Exploring a Deep Learning Pipeline for the BioCreative VI Precision Medicine Task

Tung Tran[1] and Ramakanth Kavuluru[1,2]
[1]Department of Computer Science, University of Kentucky, Lexington, KY, USA
[2]Division of Biomedical Informatics, Department of Internal Medicine, University of Kentucky, Lexington, KY, USA

***Abstract*—The BioCreative VI Track IV (Mining protein interactions and mutations for precision medicine) challenge is presented to support the Precision Medicine Initiative with the goal of applying biomedical text mining methods to extract interacting protein pairs that are affected by a genetic mutation. Herein, we describe systems that we submitted for the two challenge tasks and report our results. For the document triage task, we report a system performance of 67.23% F1 with a precision of 54.73% and recall of 87.12% on the official test set. For the relation extraction task, our models placed 2nd when using Entrez Gene IDs and 1st when using HomoloGene IDs. When using HomoloGene IDs, our system achieves 37.27% F1 with a precision and recall of 45.44% and 31.61% respectively.**

Keywords— deep learning; named entity recognition; relation extraction; protein-protein interactions; gene normalization

I. Introduction

Precision Medicine (PM) is an emerging disease treatment paradigm in which healthcare is customized to each individual patient. To support this effort, it is important to be able to extract useful translational information such as mentions of relationships between genes, mutations, and diseases. The BioCreative VI Track IV (Mining protein interactions and mutations for precision medicine) challenge is presented in an effort to identify and study mutations and their effect on molecular interactions. This particular track involves two distinct tasks: document triage and relation extraction. In the first task, participants are asked to build systems able to determine whether a PubMed citation is *relevant* or *not relevant*; i.e., whether or not it describes an interaction affected by a genetic mutation. The training set for this first task contains 4082 articles with 1729 of them deemed relevant.

A subset of the aforementioned training documents that are *relevant*, consisting of 597 documents, has been additionally annotated with gene mentions and relevant interaction pairs. This subset is supplied as training data for task 2. In the second task, participants are asked to extract interacting protein pairs that are affected by a mutation. We treat the two tasks independently and built separate systems for each. We describe the system used for the document triage task in Section 2. We then describe the system pipeline used for the relation extraction task in Section 3. Finally, we discuss our validation results in Section 4.

Our work is primarily supported by the National Library of Medicine through grant R21LM012274 and the National Cancer Institute through grant R21CA218231.

II. Document Triage System

We propose using a deep neural network architecture based on convolutional neural networks (CNNs) to tackle the document triage task. The following model is based on the one proposed by Kim et al. (1) for text classification. The input is a document with words $\boldsymbol{w} = (w_1, w_2, .., w_n)$ each represented by their corresponding index to the vocabulary V^{word}. The words are mapped to word vectors via a word embedding matrix $E^{\text{word}} \in \mathbb{R}^{|V^{\text{word}}| \times d}$ to produce a document matrix $D \in \mathbb{R}^{n \times d}$ where d is the length of the word representation vectors. More concisely,

$$D = \begin{pmatrix} E^{\text{word}}[w_1] \\ \vdots \\ E^{\text{word}}[w_n] \end{pmatrix}$$

where $E^{\text{word}}[i]$ is the i^{th} row of E^{word}. The word embedding matrix can be initialized to random or pretrained values; in either case, the word vectors are (further) modified via backward propagation. The central idea in CNNs is the so called *convolution* operation over the document matrix to produce a feature map representation using a convolution filter (CF). The convolution operation $*$ is formally defined as the sum of the elementwise products of two matrices. That is, for two matrices A and B of same dimensions, $\mathrm{A} * \mathrm{B} = \sum_{\mathrm{j}} \sum_k A_{j,k} \cdot B_{j,k}$. With this, a CF is the matrix $W \in \mathbb{R}^{h \times d}$ that is applied as a convolution to a window of size h over D to produce a feature map $v = [v_1, \ldots, v_{n-h+1}]$, such that

$$v_i = \text{relu}(W * D_{i:i+h-1} + b)$$

where $D_{i:i+h-1}$ is a window of matrix D spanning from row i to row $i + h - 1$, W and $b \in \mathbb{R}$ are learned parameters, and $\text{relu}(x) = \max(0, x)$ is the linear rectifier activation function. The goal is to learn multiple CF that can collectively capture diverse representations of the same document. Suppose there are k filters, then we produce k corresponding feature maps $v^1, \ldots, v^k$. We select the most distinctive feature of each feature map using a max-over-time pooling operation (2) to produce the final feature vector $p \in \mathbb{R}^k$, such that $p = [v^1_{max}, \ldots, v^k_{max}]$ where $v^j_{max} = \max(\boldsymbol{v}^j_1, \ldots, \boldsymbol{v}^j_{n-h+1})$.

We can also learn different sets of k CFs for different window sizes h as is typically the practice. Choosing a larger h provides more context and thus could be beneficial in improving predictive power but might adversely affect efficiency given the additional time needed. We can then take

the corresponding feature vector for each window size and concatenate them to form the final feature vector. More formally, we can parameterize the window sizes as a sequence $h_1, \ldots, h_H$ of H unique sizes. Suppose p^{h_i} denotes the feature vector produced on k filters with a window size of h_i, then the final $kH \times 1$ feature vector is

$$p^* = p^{h_1} \parallel \cdots \parallel p^{h_H}$$

where $\parallel$ is the vector concatenation operation. The output layer consists of m units (one per each of the m target labels) and is fully connected to the full feature vector p^*. The output vector $q \in \mathbb{R}^m$ is thus defined as

$$q = W_q p^* + b_q$$

where $W_q \in \mathbb{R}^{m \times kH}$ is a parameter matrix and $b_q \in \mathbb{R}^m$ is the vector of bias terms. In order to get a categorical distribution, we apply a softmax layer to the vector q such that

$$p_j = \frac{e^{q_j}}{\sum_{j=1}^{m} e^{q_j}}$$

where $p_j \in [0,1]$ is the probability estimate of the label at index j. We optimize on the standard categorical cross-entropy loss via a variant of stochastic gradient descent (SGD) called RMSProp (9). Much of the formulation presented is purposely kept abstract since it will be re-used in a later section. Clearly, the triage task is a binary classification problem where $m = 2$. The configuration of the model implementation is as follows. We used word embeddings of size 200 pre-trained on the PubMed corpus (3). For the convolutional component, we use window sizes of 3, 4, and 5 with 200 convolutional filters. The model is trained for 30 epochs with a mini-batch size of 8 and learning rate of 0.001. Since each instance is a collection of sentences and the window size is at most 5, we pad four zero-vectors at the beginning and the end of the input text as well as between sentences. Moreover, we use a dropout rate of 50%. During training, we checkpoint model parameters at each epoch and only keep the checkpoint resulting in the highest F1 on the development set. We train 10 such models as part of an ensemble. Each model of the ensemble is trained and tuned on a random split of 80% to 20% and seeded with a different value for random parameter initialization.

III. Relation Extraction System

For the second task we propose a pipeline system that consists of three components: supervised named entity recognition (NER) for gene mention detection, knowledge-based gene normalization, and supervised relation classification to predict each pair of genes found as either positive or negative for an interaction. It is possible to use an "out-of-the-box" solution such as GNormPlus (4) that identifies both gene mentions and their corresponding gene identifier directly; however, we opted for a supervised approach that allows us to leverage the generous gene annotations provided with the training corpus for this task. We describe the NER system used to identify potential gene mention spans along with the training corpus in Section 3A. We then describe our system for gene normalization in Section 3B. Lastly, we describe the model we used for relation classification in Section 3C.

A. Gene Mention Identification

The aim of the first component in the pipeline is to identify spans of text that refer to specific genes. To that end, we propose the use of a deep neural network system based on a CNN-LSTM hybrid model proposed by Chiu et al. (5) for NER. This sequence-to-sequence model composes word representations with CNNs by convolving over character n-grams. At the word level, contextual word representations are composed using a bi-directional LSTM layer. A separate fully-connected softmax output layer is present at the output of each LSTM unit such that an Inside, Outside, Beginning (IOB) label prediction can be made for each token.

Herein, we formulate the model from the bottom up while re-using some notations and definitions established in Section 2. In this formulation, a word w_i for $i \in [1, n]$ is treated as a lowercased character sequence $c_1^i, \ldots, c_{T^i}^i$ representing their index into the character vocabulary V^{char}. The corresponding character embedding matrix $E^{\text{char}} \in \mathbb{R}^{|V^{char}| \times 32}$ embeds each character as a vector of length 32. We use the same embedding setup to produce character type embedding vectors of length 8 indicating the type of character: lowercase, uppercase, punctuation, or other. This allows the model to generalize specific words while still considering the nuances involved in how named entities presented. Suppose the embedding matrix for *character type* is $E^{\text{ctype}} \in \mathbb{R}^{4 \times 8}$ and $z_1^i, \ldots, z_{T^i}^i$ represents the sequence of enumerated character type for the word at position i. The word at position i can then be represented as a matrix composition B^i of its character embeddings, or concretely

$$B^i = \begin{pmatrix} E^{\text{char}}[c_1^i] \parallel E^{\text{ctype}}[z_1^i] \\ \vdots \\ E^{\text{char}}[c_{T^i}^i] \parallel E^{\text{ctype}}[z_{T^i}^i] \end{pmatrix}$$

where $E^{\text{char}}[j]$ and $E^{\text{ctype}}[j]$ are the j^{th} rows of E^{char} and E^{ctype} respectively. We perform a convolution operation over B^i of window size 3 to obtain the feature map $v^i = [v_1^i, \ldots, v_{T^i-2}^i]$ such that

$$v_j^i = \text{relu}\left(W^{\text{char}} * B_{j:j+2}^i + b^{\text{char}}\right)$$

where $B_{j:j+2}^i$ is a window of matrix B^i spanning from row j to row $j+2$ and W^{char} and b^{char} are network parameters. We apply 50 filters to obtain 50 corresponding feature maps denoted as $v_1^i, \ldots, v_{50}^i$. Let $v_j^i(x)$ be the x^{th} value of v_j^i, then the word representation at position i is $u_i = [\hat{v}_1^i, \ldots, \hat{v}_{50}^i]$ where $\hat{v}_j^i = \max\left(v_j^i(1), \ldots, v_j^i(T^i - 2)\right)$.

Once a word representation is composed for each word, we can use a bi-directional LSTM to model the sequence. It is important that we also include actual word embeddings as well as *word type* embeddings as input. The latter embeddings serve a similar purpose to that of the character types and can correspond to one of the five following classes: all lowercase, mixed-cased, capitalized, all uppercase, or other. Drawing from

the notation presented in Section 2, the input is a sequence of word indexes $w_1, w_2, .., w_n$ into the word vocabulary V^{word} and the corresponding embedding matrix is denoted as $E^{\text{word}} \in R^{|V^{\text{word}}| \times d}$. In addition, we denote $\bar{z}_1, \bar{z}_2, .., \bar{z}_n$ as a sequence of enumerated *word types* corresponding to the embedding matrix $\text{E}^{\text{wtype}} \in \text{R}^{5\times 32}$. The bi-directional LSTM can then be composed as

$$\vec{h}^i = \text{LSTM}^{\rightarrow}\left(u_i \parallel E^{\text{word}}[w_i] \parallel E^{\text{wtype}}[\bar{z}_i]\right),$$

$$\overleftarrow{h}^i = \text{LSTM}^{\leftarrow}\left(u_i \parallel E^{\text{word}}[w_i] \parallel E^{\text{wtype}}[\bar{z}_i]\right),$$

$$h^i = \vec{h}^i \parallel \overleftarrow{h}^i \quad \text{for } i = 1, \ldots, n$$

where $E^{\text{word}}[j]$ and $E^{\text{wtype}}[j]$ are the j^{th} rows of E^{word} and E^{wtype} respectively. Moreover, $\text{LSTM}^{\rightarrow}/\text{LSTM}^{\leftarrow}$ represent an LSTM unit composition in the forward/backward direction. The output at each timestep necessarily has its own softmax output layer in order to be able to tag each word with an IOB label. The output layer at position i, or $q_i \in \mathbb{R}^m$, is defined as

$$q_i = W_q p_i + b_q$$

where $W_q \in \mathbb{R}^{m\times 50}$, $b_q \in R^m$ are parameters. In order to get a categorical distribution, we apply a softmax layer to the vector q_i such that

$$p_i^j = \frac{e^{q_i^j}}{\sum_{j=1}^{m} e^{q_i^j}}$$

where $p_i^j \in [0,1]$ is the probability estimate of the label at index j for the word at position i. Again, we optimize on the standard categorical cross-entropy loss. However, since each instance may be of a different sequence length, we optimize on the mean loss computed over the n output layers.

Since we are only concerned with gene entities, the label space consists of: B-GENE, I-GENE, and O. These labels are sufficient to indicate whether the token is part of a gene mention. This model is trained on the supplied training data and additionally on the GNormPlus corpus (4) which include re-annotations of the BioCreative II GM/GN corpus (6). The core training data consists of 5668 sentence-level training examples while the GNormPlus corpus constitutes an additional 6389. We used the same pre-trained word embedding vectors of size 200 as described in Section 2 for the document triage task. The network was trained using SGD with an exponential decay rate of 0.95 for a maximum of 10,000 iterations. On each iteration, we trained the network using a mini-batch of 20 random examples. We check-pointed every 100 iterations and saved only the checkpoint with the best F1 on the development set. We also deployed *early stopping* such that training is stopped if there are no improvements for 10 checkpoints. Like in the document triage task, we train 10 such models (each with a different seed) as part of an ensemble where each model is trained on a smaller random subset of only 50% of the original training set.

B. Entrez Gene ID Normalization

For the gene normalization component, we initially experimented with a naive approach using a the gene lexicon provided with the BioCreative II Gene Normalization training data as well as mappings provided with the training corpus. This served as a reasonable baseline; however, it does not take context into consideration during the mapping process. A gene mention may be incorrectly mapped to one of its many homologs resulting in increased false positives. We report the performance of this baseline system in Section 4. The final version of our gene normalization system is knowledge-based and more sophisticated in that it takes into consideration both the gene mention and the context. This system relies on the NCBI gene database [7] to identify the candidate gene IDs for a particular mention and further narrows it down to a best guess based on the document in which it occurred. We define two utility functions that serve as the basis for this system. The first function, $gene\_name\_lookup$, takes as input a mention span and returns a list of candidate gene IDs sorted by relevance. This is achieved by querying the NCBI gene database via the E-utilities API[1]. This provides a ranked of candidate genes for a given gene mention. The intuition here is that the top few in this list are either the correct gene or at least homologs of the correct gene. We now define the second function, $gene\_pmid\_lookup$, which takes as input a PMID and returns a list of candidate gene IDs for the article. We achieve this by making another query to the NCBI gene database using the PMID of the current document as query input[2]. This allows us to narrow down the list of candidate gene IDs to ones that have already been identified as appearing in the document. The final gene normalization algorithm takes as input a gene mention and a PMID and returns either a gene ID or *NULL*. The latter indicates that no match can be found, in which case we simply ignore the span entirely for the remainder of the pipeline. The algorithm is defined as follows.

Algorithm 1 Gene Normalization

Input a: gene mention
Input b: document PMID
  $X \leftarrow$ gene_name_lookup(a)
  $Y \leftarrow$ gene_pmid_lookup(b)
  for $x \in X$ **do**
    if $x \in Y$ **then**
      return x
    end if
  end for
  return *NULL*

For example, suppose the document in question has PMID 18725399 and we wish to map the gene mention "Utp21" to a gene ID. The above algorithm will correctly return 851125 as the gene ID which can be verified via footnotes 1 and 2.

[1] An example query for the gene span "Utp21": https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=gene&term=Utp21&retmax=100&sort=relevance

[2] An example query for the PMID 18725399: https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=gene&term=18725399[PMID]

C. Relation Classification of Gene Pairs

For the relation classification component, we use the same CNN architecture introduced in Section 2 modified to suit the problem of relation classification. The model hyper-parameters mirror that of the triage task and the overall task remains binary classification. However, each pair of candidate genes in an article constitutes a separate instance. For each pair of candidate genes, we generate a training instance by performing *entity binding*; i.e., we replace mentions of the pair with GENE_A and GENE_B in the corresponding document text. For a gene pair (A, B), we also generate an additional instance for the reverse case (B, A) given directionality does not matter. Note that we run both cases of a candidate pair during testing and consolidate the predictions. We also generate examples for the exception case when the candidate pairs are the same genes, i.e. $A = B$, in which case GENE_S is used for entity binding of the single gene ID. We also replace mentions of other genes with GENE_N in either case. In total, we generated 2972 instances from the 597 articles in the training set. At test time, we only predict pairs as positive where the mean probability is above 50% for the instance generated from (A, B) and its reverse case (B, A). In case no pairs meet the threshold, we try to make at least one prediction by predicting the pair with the highest probability (even if it is ≤ 50%) as positive for an interaction.

IV. Results and Discussion

For the document triage task, we evaluated performance by training on a held-out set of 70 documents. The system achieved 73.88% F1 with a precision and recall of 67.54% and 81.53% respectively. There may be nuances to this task that is not being effectively captured by the model. On the official test set, the system achieved 67.23% F1 with a precision of 54.73% and recall of 87.12% respectively

TABLE I. System Test Performance

System Evaluation Method	*P (%)*	*R (%)*	*F (%)*
Using Entrez Gene ID	36.53	25.61	30.11
Using HomoloGene IDs	45.44	31.61	37.29

To evaluate the performance of our system pipeline for the relation extraction task, we tested the system on a held-out set of 70 documents. The following are evaluations based on the individual components and are mutually independent. For the NER system, the performance is at 59.48% F1. For the gene normalization component, our initial naive approach using a gene mention lexicon yielded an accuracy of 49.67%. This poor performance may be due to the fact that the lexicon is outdated (2008 release) and that the context of a gene mention is generally ignored with this approach. The more sophisticated version that operates by cross-referencing the NCBI gene database yielded a much improved accuracy of 78.90%. The final component in the pipeline responsible for relation classification yielded an F1 of 81.80% with a precision of 72.57% and recall of 93.71%.

We now report *end-to-end* performance of the system for relation extraction on the held-out validation set. If we assume perfect gene annotations, we can extract pairs with reasonable performance at an F1 of 83.72%. NER and gene normalization represent a major barrier to achieving such promising results overall. Our initial experiments show that with naive gene normalization, the system performed with an F1 score of 20.62% with a precision and recall of 20.41% and 20.83% respectively. The low performance here is expected since incorrect gene annotations will propagate to the end of the pipeline; nevertheless, it serves as a baseline for comparison. The PubTator tool (8), which uses GNormPlus as the backend for gene annotations, results in more than double the recall at 43.72% but still suffers from low precision at 19.27% with an overall F1 of 26.75%. If we use the more sophisticated gene normalization method from Section 3.2, we observe improved precision and recall at 28.82% and 51.04% respectively and an overall performance of 36.84% F1 on the validation set.

The performance of our system on the official test set is recorded in Table 1. When evaluating on original Entrez Gene IDs, we achieved a system performance of 30.11% F1 with a precision and recall of 36.53% and 25.61% respectively which ranked second among systems that were submitted. When evaluating on HomoloGene IDs, which accounts for homologous genes, we achieved a system performance of 37.27% F1 with a precision and recall of 45.44% and 31.61% respectively which ranked first among submitted runs.

References

1. Kim, Y. (2014) Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, 1746–1751.

2. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011) Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, vol. 12, 2493–2537.

3. Moen, S. and S. S. Ananiadou, T. (2013) Distributional semantics resources for biomedical text processing. In *International Symposium on Languages in Biology and Medicine (LBM)*, 39-44.

4. Wei, C.-H., Kao, H.-Y., and Lu, Z. (2015) GNormPlus: an integrative approach for tagging genes, gene families, and protein domains. *BioMed research international*, vol. 2015.

5. Chiu, J. P. and Nichols, E. (2016) Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, vol. 4, 357–370.

6. Morgan, A. A., Lu, Z., Wang, X., Cohen, A. M., Fluck, J., Ruch, P., Divoli, A., Fundel, K., Leaman, R., Hakenberg, J., et al. (2008) Overview of biocreative ii gene normalization. *Genome biology*, vol. 9, no. 2, p. S3.

7. Maglott , D., Ostell , J., Pruitt , K. D., and Tatusova, T. (2010) Entrez gene: gene-centered information at ncbi. *Nucleic acids research*, vol. 39, no. suppl 1, D52–D57.

8. Wei, C.-H., Kao, H.-Y., and Lu, Z. (2013) PubTator: a web-based text mining tool for assisting biocuration. *Nucleic Acids Res*, 41(W1):W518-22.

9. Tieleman, T. and Hinton, G. (2012) Lecture 6.5---RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning