Hierarchical Bidirectional Attention-Based RNN in BioCreative VI Precision Medicine Track, Document Triage Task

Aris Fergadis[1,3], Christos Baziotis[1,2], Dimitris Pappas[2,3], Haris Papageorgiou[3], Alexandros Potamianos[1,3]
[1]School of ECE, National Technical University of Athens, Zografou 15780, Athens, Greece
[2]Department of Informatics, Athens University of Economics and Business, Patission 10434, Athens, Greece
[3]"Athena" Research and Innovation Center, Maroussi 15125, Athens, Greece

***Abstract*— In this paper, we describe our submission to the "Document Triage Task", of the BioCreative VI Precision Medicine Track, in which we ranked *first* among ten teams. The submitted system is a Hierarchical Bidirectional Attention-Based Recurrent Neural Network (RNN). Our approach utilizes the hierarchical nature of documents, which are composed of sequences of sentences, where sentences are composed of sequences of words. We propose a reusable sequence encoder architecture, which is used as sentence and document encoder. The sequence encoder, is composed of a bidirectional RNN, equipped with an attention mechanism, which identifies and captures the most important elements (words or sentences) in a sequence. Furthermore, we argue that the title of the paper itself, usually contains important information, compared to the other sentences of the abstract. For this reason, we propose a shortcut connection, which integrates the title's vector representation, directly to the final feature representation of the document. We leverage word embeddings, trained on PubMed, for initializing the embedding layer of our network. Moreover, our system does not rely on handcrafted features. Furthermore, we train our system end-to-end using back-propagation, with stochastic gradient descent. We make the source code available to the research community[1].**

***Keywords*— Document Classification; Hierarchical Recurrent Neural Network; GRU; Attention Layer**

I. Introduction

The Document Triage Task is one of the two tasks of the Precision Medicine Track in BioCreative VI. The community challenge that addresses is *"identification of relevant PubMed citations describing mutations affecting protein–protein interactions"* (11).

In this paper we describe our submission to this track, which we approach as a document classification task, where we define the document as the concatenation of the title and abstract of the citation. Usually, document (or text) classification is approached with methods that represent documents with sparse lexical features such as bag-of-words, n-grams, words frequencies (term-frequency and/or inverse-document-frequency – tfidf) and other sophisticated handcrafted features. Linear models or kernel methods, such as variants of Naive Bayes and Support Vector Machines classifiers, are used to train models from those features (20).

Recently, deep neural networks have become popular in NLP tasks, because they can learn underlying features automatically. RNNs have shown great results processing text, especially the variants Long Short-Term Memory (LSTM) (10) and Gated Recurrent Units (GRU) (6) in a variety of tasks (2, 22, 23). Baziotis et al. (2) used a two-layer bidirectional LSTM equipped with an attention mechanism for sentiment analysis in Twitter messages. Yang et al. (22) used a Hierarchical Attention Network with GRU units for document-level sentiment analysis and (23) used a Attention-Based Bidirectional LSTM for relation classification. We employ a hierarchical bidirectional GRU network, equipped with attention layers, which generates dense vector representations for each document and uses those representations as features for classification.

[1]https://github.com/afergadis/BC6PM-HRNN

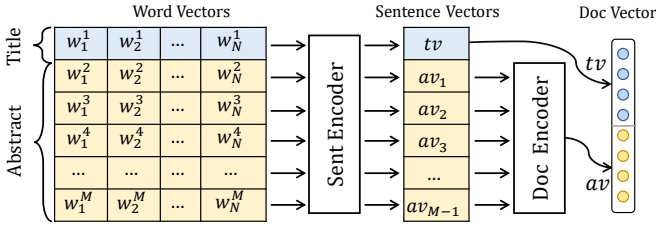

Fig. 1: Overview of our proposed system. *Word Vectors* is an matrix of word embeddings, where M is the maximum number of sentences and N the maximum number of words in a document. tv refers to the *Sent Encoder* representation for the document title and $av_1, av_2, \ldots, av_{M-1}$ to the representations of the abstract's sentences.

II. System Overview

The model we propose is a hierarchical RNN network as shown in Fig 1. We equip the RNN layers with an attention mechanism for identifying the most informative words and sentences in each document. The first level consists of an RNN that operates as a sentence encoder, reading the sequence of words in each sentence and producing a fixed vector representation (sentence vector). Then, a second RNN operates as a document encoder, reading the sequence of sentence vectors and producing the final vector representation for the whole document (title and abstract), which is used as a feature vector for classification. We propose a shortcut connection, which integrates the title's vector representation, directly into the document's vector representation.

A. Text Preprocessing

As a first preprocessing step, we perform sentence segmentation, for splitting the document in it's constituent sentences and tokenization, for splitting the sentences in tokens. We used the Natural Language Toolkit (NLTK) sentence splitter

and word tokenizer to perform these steps (14). We represent each document as a matrix $\mathbf{S} \in \mathbb{R}^{M \times N}$, where M is the maximum number of sentences that a document may have and N is the maximum number of words a sentence may have.

B. Word Embeddings

Word embeddings are dense vector representations of words, capturing their syntactic and semantic information. We leverage the pre-trained word embeddings provided by (18), in order to initialize the weights of the embedding layer of our network. These word embeddings are trained on PubMed articles using word2vec (15) with the skip-gram model and a window size of 5. The dimensionality of the word vectors is 200. Out of vocabulary words, for which we do not have a word embedding, are mapped to a common <UNK> token. We generate the word embedding of <UNK>, by sampling from a uniform distribution, with range $(-0.05, 0.05)$.

C. Recurrent Neural Networks

An RNN processes an input sequentially by performing the same operation $h_t = f_W(x_t, h_{t-1})$ on every element of a sequence, where h_t is the hidden state at time-step t, x_t the input at time-step t, h_{t-1} the hidden state at the previous time-step and W the weights of the network. After reading the whole input sequence, h_t holds a summary of the input, which is used as it's vector representation. Vanilla RNNs suffer for the problem of vanishing gradients (3), which limits their ability to learn long-term dependencies. In order to overcome this limitation, more sophisticated variants of the RNN have been proposed, such as the LSTM (10) or GRU (6), which introduce a gating mechanism in order to ensure proper gradient propagation through the network. In our experiments we used the GRU variant, which has a simpler architecture and achieved better results in our experiments.

D. Attention Mechanism

RNNs have the ability to produce fixed vector representations, for sequences of variable length. The RNN reads each element from the sequence and updates it's hidden state, which by the end, holds a summary of the information contained in the sequence. The output produced at the last time–step, usually a learned non-linear transformation of the hidden state, is used as the vector representation of the whole sequence. However, especially in longer sequences, the RNN does not have the capacity to hold all the relevant information in it's state. This means that the information of certain elements may fade away. In order to amplify the contribution of the important elements in each sequence (words or sentences) we utilize an attention mechanism (6, 8). The attention mechanism assigns a weight to the output from each timestep and the final representation is a weighted combination of all the outputs.

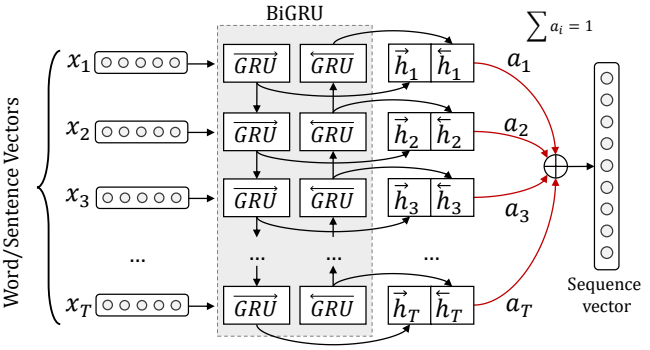

Fig. 2: Architecture of our proposed sequence encoder. The same architecture is used for encoding a sequence of word vectors to a sentence vector (sentence encoder) and a sequence of sentence vectors to a document vector (document encoder).

III. System Architecture

The input to the network is a document, which is composed of a title and an abstract. We perform sentence segmentation and obtain a list of sentences for each document, where the first sentence is the title and the rest of the sentences are the abstract. We feed the list of M sentences to our neural network.

A. Embedding Layer

Given a sentence s^i, which is the i-th sentence of a document, consisting of a sequence of N words $s^i = (w_1^i, w_2^i, \ldots, w_N^i)$, we use an embedding layer to project them to low-dimensional vector space $\mathbb{R}^E$, where E is the size of the embedding layer. We initialize the weights of the embedding layer, with the pre-trained word2vec word vectors (Section II-B).

B. Sentence Encoder

After embedding the words to the low-dimensional semantic space, we use the sequence encoder, in order to obtain a vector representation for each sentence. The sequence encoder consists of a Bidirectional GRU (BiGRU) with an attention layer, which reads the sequence of word vectors of each sentence and produces a sentence vector. The architecture of the sequence encoder, is shown in Fig. 2.

Bidirectional GRU. A GRU takes as input the sequence of word vectors of a sentence and produces a sequence of word annotations (output), $H = (h_1, h_2, ..., h_N)$, where h_i is the hidden state of the GRU at time-step i, summarizing all the information of the sentence up to w_i. We use bidirectional GRU (BiGRU) in order to capture the contextual information of the words from both their left and their right context. A bidirectional GRU consists of a forward GRU $\overrightarrow{f_W}(\cdot)$ that reads the sentence from w_1 to w_N and a backward GRU $\overleftarrow{f_W}(\cdot)$ that reads the sentence from w_N to w_1. We obtain the final annotation for each word w_i, by concatenating the annotations from both directions,

$$h_j^i = \overrightarrow{h_j^i} \parallel \overleftarrow{h_j^i}, \quad j \in [1 \ldots N], \quad h_j^i \in \mathbb{R}^{2S} \tag{1}$$

where $\|$ denotes concatenation, $\overrightarrow{h_j^i}$ and $\overleftarrow{h_j^i}$ are the hidden states for the forward and backward GRU respectively at time-step *j* and S the size of the sentence-level GRU layer.

Attention Layer. We use an attention layer in order to identify the most important words in each sentence and enforce their contribution to the final sentence vector. The attention layer, assigns a weight a_j^i to each word annotation h_j^i. The sentence vector sv^i, which is the vector representation of the whole sentence, is computed as the weighted sum of all the word annotations h_j^i.

$$e_j^i = tanh(W_w h_j^i + b_w) \tag{2}$$

$$a_j^i = \frac{exp(e_j^i)}{\sum_{t=1}^{N} exp(e_t^i)}, \quad j \in [1 \dots N] \tag{3}$$

$$sv^i = \sum_{j=1}^{N} a_j^i h_j^i, \quad sv^i \in \mathbb{R}^{2S} \tag{4}$$

where W_w, b_w are the attention layer's weights and bias and sv^i is the vector representation of the *i*-th sentence.

Moreover, we denote the sentence vector of the title as $tv = sv^1$ and the sentence vectors of the abstract as $av_i = sv^i, i \in [2 \dots M]$.

C. Document Encoder

After producing the vector representations for each sentence, we feed them to the document encoder, in order to obtain the final vector representation for the whole document. Notably, we do not feed the vector of the title tv to the sentence encoder, but only the vectors of the abstract av_i. The rationale behind this decision is discussed in Section IV. The remaining sentence vectors, $av_1, av_2, \dots, av_{M-1}$ are feed to the document encoder, in order to get the vector representation of the whole abstract av. The architecture of the document encoder, which is identical to the sentence encoder, is shown in figure 2.

Bidirectional GRU. Similar to the sentence encoder, we use a BiGRU in order to get annotations for each abstract vector av_i, which summarizes the information form the sentences around sentence *i*.

$$h_i = \overrightarrow{h_i} \parallel \overleftarrow{h_i}, \quad i \in [1 \dots M-1], \quad h_i \in \mathbb{R}^{2D} \tag{5}$$

where $\|$ denotes concatenation, $\overrightarrow{h_i}$ and $\overleftarrow{h_i}$ are the hidden states for the forward and backward GRU respectively at time-step *i* and D the size of the document-level GRU layer.

Attention Layer. We use an attention layer in order to identify the most informative sentences of the abstract and enforce their contribution to the final vector representation av. The attention layer assigns a weight a_i, to each sentence annotation and we aggregate them by computing the weighted sum of all the sentences annotations.

$$e_i = tanh(W_a h_i + b_a) \tag{6}$$

$$a_i = \frac{exp(e_i)}{\sum_{t=1}^{M-1} exp(e_t)}, \quad i \in [1 \dots M-1] \tag{7}$$

$$av = \sum_{i=1}^{M-1} a_i h_i, \quad av \in \mathbb{R}^{2D} \tag{8}$$

where W_a, b_a are the layer's weights and bias.

D. Output Layer

The final document vector d is computed by concatenating the representations of title and abstract vectors

$$d = tv \parallel av, \quad d \in \mathbb{R}^{2S+2D} \tag{9}$$

The output layer is a fully connected layer with single neuron and a logistic (sigmoid) activation function, which performs the binary classification (logistic regression). It uses the documents vector representation d as feature vector to predict the probability of the two classes.

IV. Hyper-parameters and Training Details

Title Vector Shortcut Connection. Instead of feeding the title vector tv in the sentence encoder with the rest of the sentence vectors (abstract), we create a shortcut connection, by integrating it directly to the final document feature vector d. The reasons for this design decision are twofold. First of all, we argue that the title of a paper contains a lot of important information, which will be diluted if passed in the document encoder with the other sentences, even with the addition of the attention mechanism. By integrating the title vector tv directly into the document feature vector d, we keep the title information intact. Secondly, the sentence encoder is regularized, because (1) it has to produce self-contained representations, which can be used directly as features for classification and (2) it has to produce informative representations as input to the document encoder. This means that the weights of the sentence encoder (sentence BiGRU and Attention layer), receive gradients from both directions: from the output layer and from the document encoder.

Regularization. Neural networks are notoriously prone to over-fitting (13). For this reason we adopt a series of measures, in order to regularize our model.

First, we add Gaussian noise to the input (embedding layer), which limits the amount of information that can be stored in a network (9). This can also be interpreted as a random data augmentation technique, distorting the representation of the words, which means that practically the network never sees the exact same sentence more than once, during training. We add noise by sampling from a zero-mean Gaussian distribution at each batch.

Moreover, we apply dropout to the layers of the network. Dropout randomly disables a certain proportion of the neurons in a layer on each training example (or batch). This means that for each training example, a subpart of the network is trained, which can be thought as a model

ensembling technique. Dropout improves the network's performance because it forces each neuron to learn disentangled features. This way the network learns to recognize the same patterns in multiple ways, which leads to a better model (19). We apply dropout on the embedding layer and on the sentence and document encoders, both on their BiGRU layers and their attention layers.

Many methods have been used to improve stochastic gradient descent such as momentum, annealed learning rates and L2 weight decay. We use Adam (12) optimizer, with the standard deterministic cross-entropy objective function. We add a L2 penalty term (weight decay) to the objective function, in order to discourage large weights and we clip the norm of the gradients at 5 to prevent exploding gradients (16).

As a last step, we perform early-stopping. We stop the training of the network, when the f1-score of the development set stops increasing for a certain number of epochs (17). We decided to monitor f1-score instead of the loss of the development set because its the official evaluation metric used and this way we directly optimize our model for the task.

Hyper-parameter Optimization. The hyper-parameter tuning in neural networks is a very challenging process. In addition to the time consuming training of the neural network, usually we have to tune a lot of hyper-parameters, which are highly correlated (e.g. increasing the number of neurons, changes the optimal dropout rate). As it has been shown in (4), grid search is very inefficient and random search finds consistently better models. However, in our work we adopt the Bayesian optimization approach (5), in order to perform a smart search in the high-dimensional hyper-parameter space. This way, we obtain a set of reasonable hyper-parameters, in a very small number of trials. Table I shows the optimal hyper-parameter values that we obtained.

Experimental Setup. We used Keras (7) to develop our model with Tensorflow (1) as backend. The network was trained on a GTX1070 for approximately 30 minutes.

V. Results

We participated in BioCreative VI Precision Medicine Track, in the "Document Triage Task" as team *418*. According to the official ranking, based on the F1 evaluation metric, our team ranked 1st out of 10 teams, as shown in the Table II. The training dataset, consists of 4082 PubMed citations (abstracts) as shown in (Table III). 1730 are annotated as positive (*relevant*), which means that these abstracts specifically describe protein–protein interaction influenced by genetic mutations. The *test* dataset has a total of 1500 PubMed abstracts.

TABLE I: Hyper-parameter values of our proposed model.

	Layer	Size	Dropout	Noise (σ)
	Embedding	200	0.2	0.2
Sent. Encoder	GRU	150 (x2)	0.3	-
	Attention	1	0.3	-
Doc. Encoder	GRU	150 (x2)	0.3	-
	Attention	1	0.3	-

TABLE II: Official Ranking. Displaying the best F1 score for each team.

Rank	Team #	Avg. Prec.	Precision	Recall	F1
1	418	0,7195	0,6026	0,8205	**0,6949**
2	374	0,6654	0,5747	0,8699	0,6921
3	421	0,7284	0,6112	0,7945	0,6909
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

TABLE III: Training data set of Document Triage Task.

Positive	Negative	Total
1730 (42%)	2352 (58%)	4082

For our submission we trained our model on 95% of the training corpus and used a held-out 5% as a development set for early-stopping. We adopted this approach in order to ensure that the model used for our submission, did not overfit on the training set. We stopped training when the $F1$ of the development set stopped increasing, optimizing this way our model, directly for the evaluation metric of the task.

VI. Conclusion

We presented a system for the "Document Triage Task" of the BioCreative VI Precision Medicine Track. The key points of the system are: (a) utilizes the hierarchical nature of documents, which are composed of sequences of sentences and sentences are composed of sequences of words, (b) uses a shortcut connection which integrates the document's title directly to the final feature representation of the document, (c) does not utilize any handcrafted feature, and (d) our system is trained end-to-end using back-propagation, with stochastic gradient descent. As future work in this domain, in order to further improve our model, we propose a series of approaches.

Text Preprocessing. Sentence splitting, word tokenization and named entity extraction are crucial steps in text preprocessing. The output of these preprocessing steps has a big impact on the performance of the models. Using a tokenizer tailored for biomedical data, will have the ability to better identify sentence boundaries and difficult tokens such as genes names.

Entity Embeddings. Entities annotations for genes and mutations mentions can be obtained using tools, such as NCBI text-mining web services (21). We plan to utilize these annotations, by representing them as dense vectors which can be appended to the word vectors fed in our model. We hypothesize that this is will improve even further our performance.

Character-level model. An alternative approach would be to design a model, which would operate on the character-level. We argue that such a model, will be able to better model expressions like protein or gene names, reducing the need for careful and laborious text preprocessing.

REFERENCES

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *Computing Research Repository (CoRR)*, abs/1603.04467.

2. 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. *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, (1):747–754.

3. Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

4. Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

5. Bergstra, J., Yamins, D., and Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pages 115–123.

6. Cho, K., van Merrienboer, B., Gülçehre, c., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Computing Research Repository (CoRR)*, abs/1406.1078.

7. Chollet, F. and others (2015). Keras.

8. Graves, A. (2013). Generating sequences with recurrent neural networks. *Computing Research Repository (CoRR)*, abs/1308.0850.

9. Hinton, G. E. and van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, pages 5–13. ACM.

10. Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

11. Islamaj, R., Chatr-aryamontri, A., Kim, S., Wei, C.-H., Peng, Y., Comeau, D. C., and Lu, Z. (2017). BioCreative VI Precision Medicine Track : Creating a training corpus for mining protein-protein interactions affected by mutations. *Association for the Computational Linguistics (ACL) Anthology*, pages 171–175.

12. Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *Computing Research Repository (CoRR)*, abs/1412.6980.

13. Lawrence, S., Giles, C., and Tsoi, A. C. (1997). Lessons in neural network training: Overfitting may be harder than expected. *14th National Conference on Artificial Intelligence*, pages 540–545.

14. Loper, E. and Bird, S. (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA. Association for Computational Linguistics.

15. Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *Neural Information Processing Systems (NIPS)*, pages 1–9.

16. Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

17. Prechelt, L. (2012). Early stopping—but when? In *Neural Networks: Tricks of the Trade*, pages 53–67. Springer.

18. Pyysalo, S., Ginter, F., Moen, H., Salakoski, T., and Ananiadou, S. (2013). Distributional Semantics Resources for Biomedical Text Processing. In *Proceedings of Languages in Biology and Medicine (LBM), 2013*.

19. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

20. Wang, S. and Manning, C. (2012). Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 90–94.

21. Wei, C.-H., Leaman, R., and Lu, Z. (2016). Beyond accuracy: creating interoperable and scalable text-mining web services. *Bioinformatics*, 32(12):1907–1910.

22. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical Attention Networks for Document Classification. *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2016)*, pages 1480–1489.

23. Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., and Xu, B. (2016). Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 207–212.