A Bidirectional Gated Recurrent Neural Network and Capsule Network Based Approach for Identification of Relevant Literature Regarding Protein Interactions and Mutations for Precision Medicine

Bruno Magalhães and Rui Souto
Departamento de Informática, Universidade do Minho

***Abstract*— According to the Precision Medicine Initiative, precision medicine is an emerging approach for disease treatment and prevention that takes into account individual variability in genes, environment, and lifestyle for each person [1]. In this paper, we describe our approach to the Document Triage Task of the BioCreative VI Precision Medicine Track 4, which focused on identifying relevant PubMed citations describing genetic mutations affecting protein-protein interactions by taking advantage of text mining algorithms. We propose a Bidirectional Gated Recurrent Neural Network (GRU) and a Capsule Network based model for the task. In order to convert the PubMed corpus into the corresponding vector representations and feed them into the neural network, we use a pre-trained word2vec model to get word representations. When using the evaluation script provided by the organization, our system achieves a precision of 0.6295, recall of 0.7386 and F-score of 0.6797. The source code is available and can be found in the corresponding GitHub repository[1].**

***Index Terms*— Document Classification; Protein Interactions; Deep Learning; Gated Recurrent Unit: Capsule Neural Network**

I. Introduction

As the number of PubMed articles published daily increases, the development of automated methods are becoming more and more important in order to facilitate the retrieval of information regarding biological information. In this context, text mining tools can contribute significantly to this effort by allowing the identification of scientific literature of known relationships between gene mutations and protein-protein interactions in an automated way. For this reason, it has received special attention in recent years.

The goal of BioCreative VI Precision Medicine Track 4, challenge on which this project is based, was to promote the development of text mining algorithms, bringing together the biomedical text mining community and focusing on scanning the published biomedical literature and to extract the reported discoveries of protein interactions affected by the presence of genetic mutations. The challenge is subdivided in two different tasks: document triage and relation extraction. In this paper, we describe our approach to the first subtask, trying to build a state of the art text mining system capable of detect whether a PubMed article, mentioning genetic mutations that affect protein-protein interactions (PPIs), is relevant or not, based on its title and abstract [2].

Deep neural networks, due to its flexibility and ability to learn underlying features automatically with less need of engineered features, have become popular in text classification tasks. We first introduce some state-of-the-art neural networks architecture concepts based on Recurrent Neural Networks and their recent variants Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), specialized in remembering information for a long period of time. Furthermore, Convolutional Neural Networks, originally aimed for computer vision, on Convolutional Neural Networks, which achieved excellent results on multiple benchmarks in a variety of previous NLP tasks [3] due to its capacity of extracting local and position-invariant features. However, as CNNs present some important drawbacks, Capsules Networks were introduced recently as a novel alternative to the convolutional networks as research to explore and evolve this models is still undergoing [4], [5].

We used the Bayesian optimization technique to select the best hyperparameters in which our model gave the best overall results.

Finally, in the evaluation phase, the organisers provided a script for the teams to use on their results, generating scores on standard evaluation procedures: precision, recall, F-score and average precision. The same script was used here for comparison purposes. The organisers also developed a baseline model whose scores the teams should surpass in order for their results to be considered. The baseline model scored 65% on avg. precision, 60.97% on precision, 63.56% on recall and 62.24% on F-score. According to table II in [6], which shows the official ranking for the triage task, the team that ranked first in the challenge scored 71.95% on avg. precision, 60.26% on precision, 82.05% on recall and 69.49% on F-score, being this last metric the one used for team ranking. Our goals are to achieve the baseline scores and, if possible, beat the first ranked team.

II. Approach Overview

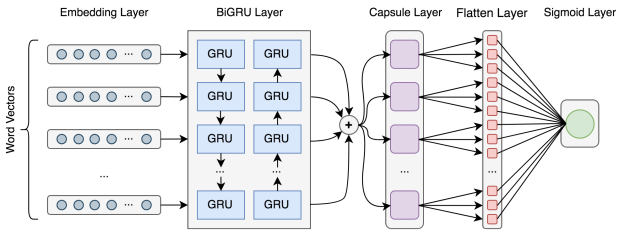

Fig. 1. Overview of our proposed architecture.

[1] https://github.com/ruipsouto/biocreative-vi-track4

In figure 1 we present an overview of our proposed model architecture. We've used a bidirectional Gated Recurrent Neural Network in combination with a Capsule Network. The documents are fed as sequences of tokens into the embedding layer where those tokens are converted to their respective embedding. With the use of the BiGRU and capsules layers we encode the context of the words and identify the most important features, respectively, being then transformed into a 1 dimension vector on the flatten layer so that the sigmoid layer can calculate the final output. In the next sections we explain in a more in-depth manner how we reached such an architecture.

A. Data

Both training and testing datasets are provided by the organizers in JSON and XML formats. The triage training dataset consists of 4082 annotated PubMed documents with and infon tag marking the document as relevant (1729 documents) or not relevant (2353 documents). Each document contains two passages: the title and abstract, and has a document ID corresponding to the article's PubMed ID. The testing dataset consists of 1464 documents, manually classified as relevant (730 documents) or not relevant (734 documents) [7]. We use the BioC package [8] to parse the documents and annotations from XML format.

B. Text Preprocessing

Due to the fact that "pure" text cannot be interpreted by deep learning models, it is necessary to consider some methods with the purpose of obtaining vector representations of words while preserving their syntactic and semantic information. In a bag of words (BoW) method, each document is represented as a vector with the size of the vocabulary where each index indicates the presence or absence of the corresponding word in the corpus. Despite the low complexity, there is a significant loss concerning the syntactic and semantic information and the feature vector space grows with the expansion of vocabulary, which in deep learning networks translates into a high computational cost.

Rather than using a large feature vector space from the bag of words method, we take advantage of another word representation method called Word Embeddings. In word embeddings, each word is represented by a dense vector in a N-dimensional Embedding Space. Unlike the BoW approach, these vector representations are learned by a machine learning model to compact more information on a relatively smaller dimension. One of the advantages of the embeddings is the ability to preserve the syntactic and semantic details of the words, by mapping similar words closer on the embedding space.

For this project, we employed a method of transfer learning using a pre-trained word embedding layer provided by [9], with the purpose of reducing training time and improving overall performance considering that the embeddings were previously estimated on a much larger biomedical corpus.

To begin with the text transformation we performed some basic text preprocessing steps, including a Tokenization task. We break up the data into words, called tokens, that can be used in further processing [10]. Instead of using the Tokenizer utility class from the Keras API, we did our own tokenization in order to use the direct indices of the embeddings, from the pre-trained word embedding vocabulary. Our hypothesis is that the most meaningful words for the model to use in documents classification might be the less frequent ones. Therefore, with a bigger vocabulary there will be lesser unusual words converted to the unknown token on new, unseen documents.

We also decided to use 10% of the training set as validation during fitting.

C. Recurrent Neural Networks

When working with data that has sequential nature, simple neural networks failed on understanding the contextual structure of the sequence. It was unclear how they could reason about the influence of previous output on later input of the sequence. With the development of simple recurrent neural nets [11], a slight change to the classic architecture of the network made the processing of sequential input more efficient. The idea was simple: introducing recurrent connections (as shown in figure 2) allowed the hidden units of the network the usage of previous output as input in the current time step, thus providing the network with a memory mechanism.

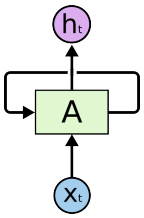

Fig. 2. Folded simple RNN unit. Source: Understanding LSTM Networks

To train this new type of network, a different gradient-based technique was developed, named back-propagation through time (BPTT) [12]. This change in the architecture has helped networks identifying dependencies in the sequences, leading to great advancements in tasks such as natural language processing where the context is important to interpret the meaning of the constituents of the corpus.

Although RNNs can theoretically capture any long-term dependency in an input sequence, it is well-known to be difficult to train an RNN to actually do so[13]. As the BPTT algorithm propagates the error back through the network to update the weights, the gradient is being multiplied by the weights of the recurrent connections. Theses weights are initialised as values close to zero and as these multiplications take place, the gradient gets smaller until it is insignificant. As a result, the initial layers of the network are not trained properly and take more time. This is known as the vanishing gradient problem.

1) Long Short-Term Memory Network: Long Short-Term Memory network (LSTM) were introduced as a variant of the RNNs to solve the problem of the vanishing gradients

[14]. In conjunction with an appropriate gradient-based algorithm, LSTMs back-propagation enforce constant error flow through internal states of the special units, providing that the gradients neither explode nor vanish.

2) Gated Recurrent Unit: The Gated Recurrent Unit (GRU), a variation on the LSTM, appeared later [15]. It adaptively remembers and forgets its state based on the input signal to the unit, by combining the forget and input gates into a single "update gate". This results in a simple and less computationally intensive but competitive model. More recently [16], Bidirectional formats of this variants were introduced, which consists in putting two independent LSTMs (or GRUs) side-by-side. The input sequence is fed in normal time order for one network, and in reverse time order for another. The outputs of the two networks are usually concatenated at each time step.

D. Convolutional Neural Networks

Convolutional Neural Networks, or CNNs, were initially designed to map image data to an output variable, however they've shown promising results when applied to various NLP tasks as well [3]. A simple CNN architecture includes several layers such as embedding, convolution, max-pooling and softmax. Essentially, convolutions are performed on the embedding matrix via linear filters of different sizes. Each filter is applied to each possible window of words to produce a feature map. This is followed by a max-pooling operation, which is applied over the feature map and take the maximum value as as the feature corresponding to this particular filter. The general idea consists in using multiple filters, with varying window sizes, to obtain multiple features. These features form the penultimate layer and are passed to a fully connected softmax layer whose output is the probability distribution over labels. CNN's are good at extracting local and position-invariant features whereas RNN's are better when classification is determined by a long range semantic dependency , when the current step has some kind of relation with the previous steps, rather than some local key-phrases.

E. Capsules Neural Networks

The intuition behind the creation of the capsules network is that CNNs have important drawbacks. One of these drawbacks is that CNNs don't take into account the orientational and relative spatial relationships between the lower-level features that make up the higher-level ones. What this means is that no matter the position or orientation of the main components of an object in a frame, the network can still identify the object given those components are present. Another drawback is the fact that CNNs struggle to identify the same object on different viewing angles. They do not have an understanding of 3D space.

Capsules on the other end, explicitly model this relationships by encapsulating all important information about the state of the feature they are detecting in vector form. They also encode the probability of detection of a feature as the length of their output vector. This is what sets capsules apart. Their ability to store orientational and spatial information without losing the probabilities of detection.[17], [18]

III. System Architecture

A. Input

We have a training set $D = \{d_1, d_2, ..., d_n\}$ of documents, such that each document d_i is labeled with a label l_i from the set $L = \{l_1, l_2, ..., l_n\}$ [10]. The input to the neural network is a document, consisting of a sequence of n words $w = (w_1, w_2, ..., w_n)$, each represented by their corresponding index to the vocabulary V, a set of distinct words in the collection D, where n is the maximum corpus size, that is, the number of words in the document that contains the most words.

B. Embedding Layer

Initially, we use an embedding layer to project the words to the dense embedding space using a word embedding matrix $M \in \mathbb{R}^{|V| \times dim}$ to produce a document matrix $D \in \mathbb{R}^{n \times dim}$, where $|V|$ is the number of words in the vocabulary V and dim is the dimensionality of the word representation vectors, as stated in [19].

$$D = \begin{pmatrix} M[w_1] \\ \vdots \\ M[w_n] \end{pmatrix}$$

As mentioned earlier, the weights of the embedding layer are initialized with the pre-trained word2vec word vectors.

C. Bidirectional Gated Recurrent Unit Layer

The network contains two sub-networks for the left and right sequence context, which are forward GRU $\overrightarrow{f_W}$ and backward GRU $\overleftarrow{f_W}$ pass respectively.

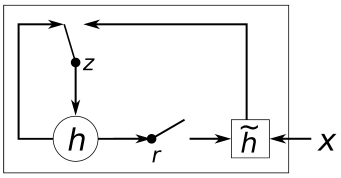

Fig. 3. An illustration of the gated recurrent unit activation function [15].

As shown in Fig. 3, typically, two fundamental components composite the GRU-based recurrent neural networks: one update gate z, which selects whether the hidden state is to be updated with a new hidden state $\tilde{h}$ and a reset gate r to decide whether the previous hidden state is ignored or not with the following set of equations, according to [15].

$$r_j = \sigma([W_r w]_j + U_r h_{\langle t-1 \rangle}]_j) \quad (1)$$

$$z_j = \sigma([W_z w]_j + U_z h_{\langle t-1 \rangle}]_j) \quad (2)$$

where σ is the logistic sigmoid function, and $[.]j$ denotes the j-th element of a vector. w and h_{t-1} are the input and

the previous hidden state, respectively. W_r and U_r are weight matrices which are learned.

The actual activation of the proposed unit h_j is then computed by

$$h_j^{\langle t \rangle} = z_j h_j^{\langle t-1 \rangle} + (1 - z_j)\tilde{h}^{\langle t \rangle} \tag{3}$$

where

$$\tilde{h}_j^{\langle t \rangle} = \phi([Ww]_j + [U(r \odot h_{\langle t-1 \rangle})]_j) \tag{4}$$

The output of the i^{th} word is shown in the following equation.

$$h_i = [\overrightarrow{h_i} \oplus \overleftarrow{h_i}] \tag{5}$$

Here, we use element-wise sum to combine the forward and backward pass outputs.

Long story short, this allows us to get more fine-grained sentence information and ability of capturing semantic dependencies in both directions, that is, from the front to the back of the sentence and from the back to the front.

D. Capsule Layer

Prediction vector $\hat{u}_{j|i}$ is computed via a matrix multiplication of the output vector h_i from the BiGRU layer with a transformation matrix W_{ij}. This encodes really important spatial relationships between low-level features and high-level features within the document.

$$\hat{u}_{j|i} = W_{ij} h_i \tag{6}$$

Following this, we perform a weighted sum of input vectors $\hat{u}_{j|i}$, where c_{ij} are coupling coefficients found using the dynamic routing algorithm [17]. This process determines which higher level capsule the current capsule will send it's output to.

$$s_j = \sum_i c_{ij} \hat{u}_{j|i} \tag{7}$$

Capsules networks use a non-linear "squash" activation function, which takes a vector and shrinks it to have a maximum length of 1 and a minimum length of 0 to still represent probability, while preserving it's orientation.

$$v_j = \frac{\| s_j \|^2}{1 + \| s_j \|^2} \frac{s_j}{\| s_j \|} \tag{8}$$

E. Output Layer

The output of the last capsule layer is then transformed to a vector by the flattening operation and this contains the final and the highest-level features of the corresponding document. The output layer is a fully connected dense layer with a single neuron and a logistic (sigmoid) activation function, which *squashes* a vector in the range (0,1) to predict the probability of the two classes $C = \{C_1, C_2\}$, which can be expressed as relevant and not relevant respectively.

$$p = \sigma(v \times W_c + b_c) \tag{9}$$

v is the document vector, W_c is the parameter matrix containing the neuron weights and b_c is a bias terms vector. We use the binary cross-entropy of the correct labels as training loss, expressed as

$$L = \begin{cases} -\log(f(s_1)) & \text{if } t_1 = 1 \\ -\log(1 - f(s_1)) & \text{if } t_1 = 0 \end{cases} \tag{10}$$

where s_1 and t_1 are the score and the groundtruth label for the class C_1 and $s_2 = 1 - s_1$ and $t_2 = 1 - t_1$ are the score and the groundtruth label for the class C_2.

IV. Results

A. Evaluation

For evaluation purposes, we took advantage of the evaluation script provided by the BioCreative in the original challenge, where text mining predictions were compared to manually annotated data using the standard evaluation metrics, such as precision, recall, F-score and average precision. Doing so, we were able to make a reliable comparison between our proposed solution and the others previously submitted in the original challenge.

B. Training Settings

The weights of the embedding layer are initialized with the pre-trained word2vec word vectors and all models were trained using the binary cross-entropy using Adam algorithm as a optimizer. Our models were implemented using the Keras framework [20] with TensorFlow backend [21] and trained on a machine with 8 CPU cores, 52 GB of memory and a NVIDIA Tesla K80 GPU.

Neural networks are highly prone to overfitting, meaning that they may not be able to generalize to unseen data since the model learned too well the training data. In this kind of situation, it is desirable to learn less to learn better. To avoid this matter, we employed several methods. Firstly, we appllied gaussian noise to the input data, which is a layer that will add noise to inputs of a given shape. The noise has a mean of zero and requires that a standard deviation of the noise be specified as a parameter. Then, we applied dropout to the embedding layer to ignore a certain percentage of the data for updating the neuron weights during the training stage. To stop the training of the network as soon as no significant improvements were achieved, we used EarlyStopping with a patience of 10 epochs and Model Checkpoint to save the best model obtained during training for posterior testing. Both techniques were configured to measure improvements on the F-score metric.

C. Hyperparameters

Tuning the different hyperparameters of a deep learning model is often a "black art" that requires expert experience, unwritten rules of thumb, or sometimes brute-force search [22]. We use a bayesian optimization rather than more straightforward and computationally intensive methods, such as grid search and random search. With a method built upon bayesian inference and gaussian process we attempt to find

TABLE I

BAYESIAN OPTIMIZATION SEARCH PROCESS, INCLUDING SEARCH RANGE AND SELECTED VALUES FOR EACH PARAMETER CONSIDERED.

Parameter	Search Range	Selected Value
Gaussian Noise	0.0 - 0.1	0.07
Embedding Dropout	0.0 - 0.5	0.15
Number of Capsules	8 - 64	8
Dimension of Capsules	2 - 32	16
Capsule Layer Dropout	0.0 - 0.5	0.0
Learning Rate	1e-4 - 1e-2	1e-3

the maximum value of F-score, considering the parameters and search range in Table I in as few iterations as possible.

D. Analysis on Results

As per table II one can visualise that our objective of achieving the best ranking of the teams was not complete. Nevertheless, our model surpassed the baseline model and our results are competitive. Comparing with the participating teams, our model achieved the highest score on the precision metric and is balanced between its other metrics. Some other models who obtained a similar F-score are influenced by the higher value of the recall metric, which means that their model is biased and predicting the major number of samples as relevant. A model with this type of behaviour is of no use, as the number of irrelevant documents retrieved as relevant is considerable and only translates to more confusion to the user of the classifier. Although our approach presented comparative results, the model is still weak as there is a great amount of misclassified documents.

TABLE II

OFFICIAL RANKING. DISPLAYING THE BEST F1 SCORE FOR EACH TEAM IN THE COMPETITION AND OUR SOLUTION.

Rank	Team #	Avg. Prec.	Precision	Recall	F-score
1	418	0.7195	0.6062	0.8205	0.6949
2	374	0.6654	0.5747	0.8699	0.6921
3	421	0.7284	0.6112	0.7945	0.6909
–	**Ours**	**0.6239**	**0.6295**	**0.7386**	**0.6797**
4	433	0.6617	0.5482	0.8877	0.6778
5	420	0.6439	0.5473	0.8712	0.6723
6	419	0.5742	0.5718	0.8068	0.6693
7	414	0.5098	0.5075	0.9795	0.6685
8	375	0.6808	0.5821	0.7575	0.6583
9	405	0.5877	0.5478	0.5575	0.5526
10	379	0.4885	0.4622	0.3438	0.3943

V. CONCLUSION AND FUTURE DIRECTIONS

The intent for this project was to initiate the authors on the topic of NLP tasks using deep learning approaches. The presented problem was an already concluded subtask of the BioCreative VI Track 4 challenge were the participants had to develop a text mining pipeline including text pre-processing, model creation, training and hyperparameters optimisation, for the triage of scientific articles that describe protein-protein interaction.

In the duration of the project, the group encountered numerous problems mainly in the corpus preparation task. At first, there was some confusion in understanding the concept of the word embeddings and the usage of the embedding layer. The primarily tested models showed high fluctuations in the training graphs due to bad input and model preparation. We acknowledge that too much time was spent solving problems on the pre-processing step which left less time to focus on the exploration of different types of model architectures.

Wrapping up on the analysis made on IV-D, even though our results are satisfactory for the intent of this project, there was plenty we wanted to do that we'll leave as future work. From this point, our goal should be increasing the recall without compromising the precision.

On the corpus preparation side, we should thoroughly analyse what portion of the text is being converted to the unknown token. We have the hypothesis that significant words for the model to use in the classification might not be present in the pre trained vocabulary, thus being represented as unknown information. One solution we propose is the substitution of that unknown words by similar words found using the embedding capabilities. We would also like to do experiments using a bigger pre-training embedding.

On the model side, as the Capsule Network showed promising results, we thought of coupling an attention mechanism to the capsules. The premise is that the lower-level capsules could leverage on the encoding made by the attention layer to better identify the intended features.

We have no assurance that this solutions would bring any improvement to our model. Yet, the usage of the CapsNet should be explored as it is a novel implementation backed by a strong intuition and few research has been done regarding its application.

REFERENCES

[1] L. Hill, "Help me understand genetics precision medicine."
[2] R. I. Doğan, S. Kim, A. Chatr-aryamontri, C. Wei, D. Comeau, and Z. Lu, "Overview of the biocreative vi precision medicine track: Mining protein interactions and mutations for precision medicine."
[3] Y. Kim, "Convolutional neural networks for sentence classification."
[4] V. Jayasundara, S. Jayasekara, H. Jayasekara, J. Rajasegaran, S. Seneviratne, and R. Rodrigo, "Textcaps : Handwritten character recognition with very small datasets."
[5] M. Y. Z. L. S. Z. Z. Z. W. Zhao, J. Ye, "Investigating capsule networks with dynamic routing for text classification."
[6] A. Fergadis, C. Baziotis, D. Pappas, H. Papageorgiou, and A. Potamianos, "Hierarchical bidirectional attention-based rnn in biocreative vi precision medicine track, document triage task."
[7] R. I. Doğan, A. Chatr-aryamontri, C. Wei, C. S. Chang, R. Oughtred, J. Rust, L. Boucher, S. Kim, D. C. Comeau, Z. Lu, K. Dolinski, and M. Tyers, "The biocreative vi precision medicine track corpus: Selection, annotation and curation of protein-protein interactions affected by mutations in scientific literature."
[8] W. Liu, R. I. Doğan, D. Kwon, H. Marques, F. Rinaldi, W. J. Wilbur, and D. C. Comeau, "Bioc implementations in go, perl, python and ruby."
[9] S. Pyysalo1, F. Ginter, H. Moen, T. Salakoski, and S. Ananiadou, "Distributional semantics resources for biomedical text processing."
[10] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, and J. B. Gutierrez, "A brief survey of text mining: Classification, clustering and extraction techniques."
[11] J. L. Elman, "Finding structure in time."
[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation."
[13] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult."
[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory."

[15] K. Cho, B. van Merriënboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation."
[16] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks."
[17] G. E. Hinton, S. Sabour, and N. Frosst, "Dynamic routing between capsules."
[18] ——, "Matrix capsules with em routing."
[19] T. Tran and R. Kavuluru, "Exploring a deep learning pipeline for the biocreative vi precision medicine task."
[20] F. Chollet. Keras: Deep learning for humans.
[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, C. O. D. Murray, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems."
[22] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms."
[23] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification."