Handling Rare Words in Medical Texts

Brenton Arnaboldi
ba1303@nyu.edu

Yuqiong Li
yl5090@nyu.edu

Abstract

Medical texts may have high numbers of rare words, as many documented diseases appear only in a tiny share of the population. Therefore, for natural language processing (NLP) tasks on medical text, using a simple fixed vocabulary may not be the best approach because rare (but important) medical terms might frequently fall outside the vocabulary. To deal with out-of-vocabulary (OOV) words, we experiment with a variety of pre-processing techniques, including byte pair encoding. Using publicly available data from MIMIC II, we then compare the effectiveness of our different pre-processing methods by predicting the ICD codes associated with a discharge summary (a physicians note describing the patients symptoms). Our main model is a Gated Recurrent Unit (GRU) with hierarchical attention over words and sentences.

1 Introduction

For Electronic Health Records (EHR), physicians need to assign a five-digit code for each symptom observed in a patient. The International Classification of Diseases (ICD) is a standardized list of codes for classifying diseases. These codes are often used for billing and reporting purposes, so assigning accurate ICD codes after a patients clinical visit is critical. Unfortunately, this labeling task requires expert medical knowledge, and the process is expensive and prone to errors. The approximate cost of ICD coding and error correcting is roughly $25 billion per year in the United States (Farkas and Szarvas, 2008). ICD9 codes are also used to determine the eligibility of patients for clinical trials. Automating the assignment of ICD codes from medical text, therefore, is an extremely important endeavor.

Predicting ICD9 codes from discharge summaries is a challenging task for multiple reasons. First, the label space is very large; there are over 5000 unique five-digit ICD-9 codes in the MIMIC II dataset. These codes can be rolled up to 970 three-digit codes, but the label space remains big even with this contraction. Second, the discharge summaries (input texts) are extremely long; the average length of a patient note is 2,018 tokens. At a minimum, we needed a model that can handle long-range dependencies. However, since the discharge summaries are so long, training a regular GRU would be too slow. As such, we implemented a Hierarchical-Attention GRU model (HA-GRU) from Baumel et al. (2014), who used the model for the same prediction task on MIMIC ICD-9 codes. The model contains two GRUs: one operates on words to encode sentences, while the other operates on sentences to encode documents. This structure allowed us to train the model more quickly.

In addition, running NLP tasks on medical corpora can be challenging because of the plethora of rare diseases that exist. Many NLP models rely on a fixed vocabulary consisting of words that appear at least X times in the corpus. However, for medical documents, a fixed vocabulary approach might be problematic because it might exclude rare conditions. These rare conditions might be important to the prediction task, even if they dont appear frequently in text. Because a fixed vocabulary was not ideal, we adopted an open dictionary approach, using Byte Pair Encoding to split rare words into subword units (Sennrich et al., 2016). We thought this method would be useful for medical text because similar diseases often share the same roots, and these roots can be used to identify diseases. For example, consider the condition afibrinogenemia. Even if the word appears rarely in text, under byte pair encoding the model can read this term as a—fibrinogen—emia. Fibrinogen is a protein involved in the formation of clots, while the suffix emia refers to blood, so the model can quickly pick up an accurate representation of afibrinogenemia (disease related to blood clots).

Finally, the MIMIC II discharge summaries had many spelling errors and typos. Baumel et al. (2014) address this issue by mapping every OOV word to its nearest vocabulary word based on Levenshtein edit distance. Using edit distance to alter OOV words is productive for some terms but detrimental for other terms. We tried experimenting with a hybrid approach where OOV words just 1 edit operation away from the closest vocabulary word were corrected, while applying Byte Pair Encoding to the remaining rare terms.

2 Background and Literature Review

2.1 Previous Work: ICD Coding

Given the importance of assigning correct ICD9 codes to patients, many studies have proposed methods to automate this task. Previous work has explored the use of rule-based systems and supervised machine learning methods to predict ICD9 codes. Researchers have only just begun to use neural networks for this task. Rule based systems, such as Goldstein et al. (2007), have consisted mainly of hand-crafted rules to capture lexical elements (e.g. short, meaningful sets of words). Others have tried simple machine learning techniques. Larkey and Croft (1995) used a combination of K-nearest-neighbor and Bayesian classifiers to predict ICD9 codes, while Lita et. (2008) and Perotte et. (2014) used ridge regression and support vector machines (SVM). In general, rule based methods have usually outperformed machine learning classifiers, although comparisons between experiments are tricky because different studies may not use the same subset of ICD9 codes.

Neural networks have only recently been applied for this problem. Deep learning has the potential to make this task far more efficient by eliminating the need to describe explicit features or rules (as required in traditional machine learning and rule-based systems). Ayyar (2017) used a simple Long Short Term Memory (LSTM) network with pre-trained word vectors from GloVe (Common Crawl, 840 billion tokens). Shi, Xie, et al. (2017) used both character-level and word-level LSTMs to encode the discharge summaries and the titles associated with each ICD9 code. They then applied an attention mechanism to link the hidden representations of the discharge summaries and the code titles. Our main reference paper (Baumel et. 2017) tested four different models: 1) support vector machines (SVMs); 2) continuous-bag-of-words (CBOW); 3) convolutional neural network (CNN); and 4) the Hierarchical-Attention GRU model HA-GRU). The authors used the entire set of ICD9 labels, and found that the HA-GRU was the highest-performing model. The HA-GRU achieved an F-score of 0.54, while the other models ranged from 0.33 to 0.46. These results are not high enough yet to apply the model to a real-world setting, but they demonstrate a big improvement over traditional methods.

2.2 Previous Work: Handling Out-of-Vocabulary Words

Much of the existing literature on handling out-of-vocabulary words comes from neural machine translation (NMT). Most traditional NMT models use a fixed vocabulary, but the architecture of NMT models generally do not scale well to large vocabularies. To circumvent this issue, Jean et. (2015) proposed an importance sampling method that takes subsets of a larger target vocabulary. Because language-to-language translation requires an enormous fixed vocabulary, another efficient approach is to create an open vocabulary that decomposes words into character blocks. Many recent studies (such as Lee et., 2017) have examined the effectiveness of character-level models in machine translation. Meanwhile, Chitnis and DeNero (2015) use Huffman encoding to represent rare words as a pair of two symbols. Sennrich et al. (2016) use a byte pair encoding algorithm to represent rare words as subword units.

Byte Pair Encoding (BPE) is a data compression technique that replaces the most frequent pair of consecutive bytes in a sequence with a single unused byte (Gage, 1994). Shibata et al. (1999) used byte-pair encoding to compress text files and found that pattern matching (finding patterns in text) was more efficient under this method. However, to our knowledge, Sennrich et al. (2016) was the first to apply byte pair encoding for a natural language processing task. The authors use BPE to segment rare words into variable-length character blocks or subword units. Instead of replacing frequent pairs of bytes, the authors merge frequent pairs of characters or character blocks. Our paper attempted to follow their process.

3 Preprocessing Methods

3.1 Representations of Input Text

Given that Baumel et al. (2017) is our primary reference for the HA-GRU model, we started off by trying similar preprocessing steps. We tokenized the input text (using NLTK), turned all text to lowercase, and converted all numbers to the generic character *. Following Baumel et al., we constructed a vocabulary consisting of words that appeared at least 5 times in the training set corpus. This method resulted in a vocabulary of 38,119 unigram tokens. However, our training corpus had roughly 157,000 unique tokens. (There was also an additional 12000 unseen test set words). Out-of-vocabulary (OOV) words were therefore a major problem that we needed to resolve.

Ultimately, we built six different representations of the text to deal with out-of-vocabulary words:

- ’Regular Vocabulary’ version: pad unknown words with UNK token
- ’Levenshtein’ version: map each unknown token to its nearest word in the vocabulary (using edit distance)
- ’Byte Pair Encoding with 25K merges’: use a byte pair encoding algorithm (Sennrich, 2016) to map rare words to blocks of characters. The number of ”merges” is a hyper-parameter.
- ’Byte Pair Encoding with 10K merges’
- ’Byte Pair Encoding with 5K merges’
- ’Hybrid with 10K merges’: use edit distance to map some OOV words to nearest vocabulary word, then apply byte pair encoding for the remaining rare words.

3.2 Byte Pair Encoding Algorithm

The procedure starts by representing each word in the corpus as a sequence of characters, plus an end-of-word symbol ’/w’. (This symbol helps preserve the original tokenization of the vocabulary). Once words are divided into characters, we count all pairs of consecutive characters and find the most frequent pair (e.g., (’A’, ’T’)). We then replace each occurrence of this pair with a merged block (’AT’). Eventually, with enough iterations of merging characters, common words are cobbled back together while rare words are represented as blocks of multiple characters.

For example, consider the text: ”harry had had enough”. A Counter object on the tokenized text would yield (’had’: 2, ’harry’: 1, ’enough’: 1). Each word is split into characters: (h a d /w: 2, h a r r y /w: 1, e n o u g h /w: 1). Now, we find the most common pair of characters. We can see that (h, a) appears three times, while (a, d) and (d, /w) each appear twice. So we would merge ”h” and ”a” for each entry in the sequence above to get (ha d /w: 2, ha r r y /w: 1, e n o u g h /w: 1) after one merge operation. In this new configuration, (ha, d) and (d, /w) each appear twice. For the second merge operation, we can pick either option, and so if we decide to merge (d, /w) we would get (ha d/w: 2, ha r r y /w: 1, e n o u g h /w: 1).

The number of merge operations is the only hyper-parameter of the algorithm. In this case, where the number of merge operations = 2, the building blocks for our new open vocabulary would be (”ha”, ”d/w”, ”r”, ”y”, ”e”, ”n”, ”o”, ”u”, ”g”, ”h”, ”/w”). For our paper, we tried 5000, 10000, and 25000 merge operations on the original text (as well as 10000 merges under the hybrid approach we will discuss later). As the number of merge operations increases, the closer the text representation will be to the original text.

3.3 Advantages of Byte Pair Encoding vs. Levenshtein Distance

The Baumel paper, our key reference for the HA-GRU model, used edit distance to map unknown words to their nearest neighbor in the vocabulary. However, edit distance can be an overly simplistic approach to fixing OOV words, especially for long scientific terms common in medical corpora. For example, consider the rare word ”angiographically”. This word is an adverb closely related to the term angiography, a method of obtaining an X-ray of blood vessels. The edit distance between ”angiographically” and ”angiography” is 5. However, the edit distance between ”angiographically” and ”geographically” is only 3. In this case, we would com-

Preprocessing	**Vocab Size**	**Avg. Length**
Regular Vocab	38,119 words	2,018
Levenshtein	38,119 words	2,018
Byte25K	24,757 blocks	2,045
Byte10K	10,013 blocks	2,124
Byte5K	5,036 blocks	2,303
Hybrid10K	9,977 blocks	2,121

Table 1: Avg. Length is mean number of tokens in a summary

Preprocessing	**1 blk. words**	**+2 blk. words**
Regular Vocab	38,119	0
Levenshtein	38,119	0
Byte25K	15,692	153,742
Byte10K	5,796	163,638
Byte5K	2,666	166,768
Hybrid10K	5,661	104,656

Table 2: The original corpus includes 169,434 unique unigrams. 1 block words are unigrams that can be represented by one subword unit. +2 block words are unigrams that must be represented by multiple subword units. Note that the first two rows and final row do not add up to 169,434, as rare words were either ignored or mapped to a vocabulary word.

pletely distort the meaning of the word by mapping it to geographically. By contrast, Byte Pair encoding might segment the word as "angiograph"—"ically", which preserves the root "angiograph". Byte pair encoding can potentially do a good job of capturing common prefixes and suffixes found in medical terms. For example, the suffix "-itis" is used to describe inflammation (arthritis, hepatitis, sinusitis, etc.).

However, byte pair encoding might be less effective than Levenshtein distance mapping if the text has many spelling errors. The MIMIC discharge summaries in our corpus seemed to have many typos. Byte pair encoding might struggle to pick up misspellings. For example, consider the out-of-vocabulary token "ampuated". Using Levenshtein distance, this word is edited to amputated. However, under byte pair encoding, the misspelling distorts the key root "amputat" [amputation, amputating, amputate]. Instead, BPE may parse the token into "amp—u—ated". Even with attention applied over the character blocks, its hard to see how this representation would be as effective as simply correcting the spelling of the word.

Given these issues, our final preprocessing strategy was a "hybrid" approach between Levenshtein mapping and byte pair encoding. First, we took all the OOV words and determined whether their nearest neighbor in the vocabulary could be reached by only one insertion or deletion. This step removed 57,201 (43 percent) of 131,315 OOV words. To deal with the remaining rare words, we used byte pair encoding.

3.4 Impact of Preprocessing on Text

Our entire corpus contained 169,434 unique words, but 12,322 of the terms appeared only in the test set. Under the Regular Vocabulary and Levenshtein methods, we restricted our vocabulary to words that appeared at least five times in the training set, resulting in a vocabulary of 38,119 terms. The average summary length remained untouched at 2018 tokens, as shown in Table 1.

With Byte Pair Encoding, we ran the algorithm for a different numbers of merge operations, building a vocabulary of character blocks for each option (5K merges, 10K, 25K). Each character-block vocabulary includes a combination of 1) complete words and 2) subword units.

The size of this character block vocabulary increases almost perfectly linearly with the number of BPE merge operations. With zero BPE merges, a vocabulary of character blocks might be around 40 to 50; one entry for each letter of the alphabet, plus random punctuation. Generally speaking, each byte pair merge creates one additional character block (in rare cases, a merge operation might simultaneously delete one of the original characters, so the relationship between vocabulary size and BPE operations is not perfectly 1-to-1.)

For each vocabulary (25K, 10K, 5K), we found the number of our original words (the 169,000) that could be represented as 1) one character block or 2) multiple character blocks. For example, the 25K vocabulary had 15,692 character blocks as complete words, which means that 15,692 of the 169,000 original words could be represented with only a single character block. The other 153,000 words are mapped to sequences of multiple character blocks.

Interestingly, large changes in "Vocab Size" and "1 block words" only cause only small changes in average summary length (as measured by the relevant vocabulary units). Under the BPE methods, while a majority of unique words must be represented by multiple character blocks, the majority of words that actually appear in the text are represented by a single character block.

4 Hierarchical-Attention GRU model

We tested the Hierarchical-Attention GRU (HA-GRU) model on this task (Yang, 2016; Baumel 2017). This model's main characteristics are that 1) it uses a hierarchical model structure to encode at sentence level and document level seperately, and 2) it has two levels of attention mechanisms applied at the word and sentence-level. All data manipulation and model training was done in Python 3.6.0, Pytorch 0.2.0. One immediate challenge was that Baumel's model (the one used to predict ICD9 codes) used DyNet as opposed to PyTorch. We borrowed a PyTorch implementation from a GitHub repository from EdGENetworks titled "Hierarchical Attention Networks for Document Classification in PyTorch", which tries to emulate the model proposed in Yang, 2016, but transferring the model to a multi-label classification problem still required some work.

4.1 Model description

The specific architecture of HA-GRU are described below:

At the first level we encode every sentence in a document into a vector. This is achieved by first embedding words into word vectors, then applying a bi-directional GRU (word GRU) with neural attention mechanism on the embedded words. The output of this step is sequences of sentence vectors.

At the second level, after acquiring sentence vector for all sentences in a document, we apply a bi-directional GRU layer (sentence GRU) on the sequence of sentence vectors. We then apply seperate neural attention mechanisms to encode the GRU outputs for classifying each label in the ICD code. This is to mimic Baumel's (2017) approach in that different labels share the GRU structure, but have different attention mechanisms. Finally, we apply a fully connected layer with softmax to the document vectors for each classifier to determine the code.

Intuitively, setting different attention mechanisms for different codes is justified because labels correspond to different diseases, the same sentence might have different weights in predicting different diseases. For example, a sentence containing the phrase "heart attack" might be very informative in predicting code associated with heart disease, thus has a high attention score. However when used to predicting code associated with high blood pressure, it might not be informative so have a lower attention score.

4.2 Training

We train our HA-GRU model with the preprocessed discharged summaries mentioned aforehand. We encode each target label as binary variables. When a code is present in the the correspond variable value is 1, else is 0. We use Binary Cross Entropy as our loss function and implemented it using pytorch's torch.nn.BCELoss function.

We tested with the size of embedding layer, word bi-GRU layer and sentence bi-GRU layer being (50, 100, 100), (100, 200, 200) and (200, 400, 400) respectively for first stage of experiment with 972 ICD code. With unsatisfacotry results we only tested the configuration of (50, 100, 100) in our second stage of experiment with six labels.

We use mini-batch gradient descent to train the HA-GRU model. Each mini-batch is of the same size (batch size) $\times$ (mean number of sentences in summaries in training data) $\times$ (mean length of tokens all sentences in training data). Batch size is a hyperparameter currently set to 64. Long sentences / summaries are truncated while short entences / sentences are padded with token value being 1, equivalent to "unknown" in preprocessing. By taking the mean we risk losing information. However, since 1) our implementation of both attention mechanisms requires a fixed length for number of tokens and sentences, and 2) padding all sentences and summaries to the max length will exponentially increase memory requirement, we trade off accuracy for efficiency.

With SGD we set the learning rate to be $1e-1$ for quicker convergence, and set the momentum to be 0.9. For stage one, we tested the number of epochs to be 10. For stage two, we set the number to be 5 due to time constraints.

Early stopping: to avoid overfitting we set an early stopping mechanism when validation loss is greater than training loss for a mini-batch, the model stops trianing. Validation set is randomly drawn from the validation set. This condition is checked at every step of mini-batch gradient descent.

5 Results

We first tried running the model on our full dataset (over 29,000 summaries and 936 labels), but because of the extreme sparsity of the label set, the results were negligible. We evaluated our models using Macro F-Score, as it is the most commonly used metric for this task. Our macro F-score is calculated by $\frac{2}{1/(precision+0.001)+1/(recall+0.001)}$. Precision and recall here are calculated by averaged each sample's precision and recall in the validation set. We add 0.001 as a smoothing factor because for some samples none of the labels are correctly predicted. Unfortunately, our Macro F-scores using the full label set of 936 ICD9 codes ranged from 0.01-0.04.

After getting these results, we decided to reduce the label space to six frequent ICD9 codes. These codes corresponded to: 1) Acute myocardial infarction (heart attack); 2) Chronic Kidney Disease; 3) Disorders of Lipoid Metabolism; 4) Disorder of Esophagus (tube between throat and stomach); 5) Other Disorders of Urethra and Urinary Tract; and 6) Other and Unspecified Anemias. Each of these six labels appears in roughly 10-to-25 percent of cases in our corpus. We did not pick the most frequent ICD9 codes (e.g. those that appear in 30-to-40 percent of cases), but we also wanted fairly common labels. Furthermore we thought this set of six labels represented a good range of diseases.

Unfortunately, our results lag behind those achieved by other studies on the topic, as indicated in Table 3 above. Strangely, even though the validation loss generally decreased across more epochs, our highest test-set F-scores typically occurred after one epoch. (Note: we took snapshots of test set F-score after each epoch because of time constraints; we weren't sure we would be able to run the full models).

To be honest, the results probably don't tell us much about the relative effectiveness of each preprocessing technique. The differences between each row are suspiciously high.

Preprocessing	**Macro-F**
Regular Vocab	0.089
Levenshtein	0.225
Byte25K	0.260
Byte10K	0.181
Byte5K	0.175
Hybrid10K	0.129

Table 3: The macro-F score achieved on validation sets.

Conclusion

While our results were not great, the project was still a good exercise for thinking about how the optimal preprocessing strategy depends on the particular corpus. Because medical discharge summaries tend to include both obscure scientific terms and informal language/typos, we thought a combination of Byte Pair Encoding and Levenshtein distance mapping would improve results. Looking back, it would have been interesting to apply deep learning techniques to correct spelling errors, which would be more rigorous than simply depending on edit distance. With more time, we also would have re-assessed the architecture of our HA-GRU model.

Contributions

Brenton worked on text preprocessing, including implementation of the Byte Pair Encoding, and wrote most of the report. Yuqiong set up the HA-GRU model and ran experiments. Aodong set up the GitHub repository but did not contribute much after submitting the proposal. We hope our project will be evaluated as a two-person effort.

References

Sandeep Ayyar. 2017. Tagging patient notes with ICD-9 codes. Available from the following URL: https://web.stanford.edu/class/cs224n/reports/2744196.pdf.

Tal Baumel, Jumana Nassour-Kassis, Raphael Cohen, Michael Elhadad, and Noemie Elhadad. 2017. Multi-label classification of patient notes: Case study on ICD code assignment. *CoRR*, abs/1709.09587.

Rohan Chitnis and John DeNero. 2015. Variable-length word encodings for neural translation models. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Richard Farkas and Gyorgy Szarvas. 2008. Automatic construction of rule-based ICD-9-CM coding systems. *BMC Bioinformatics*.

Philip Gage. 1994. A new algorithm for data compression. *C Users J*.

Ira Goldstein, Anna Arzumtsyan, and Ozlem Uzuner. 2007. Three approaches to automatic assignment of ICD-9-CM codes to radiology reports. *AMIA 2007 Symposium Proceedings*.

Sebastian Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*.

AEW Johnson, TJ Pollard, L Shen, L Lehman, M Feng, M Ghassemi, B Moody, P Szolovits, LA Celi, and Mark RG. 2016. *MIMIC-III, a freely accessible critical care database*.

Leah Larkey and W. Bruce Croft. 1995. Automatic assignment of icd9 codes to discharge summaries. *Technical report, University of Massachusetts at Amherst*.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully character-level neural machine translation without explicit segmentation. *arXiv preprint arXiv:1610.03017*.

Lucian Vlad Lita, Shipyeng Yu, Radu Stefan Niculescu, and Jinbo Bi. 2008. Large scale diagnostic code classification for medical patient records. *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*.

Adler Perotte, Rimma Pivovarov, Karthik Natarajan, Nicole Weiskopf, Frank Wood, and Noemie Elhadad. 2014. Diagnosis code assignment: Models and evaluation metrics. *Journal of the American Medical Informatics Association*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Haoran Shi, Pengtao Xie, Zhiting Hu, Ming Zhang, and Eric Xing. 2017. Towards automated ICD coding using deep learning. *https://arxiv.org/pdf/1711.04075.pdf*.

Yusuke Shibata, Takuya Kida, Shuichi Fukamachi, Masayuki Takeda, Ayumi Shinohara, Takeshi Shiohara, and Setsuo Arikawa. 1999. Byte pair encoding: A text compression scheme that accelerates pattern matching. *Technical Report DOI-TR-161, Department of Informatics, Kyushu University.*

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy.. 2016. Hierarchical attention networks for document classification.