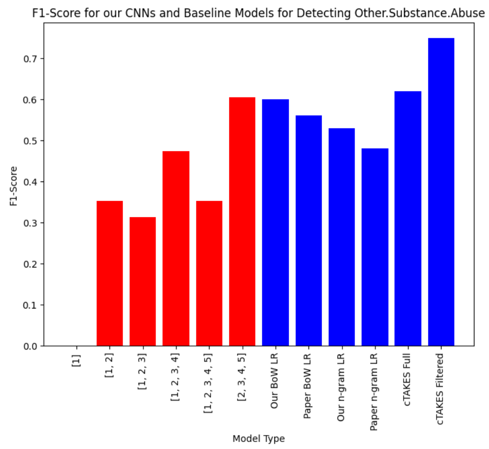
Model	P	R	F1	AUC
1	0, 17	0, 50	0, 25	74, 69
1-2	0, 95	0, 95	0, 95	69, 100
1-3	0, 95	0, 95	0, 95	70, 100
1-4	0, 100	0, 95	0, 97	69, 100
1-5	0, 95	0, 95	0, 95	68, 100
2-5	67, 100	8, 95	14, 97	75, 100

In the above table, "1-3" refers to a CNN which used 1-grams, 2-grams, and 3-grams to decide if patients described by discharge summaries had or lacked obesity. As you can see, most of the CNNs designed to detect obesity did very well in the paper, while 5 of our models totally failed by classifying every patient as non-obese, and the last also did poorly. Despite spending some time investigating, we could not rectify this issue.

On the whole, while our results did not conclusively support our claims from section 2, our replication study does not prove that they are incorrect or that this paper is flawed, as we are not confident that we perfectly replicated the original study.

4.1 Result 1

Our first claim is that the CNNs proposed by the paper are better at predicting if patients have various conditions based on discharge summaries than various non-deep learning methods. Our results did not support this claim. Due to the sheer number of models involved, we will not display all of our findings in detail. However, you can see by looking at the bar charts comparing the F1-scores and AUCs for our CNNs and various baseline models (see "Display Notebook.ipynb" for visualizations) that our implementations of the paper's CNNs fail to consistently outperform the baseline models, with the best CNN we implemented for predicting each condition usually being outperformed by the best baseline method for that condition, and the CNNs sometimes totally failing.



As an example, the bar chart above shows the F1-scores from using our implementations of the paper's CNNs and various baseline models to predict if patients abuse non-alcohol substances (see "Display Notebook.ipynb" for more detail). The best baseline model outperformed all of our CNNs, and one of the CNNs totally failed (the CNN using unigrams to detect substance abuse classified nearly every example as not exhibiting substance abuse and has precision and recall values of 0). This pattern can be seen across most of the conditions under investigation. As a result, we conclude that our findings fail to demonstrate the superiority of this CNN architecture over past methods for extracting patient characteristics from text documents, but we do not suggest that the paper is wrong, as we are not confident that we have replicated the procedures from the paper with sufficient accuracy.

4.2 Result 2

Our second claim is that CNNs considering larger n-grams will tend to yield better results than those considering smaller ones, with diminishing returns. Our results are somewhat supportive of this claim. That pattern does hold in the bar chart above, with F1-score tending to increase with n-gram size for CNNs predicting substance abuse, but if you look at all our bar charts showing the values of F1-score and AUC for different n-grams considered, for different conditions, it is not immediately obvious that large models are better. Here is a table of the best model (denoted by which n-grams are considered) for each condition, considering F1-score and AUC:

Condition	F1	AUC
Cancer	(1, 2)	1
Heart Disease	(1, 2)	(1, 2, 3)
Lung Disease	(1, 2, 3, 4, 5)	(2, 3, 4, 5)
Chronic Neuro	1	(2, 3, 4, 5)
Chronic Pain	(1, 2, 3, 4, 5)	(2, 3, 4, 5)
Alcohol Abuse	(1, 2, 3, 4)	(1, 2, 3, 4)
Substance Abuse	(2, 3, 4, 5)	(1, 2, 3)
Obesity	(2, 3, 4, 5)	(2, 3, 4, 5)
Psychiatric	(2, 3, 4, 5)	(1, 2)
Depression	(1, 2, 3)	(1, 2)

This table suggests that larger models tend to be better, as the models with the best F1-score and AUC for each model tend to be larger ones, but the actual differences between models are fairly small, we do not see a clear trend of increasing performance metrics with increasing n-gram size in most of our bar charts, many of our models did not even work, and many others functioned fairly poorly. As a result, we think that this claim may be true, but are not confident that it is the case.

4.3 Additional results not present in the original paper

We performed two ablations in our replication study. Due to space constraints, we will not include any visuals regarding them here, but many can be found in "Display Notebook.ipynb". Firstly, we tested how not applying ReLU to the results of the convolutional layer would affect CNN performance. We thought that this would be interesting to investigate, as in the proposed CNN architecture, a global max pooling operator is applied to each filter's output after the ReLU, which would make the ReLU do nothing if the kernel returns even a single positive value. We found that removing the ReLU sometimes reduces the F1-score and sometimes increases it, but consistently increases the AUC. Whether ReLU is included or not, some of the models fail totally and return an F1-score of 0. We do not know what these results indicate.

In the second ablation, we investigated the effects of removing the max pooling operation applied to the results of each filter (with the ReLU included). We thought this would be interesting, as although we learned in class that pooling is important for reducing the effects of small translations, we thought that condensing the results of each filter to a single value may have discarded too much information. We therefore implemented a version of the CNN architecture without pooling and trained and tested 60 CNNs following that architecture. In "Display Notebook.ipynb", you can see the results of this ablation. The F1-scores of the CNNs without pooling tended to be significantly lower than those of the corresponding CNNs which included it, while the AUCs were sometimes higher and sometimes lower. All in all, we conclude that the pooling step was useful for improving performance in this model. However, even better results might have been obtained by pooling smaller areas of convolutional output, as this would still help reduce the effects of local translations while transmitting much more information to the fully-connected layer. We did not have the time to test this.

5 Discussion

We do not feel that our replication study has yielded significant insights into the validity of this paper. While we were unable to definitively prove the claims we set out to investigate, that may well be due to problems with our reproduction and not problems with the proposed CNN architecture. Our ablations do suggest that the application of global maximum pooling to the results of the convolutional layer was of use.

5.1 What was easy

It was easy for us to understand the general structure of the CNNs from the paper due to their simplicity (only 2 layers with learnable parameters were present). We also found it fairly easy to create word embedding vectors using Word2Vec, because gensim provided good tutorials on this. We originally thought that it might take too long to run Word2Vec on all discharge summaries in MIMIC-III, but this actually only took 35.77 minutes. The ease of accomplishing these two tasks makes it possible to use the CNN architecture from this paper to determine if patients in novel data have various conditions, provided you managed to fix the issue causing models to sometimes totally fail.

5.2 What was difficult

It was difficult to figure out what was causing our CNN implementations to fail. We spent many hours tinkering with the optimization method, loss function, activation functions, various hyperparameters, parameter initialization, and normalization methods (the instructions provided were rather vague), seeing if that improved the results, and printing out both the final results and many intermediate values to see if we could figure out where the error was. Implementing normalization of the fully-connected layer's parameters seemed to reduce the proportion of models which totally failed, but nothing else helped. We tried contacting the other groups working on this paper, and the group which replied noted that they experienced the same issue as us and did not know how to solve it.

One strange thing that we noticed is that, for the failed models, even though training error decreased with each epoch and a decent number of training examples would be classified as having the condition, even during the last training epoch, they would still totally fail during testing. No matter if we used the training set or the test set for evaluation, every (or nearly every) example would be classified as not having the relevant condition during evaluation, which makes us think that overfitting is not the problem. We created boxplots of the probabilities returned by both working and non-working CNNs. The overall shape of the data was the same in both cases, but in the case of the failed CNN, all the probabilities were below 0.25, while in the case of the successful one, they ranged from a bit below 0.2 to above 0.8. Also, we noticed that for some of the failed models which actually marked some test examples as positive, precision was very high and recall was low. We do not know what this might indicate or how our problems could be fixed.

5.3 Recommendations for reproducibility

Many improvements could be made to make this paper more reproducible. Firstly, including the code for creating word embeddings would have made it possible for us to run the provided CNN code, which would have helped us understand it better. A link to the word embeddings used was included, but it was dead. The word embeddings we made were in a different format. Secondly, the information presented in the PDF describing model details should be corrected, as the information there about parameter initialization differs from what was done in the code. Thirdly, more comments should be added to the code, to explain what was going on. As it was, I had to figure out what was going on based on trying to interpret code from a language I do not know (Lua) and which was basically completely undocumented without being able to run the code. Fourthly, the creation times of the documents used should have been recorded, as that would have allowed us to uniquely identify which set of labels corresponded to which discharge summaries in all cases and thus not have to omit 269 documents from our replication study.

6 Communication with original authors

Our communication with the original authors were unsuccessful. We reached out to the first author listed (Sebastian Gehrmann) via email, and he did not provide us with much useful information. For instance, when we asked about how we might deal with the mislabelled annotations, he sent us an article about matching MIMIC-II documents with MIMIC-III documents, which was totally irrelevant for us. When asked for some clarification regarding his architecture and Lua code in our third email, he stopped responding. However, we understand that Dr. Gehrmann is probably very busy and might not remember much about some project he worked on 5 years ago.

References

[1] Sebastian Gehrmann et al. "Comparing deep learning and concept extraction based methods for patient phenotyping from clinical narratives". In: *PloS one* 13.2 (2018), e0192360.