A Transformer-Based Model to Classify Medical Abstracts by Patient Condition

Sanjana Krishna[1], Deepti Murthy[2]
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
[1]sanjanak@cmu.edu, [2]dmurthy@cmu.edu

1 Introduction

Due to the dramatic increase in the generation of medical literature in recent years, driven both by rapid advancements in the medical field and a diversification in the types of studies being conducted, there is a growing need for the efficient and accurate categorization of papers relating to specific fields. The use of machine learning methods for the classification of medical documents has become popular in recent years due to their impressive performance on real world medical data, which is often convoluted with repetitive information and noise, or conversely, a lack of information. Therefore, to better enable medical professionals and researchers to screen for studies relevant to their fields, we propose an optimized transformer classification model with the ability to categorize medical abstracts by patient condition. For the problem of text classification, we employed a LSTM (long short term memory) which is a type of Recurrent Neural Network (RNN) as our baseline model due to its ability to process sequentially-structured data and the plethora of recent publications detailing the success of RNN-based models on scientific texts. To improve baseline performance, we experiment with transformer models with various word embedding methods, methods to encode word similarity, transformer architectures, and hyper-parameter settings.

Our models are trained on abstracts of studies focusing on common patient conditions and aim to predict the patient condition associated with abstracts in testing data of the same type. Specifically, we utilize the "Medical-Abstracts-TC-Corpus", which contains an average of 2300 training and 600 testing abstracts across 5 different patient conditions: "neoplasms", "digestive system diseases", "nervous system diseases", "cardiovascular diseases", and "general pathologies". These abstracts contain on average 212 words, varying from commonly used words in English, such as "characteristics", "difference", and "between", to complex medical terminology, such as "malignant", "beta" "D" "galactopyranosyl" "(" "1" "4" ")", and "pseudo" "obstruction". We generate a suitable vocabulary, use appropriate word embeddings for training the model, and find an optimal model architecture such that combinations of these complex medical terminology are efficiently used for accurate abstract categorization. We discuss our results in the following sections, evaluating performance using loss and accuracy plots and confusion matrices. Our best performing model was a transformer with one layer and one attention head, using pre-trained medical embeddings.

2 Background

For our baseline model, we generated a vocabulary from the "Medical-Abstracts-TC-Corpus" dataset and implemented an RNN/LSTM via PyTorch to classify the testing dataset. We performed dropout (0.2) on the LSTM's hidden outputs and they were passed to a linear layer and softmax of length 5 (the number of classes), and softmax cross entropy was used. We aimed to create a vocabulary specific to the words seen in the medical data, as this would allow more efficient training and higher overall performance. To generate our vocabulary, we created a unique token for each unique word in

10417 Class Project.

the training dataset, first by tokenizing each sample by splitting by "splitting characters" and spaces, and then assigning each unique new string to a unique random integer in the range of the number of unique strings. We then prepared our data for training by iterating through every sample in our training dataset, tokenizing each sample, and then setting each sample to the average sample length; if the sample had fewer words than the average, we repeated and concatenated the sample text until necessary, and if it was greater, we simply cropped. This is because the pytorch RNN layer required the inputs to be of the same length. For example if the average sequence length was 40, we would want all sequences to have that length, since that is a hyper-parameter of the RNN, so sometimes we had to cut the sequences or concatenate shorter sequences together to reach the sequence size. We also ensured the dataset was balanced such that there were 1100 training examples in each of the 5 classes. The first architecture we tried had an RNN instead of RNN-LSTM and only used the last hidden output and no dropout, and resulted in .2 test accuracy (which was at chance). This was flawed because it only used information from the last hidden layer. So, we tried a new architecture: tokenzing –> Embedding –> LSTM –> Linear layer with all hidden layers of RNN –> Softmax. The accuracy and loss on the training and testing data from the "Medical-Abstracts-TC-Corpus" dataset are displayed in Figures 1 and 2.

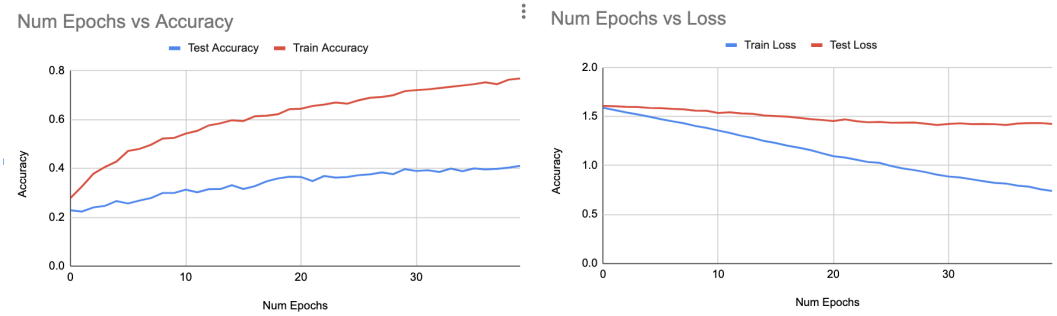

Figure 1 & Figure 2

We used a batch size of 64, learning rate of .01, RNN sequence length of 30, RNN embedding dimension of 400, and hidden dimension of 128. The baseline reached test accuracy of 41% and loss decreased from 1.6 to 1.42 before plateauing, which is better than chance. The train loss and accuracy were improving, such that after just 40 epochs, the train accuracy reached .72 and loss decreased from 1.6 to .77. This indicates that the current model is likely too over-fitted, since the test and train losses/accuracies are very different.

3 Related Work

The paper we chose for inspiration is “Medical Text Classification Using Hybrid Deep Learning Models with Multihead Attention” by Prabhakar et al, published in Hindawi in August of 2021. The authors’ goal was to classify text documents consisting of three categories of medical diseases. This is very similar to our problem in which we aim to classify research paper abstracts with medical terminology into five classes of patient conditions. This paper used GloVe embeddings instead of the regular learned weights (via torch.nn.embeddings). They also used an LSTM; according to the paper’s discussion, it prevents the vanishing gradient problem and it allows for prior history of words - hidden embedding and the input embedding - to be weighted more or less which is not possible in the RNN. They had four blocks of LSTM outputs, so that each block could learn a different set of features from the input text. They also used attention (learned Q, V, and W) to better allow for certain words to depend on previous words, and used softmax to predict the class. This proposed model worked very well on this task, reaching a maximum of 96% test accuracy, but the authors’ did try many other experiments (across learning rates .01, .001, .0001, different optimizers such as SGD and Adam optimizers), as well as used a CNN-based approach in a hybrid model to enhance learning. Overall, this paper’s interesting use of an optimized LSTM model for text classification motivates our desire to apply a similar approach for classifying medical abstracts in our project.

4 Methods

Our baseline model was an LSTM, however, given the advances in transformer architectures, we explored using a transformer model to improve upon this baseline. We also implemented strategies including padding our training sentences, using different types of word embeddings, and incorporating similarity calculations to improve learning.

To improve the baseline, we implemented padding (to mask out words for which classification of medical text was difficult, for a few reasons. One reason is that the medical text sentences in our dataset are of variable length - in fact the number of words in our training sentences (as determined by the tokenizer we used) ranges from 30 to 300 words, and the average length is 212 words. For our previous LSTM model, we limited the word length to 40, since we did not use padding, limiting the model's ability to learn useful information beyond 40 words. Further, medical abstracts are littered with words that are not commonly found within other disciplines; medical terminology is complex, and as such, we found that several words in our training and testing abstracts are unique to the abstract it is contained in. Further, although medical words contain suffixes and prefixes that are common to other words, the combinations of these are limitless. For example test words like "tachydysrhythmia" and "xanthogranulomatosis" in the testing abstracts are unlikely to be found in the set of words that were trained on. So, if we were to self-learn embeddings in our model, all of the words in the testing set that did not appear in the training set would have to be replaced using our padding method. This is unfortunate, given that approximately 30 percent of the testing words do not appear in the words trained on. Therefore, we decided to creatively tackle this problem in a few ways.

First, we attempted to use pre-learned embeddings of the words in both the training and testing sets using an online word embedding tool (a tool that maps medical words to pre-made 300-length embeddings). Specifically, this is a pre-made set of 300-dimensional vector embeddings for words that are commonly found in clinical texts. Therefore, in this case, we did not need to add an embedding layer in our model's neural network. To implement this strategy, we used the clinical embeddings for all words in training and testing that had a mapping, and otherwise if the word was not mapped to, we used a random 300-dimensional vector via our padding method for the embedding. The tool for clinical embeddings we used can be accessed via the following link: `https://github.com/gweissman/clinical_embeddings`.Creating good pre-learned embeddings can be done a few ways. Skip-gram and CBOW are techniques employed by Word2Vec models. Skip-Gram models' learning objective is to maximize the probability of filling in a word based on the surrounding N grams. GloVe Embeddings learn embeddings by finding the frequency that certain sets of words appear together and match their embeddings, hence encoding semantics within the embeddings. We picked the dimension 300 Word2Vec trained embeddings, as they were recommended by the site (Flamholz et al).

Next, we attempted to self-learn the word embeddings by implementing an embedding layer in our model's neural network. Without any additional improvements, the self-learned embedding model works the same way as the medical embeddings: if the testing word is not in the training word dictionary it is replaced using padding, otherwise the self-learned embedding is used. We hypothesized that self-learning the embeddings on its own would not perform any better or as well as our method of using pre-learned embeddings, since as previously mentioned, 30% of the testing words were not seen in the training data due to the uniqueness of medical terminology. Thus, we decided to implement a similarity calculation between testing and training words such that we could replace unseen testing words with their most-similar word in the training dataset.

The first method of encoding similarity was using a built in function that came with the glove vectors medical embeddings:

```
modelemb.wv.most_similar(word)
```

that outputs the top 10 most similar words to the testing word in the medical embedding lexicon. For those 30% of unseen test words, we looped through each of the 10 matches and identified the first word that was also in the training set, and replaced the test word with this similar word. According to online documentation, this medical embeddings function uses a cosine similarity function across the embeddings to find the top 10 matches.

An issue with this built-in function is that at least one of the top 10 most similar words found for each testing word had to also be present in the training dataset. This was not always the case, so some testing words both did not have a medical embedding and did not have a usable most-similar word to replace it. Thus, these words had to be replaced using the padding method. Specifically, of the 30% of testing words that were not seen in the training data, only 37% of these words had at least one word in the top 10 most similar words as calculated by the pre-built function that existed in the training data. Thus, using this pre-built function for finding similar words allowed only 79% of testing words to avoid being replaced by padding, which is only slightly better than 70 % previously.

To counter this issue, we creatively implemented our own method of calculating the most similar training word for each testing word. To do this, prior to training, we got an embedding of each word in training and testing to calculate the most similar word in training to each testing word. We calculated the dot-product of the embedding of each testing word with each training word (to calculate the similarity between these words), and then generated a matrix containing the similarity scores of the words in the training dataset to each testing word. Then, in our model, whenever a testing word was not present in the training dataset, we replaced the word with the most similar word from the testing dataset using this pre-made matrix. Approximately 70% of testing words were already present in the training dataset. Of the remaining 30%, 70% of these words had a mapped medical embedding, allowing us to compute the embedding dot product. Therefore, using this matrix method, we increased the total number of non-padded testing words to approximately 91%. The dot products were in a range from 2 to 20, which is not very small and therefore in a good range to aid learning.

Further, we made additional changes to improve performance. First, we increased the amount of information trained on, as compared with our LSTM model. To do this, we implemented padding, so that we could increase the number of words (the sequence length used for training and testing) to 50 instead of 40 or 30. Padding is important because the weights of the padded words or tokens will not be updated, which is important, since we don't want training on random tokens/embeddings. Thus, we set the fixed word length to 50 words whenever a training or testing sentence had less than 50 words, we used padding to extend the sentence length. Additionally, in our previous LSTM implementation, we had used a basic English language tokenizer, which only split the sentences on spaces. So, when we changed the tokenizer to also split on "-" dashes, we found that 85 % of words in the training and testing datasets had pre-made embeddings vs. the 75 % using the original basic-English tokenizer. This makes sense, because there is unlikely to be a pre-made medical embedding for words like 'parkinsons-disease', but there is likely to be a medical embedding for 'parkinsons' and 'disease'. Many terminology specific to medical texts include dashes, so using improving our tokenization method this way improved model performance.

The architecture and hyper-parameters of our transformer models consisted of positional encoding, 0% dropout rate, transformer encoder layers with 0.3 dropout rate, a variable number attention heads (1-3), and variable number of network layers. This layer was followed by a flattening layer (batch size, sequence length, embedding dim) —> (batch size, seq length*embedding dim) followed by a dropout layer of rate 0.3. We then passed the vector through a linear layer with 5 output nodes corresponding to each of the 5 classes for classification and outputted the argmax of the softmax. The model was trained via softmax cross entropy loss.

For training with pre-trained medical embeddings, we used the Adam optimizer and a learning rate of 1e-3, which we found to be optimal via hyper-parameter tuning. Using the fixed learning rate was also optimal as opposed to using a variable learning rate, which took many more epochs to converge and lacked an increased in overall performance. However, when using self-learned embeddings, we used the Adam optimizer, with a variable learning rate - such that the step size followed an Exponential decay with gamma = .9.

To optimize our transformer model architecture, we varied the sequence length limit, the number of layers, and the number of attention heads. We tested using 1-3 layers, 1-3 attention heads, and 50-70 word lengths (increasing by 10). The following are the results for the following combinations: (1 layer, 1 attention head), (1 layer, 2 attention heads), (1 layer, 3 attention heads), and (2 layers, 3 attention heads). The model was run until train loss started increasing back up. For the selected architectures below, we noted the following train accuracy, train loss, test accuracy, test loss (left to right) where the test accuracy was best:
2 layer 3 Heads (50 word length): 1.724 .203 1.695 .223 (at 2 epochs)
1 Layer 3 Heads (50 word length): 0.895 0.663 1.107 0.568 (at 4 epochs)

1 layer 2 heads (50 word length): 0.789 0.701 1.038 0.582 (at 11 epochs)
1 layer 1 head (70 word length): 0.931 0.653 1.114 0.565 (at 5 epochs)
1 layer 1 head (50 word length): 0.802 0.706 1.096 0.596 (at 12 epochs)

Our best performing model was a 1 layer, 1 head transformer: more attention heads and layers likely resulted in over-fitting preventing generalization to the testing set. A word length of 50 was found to perform the best on testing data, better than 70 for a few reasons, even though we used padding. While the gradients are not back-propagated with padding, the padding token embeddings are still used in the forward pass to calculate the resulting class - when the word length was 70, many of the sentences likely had more padding tokens causing more noise (hence the accuracy went up by 3 % when the word length was decreased back to 50).

5 Results

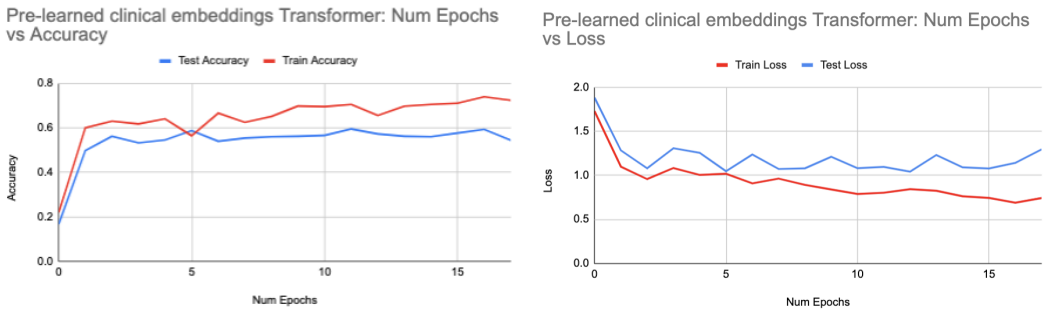

Figure 3a: Pre-learned Medical Embeddings: Best Test Accuracy: 59.6%, Best Test Loss: 1.096

```
[[304  18  86   9  20]
 [ 25 137  23   4  19]
 [ 11   3 244  16   3]
 [  4   3  92 311  11]
 [ 95  83 281 148  50]]
      Rows: True Class, Columns: Predicted Class
Classes In Order: neoplasms, digestive, nervous system, cardiovascular, general pathologies
```

Figure 3b: Representative Confusion Matrix Plot (from model of Pre-learned Medical Embeddings)

```
Predicted Label: Cardiovascular
True Label: General pathology
 Text: [...The aberrant right coronary artery was thought to be a possible cause of the
 cardiopulmonary arrest in this patient. Both lesions were corrected at a single operation....]
```

Figure 3c: Example of Mistake by Model)

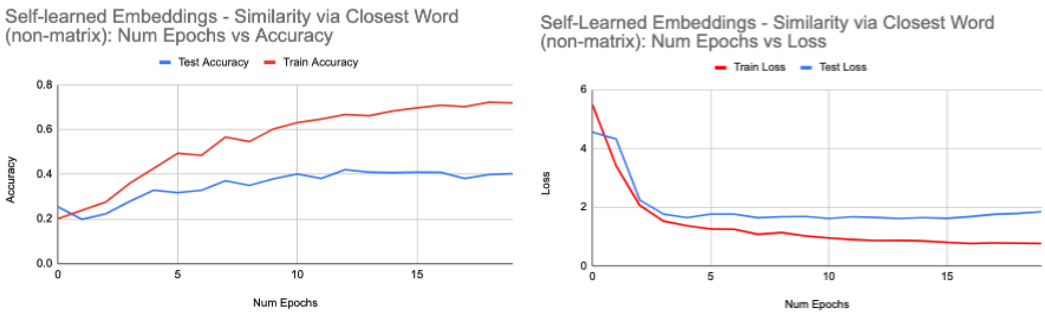

Figure 4a: Self-learned Embeddings via Built-In Fcn: Best Test Accuracy: 42.05%, Best Test Loss: 1.61

```
Predicted Label: Nervous System
True Label: Cardiovascular
Text: ['efficacy' 'of' 'propafenone' 'in' 'preventing' 'ventricular' 'tachycardia'
'inverse' 'correlation' 'with' 'rate' 'related' 'prolongation' 'of' 'conduction'
'time' 'the' 'efficacy' 'of' 'propafenone' 'in' 'preventing' 'induction' 'of' ...]

Predicted Label: Neoplasm
True Label: Cardiovascular
Text: ['a' '47' 'kda' 'human' 'nuclear' 'protein' 'recognized' 'by' 'padding' 'autoimmune'
'sera' 'is' 'homologous' 'with' 'the' 'protein' 'encoded' 'by' 'padding' ',' 'a' 'gene'
'implicated' 'in' 'onset' 'of' 'chromosome' 'multipolar' '.' '...]
```

Figure 4b: Example of Mistake by Model

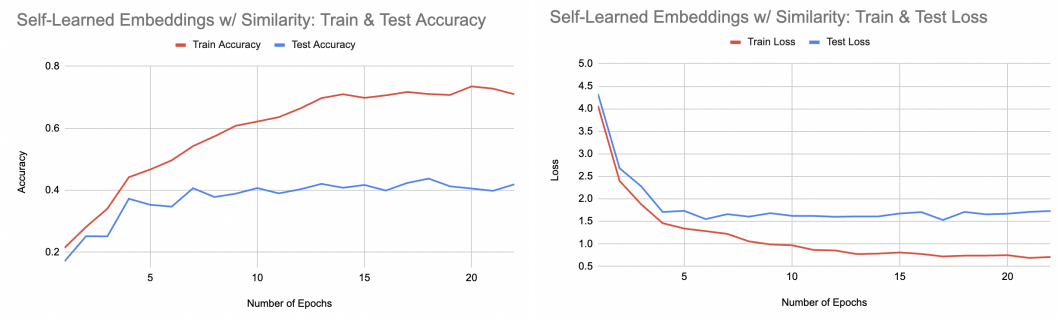

Figure 5: Self-learned Embeddings via Matrix Method: Best Test Accuracy: 43.75%, Best Test Loss: 1.53

Figure 6: Self-learned Embeddings without Similarity Encoding: Best Test Accuracy: 42.0%, Best Test Loss: 1.62

First, with medical embeddings, as observed in Figure 3a, the transformer reaches a training accuracy of 74 % and a testing accuracy of 59.6%. The train loss was 0.68 and test loss reached 1.096. From our baseline LSTM (with learned embeddings) performance of 72% training accuracy and 41% testing accuracy, this model had an 18.5% increase in test accuracy. Compared to the 0.77 train loss and 1.42 test loss in the baseline, there was a slight improvement in the loss with the transformer. We wanted to see if this was because of the type of embeddings used (pre-learned medical or learned) or because of the model architecture (LSTM vs transformer), so, we used the medical embeddings for the original LSTM, and the best test loss was 1.15, and test accuracy 53 % , which was more comparable to the transformer with medical embeddings. However, the transformer still performed slightly better overall (59.6 % test accuracy > 53 %). The improvement was therefore from both the new medical embeddings and transformer architecture used. Transformers avoid the vanishing

gradient problem that LSTM are still subject too and can do global as opposed to solely sequential attention.

After experimenting with pre-determined clinical embeddings, we opted to try to improve performance by learning our own embeddings. Here, we first tried learning our own embeddings without any additional modifications to the test data. As observed in Figure 6, without using the previously discussed similarity approaches to replace test words, the transformer reached a training accuracy of 70% and a testing accuracy of 40%. This is neither an improvement from our baseline LSTM model nor is it competitive with our transformer model using medical embeddings. In terms of loss, our model with self-learned embeddings reached a minimum train loss of 0.82 and minimum test loss of 1.62. Again, there is no improvement in loss from our baseline model.

As displayed in Figure 4, when replacing the test words with the most similar word from the training dataset using the built-in function, training accuracy reached 72.2 % and the testing accuracy reached 42.05%. This is a slight (1.05%) improvement from the test accuracy via our baseline LSTM model and a 2.05% improvement from the test accuracy of the transformer using pre-trained embeddings without replacing similar test words. However, this model is still less successful than our transformer model that uses medical embeddings instead of pre-trained embeddings. In terms of loss, the minimum training loss of this model was 0.761 and minimum test loss was 1.61. There is no improvement in loss from our baseline model and the loss is seemingly on par with that of the transformer using pre-trained embeddings without replacing similar test words.

As observed in Figure 5, when replacing the test words with the most similar word from the training dataset using the matrix method, the training accuracy reaches 80% and the testing accuracy reaches 43.75%. This is the best test accuracy achieved by all our transformer models that use pre-trained embeddings (2.75% improvement from the LSTM baseline), but the model still performs worse than our transformer implementation using pre-trained medical embeddings. In terms of loss, this model's minimum training loss was 0.709 and minimum test loss was 1.53. This is again not an improvement in test loss from the baseline LSTM, but is improved loss as compared to all other transformer approaches with learned embeddings.

We expected that using the medical embeddings would improve performance as compared to using regular English word embeddings due to the uniqueness of medical terminology, which they did. We also hypothesized that self-learned embeddings would further improved knowledge learned, however the performance of our models indicates otherwise. The explanation for this is provided in the discussion.

Metrics used for evaluation include training and test loss and training and test accuracy, as well as confusion matrices to gain a deeper understanding of the misclassification of abstracts. As observed in Figure 3b, the "General Pathologies" class contained the most misclassified cases. This was true across all model implementations, and is likely due to the fact that abstracts in this general class span various fields, and thus include terminology and subject matter that are included in the other classes. For example, in Figure 3c, we observe an example where text from the "General Pathologies" class was mislabeled as the "Cardiovascular" class. The text includes phrases like "coronary artery" and "cardiopulmonary arrest" and it's subject matter falls under cardiovascular disease, so it makes sense that this case was misclassified in the way that it was. This is the case for many of the test abstracts that were misclassified as "General Pathologies."

We identified another paper "Evaluating Unsupervised Text Classification: Zero-shot and Similarity-based Approaches" from December 2022, that classified the same dataset we used. They used two methods. For similarity based approaches, embeddings of the words and texts were trained to semantically match the words/texts of those in the same class using a similar approach to Word2Vec training. Then, the label would be the label of the texts that had the highest cosine similarity with the sample text. In zero-shot classification, a transformer was trained to output the class label and could do so on completely new test samples, possibly generating new classes. They found the highest accuracy/f1-score of class labels resulted from their zero-shot model and was 57.28 % (Schopf et al). So, compared to this prior work, our maximum accuracy of the model for pre-learned medical embeddings of 59.6 % performed more than 2 % better.

6 Discussion and Analysis

Overall, our best performing model was the transformer with one layer and one attention head that used the pre-trained medical embeddings. We achieved higher test accuracy (59.6 %) than our baseline LSTM with this implementation and similar performance to a model in literature (Schopf et al), as previously discussed. The medical clinical embeddings were trained using 18 million Wikipedia articles, 276,000 MIMIC clinic notes, and 27,575 PubMed Open Access Case Reports, so it makes sense that these embeddings are representative. That is why we decided not to use Word2Vec (general English word embeddings) to create pre-trained embeddings.

We gained insight into the transformer architecture by experimenting with different features. In our text classification, dropout (which helps generalize the training model) and positional encoding were very important - no text could be thought of as a bag of words even if the sentences were simple. Further, it makes sense that using one attention head and one layer performed best because we observed significant over-fitting to the training data across all models. However, this means also that the model cannot attend to other information that would lead to a more complex understanding of the class (it can only make correlations between words and class). For instance in Figure 3c, the model likely could only learn the correlation of cardiovascular words with that class, but if it also attended to "cardiopulmonary" it would identify that the class is a general pathological disease. However, using more heads and layers over-fitted to the training data due to the lack of diverse data, preventing us from using more complex models. However, if there was more training data, a more complex model may be able to learn such distinctions. Also, the example in Figure 3c shows, sometimes the true labels can be contradictory or subjective, restricting the models' ability to generalize correctly.

From our models, self-learned embeddings was not able to improve performance as compared to pre-trained embeddings. We used two approaches to replace unseen test words with words in the training set (since only the training words have trainable embeddings). We hypothesized that self-training would be optimal, since the learned embeddings would be more task relevant that the pre-learned embeddings. However, this was not the case - they were too over-fitted on the training data. The matrix similarity method test accuracy peaked at 43.75 % and the built in similarity function peaked at 42.05 % test accuracy. The matrix similarity approach finds the word in the training set with the highest dot product of embedding to the sample word. However, in many cases the argmax was not semantically related to the target test word - for example the test word - "fractional" was replaced by "lower" and "diastole" (heart rate term) was replaced by "glioblastomas" (cancer term). Yet, this model performed best out of the learned-embedding methods, likely because many of the test words could be replaced with actual training words - these words are likely more useful to the model's understanding than to that of humans, assisting in classification. Interestingly, the pre-built function for similarity resulted in replacing unseen test words with more semantically-related words, but replaced fewer of the testing words - hence it only reached an accuracy of 42 %. For example: "hypoapneas" with "apneas", "phagocytic" with "phagocytes", "skilled" with "trainees", etc. These methods did not perform much better than if the unseen test words were not replaced. This finding suggests that the learned embeddings are over-fit to the train data, and therefore unable to generalize to new contexts - possibly due to the small training batch of 5500 examples. For example in testing, if a training sentence was "Neurological *disorders* affect people's brains" (with class nervous system disease), and a test example was "*Disorders* affecting the arteriole ...", it may be misclassified as a Neurological disease, since "disorders" was previously paired with Neurological disease (due to the small and non-diverse training set).

Self-learned embeddings did, however, allow a small increase in success when using similarity techniques to replace unseen words. This is because there were over 900-2000 words in testing (out of the 10000 testing words) that weren't trained on, which is better than the 3000 words that weren't be replaced in testing previously. A limitation of all our models is that the sequence length used was 50 with some words replaced by padding, while the average abstract length was 212, further contributing to over-fitting and lack of extra context to learn successfully.

In the future, it may be possible to implement a more sophisticated vocabulary that includes prefixes and suffixes of training words, takes into account word frequency across the data, and potentially uses a set of words sourced from other open-access medical text databases. This enhanced vocabulary will enable better identification of similar words in the training data for each testing word. We also may attempt to expand the training set to include more than 5500 examples which may allow us to learn better embeddings that are not over-fitted to the training data.

References

Flamholz, Zachary N., et al. Word Embeddings Trained on Published Case Reports Are Lightweight, Effective for Clinical Tasks, and Free of Protected Health Information. Cold Spring Harbor Laboratory, 4 Dec. 2019, http://dx.doi.org/10.1101/19013268. Accessed 6 Dec. 2023. (Embeddings citation)

Pandey, Sandeep. "Text Classification Using Custom Data and PyTorch - Sandeep Pandey." Medium, 17 May 2023, https://medium.com/@spandey8312/text-classification-using-custom-data-and-pytorch-d88ba1087045.

Prabhakar, Sunil Kumar, and Dong-Ok Won. "Medical Text Classification Using Hybrid Deep Learning Models with Multihead Attention." Computational Intelligence and Neuroscience, vol. 2021, Sept. 2021, doi:https://doi.org/10.1155/2021/9425655.

Sebischair. "Medical-Abstracts-TC-Corpus/README.Md at Main · Sebischair/Medical-Abstracts-TC-Corpus." GitHub, https://github.com/sebischair/Medical-Abstracts-TC-Corpus/blob/main/README.md. Accessed 30 Oct. 2023.

Schopf, Tim. "Evaluating Unsupervised Text Classification: Zero-Shot and Similarity-Based Approaches." ACM Other Conferences, Dec. 2022, https://dl.acm.org/doi/10.1145/3582768.3582795.

Schopf, Tim and Braun, Daniel and Matthes, Florian, Evaluating Unsupervised Text Classification: Zero-Shot and Similarity-Based Approaches, 2023, https://doi.org/10.1145/3582768.3582795, 10.1145/3582768.3582795, (Dataset citation)

Soma, Basu. "17 Best Text Classification Datasets for Machine Learning." iMerit, 16 July 2021, https://imerit.net/blog/17-best-text-classification-datasets-for-machine-learning-all-pbm/.

Torchtext.Models — Torchtext 0.16 Documentation. https://pytorch.org/text/stable/models.html.

Wherll. "How to Use Datasets and DataLoader in PyTorch for Custom Text Data." Towards Data Science, 14 May 2021, https://towardsdatascience.com/how-to-use-datasets-and-dataloader-in-pytorch-for-custom-text-data-270eed7f7c00.