Text Mining Adverse Drug Reactions from Social Media

University of Manchester

Edward Toms

Supervisor: Sophia Ananiadou

May 2022

Abstract

This piece of work explores various neural network architectures in order to produce a named entity recognition sequence labelling model. The goal of the research is to identify the most effective architecture for producing this high-performance model. The objective of the model is to accurately extract mentions of adverse drug reactions specifically from social media posts. Through various iterations, a model was produced with an F1-score of 85.8%, comparable with some of the highest performing research within the field. This is achieved through the use of two bidirectional LSTMs to predict the span of a named entity within a sentence. The domain being both social media and medical related poses a number of difficulties, which are discussed as part of the research.

Contents

Chapter 1

Introduction

1.1 Adverse Drug Reactions (ADRs)

An Adverse Drug Reaction (ADR) refers to an unintended response from a user to a drug they are taking, resulting in harm [CDC, 2017]. This can be anything from less severe symptoms such as insomnia or headaches, all the way up to serious responses such as cancers. Detecting ADRs is very important to the monitoring of drugs once they have been released onto the market, as it can help to ensure the safety of the users, for example if there has been a manufacturing flaw resulting in a sudden spike in ADRs. 28% of all ADRs are classed as preventable, and of those that are serious and life threatening, 42% are preventable [Bates et al., 1995]. It has been found that 6-7% of hospital admissions in the UK are due to ADRs, costing the NHS around £380 million a year [NICE, 2022]. Clearly, ADRs are a cause of concern in the field of medicine, and aiming to reduce and identify them early is critical to preventing unwanted responses and even saving many lives. However, a large number of ADRs go unreported via official channels, and are instead mentioned informally via other platforms; namely social media [Mao et al., 2013].

Due to this lack of reporting of ADRs, it is challenging for drug companies and users to keep track of how products are performing, and if they are safe for use. As social media is often used instead of official channels, it provides us with a large volume of data, either casual or more serious complaints. However, the scale of this data introduces further problems; it would not be possible for companies to manually monitor Twitter or similar sites to identify spikes in ADRs. For example, around 500 million Tweets are posted every day [Internet Live Stats, 2022], an impossibly large number for a human to monitor. Only a very low proportion of these Tweets would even mention a drug by name, and an even lower proportion will actually be relevant to ADR detection. One could argue that you could search by term or hashtag, but none of these solutions could ever bring the search space down to one that is human-readable. An automated solution is required to quickly and efficiently detect ADRs as they are occurring, while effectively being able to understand if a user is praising a drug, or suffering severe side effects. This concept of determining whether a user is pleased or disappointed with the drug in question is known as sentiment analysis. This is generally an easy task for humans, but much harder for computers to carry out effectively. For example, the statement: "Having no Vyvanse is killing me" suggests a user expressing that their drug is effective, and they are suffering without it. While this is easy for a human, it is difficult for machines to disambiguate it. In isolation, "Vyvanse is killing me" appears as an ADR, but we know given context that it is in fact the opposite, so this is something to bear in mind.

This piece of work aims to automatically detect ADRs through the text mining of social media, in order to be able to alert drug companies if a certain drug is spiking in mentions of ADRs online. This is achieved through experimenting with different neural network architectures, with a particular focus on bidirectional Long Short-Term Memory (bi-LSTM) layers and Convolutional Neural Network (CNN) layers. Named Entity Recognition (NER) in the domain of social media is a well explored field [Aguilar et al., 2019], which has had rapid development and numerous breakthroughs over the last few years [Hashimoto et al., 2016]. NER is the task of locating and classifying specific entities within text, for example names, locations, or in this case, adverse drug reactions [Wikipedia contributors, 2022b]. Classical techniques such as Support Vector Machines (SVM) and Bayes have been replaced in the modern era, with the field now being dominated by (deep) machine learning techniques [Chiu and Nichols, 2015]. It has been shown that these deep learning methods have very high performance measures [Huynh et al., 2016], which is the motivation towards taking this approach. Neural networks reduce the need for lots of hand-crafted features, which are both time consuming to produce as well as causing potential overfitting [Huynh et al., 2016]. Instead of using these hand-crafted features, neural networks can be created using pre-trained word representations, which are accessible (due to numerous models already being available), as well as proven to be effective in the past. Labelled test data is of course an extremely useful resource, but this data can be hard to come by due to the domain being rather specific. Regardless, such datasets do exist, namely CADEC and the Social Media Mining for Health (SMM4H) shared tasks, and although they contain some deprecated data, they will certainly be used for both training and testing [Karimi et al., 2015].

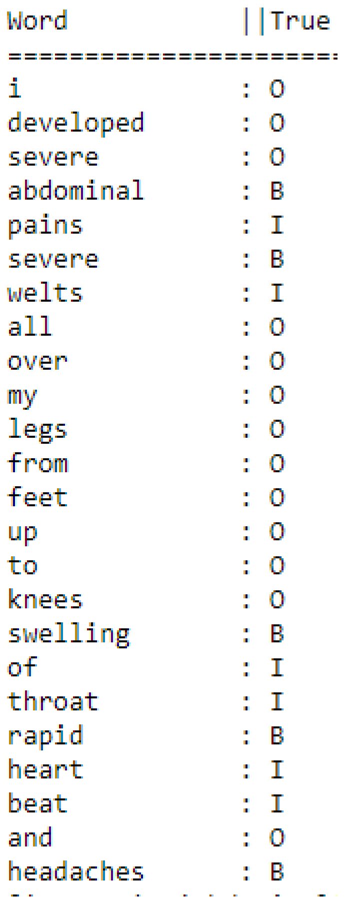
Much of the most current and cutting edge research comes from the annual SMM4H workshops as stated above. These workshops propose a set of shared research tasks, allowing any research group who wishes to enter to submit a solution to these tasks. The tasks mainly revolve around:

"automatic methods for the collection, extraction, representation, analysis, and validation of social media data (e.g., Twitter, Facebook) for health informatics" [SMM4H, 2020].

In particular, task 1 from the 2019 SMM4H set of tasks looks at the very problem proposed above. The papers submitted to this task are extremely relevant to my research, and contain a vast number of novel and unique approaches, which I will aim to combine into an effective model. Due to the fields of interest of SMM4H, papers from other tasks and years will still be very helpful, so it will be the main source for finding relevant pieces of research. Having so much research focussed on social media mining specifically within the health domain makes this too valuable a resource to ignore.

1.2 Named Entity Recognition (NER)

NER is generally treated as a sequence labelling task, where the model is aiming to predict correct tags for each word in a sentence. In order to achieve this, we need to first understand the system of tagging sequences. NER systems generally use the IOB2 tagging scheme, which stands for inside, outside, and beginning (of a named entity) respectively [Ramshaw and Marcus, 1999]. Tags that are part of a NE also contain information about what type of entity they are, for example B-PER or B-LOC refer to the start of a named entity that is a person or a location respectively. For this task, we are only interested in extracting the ADR named entities, so clarifying the type of entity in tagging is not important. This style of tagging is useful when there are multiple classes of NE and you wish to assess the performance of your model on each class. Because of this, it is acceptable to simply apply the tags “B” or “I” without other contextual information, as there is only one class.



```
Word           ||True
=====================
i              : O
developed      : O
severe         : O
abdominal      : B
pains          : I
severe         : B
welts          : I
all            : O
over           : O
my             : O
legs           : O
from           : O
feet           : O
up             : O
to             : O
knees          : O
swelling       : B
of             : I
throat         : I
rapid          : B
heart          : I
beat           : I
and            : O
headaches      : B
```

Figure 1.1: An example tagged sequence from the dataset

1.3 Word Embeddings

In natural language processing (NLP) , word embeddings are a vector-based representation of every word within a vocabulary. These help models to understand the meaning of words, the context they are used in, the similarity of words, and much more. As proposed by Harris (1954), words in similar contexts have similar meanings. Embeddings can have low dimensionality; containing a small amount of information, or they can have a high dimensionality; offering rich and specific contextual information. They are passed into models as weights for training in order to expedite the training process. Having these weights from the start rather than having to train them from scratch allows a model to make meaningful predictions sooner. The importance of word embeddings cannot be overstated; training unique weights for a machine learning model is a long process and the performance will still likely be lower than if embeddings were used. Embeddings are usually trained to be tailored towards a specific style of language. In my case, social media embeddings will be critical to understanding how this style of language is used, and how it differs from traditional written English. It is important to understand the use of word embeddings in order to understand how these NLP models learn.

Chapter 2

Related Work

The use of neural networks for text classification and sequence labelling is a well documented area with much research to draw inspiration from. The task of NER with these methods is a little less thoroughly researched, but there are still vast amounts of useful papers in the field.

2.1 Task-Specific Related Work

Towards text processing pipelines to identify adverse drug events-related tweets [Vydiswaran et al., 2019] uses a bi-LSTM on top of a convolutional layer to perform classification of tweets. Along with this architecture, word2vec is used to create pre-trained Twitter word embeddings as input features. The team created three models for comparison. The first model was a linear kernel SVM classifier with balanced class weights. This model was then trained on unigram features from tweets. Their second model was similar, but involved much more pre-processing on the data. This included a runthrough of the Ekphrasis unpacking tool [Baziotis et al., 2017], tokenization and lemmatization through NLTK [Bird et al., 2009], semantic type extraction through MetaMap, concepts from SNOMED extracted as features through cTAKES, pattern matching feature detection (eg through regex), as well as simply removing UTF-8 characters. This resulted in increases in both precision and F1 scores over the first model. The final model began as two variants of neural networks; a bi-LSTM and a bi-LSTM with a convolutional layer. However, they used the highest performing model based on validation accuracy, which was the bi-LSTM model. This performance was achieved through the use of GloVe word embeddings, along with pre-trained word2vec Twitter embeddings. Of these three models, they found the first two to perform best over recall and F1, while the second two models performed better over precision. Aspects of each of these models can certainly be used and refined in order to tweak an optimal model, with the discussion in the paper of word representation for neural models being particularly relevant.

Detecting Tweets Mentioning Drug Name and Adverse Drug Reaction with Hierarchical Tweet Representation and Multi-Head Self-Attention [Wu et al., 2018] is another paper using similar techniques. The first phase is to learn character level word representations through the use of a CNN. A bi-LSTM paired with another CNN then learns tweet representations of words. They then use multi-head self-attention (MHSA) to gather further information regarding the tweet's context by looking at interactions between all words in a tweet. The combination of these structures led them to achieve 91.83% F1-score for detecting if a drug is mentioned in a tweet, and 52% F1-score in detecting ADRs in tweets. This performance is very high, so is an approach to pay close attention to. Their use of a character-level CNN to analyse word embeddings is something that they have praised highly, due to its ability to deal with misspellings and out-of-vocab words effectively. This is certainly a relevant issue in the domain of social media as proper language is not necessarily to be expected. They also found that the addition of word embeddings, POS tags embeddings, sentiment scores and medical lexicon features were each effective in improving the performance of their classifier. The most impactful of these was the addition of word embeddings, which they have established is due to the level of semantic information this provides.

Adverse Drug Reaction Classification With Deep Neural Networks [Huynh et al., 2016] explores various different architectures for deep neural networks within the domain of ADR classification. This paper is a very useful point of reference for comparing different network structures, as it explores many more types of network compared to other papers. They treat the task as purely classification, rather than NER as other papers have, so although the proposed structures are relevant, they will need to be adapted for NER. All of their networks used are variations on a classic CNN, with two new unique structures being proposed; Convolutional Recurrent Neural Network (CRNN) and Convolutional Neural Network with Attention (CNNA). They compare these with a traditional CNN and a Recurrent Convolutional Neural Network (RCNN, ie the opposite of a CRNN in structure). The dataset used consists of Twitter data and MEDLINE reports of ADRs, with this combination making this piece of work an excellent baseline for comparison. Once again, they use word embeddings as input data for the networks, which has been shown to be a very common and effective approach. Through their research, they have found that the classic CNN structure performs best on the ADR dataset compared to their more complicated proposed network structures. Despite this, they note that CNNA is more appropriate for extracting word subsequences for ADRs, due to allowing the visualisation of attention weights during classification. On the ADR dataset, CNN performed the highest with an F1 score of 87%.

Detecting Adverse Drug Events from Swedish Electronic Health Records using Text Mining [Bampa and Dalianis, 2020] has a focus towards predicting ADRs based on the medical records of patients. They trained three classifiers: random walks; SVMlinear; and SVMrbf. With this, they used various various forms of word-level representations, including tf*idf and several different n-grams. From their results, unigrams outperformed all other n-grams, although these were in turn almost always outperformed by tf*idf. An important takeaway point they discovered was that considering textual features can increase detection performance by up to 15%. Although the domain is right for the task of ADR detection with NER, they have treated this as a classification task with a simpler architecture. However, this research will still be useful when it comes to selecting an effective form of word representation to pass to my architecture.

2.2 NER Papers not Related to ADRs

Recurrent Convolution Neural Networks for Text Classification [Lai et al., 2015] explores this task in a broader sense, performing classification of texts from various different domains through their testing process. The main focus for their task was producing a general classifier that could be trained on different datasets, rather than tailored towards a specific dataset (ADRs) like in my task. They use a single convolutional layer on top of pre-trained word embeddings from word2vec, with these word embeddings being very common in other research. As they have stated, this is a popular approach for when a large supervised dataset is unavailable. They trained 4 different models, with each scoring over 80% for precision across various datasets. This paper seems like it will be excellent for fine tuning my model later down the line, as it contains vast amounts of information about the variants in their models and how they perform against one another in different tasks.

Bidirectional LSTM for Named Entity Recognition in Twitter Messages [Limsopatham and Collier, 2016] is targeted towards NER for Twitter in general, rather than being specific to the domain of ADR detection like previous papers have been. The initial step of their approach revolves around converting sentences into orthographic features, which is not something I had explored in my research up to this point. An example they have given is: "Nintendo 3DS released in north America" becomes: "Cccccccc nCC cccccccc cc cccc Ccccccc". After this, character based word representations as well as pre-trained word embeddings are created for each sentence. Then, the data is input into their bi-LSTM in the following sequence: character representation; word embedding representation; orthographic character representation; orthographic word embedding representation. These are concatenated, with the sequencing allowing the network to infer contextual information of words. They then calculate the probability of labelling the whole sentence correctly by modelling interactions between successive labels. As this was part of a shared task, there are many comparable approaches. For the task of segmentation and categorisation, their approach had the highest performance, with F1, precision and recall scores of 52.41%, 60.77% and 46.07% respectively. This is a very novel approach, and one to explore in further detail.

2.3 Approaches using Pre-Trained Language Models

Detection of Adverse Drug Reaction mentions in tweets using ELMo [Sarabadani, 2019] uses Embeddings for Language Models (ELMo) [Gardner et al., 2018] in order to create word embeddings based on their context in a sentence. ELMo is "able to capture both syntactic and semantic characteristics" [Sarabadani, 2019]. These embeddings are fed into a series of convolutional layers, with the outputs of them being fed into a bi-LSTM. They use several variants of this architecture to compare to one-another, with many of the differences being to do with the input features. Along with the ELMo embeddings, they have also experimented with GLoVe embeddings, character-level embeddings and a concatenation of word and character embeddings. Unfortunately, their models did not perform as well as many others, with the ELMo model giving the best F1-score of 64%. They suggest a number of improvements, including; using BERT (another language model) for embeddings; using more complex architectures after the embedding layer; adding Parts of Speech (POS) tags to aid in training; adding tweet cluster features and topic modelling; and augmenting the training set with new tweets mentioning ADRs.

Want to Identify, Extract and Normalize Adverse Drug Reactions in Tweets? Use RoBERTa [Subramanyam Kalyan and Sangeetha, 2020] is another paper using a pre-trained language model; in this case RoBERTa, a fine-tuned and optimised version of BERT. It is noted in this paper that many modern approaches fine-tune pre-trained models (eg BERT) rather than training their own models from scratch. In order to then make this model appropriate for the task at hand, a task-specific layer is added on top of the language model; in this case a sigmoid layer. This concept is used for both of the tasks being attempted by this research group, which are classification and ADR extraction. This task of ADR extraction is viewed as a sequence labelling problem, which they achieve through the use of IOB tagging, by tagging tokens as:

I: tokens inside entity span
O: tokens outside entity span
B: beginning of entity span

This method is something which will be explored again later [Miftahutdinov et al., 2019]. In terms of performance, their models scored above average for both tasks. For the classification of ADRs task, they achieved a precision of 52%, recall of 65% and F1 of 58%. For the NER task, multiple models were trained. However, the model they chose to write most about was the base RoBERTa model, with precision of 63%, recall of 78.9% and F1 of 70.1%. The recall and F1 scores were the highest of any of their trained models.

KFU NLP Team at SMM4H 2019 Tasks: Want to Extract Adverse Drugs Reactions from Tweets? BERT to The Rescue [Miftahutdinov et al., 2019] once again uses a pre-trained language model; this time being the generic BERT. They took place in the same set of shared tasks as the previous paper's group, so once again they were faced with classification of ADR tweets and NER of ADRs in tweets. Through their research, they found BERT to outperform the previous baselines of RNNs and LSTM, so it is an important point of comparison for this piece of research. For the classification task, they compared their BERT-based model to a SVM-based model, with the BERT model outperforming the SVM with an F1 of 57.38% vs the SVM's score of 51.64%. For the NER task, this team once again viewed it as a sequence labelling problem, so also used the BIO tagging scheme. They compared several models, including bi-LSTM with CRF; BERT; and BERT for Biomedical Text Mining (BioBERT). They also used a CRF tagger on top of BioBERT. Due to encountering unstable results, they opted to combine the results of ten BioBERT-CRF and implement a voting scheme. This led their relaxed F1, precision and recall to be 65.8%, 55.4% and 81.0% respectively, while their strict scores were 46.4%, 38.9% and 57.6% respectively. All of these scores are above average, with their model achieving the highest relaxed F1 score among all teams.

Chapter 3

Design and Implementation

3.1 Dataset

I chose to use the CADEC corpus [Karimi et al., 2015] for this research as it contains the data and labels required in a relatively convenient format. The corpus consists of 1250 posts from social media (askapatient.com) detailing adverse drug events, symptoms, diseases, and drug names, coming to a total of 101,486 words. These data are labelled at the character level, labelling the span (ie start and end locations) of a named entity. Although having labelled data is critical to the training of a sequence labelling model, this character-level annotation did present some problems which will be discussed further in chapter 3.2. Due to the nature of the task, I will only be using the ADR labels, but it could be worth using the other labels in future as an extension to this task, for example to detect the drug name related to an ADR. As the dataset is relatively small, it will be difficult to train an accurate classifier for this many classes, so I instead opted to focus on a high-accuracy ADR span classifier.

Alternatively, I initially looked at the SMM4H shared task dataset. This is produced as part of the shared task so that each participant has access to the same labelled data. It is a collection of Twitter data in which users are expressing that they been affected by some sort of ADR. As it was provided to participants at the time, it was fairly difficult to gain access to this dataset. However, once I had found a source, I quickly found it to be less helpful than anticipated. Firstly, the raw data was not stored in the dataset; instead, there was a list of user IDs and tweet IDs, along with a script to fetch the tweets. The ability for users of social media to delete and update posts was problematic, specifically because a huge amount of tweets had been deleted since the corpus was produced. Not only was the data looking sparse, but the script provided to fetch the data was written in Python 2.x, and contained a lot of deprecated code that would no longer work. In order to even access this data I would need to translate the code into Python 3.x, which I deemed an unnecessary time sink when I had CADEC readily available. SMM4H was preferable as the tagging was done at a word level rather than character level, but the difficulty in accessing this data along with the large amount of missing data points made CADEC the clear best choice.

3.2 Data Pre-Processing

I created IOB tags corresponding to the dataset based on their corresponding annotations. The first stage of the pre-processing was reading in these labels and producing two lists: one containing each word in a sentence and the other containing the tag for the word at that index. As the annotations were at the character-level, I opted to initially apply the annotations in-line to the start of each word, and then increment the end index of the span appropriately. I experimented with the use of BIOLU tags; an extension of IOB2 that includes tags for the last (L) word in a NE and single unit (U) words. Due to the small size of the corpus, the L and U tags were fairly sparse throughout, so the model lacked enough data points to make accurate predictions. This massively hindered the overall performance; there were simply too many classes and not enough data, so I chose to stick to IOB tagging.

Once these lists had been created, it was then possible to strip punctuation and make the text lowercase without putting the indices out of step. After the text had been cleaned, each number was then replaced by the word 'number' in order to help the model with contextual information, as a unique word embedding for every number is not useful. At this point, around 20% of the dataset were words that were not in the English language, as they were either misspellings, accidentally concatenated words (where the user had missed the space by accident), or domain-specific words like drug names. From here I was able to pass each out of vocabulary (OOV) word through the Ekphrasis spell checker [Baziotis et al., 2017], which removed half of these incorrect words. The words that were left (~10%) nearly all consisted of accidental concatenations. I intended to then use the Ekphrasis word segmenter to separate these, but there was no elegant solution to determine what words to segment. The initial plan was to segment any word that was left OOV, but that led to too many words being incorrectly segmented. For example, the word “I’ve” becomes “ive” after pre-processing, but Ekphrasis would segment this into “i” “ve”, which is not what we are looking for. This has further implications as the lists of word and tag need to be updated, which can easily cause problems if not handled correctly. 10% of words being OOV was deemed an acceptable amount, so each word from each sentence was entered into a dataframe, along with its corresponding tag. I experimented with the removal of stop words, but actually found it to not have a noticeable effect on results. Because of the way the data is labelled, removing stop words can actually harm the extraction of ADRs in some cases. For example, one such ADR from the dataset was the phrase “pain in the back of my head”. With stop word removal, “in the” and “of” will be removed, but that means the model will never see these words, thus incorrectly labelling them, lowering overall performance. From here it was possible to extract the unique words, and move on to word embeddings.

An embedding index was produced using GloVe vectors [Pennington et al., 2014] for each unique word in the dataset. Vectors with a dimensionality of 200 were used in order to extract rich and accurate information about each word. These were the GloVe Twitter embeddings, which were trained on tweets, making them ideal for this task. As CADEC contains data from social media posts, the style of language is exactly right for this type of embedding. These were then passed as weights into the model during the training process. I chose this approach over a pre-trained language model such as BERT in order to research the training process and to see how well (relatively) domain-specific embeddings would perform.
I considered word2vec, another industry standard word representation, as it is similar to GloVe in practice but has a different training process. Word2vec also offers a higher dimensionality of 300 (as opposed to GloVe’s maximum of 200), but this is a small corpus and vocabulary, so that extra dimensionality is not worth the additional time requirements it will impose. In my opinion, the global vectors from GloVe are more useful than the local co-occurrence vectors produced from word2vec, as social media text is often fragmented and global representations will deal with this style of language better. Despite being similar in use, it has been found that GloVe has a slightly higher

overall performance in this specific domain [Vydiswaran et al., 2019].

3.3 Network Architecture

In order to produce a sequence labelling model I opted to begin with a bi-LSTM. LSTMs are a type of recurrent neural network that are able to process entire sequences rather than single data points through the use of feedback [Hochreiter and Schmidhuber, 1997]. Bi-LSTMs have become the industry standard high-performing benchmark [Du et al., 2019] for their sequence labelling abilities. This is a stack of two LSTM units, with one taking the reverse sequence as input. The hidden representations from each LSTM are then concatenated, forming a hidden representation of both the forward and reverse sequences. Doing this allows the model to learn better contextual information for a word, as a unidirectional architecture will only look at the context preceding a word. This is evidently important when we look at the patterns tags can take. For example, The tag 'I' must always follow a 'B' or an 'I', but never an 'O'. 'B' and 'O' can follow any tag. Without being able to look forward past the current target word, the model would be unable to learn these patterns.

3.3.1 Iteration 1

The baseline structure for this model consisted of: an embedding layer with dimensionality of 200 (to match GloVe); a 1-dimensional spatial dropout layer; a bi-LSTM layer with 2*256 units; a bi-LSTM layer with 2*128 units; and a time-distributed dense output layer. I used Adam as the optimiser and mean squared error as the loss function. The model was trained with a batch size of 64 over 11 epochs. This model performed fairly poorly overall, only achieving an F1-score of 38.6%. Generally, this model (along with many subsequent models) tends to get the span of the named entity correct for the most part, but struggles with the correct labels. For example a sequence that should be tagged "BIIO" might be tagged "BBBO". It was clear from this iteration that the model had not gathered enough contextual information from training, as impossible sequences such as "OIIO" would be predicted.

3.3.2 Iteration 2

The second iteration involved the addition of a 1D convolutional layer to the architecture of iteration 1. This layer trained 128 filters with dimensionality of 1 to act as a projection layer to get past some limitations of pooling [Kiranyaz et al., 2021]. The other network parameters were the same as in iteration 1. The purpose of this stage of experimentation was to determine the best position for the convolutional layer, along with how many of these layers to include. Through this experimentation process, I created models with one convolutional layer before the first bi-LSTM, a model with one convolutional layer in-between the bi-LSTMs, and finally a model with the convolutional layer after both bi-LSTMs. None of these models reached over 40% F1-score - in fact most of them performed considerably worse than the standard bi-LSTM model. In tag prediction, the model seemed to have lost some contextual information, as many of them would have a bias towards predicting "I" tags, which is the most surprising tag to see over-represented (there are more "O" and "B" tags than "I" in the dataset).

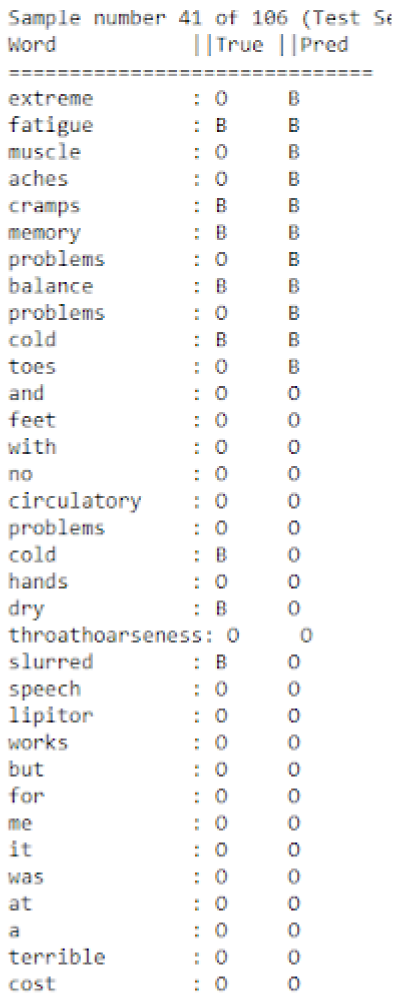
3.3.3 Iteration 3

The final iteration during experimentation involved sandwiching bi-LSTM layers in-between convolutional layers. The aim was to smooth the output at each stage, so that similar word representations could be more easily identified by the bi-LSTM layers and thus perform better on NEs from outside of the training set. The worst performance from these was the architecture consisting of a convolutional layer before and after the first bi-LSTM layer, achieving an F1-score of only 24.7%. This model was better than the results imply at span-based classification, but the tags were generally wrong (for example mostly "I" tags where there should be "B" tags). The best performing model consisted of two bi-LSTM layers (2*128 units) with a convolutional layer after each (training 200 and 100 filters for the first and second layer respectively). This model achieved an F1-score of 38.2%, one of the highest performing so far, but it had a few issues with its prediction. The NE tagging seemed to be all or nothing; the model would either correctly predict a whole span as being an NE, or it would completely miss the entirety of the NE. A good example for this is the phrase "pain in head". The gold-standard tags for this phrase are "BII". A model with low performance may correctly label "pain" as "B", but miss the other tags. This was fairly common in these early models. In this iteration however, the NE would either have the whole span labelled (but perhaps slightly inconsistent tags), or completely miss the sequence. “Pain” by itself would be picked up as “B”, but “pain in head” would be labelled “OOO”. This shows that the model is struggling to extract the contextual information around a target word, so only performs well at tagging words in isolation.

3.3.4 Iteration Summary

From working through these iterations, it was clear that adding convolution was not aiding the overall performance of the model. In fact, the highest performing model was the simple stacked bi-LSTM layers without convolution. This is in line with other research in which traditional models tend to perform better than those using convolution [Vydiswaran et al., 2019]. This is in part due to the style of word representation I have chosen. Many convolutional models use character level vectors in which words are made up of strings of features rather than actual characters themself [Limsopatham and Collier, 2016]. This approach has proven to be effective, but with this limited dataset there are not enough data points to train on features that are this abstract. Because of this, I stuck to GloVe embeddings at the word level, due to their rich dimensionality and how well they have been tailored towards this specific domain.

After this experimentation, it became clear that using a standard bi-LSTM was the best approach to take, as convolution would often lead to poorly tagged sequences. These sequences were much more likely to have “impossible” tags as prediction, which massively hurts the overall performance. However, in the bi-LSTM models, there seemed to be a fair amount of over-fitting taking place as the model exploited the large imbalance of tags in the dataset. This can be fixed by tweaking the model and tuning the hyperparameters, so I was confident that this model would perform very well after optimisation. Because of this, iteration 1 was used as the starting point for the non-optimised model.



```
Sample number 41 of 106 (Test S
Word          ||True ||Pred
==============================
extreme        : O     B
fatigue        : B     B
muscle         : O     B
aches          : O     B
cramps         : B     B
memory         : B     B
problems       : O     B
balance        : B     B
problems       : O     B
cold           : B     B
toes           : O     B
and            : O     O
feet           : O     O
with           : O     O
no             : O     O
circulatory    : O     O
problems       : O     O
cold           : B     O
hands          : O     O
dry            : B     O
throathoarseness: O     O
slurred        : B     O
speech         : O     O
lipitor        : O     O
works          : O     O
but            : O     O
for            : O     O
me             : O     O
it             : O     O
was            : O     O
at             : O     O
a              : O     O
terrible       : O     O
cost           : O     O
```

Figure 3.1: A sequence of tags in an over-fitting model

3.3.5 Optimisation

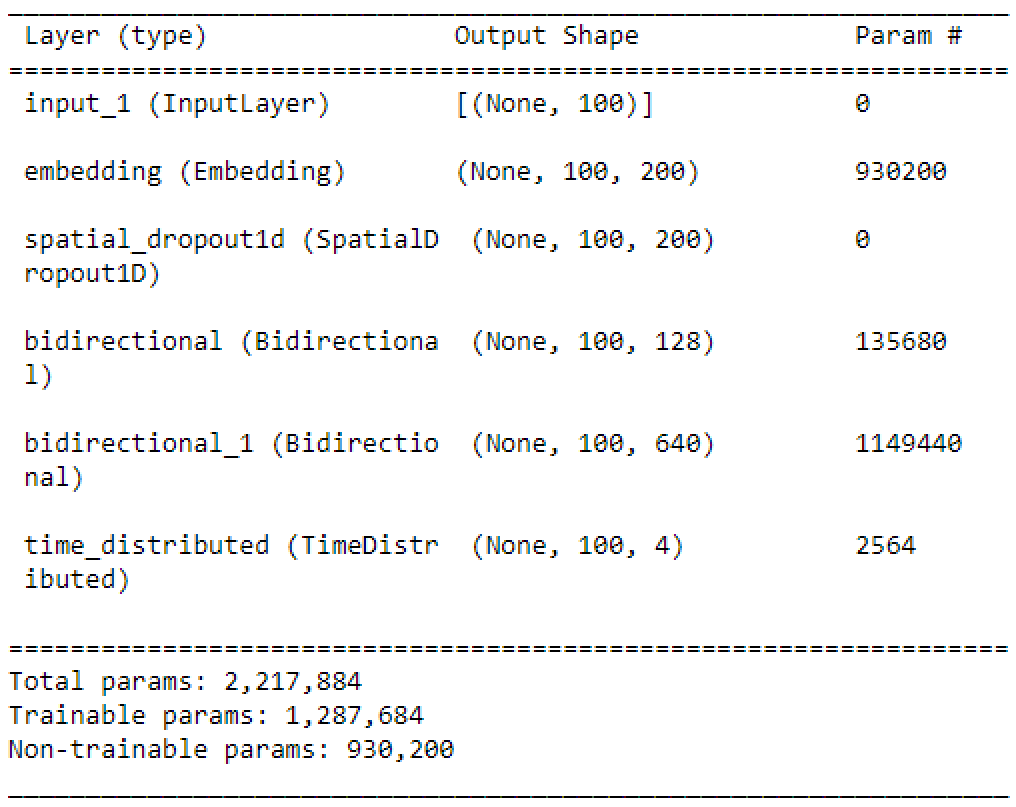
Using the baseline model, I performed a HyperBand hyperparameter search over the number of units in each bi-LSTM layer. This algorithm is based on a random search but uses some exploration-exploitation policies in order to reduce time wasted on new settings which perform worse than the current best (early stopping) [Li et al., 2017]. After 50 epochs, the optimal settings were determined to be 2*64 units in the first layer and 2*320 units in the second layer, along with a learning rate of 0.001 for the optimiser. Using these new settings, I then performed an optimisation to find the best number of epochs. As the optimal epochs changes between each iteration, it is something that would have likely needed further experimentation to get the best possible results. Regardless, after 50 epochs the best was determined to be epoch 30, so the model was retrained using these settings.

3.4 Model Implementation

The final model was implemented using Keras for Python, an API using the TensorFlow backend [Chollet et al., 2015]. Once the data had been processed as detailed previously, it was able to be converted into a vector format. A dict containing each word in the vocabulary along with an index was produced, and each word in a sequence was converted to its corresponding index. These indices were shared with the GloVe dictionary in order for the model to learn the correct weights. The sequence was then padded (or truncated) to a length of 100 words, as this was roughly the average sequence length. This is vector X. Similar to the word index, a tag index was produced to give a unique index for each tag. This was then used to carry out the same process for the sequence of tags (as opposed to sequence of raw text), except after this sequence was produced each tag representation was converted to a one-hot vector. This is vector Y. The data was then split into X and Y training and test sets using sklearn train_test_split, ready to be fed into the model [Pedregosa et al., 2011].

The input layer of the model has a dimensionality of 100 to match the sequence length of X and Y. This passes into the embedding layer, which has an input dimension of the size of the vocabulary + 2. This is due to the addition of "PAD" and "UNK" words, to represent padding and OOV words respectively. The embedding matrix of GloVe vectors is passed into the embedding layer as weights, and these weights are set to be untrainable. This approach was chosen as the weights are already trained towards this style of data, so updating the weights detract from learning the tags. This feeds into a 1D spatial dropout layer, with a rate of 0.05. Next, there are two stacked bi-LSTM layers. The first layer has 2*64 units and the second has 2*320 units. Both have a recurrent dropout of 0.1. Finally, these feed into a time-distributed dense output layer with 4 units and rectified linear units activation function. The model is optimised using the Adam optimiser, with a learning rate of 0.001, and using mean squared error as the loss function. The model architecture can be seen in figure 3.2.

Using these hyperparameters, the model was then trained for 30 epochs, achieving an F1-score of 85.8%.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 100)]             0

 embedding (Embedding)       (None, 100, 200)          930200

 spatial_dropout1d (SpatialD  (None, 100, 200)         0
 ropout1D)

 bidirectional (Bidirectiona  (None, 100, 128)         135680
 l)

 bidirectional_1 (Bidirectio  (None, 100, 640)         1149440
 nal)

 time_distributed (TimeDistr  (None, 100, 4)           2564
 ibuted)

=================================================================
Total params: 2,217,884
Trainable params: 1,287,684
Non-trainable params: 930,200
_________________________________________________________________
```

Figure 3.2: The final model architecture

This model could then be accessed through a simple frontend, allowing a user to open the page via localhost and provide queries. The model will return a list containing sublists of each NE. The purpose of this was to demonstrate the performance of the model on any query, rather than having to go through the entire codebase and get predictions only for entries in the dataset.

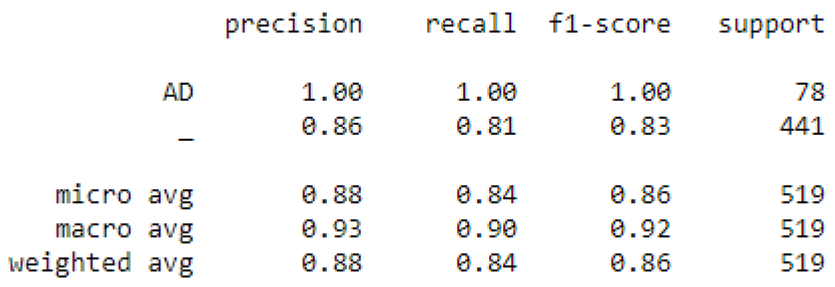
Chapter 4

Evaluation

4.1 Performance

Results were assessed using seqeval [Nakayama, 2018], an evaluation library designed for sequence labelling tasks, with a specific focus towards accurate metrics for NER (amongst other things). Having this span-level evaluation is critical to assessing the overall performance of this model, as there is more to it than just the correct label in the correct position. A correctly identified span with the wrong labels is still a correctly identified span, but traditional evaluation techniques would simply consider these incorrect. This leads to the performance appearing far worse than it is in reality, as a model that can identify the span but that struggles with the exact label is still doing well at the overall task, and that should be indicated by the metrics.

The model detailed in the chapter 3.4 achieved an F1-score of 85.8%, with precision of 88% and recall of 84%. I am extremely happy with these results as the performance has increased significantly since the initial baseline implementation with an F1-score of only 38.6%. The classification report can be seen in figure 4.1:



```
              precision    recall  f1-score   support

          AD       1.00      1.00      1.00        78
           _       0.86      0.81      0.83       441

   micro avg       0.88      0.84      0.86       519
   macro avg       0.93      0.90      0.92       519
weighted avg       0.88      0.84      0.86       519
```

Figure 4.1: Classification Report

When compared to related work, this model performs extremely well, but with some room for improvement. For NER, F1 score is the most important metric out of the three listed due to the data. For this dataset, around 85% of all tags are "O" tags, meaning a model that tags every word as "O" would still achieve a precision of 85%. As F1 is a weighted average, it shows us the performance out of the relevant predictions, allowing us to compare models more accurately. All of the statistics detailed below are for models also trained on CADEC, meaning we can compare to each of them.

Model Performance			
Model	P	R	F1
(Metke-Jimenez and Karimi, 2016)	64.4	56.5	60.2
(Stanovsky, Gruhl and Mendes, 2017)	92.2	94.5	93.4
(Tang et al., 2018)	67.8	64.9	66.3
My Model	**88.2**	**84.4**	**85.8**

The research performed by Metke-Jiminez and Karimi (2016) along with that from Tang et al. (2018) is a good point of comparison. Both papers had a similar amount of complexity to their model in terms of architecture, processing and embeddings. Outperforming these in all metrics represents a significant achievement and I am very pleased with how this model performs compared to related work. The work proposed by Stanovsky, Gruhl and Mendes (2017) contains a far more sophisticated approach, so their incredibly high results are to be expected. Not only have they augmented

their data using DBpedia concept embeddings, but they have also developed and implemented a tool (RASCAL) to check the validity of the annotations within the dataset. This requirement is something that will be discussed in the following chapter. As well as this, they used additional embeddings specifically trained towards the medical field, from the Blekko medical corpus [Wikipedia contributors, 2022a]. Finally, the use of knowledge graphs on top of these other optimisations shows that this piece of work demonstrates a level of complexity above the other pieces mentioned. Regardless of all of their improvements and added complexity, my model was still close in performance to this state-of-the-art research, so I am more than happy with the overall results.
The main problem causing the incorrect tags was largely related to the rules of how one tag can transition to another. This is referring to patterns in the tags as there are certain "impossible" sequences. This is covered in more detail in the network architecture section, but to recap; "I" tags may only follow "B" or "I" tags, they may never follow an "O" tag. As can be seen from the model's performance, it has generally learned these as patterns rather than as strict rules. Because of this, abnormal and nonsensical sequences will occasionally be predicted, but this is infrequent enough to be acceptable. Ways to improve on this will be discussed later on.

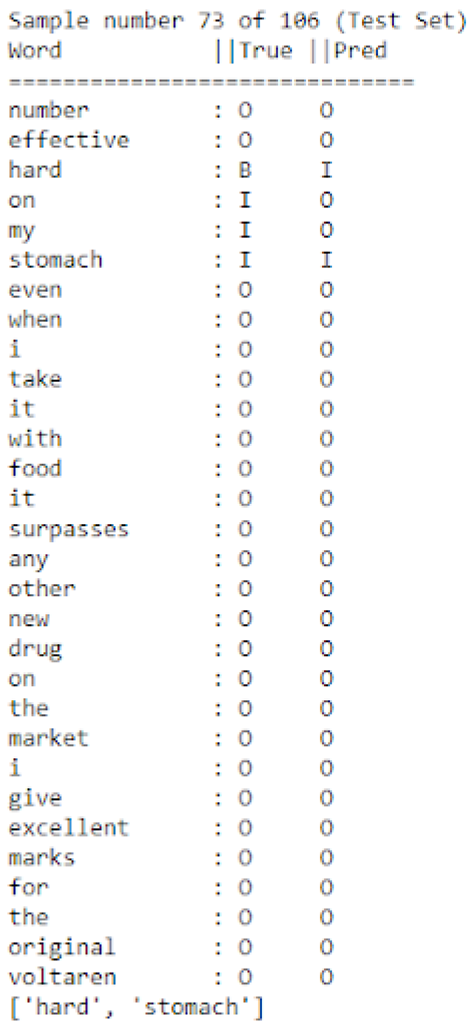
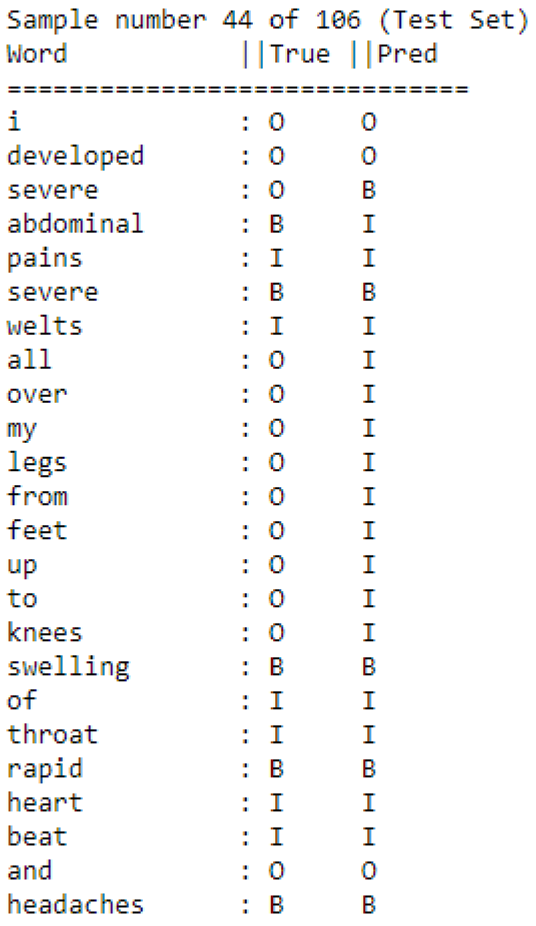

```
Sample number 73 of 106 (Test Set)
Word           ||True ||Pred
==============================
number         : O     O
effective      : O     O
hard           : B     I
on             : I     O
my             : I     O
stomach        : I     I
even           : O     O
when           : O     O
i              : O     O
take           : O     O
it             : O     O
with           : O     O
food           : O     O
it             : O     O
surpasses      : O     O
any            : O     O
other          : O     O
new            : O     O
drug           : O     O
on             : O     O
the            : O     O
market         : O     O
i              : O     O
give           : O     O
excellent      : O     O
marks          : O     O
for            : O     O
the            : O     O
original       : O     O
voltaren       : O     O
['hard', 'stomach']
```

Figure 4.2: A sequence being labelled with impossible tags

4.2 Dataset

The glaring issue of using CADEC is its small size. The Penn Treebank [Taylor et al., 2003] is a corpus used for POS tagging, a very similar sequence labelling task. This corpus produces such high performance models because it contains over 7 million accurately tagged tokens. In contrast, CADEC contains only around 100 thousand tokens. Not only are there a relatively small number of entries, but the tagging itself seems fairly inconsistent. It is clear there were multiple annotators who had some level of inconsistency between them as to what they deemed as being the full span of an ADR. An example of this is illustrated in figure 4.3.

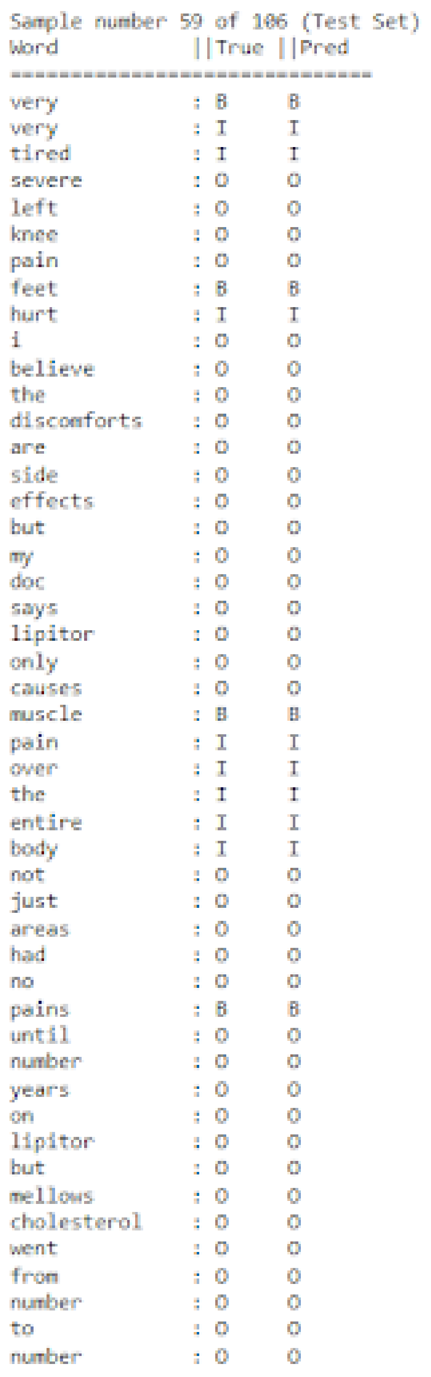


```
Sample number 44 of 106 (Test Set)
Word           ||True ||Pred
==============================
i              : O     O
developed      : O     O
severe         : O     B
abdominal      : B     I
pains          : I     I
severe         : B     B
welts          : I     I
all            : O     I
over           : O     I
my             : O     I
legs           : O     I
from           : O     I
feet           : O     I
up             : O     I
to             : O     I
knees          : O     I
swelling       : B     B
of             : I     I
throat         : I     I
rapid          : B     B
heart          : I     I
beat           : I     I
and            : O     O
headaches      : B     B
```

Figure 4.3: Inconsistency in what should count as the start of an ADR

Firstly, this annotator is inconsistent within the entry as to where the span begins. For the first ADR they have decided it begins at the word "abdominal", excluding the word "severe" preceding it. However, in the very next ADR ("severe welts"), they have decided to include "severe", rather than leaving "welts" as a single unit entity. There are numerous entries like this throughout the dataset, in which some annotators deem an ADR to begin as soon as it is described (“severe welts”), whereas others will wait for a specific symptom to be mentioned (“abdominal pains”).

A further inconsistency is to do with how strict the annotator has been. As you can see in figure 4.3, my model predicts the phrase “severe welts all over my legs from feet up to knees” as one entire ADR. In the annotation for this entry, only the symptom at the start is annotated (“severe welts”). Despite this, in other entries, the annotation is more lax and will encompass the entire description of an ADR, like in the model’s prediction. This can be seen from in figure 4.4, in which the span “muscle pain over the entire body” is labelled as a large ADR in both the gold standard tags and the predicted tags. Using the style from the figure 4.3, you would expect only the phrase “muscle pain” to be identified, showing the importance of consistency. In my opinion, the span identified in figure 4.3 by the model is correct (as it contains the whole ADR span), and the annotator is incorrect as they have only labelled the symptom. The amount of inconsistency goes on, again in figure 4.4 “very very tired” is labelled as an ADR span. Generally words like ”very” are included in the spans, but there are many files in which the same sentence would just have “tired” annotated as a single unit ADR.



```
Sample number 59 of 106 (Test Set)
Word           ||True ||Pred
==============================
very           : B     B
very           : I     I
tired          : I     I
severe         : O     O
left           : O     O
knee           : O     O
pain           : O     O
feet           : B     B
hurt           : I     I
i              : O     O
believe        : O     O
the            : O     O
discomforts    : O     O
are            : O     O
side           : O     O
effects        : O     O
but            : O     O
my             : O     O
doc            : O     O
says           : O     O
lipitor        : O     O
only           : O     O
causes         : O     O
muscle         : B     B
pain           : I     I
over           : I     I
the            : I     I
entire         : I     I
body           : I     I
not            : O     O
just           : O     O
areas          : O     O
had            : O     O
no             : O     O
pains          : B     B
until          : O     O
number         : O     O
years          : O     O
on             : O     O
lipitor        : O     O
but            : O     O
mellows        : O     O
cholesterol    : O     O
went           : O     O
from           : O     O
number         : O     O
to             : O     O
number         : O     O
```

Figure 4.4: A perfectly labelled sequence

Chapter 5

Further Work and Conclusion

5.1 Further Work

5.1.1 Dataset Augmentation

As previously mentioned, the CADEC corpus is very limited in terms of its size. In the work proposed by Stanovsky, Gruhl and Mendes (2017), they use knowledge graphs in order to help guide the model through more advanced embeddings. This is in an attempt to offset the limited number of tokens by providing richer context for the words in the dataset. The model produced in my research would benefit from this, particularly for OOV words as GloVe embeddings are only provided if a word appears in the dataset's vocabulary.

Alternatively, the use of another dataset to either add to the CADEC data or to be used instead could be beneficial. The addition of more data will be critical in improving the accuracy of the NER system for all words and phrases, rather than just having a high performance for a specific dataset. With such little data it is hard to make a robust system that can deal with more obscure, complex or grammatically ambiguous phrases. The current system is very good at identifying more obvious ADR phrases, for example symptoms, but struggles with more subtle phrasing of sentences.

5.1.2 Data Pre-Processing

In relation to the dataset, the annotations provided proved problematic due to their style. As the annotations were character-level, it made it difficult to produce a robust text-processing pipeline. This is due to textual features such as punctuation counting towards this character count, meaning the data needed to be labelled first, and then cleaned. Although it seems like a minor point, it would be much easier to process this data using a package like NLTK to perform token-level analysis. However, this has issues like splitting tokens around punctuation, causing the annotations and the text to become out of sync.

Additionally, there were a large amount of accidental concatenations of words within the dataset. This ended up being around 10% of all tokens. Although this is a low amount, it would be preferable to find a solution to dealing with these. As previously detailed, it is not as simple as just segmenting words not within the English dictionary, as lots of words will become incorrectly segmented or unnecessarily segmented. Finding a solution that was both elegant and all-purpose proved very difficult, and was something I would have liked to be able to implement. This is particularly important due to the domain of this research, as on social media there are large amounts of mistakes, misspellings, acronyms and slang that need to be handled correctly.

In terms of how the data is read, I would likely rewrite the process for applying tags. Rather than appending them to the text in-line, it would make more sense to simply count the characters, define new tokens by whitespace and just create a list of tags without altering the source text. This would be a more robust process, opening more avenues for experimentation. For example, one of these would be revisiting stop words; in my early research I found that stop words were having little effect on the performance of the model. However, these models were fairly rudimentary, and it would be interesting to see the results on this model with a much higher performance than the earlier iterations. There was no clear solution to stop words, as although they are generally deemed meaningless words, certain samples would contain ADR spans in which they were relevant. For example, in figure 4.3, "swelling of throat" is labelled as a span. With the removal of the word "of" (as it is a stop word), the model would likely struggle to guess that this is the middle of a span, due to it being OOV. Examples like this show that there is no clear-cut solution; as the dataset is so diverse in terms of writing style that a lot of relevant data will still be removed, even though stop words are generally not offering context. However, there is one solution that could help with that, which is the addition of Conditional Random Fields (CRF).

5.1.3 Conditional Random Fields (CRF)

CRFs are a form of finite state models with probabilities associated with transitioning between tags, while overcoming some of the shortcomings of Hidden Markov Models (HMM) and other traditional transitional probabilistic models [Lafferty et al., 2001]. A CRF layer on top of a bi-LSTM layer helps to teach the model which tags are allowed to transition into which other tags, along with probabilities for a tag given a previous tag and the word vector. This allows a model to get around the problem of "impossible" tags detailed in the network architecture section. A basic model without this layer can still learn this behaviour (as shown in this research), but not as a strict rule, rather a model will learn that it is just extremely unlikely for an "I" tag to follow an "O". Using CRF, the model will learn that it is a strict impossibility to produce tags of this sequence, assigning a probability of zero for this transition. This in turn can help with some other shortcomings, for example this issue of stop words. A model without CRF will have no idea how to label stop words within a NE, so will likely tag them as "O" as this is the most common tag, and a lot of this model's weakness is from OOV words. However, with the addition of CRF, the model would be able to look at the beginning and end of the NE, and infer the tags that should go in between using its transition rules.

Unfortunately, I was unable to implement the CRF layer due to Keras being in the process of migrating to a new version. This involved changing what packages were supported and included in the main installation, and CRF was not one of these. Instead, it was being added into another package, but the documentation and examples are extremely lacking and very hard to use. This pushed CRF to be beyond the scope of this project during the timeframe. In future I would very much like to successfully implement CRF as I feel it is the missing piece of this model required to push it above 90% F1-score.

5.1.4 Additional Named Entities

In order to move this from a piece of research into a viable product, training the model to predict more than just ADRs is required. Within the CADEC dataset, annotations for drug names and symptoms also exist. By updating the tagging process to include the information of what is being identified (for example tag “B” could become “B-DRUG” or “B-SYMP”), the model can be trained to detect much more rich information. This additional context to an ADR can be very helpful, as it allows the user to see what drug is causing a specific ADR, or if there is a spike in symptoms.

This is not without its drawbacks, many of which will be made more severe by the small dataset. Firstly, only 8 drugs are labelled within the dataset, but with a large bias towards 3 of them. Consequently, newly trained models may struggle with more obscure drugs, or drugs used in a previously unseen context. Perhaps an additional dataset specifically tailored towards drug names would be useful, either to augment this model or produce another model; one for drugs and one for ADRs. Additionally, there will likely be some confusion between the labelling of symptoms and the labelling of an ADR. There is a lot of overlap in these, as a symptom is generally an ADR, but not always, for example when somebody is describing a symptom being alleviated. Having such a small dataset, this is a potentially large problem, as a lot of data will be needed to learn the subtlety of differentiating between these two closely related classes.

5.1.5 Convolution

As stated in the introduction, one of the aims of this project was to explore the effectiveness of a convolutional layer within this network architecture. During my early experimentation, I quickly found that my models were more effective without using convolution. It would be worth revisiting the work proposed by Huynh et al. (2016), in which they explore a variety of architectures all relating to convolution. Studying this paper and achieving a more in-depth grasp of the inner workings of these architectures would be critical to effectively implementing convolution, as I feel my understanding at that point of experimentation was not as deep as for other architectures. Convolution has a specific purpose within the architectures I have proposed, and I feel it was under-utilised, so it would be beneficial to experiment with it in combination with my high performance model, rather than the initial iterations.

5.1.6 Richer Embeddings

A final additional feature would be to augment the word embeddings with some further information. An approach I have particular interest in is the research proposed by Wu et al. (2018). In this piece of work, they highly praise the use of character-level embeddings paired with a convolutional layer, as well as including token-level embeddings. This additional embedding data allowed them to achieve a high performance of 91.83% F1. I believe this is something I overlooked when designing my architecture, as my use of convolution was not effective. Using convolution to learn embeddings and then feeding this into a bi-LSTM as described in my research could help to boost the performance, as richer context is very helpful for a small dataset.

In relation to enriching the data, I would like to experiment with the addition of POS tags to help with OOV words. The word embeddings do consider context, but the addition of POS for words lacking embeddings will help the model to understand what the word's purpose is, even without knowing its actual meaning. A dimension could be added to the input vector to include a one-hot encoding for each POS. The importance of this is reduced as the dataset becomes larger (as the vocabulary is larger), so this addition is more relevant to my work than others due to the relatively small vocabulary.

5.2 Conclusion

The goal of this piece of work was to produce a high-performance named entity recognition model by comparing different neural network architectures in order to find the most suitable. The use of deep learning neural networks is something that I had relatively little experience with going into this project, so I am more than pleased with the results. Being able to achieve an F1 performance higher than many other pieces of research within the field is a great achievement, and it has been pleasantly reassuring to watch my understanding develop to this point. I am very curious to see how research within the field develops over time, as these models already have impressively high performance. As one of the main limitations in producing a model this specific is the lack of data, the influx of data and reporting due in part to COVID-19 has the potential to exapand this from a proof of concept to a real deployable solution.

Bibliography

[Aguilar et al., 2019] Aguilar, G., Maharjan, S., López-Monroy, A. P., and Solorio, T. (2019). A multi-task approach for named entity recognition in social media data. *arXiv preprint arXiv:1906.04135*.

[Bampa and Dalianis, 2020] Bampa, M. and Dalianis, H. (2020). Detecting adverse drug events from swedish electronic health records using text mining. In *Proceedings of the LREC 2020 Workshop on Multilingual Biomedical Text Processing (MultilingualBIO 2020)*, pages 1–8.

[Bates et al., 1995] Bates, D. W., Cullen, D. J., Laird, N., Petersen, L. A., Small, S. D., Servi, D., Laffel, G., Sweitzer, B. J., Shea, B. F., Hallisey, R., et al. (1995). Incidence of adverse drug events and potential adverse drug events: implications for prevention. *Jama*, 274(1):29–34.

[Baziotis et al., 2017] Baziotis, C., Pelekis, N., and Doulkeridis, C. (2017). Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754, Vancouver, Canada. Association for Computational Linguistics.

[Bird et al., 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

[CDC, 2017] CDC (2017). Adverse drug events in adults.

[Chiu and Nichols, 2015] Chiu, J. P. C. and Nichols, E. (2015). Named entity recognition with bidirectional lstm-cnns.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. https://keras.io.

[Du et al., 2019] Du, J., Huang, Y., and Moilanen, K. (2019). Aig investments. ai at the finsbd task: Sentence boundary detection through sequence labelling and bert fine-tuning. In *Proceedings of the First Workshop on Financial Technology and Natural Language Processing*, pages 81–87.

[Gardner et al., 2018] Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N., Peters, M., Schmitz, M., and Zettlemoyer, L. (2018). Allennlp: A deep semantic natural language processing platform.

[Hashimoto et al., 2016] Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. (2016). A joint many-task model: Growing a neural network for multiple nlp tasks.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Huynh et al., 2016] Huynh, T., He, Y., Willis, A., and Rüger, S. (2016). Adverse drug reaction classification with deep neural networks. Coling.

[Internet Live Stats, 2022] Internet Live Stats (2022). Twitter usage statistics.

[Karimi et al., 2015] Karimi, S., Metke-Jimenez, A., Kemp, M., and Wang, C. (2015). Cadec: A corpus of adverse drug event annotations. *Journal of biomedical informatics*, 55:73–81.

[Kiranyaz et al., 2021] Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., and Inman, D. J. (2021). 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398.

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

[Lai et al., 2015] Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence.*

[Li et al., 2017] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816.

[Limsopatham and Collier, 2016] Limsopatham, N. and Collier, N. (2016). Bidirectional lstm for named entity recognition in twitter messages.

[Mao et al., 2013] Mao, J. J., Chung, A., Benton, A., Hill, S., Ungar, L., Leonard, C. E., Hennessy, S., and Holmes, J. H. (2013). Online discussion of drug side effects and discontinuation among breast cancer survivors. *Pharmacoepidemiology and drug safety*, 22(3):256–262.

[Miftahutdinov et al., 2019] Miftahutdinov, Z., Alimova, I., and Tutubalina, E. (2019). Kfu nlp team at smm4h 2019 tasks: Want to extract adverse drugs reactions from tweets? bert to the rescue. In *Proceedings of the fourth social media mining for health applications (# SMM4H) workshop & shared task*, pages 52–57.

[Nakayama, 2018] Nakayama, H. (2018). seqeval: A python framework for sequence labeling evaluation. Software available from https://github.com/chakki-works/seqeval.

[NICE, 2022] NICE (2022). What are the health and financial implications of adverse drug reactions?

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[Ramshaw and Marcus, 1999] Ramshaw, L. A. and Marcus, M. P. (1999). Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

[Sarabadani, 2019] Sarabadani, S. (2019). Detection of adverse drug reaction mentions in tweets using elmo. In *Proceedings of the Fourth Social Media Mining for Health Applications (# SMM4H) Workshop & Shared Task*, pages 120–122.

[SMM4H, 2020] SMM4H (2020). Social media mining for health applications (smm4h) workshop and shared task 2021.

[Subramanyam Kalyan and Sangeetha, 2020] Subramanyam Kalyan, K. and Sangeetha, S. (2020). Want to identify, extract and normalize adverse drug reactions in tweets? use roberta. *arXiv e-prints*, pages arXiv–2006.

[Taylor et al., 2003] Taylor, A., Marcus, M., and Santorini, B. (2003). The penn treebank: an overview. *Treebanks*, pages 5–22.

[Vydiswaran et al., 2019] Vydiswaran, V. V., Ganzel, G., Romas, B., Yu, D., Austin, A., Bhomia, N., Chan, S., Hall, S., Le, V., Miller, A., et al. (2019). Towards text processing pipelines to identify adverse drug events-related tweets: university of michigan@ smm4h 2019 task 1. In *Proceedings of the fourth social media mining for health applications (# SMM4H) workshop & shared task*, pages 107–109.

[Wikipedia contributors, 2022a] Wikipedia contributors (2022a). Blekko — Wikipedia, the free encyclopedia. [Online; accessed 4-May-2022].

[Wikipedia contributors, 2022b] Wikipedia contributors (2022b). Named-entity recognition — Wikipedia, the free encyclopedia. [Online; accessed 9-Feb-2022].

[Wu et al., 2018] Wu, C., Wu, F., Liu, J., Wu, S., Huang, Y., and Xie, X. (2018). Detecting tweets mentioning drug name and adverse drug reaction with hierarchical tweet representation and multi-head self-attention. In *Proceedings of the 2018 EMNLP workshop SMM4H: the 3rd social media mining for health applications workshop & shared task*, pages 34–37.