Convolutional neural networks for text classification

Julen Miner

Abstract

In this document we report my implementation of a convolutional neural network that is capable of classifying different documents in 20 classes. The 20 topics are already defined and a lot of documents are provided for the implementation of the project. The performed tasks were: get the text from the data files and from all of the folders, preprocess the data using a tokenizer and word embedding using GloVe's pretrained model, build the convolutional neural network, train it and test it with the test data. After training the convolutional neural network, four plots are shown to represent the accuracy and loss value of the model in the training process. Finally, we can see the results of the testing data, which shows the behavior of convolutional neural networks when they are used for text classification and/or text analysis.

Contents

1 Description of the problem **3**

2 Description of our approach **3**

3 Preprocessing of the data **3**

3.1 Data source 3

3.2 Loading the documents and labels 3

3.3 Tokenization of the sentences 4

3.4 Word embeddings 4

4 Processing of the data **4**

4.1 Convolutional Neural Network 5

4.1.1 Architecture of the CNN 5

4.1.2 Training process 6

4.1.3 Testing process 6

4.2 Results 7

4.2.1 Training results 7

4.2.2 Testing results 8

5 Conclusions **9**

6 Implementation **9**

1 Description of the problem

Convolutional neural networks are mainly used for image analysis. However, recently they have been used in the analysis and classification of different texts and the results obtained using CNNs have been very good.

In this project a convolutional neural network has been implemented, capable of classifying a text in 20 different classes. For this process, the data has been taken from a public source [1] and it is provided in three different ways. I chose one that separates the data in train and test data. Then, the data was preprocessed to obtain information that the convolutional neural network could be able to comprehend. After training the model, it was tested with the test data.

2 Description of our approach

I organized the implementation of the project according to the tasks:

1. Preprocessing of the data set
2. Building the convolutional neural network
3. Training the convolutional neural network
   (a) Training of the model
   (b) Visualize the training results
4. Testing the convolutional neural network
   (a) Predicting the classes of the test data
   (b) Show the results of the model

3 Preprocessing of the data

3.1 Data source

As it is suggested in the description of this problem, the data source is taken from the 20 Newsgroups page [1]. From this page, you can download a compressed file that contains all of the documents that can be used to train this problem. Even though all of the documents are inside folders that correspond to a given topic (therefore, there are 20 folders), the data is given in three different ways:

- the original one, that contains all of the documents
- the second one, sorted by date into training (60%) and test (40%) sets and does not contain duplicates
- the third one, does not contain duplicates and headers are removed

In the same page it is recommended to use the second option, as it is already divided into train and test data and does not contain duplicates.

3.2 Loading the documents and labels

To load the documents two functions were defined, `read_files`, able to read the content of files in a given directory, and `clean_doc`, able to clean a given string by lower casing and removing the stop words, punctuation and words that have only one character.

To assign a label, since each set of documents is about one topic, it will be assigned a number, starting from 0, to each topic taking into account the number of documents in each set. Then, the function `to_categorical` function was used, imported from `keras.utils`, that will convert each label in an array that correspond to the label (see Figure 1)

```
# Consider an array of 5 labels out of a set of 3 classes {0, 1, 2}:
> labels
array([0, 2, 1, 2, 0])
# `to_categorical` converts this into a matrix with as many
# columns as there are classes. The number of rows
# stays the same.
> to_categorical(labels)
array([[ 1.,  0.,  0.],
       [ 0.,  0.,  1.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 1.,  0.,  0.]], dtype=float32)
```

Figure 1: What `to_categorical` function does to a given array

3.3 Tokenization of the sentences

The text extracted from the data source needed to be tokenized, so for this section the Tokenizer provided by keras was used, as it has been used in other papers about text analysis (for example, in *Stance Prediction for Russian: Data and Analysis* [2]). Using this tokenizer, some additional information has been extracted: the maximum length of a document is 7055 words, the minimum length is 12 words, the mean length is 168 words and the size of the vocabulary (different words overall) is 118469. Taking into account those numbers, a limit for the text size was chosen, in order to fix the length of all the texts.

Using this tokenizer and the vocabulary, a dictionary is made. Thanks to this dictionary, each of the words that appear on all of the texts will have a numerical representation. Up to this point, each of the document have a vector representation with a word in each position of the vector, but those words can not be processed by the convolutional neural network, so, using the dictionary, each of the words in those vectors, will be changed with its corresponding number. This process will help when we use an embedding layer.

3.4 Word embeddings

In order to obtain a representation of the words, so the following CNNs can be trained, some pre-trained models were used. The aim of this is to see how affects the representation of the words in the training process. The used models are the following:

- GloVe [3], developed by the Standford University, it will provide vectors for word representation. The work that they developed can be seen in the paper related to this pretrained model [4].
- Google's trained Word2Vec [5], it includes word vectors for a vocabulary of 3 million words and phrases that they trained on roughly 100 billion words from a Google News dataset.
- WikiNews [6], developed by FastText (Facebook), 1 million word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).
- Crawl [6], developed by FastText (Facebook), 2 million word vectors trained on Common Crawl (600B tokens).

Using the pretrained models, all of the words that are on the dictionary will be converted into vectors. Those will serve as the embedding matrix and will be used to build the embedding layer of the convolutional neural networks. The size of each individual word embedding is 300 for all four of the models.

4 Processing of the data

For the solution of this problem, four convolutional neural network were implemented, capable of analyzing and classifying text documents in 20 different classes. The difference between those four CNNs is that each of the networks has a different embedding layer.

4.1 Convolutional Neural Network

In deep learning, a convolutional neural network (also called CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. Convolutional networks were inspired by biological processes[4] in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. For example, given many pictures of cats and dogs it learns distinctive features for each class by itself. CNN is also computationally efficient. It uses special convolution and pooling operations and performs parameter sharing. CNNs have been successfully used for image classification, as we can see in some researches, as it is explained, for example, in the paper *A Convolutional Neural Network Cascade for Face Detection* [7] where they obtained high accuracy on face detection using convolutional neural networks. However, recently CNNs have been used for text analysis and some researches have obtained very good results, as it is explained in the paper *Convolutional Neural Networks for Sentence Classification* [8].

4.1.1 Architecture of the CNN

Convolutional neural networks are composed by a series of different layers that are able to perform some operations to the input data (see Figure 2). The CNNs that I created, have an embedding layer, a layer that is not usually seen when CNNs are used for image analysis. This layer will replace the simple vector with the numeric representation of the words in a text by a matrix that will have a vector representation of those words. The convolutional neural networks are usually composed by convolutional and pooling layers, followed by a number of fully connected layers and the softmax output, since this problem is a multiclass problem.

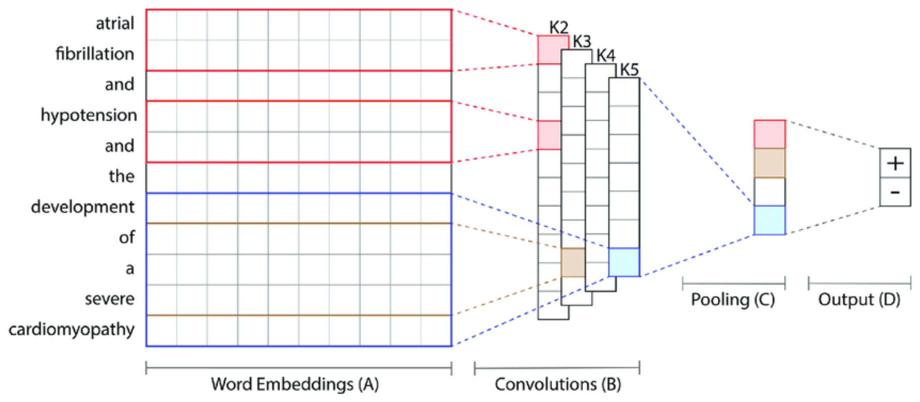

Figure 2: Example of a CNN architecture, taken from the paper Comparing deep learning and concept extraction based methods for patient phenotyping from clinical narratives [9].

The main building block of CNN is the convolutional layer. Convolution is a mathematical operation to merge two sets of information. The convolution is applied on the input data using a convolution filter (also called the kernel) to produce a feature map. The convolution operation is performed by sliding this filter over the input. At every location, element-wise matrix multiplication is done and the results are summed. This sum goes into the feature map. Multiple convolutions are performed on an input, each using a different filter and resulting in a distinct feature map. Then, they are stacked all these feature maps together and that becomes the final output of the convolution layer.

For this project, the convolutional layer used was the depthwise separable 1D convolution. Separable convolutions consist in first performing a depthwise spatial convolution (which acts on each input channel separately) followed by a pointwise convolution which mixes together the resulting output channels. The `depth_multiplier` argument controls how many output channels are generated per input channel in the depthwise step.

After a convolution operation, pooling is usually performed to reduce the dimensionality. This enables to reduce the number of parameters, which both shortens the training time and combats overfitting. Pooling layers downsample each feature map independently, reducing the height and width, keeping the depth intact. The most common type of pooling is max pooling which just takes the max value in the pooling window. Contrary to the convolution operation, pooling has no parameters. It slides a window over its input, and simply takes the max value in the window. In this project, a global max pooling operation was used.

To the convolutional neural network, it is also added a dropout. Dropout is by far the most popular regularization technique for deep neural networks. It is used to prevent overfitting and the idea is very simple. During training time, at each iteration, a neuron is temporarily dropped or disabled with probability p. This means all the inputs and outputs to this neuron will be disabled at the current iteration. The dropped-out neurons are resampled with probability p at every training step, so a dropped out neuron at one step can be active at the next one. The hyperparameter p is called the dropout-rate and in this project it has the value 0.4, corresponding to 40% of the neurons being dropped out. The reason is that dropout prevents the network to be too dependent on a small number of neurons, and forces every neuron to be able to operate independently.

4.1.2 Training process

All four of the convolutional neural network models trained in this project have the same layers, except for the embedding layer. The loss value is calculated with the categorical cross entropy function. This loss function is used for multi-class classification where each example belongs to a single class. Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0. The calculation is done by calculating a separate loss for each class label per observation and sum the result:

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

Where:

- M - number of classes (20 in this project)
- y - binary indicator (0 or 1) if class label *c* is the correct classification for observation *o*
- p - predicted probability observation *o* is of class *c*

The optimizer of the models is the Adam optimizer. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. More information of the Adam optimizer can be found on its original paper [10].

The training data is divided in two: the actual training data and the validation data. The size of the validation data is the 20% of the original training data. The original set is split using the function `train_test_split` imported from `sklearn.model_selection`, but in this project is not used to split into train and test data, but into train and validation data. The function divides the data with a given randomness in order to have data from different classes. Since the original dataset has balanced examples, I did not have to deal with unbalancement.

The training process is done with 10 and 40 epochs, so each of the model is trained twice, in total 8 models. This is done to see how the number of epochs affects in the training process of the model. After fitting the examples in each convolutional neural network, the model is saved in the ”models” folder and the history is saved in order to plot the results later.

4.1.3 Testing process

To test the models with the test data, all of the test documents need to be loaded using the same method as the beginning, when the training data was loaded. So, all of the test data is preprocessed.

Then, the models are loaded from the ”models” folder. Those loaded models are individually evaluated using the test data and the results show the loss and accuracy values. The test data, as the train data, is balanced.

4.2 Results

There are two types of results in this project: the ones that were generated by the training process and the ones that were generated in the testing process. Since there are in total eight different models, this section will compare the results among them.

4.2.1 Training results

The training results give four values for each epoch: loss value, accuracy value, loss value with the validation data and accuracy value with the validation data. There are four models in this project with four embedding layers, but those four models are trained with different number of epochs: 10 epochs and 40 epochs.

If the results of the models **trained with 10 epochs** are shown, we can see that the accuracy and loss value of the four models (see Figure 3) in the training process follows more or less the same path. In this project, the accuracy value using the WikiNews embedding layer is lower than the other model values in the first epoch, and the same happens with the loss value, where the same model gives a higher value. Nonetheless, after the 10 epochs all four of the models give almost the same accuracy and loss values.

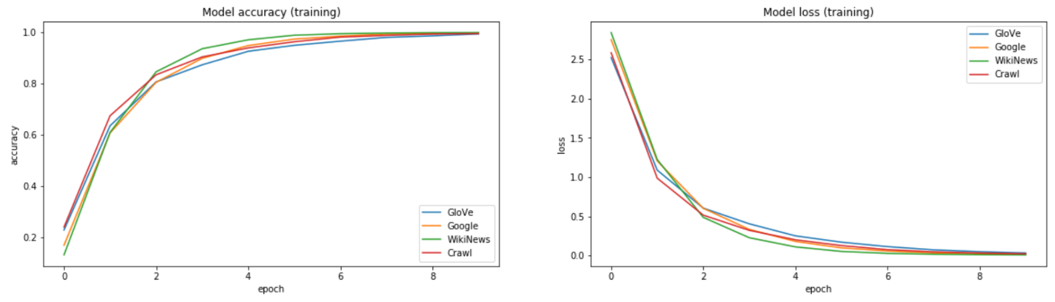

Figure 3: Plot of the training performance with 10 epochs.

If we look at the results of the validation data (see Figure 4), it can be seen that the difference in the starting epoch is greater than the difference in the training data. However, after the 10 epochs, the values stabilize and the differences among the values is not huge. As a matter of fact, the model with the highest accuracy and the lowest loss value is the model that has the Crawl embedding layer, and the model with the lowest accuracy and the highest loss value is the model that has the WikiNews embedding layer.

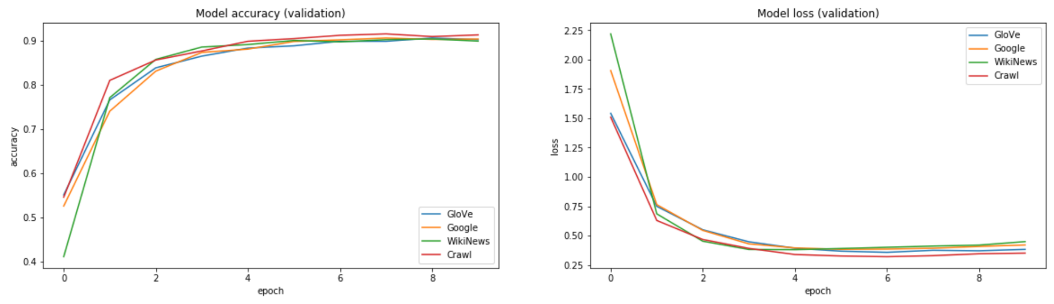

Figure 4: Plot of the validation performance with 10 epochs.

Now, if we compare the results of the models trained with 10 epochs with the models trained with 40 epochs, we can see that there is a slight difference. At first sight, there is not very relevant information in the plots of the training data results (see Figure 5). The values are very stable and almost perfect (meaning that the accuracy is almost 1 and the loss is almost 0) for the models after the tenth epoch.

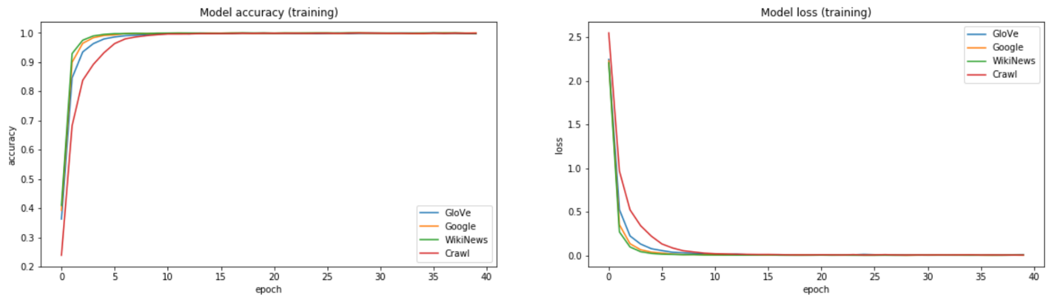

Figure 5: Plot of the training performance with 40 epochs.

But those values give more information if we compare those results with the results obtained with the validation data (see Figure 6). The lines on the plots obtained with the validation data are not as straight as the ones obtained by the training data. The accuracy value reach its highest value between the fifth and the tenth epoch, and then start to slightly decrease. As a result, the loss value increases where the accuracy decreases. This happens because as the model has more epochs in the training process, the model is more overfitted. Training loss keeps going down but the validation loss starts increasing after around epoch 10. And what is happening? The model is memorizing the training data, but its failing to generalize to new instances, and thats why the validation performance goes worse. So, the conclusion here is that the number of epochs chosen to train a model really affects the performance. If the training data would have more examples, tha overfitting would occur in a later epoch.

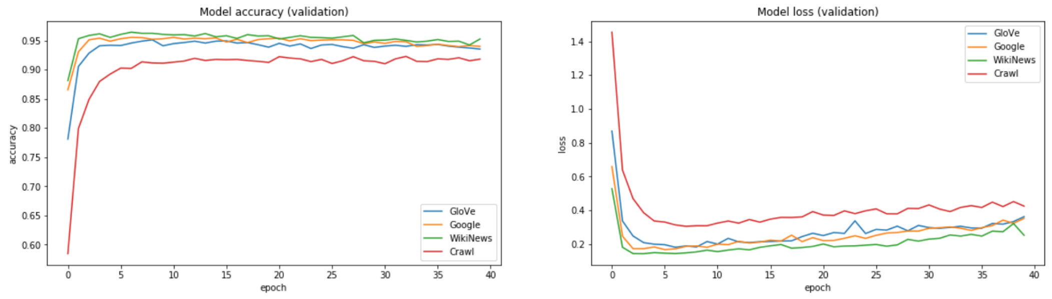

Figure 6: Plot of the validation performance with 40 epochs.

4.2.2 Testing results

Seeing the results of the testing (see Figure 7), we can see that the accuracy values for the models trained with 10 epochs are almost equal as the ones obtained with the 40 epochs. However, the loss values are higher for the models trained with 40 epochs. We can conclude that all of the time that takes to train the models with 40 epochs is not worth since the results with 10 epochs are equal or better (the accuracies are almost equal and the loss values are better).

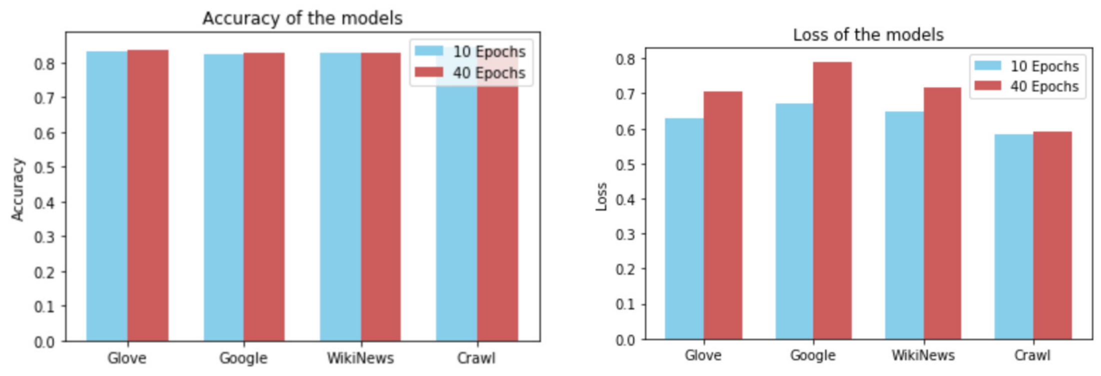

Figure 7: Plot of the testing performance.

5 Conclusions

To put it in a nutshell, the results obtained in the project have demonstrated that the accuracy of the models is really high if the number of epochs is adequately defined. Furthermore, I concluded that the usage of different word embeddings does not highly affect in the results of the model. Since all of the models with the 10 epochs give a very high accuracy, the conclusion is that the usage of convolutional neural networks for text analysis or text classification is actually a good idea.

As it can be seen in other successful researches where convolutional neural networks have been used, as in the paper Medical Text Classification Using Colvolutional Neural Networks [11], where we can see their results comparing the text classification implemented with a convolutional neural network, with other text classification methods (see Figure x). The results show that the accuracy obtained with the convolutional neural network is highest than the accuracy of the other methods.

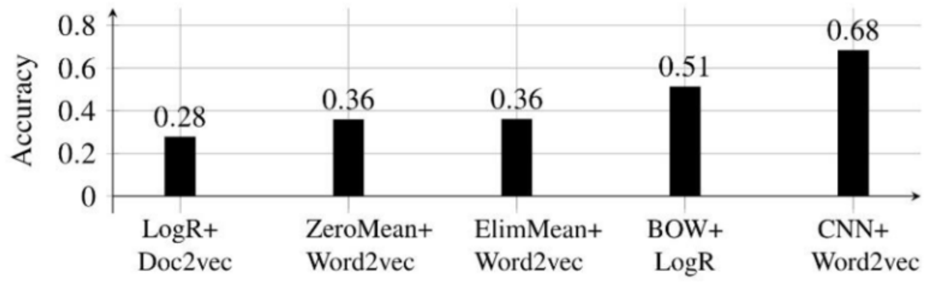

Figure 8: Classification performance, taken from the paper Medical Text Classification Using Colvolutional Neural Networks [11]

6 Implementation

All the project steps were implemented in Python. I used Numpy, NLTK and Keras' utils for preprocessing the data, Keras for building the convolutional neural network and MatPlotLib to show the results. I illustrate how the implementation works in the Python notebook `Project65-Miner-Notebook.ipynb`. Inside of the compressed file that I submitted, there should be three folders: `data` with the train and test data, `embeddings` with the pre-trained word embedding models or vectors and `models` with the trained models in this project (in order to avoid training them again). There are also two files named `history.pkl` and `history_10.pkl` which contain the results of the training process, used to plot the results. This data can be loaded following the notebook's instructions. The code of this notebook is a modified version of the code shown in the project Text classification with Convolution Neural Networks [12].

Important: the embeddings, data and models folders will be delivered empty, since the size of the files surpass the limit of the maximum 10MB size, the link to download word embeddings the files can be found in the notebook and the rest of the files can be downloaded from my GitHub repository: https://github.com/julenminer/mlnn_P65.git.

References

[1] Source of the data set: http://qwone.com/%7Ejason/20Newsgroups/

[2] Lozhnikov, N.; Derczynski, L.; and Mazzara, M; 2018. Stance Prediction for Russian: Data and Analysis. arXiv preprint arXiv:1809.01574.

[3] GloVe: Global Vectors for Word Representation, official web page: https://nlp.stanford.edu/projects/glove/

[4] Jeffrey Pennington, Richard Socher and Christopher D. Manni; 2014. Glove: Global Vectors for Word Representation. http://www.aclweb.org/anthology/D14-1162

[5] Download and usage page for Google's trained Word2Vec model in Python http://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/

[6] Source of the pre-trained word vectors by FastText: https://fasttext.cc/docs/en/english-vectors.html.
Mikolov, Tomas and Grave, Edouard and Bojanowski, Piotr and Puhrsch, Christian and Joulin, Armand; 2018. Advances in Pre-Training Distributed Word Representations. Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)

[7] Li, Haoxiang and Lin, Zhe and Shen, Xiaohui and Brandt, Jonathan and Hua, Gang; 2015. A Convolutional Neural Network Cascade for Face Detection. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 5325-5334

[8] Yoon Kim; 2014. Convolutional Neural Networks for Sentence Classification. arXiv preprint arXiv:1408.5882.

[9] Gehrmann, Sebastian & Dernoncourt, Franck & Li, Yeran & T. Carlson, Eric & T. Wu, Joy & Welt, Jonathan & Foote, John & T. Moseley, Edward & W. Grant, David & Tyler, Patrick & Anthony G. Celi, Leo. (2018). Comparing deep learning and concept extraction based methods for patient phenotyping from clinical narratives. PLOS ONE. 13. e0192360. 10.1371/journal.pone.0192360.

[10] Diederik P. Kingma and Jimmy Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980 (2014).

[11] Hughes M, Li I, Kotoulas S, Suzumura T. Medical text classification using convolutional neural networks. Stud Health Technol Inform. 2017;235:246250. [PubMed]

[12] Link to the GitHub code of the project that I modified in order to make this project: https://github.com/cmasch/cnn-text-classification