Abstract

In order to comprehend underlying biological processes, it is necessary to identify interactions between proteins. It is typically quite difficult to extract a protein-protein interaction (PPI) from text data as text data is complex in nature. Unlike sequential models, tree-structured neural network models have the ability to consider syntactic and semantic dependencies between different portions of the text and can provide structural information at the phrase level. This paper investigates tree-structured neural network models for the PPI task and the results show their supremacy over sequential models and their effectiveness for this task.

Introduction

As the scientific literature grows at an exponential rate, the vast majority of biological information is currently available in text form residing in the scientific literature. MEDLINE database’s size has grown by 4.2 percent annually during the last two decades and currently it contains approximately 26,000,000 records extracted from 5639 publications which is 23% more than what it contained in 2014 (Yadav et al. 2020). This huge amount of unstructured text from biomedical research articles is a valuable source of information for the biomedical natural language processing (NLP) domain.

As the volume of biomedical data continues to grow exponentially and due to the inherent complexity in the textual representations of these data, it is critical to pursue automatic information retrieval techniques to aid biologists in the detection and identification of useful information, and the arranging and maintaining of databases, as well as providing automatically generated decision support systems for medical professionals. Considering this issue, a lot of research has been conducted for inferring information concealed in these texts to assist health care and biomedical people, such as protein-protein interactions (PPIs), chemical-disease relation extraction, clinical relation extraction, drug-drug interactions, etc., as retrieving important information manually from this large volume of texts is both time consuming and expensive (Peng, Wei, and Lu 2016).

The majority of biological activities inside a cell, such as immune response, signal transduction, cellular organization, etc., are caused by different interactions between various proteins (Sledzieski et al. 2021). So, identifying the protein-protein interactions (PPIs) provides a better understanding of the functionalities, regulations, and communication between different proteins (Yao et al. 2019). Identifying PPIs entails figuring out how different proteins mentioned in a text are connected (Krallinger et al. 2008). This information may spread out through different parts of the whole document, however, the current work is restricted to identify PPIs present only inside single sentences (Pyysalo et al. 2008; Tikk et al. 2010). For instance, “LEC induced maximal migration of CCR1 and CCR8 transfected cells at 89.3 nmol/L and cell adhesion at 5.6 nmol/L.” (Howard et al. 2000) reflects two PPI relations: LEC-CCR1 and LEC-CCR8 and no association between CCR1 and CCR8.

For such tasks, sequential deep learning-based models have been used in multiple research works (Hsieh et al. 2017; Yadav et al. 2019). However, if the data is structured rather than presented sequentially, these models are more likely to miss the underlying semantic compositionality (Ahmed, Samee, and Mercer 2019a) as they consider word order only but no linguistic structure (Li et al. 2015). By contrast, Recursive neural networks, commonly known as tree-structured neural network models, work over parsed tree representations of the sentences and thus preserve both the syntactic and semantics in a better way.

In this paper, we investigate six tree-structured neural network models for the PPI identification task. For working with dependencies between words in different portions of the sentence, we investigate dependency tree-structured neural nets, whereas to work with the phrase level information, constituency tree-structured neural nets are explored. Finally, two ensembles of these models are used for retrieving the PPIs present in the sentences. We provide an ample analysis of these models’ performances over the benchmark PPI datasets which evince the supremacy of using tree-structured neural networks over sequential ones for this task.

Related Works

Numerous NLP methods have been developed for determining the associations between protein entities. In the initial stages, pattern-based techniques were widely used. In these approaches, on the basis of syntactic as well as lexical features, pattern-based rules were designed for extracting
the relationship (Blaschke et al. 1999; Leeuwenberg et al. 2015). However, these models were not capable of handling complex relationships specified in relational and coordinating clauses appropriately. In contrast to naïve pattern-based methods, dependency-based methods are more syntax attentive and offer a wider range of application coverage (Erkan, Özgür, and Radev 2007; Miyao et al. 2009).

Another prominent approach for extracting such relationships is to use kernel-based methods. These models learn profuse structural information by means of dependency structures and syntactic parse trees (Miwa et al. 2009; Kim et al. 2010). Some noteworthy methods use bag-of-words kernel (Sietre, Sagae, and Tsujii 2007), tree kernel (Zhang et al. 2006), convolution tree kernel (Chang et al. 2016), neighbourhood hash graph kernel (Zhang et al. 2011), and walk-weighted kernel (Kim et al. 2010).

Following work, Yadav et al. (2019) utilized structured attention over the vectors are used as the feature vectors for the classification task. Yadav et al. (2019) utilized structured attention over the sentences, and the vectors concatenated from the left and right-most LSTM output

A sentence can be represented by two tree-structured representations: constituency and dependency trees (Chen and Manning 2014). These representations provide syntactical information about the sentence by preserving word to word dependencies (dependency tree) and phrase level information (constituency tree). To utilize these structural syntactic information sources, Tai, Socher, and Manning (2015) introduced dependency (child sum tree-LSTM) and constituency (N-ary tree-LSTM) tree-LSTMs.

For the child sum tree, the internal gates of a component node are updated by the summed hidden state values of its child nodes. Then, using this updated hidden state value the other intermediate gates are updated as follows:

$$\tilde{h}_j = \sum_{\kappa \in C_j} h_{j\kappa} \tag{1}$$

$$i_j = \sigma(W_i x_j + U_i \tilde{h}_j + b_i) \tag{2}$$

$$o_j = \sigma(W_o x_j + U_o \tilde{h}_j + b_o) \tag{3}$$

$$\tilde{c}_j = \tanh(W_c x_j + U_c \tilde{h}_j + b_c) \tag{4}$$

Here, $W$’s and $b$’s are weights and bias values, and $C_j$ is the set of child nodes. In the child sum tree-LSTM, for each child node, there is a separate forget gate ($f_{j\kappa}$) which allows the model to selectively incorporate information for the parent node from the child nodes. For each child node, the corresponding cell state and forget gate values are then multiplied and finally all of these values are combined together to compute the forget gate value of the parent node. Then, the cell state ($c_j$) and hidden state ($h_j$) values of the parent node are computed using this forget gate value as follows:

$$f_{j\kappa} = \sigma(W_f x_j + U_f \tilde{h}_j + b_f) \tag{5}$$

$$\tilde{f}_j = \sum_{\kappa \in C_j} f_{j\kappa} \cdot c_{\kappa} \tag{6}$$

$$c_j = i_j \cdot \tilde{c}_j + \tilde{f}_j \tag{7}$$

$$h_j = o_j \cdot \tanh(c_j) \tag{8}$$

In the N-ary tree-LSTM, each parent node contains identical cell and hidden states for each of its children. The internal gate values and forget gates are computed as follows:

$$i_j = \sigma(W_i x_j + \sum_{i=1}^{N} U_{i,j} \tilde{h}_j + b_i) \tag{9}$$

$$o_j = \sigma(W_o x_j + \sum_{i=1}^{N} U_{o,j} \tilde{h}_j + b_o) \tag{10}$$

$$\tilde{c}_j = \tanh(W_c x_j + \sum_{i=1}^{N} U_{c,j} \tilde{h}_j + b_c) \tag{11}$$

$$f_{j\kappa} = \sigma(W_f x_j + \sum_{i=1}^{N} U_{f,j\kappa} \tilde{h}_j + b_f) \tag{12}$$

Just like in the child sum tree-LSTM, the final forget gate of the parent node is computed by multiplying the corresponding forget gate and cell state values and then summing them

**Tree-LSTMs**

A sentence can be represented by two tree-structured representations: constituency and dependency trees (Chen and Manning 2014). These representations provide syntactical information about the sentence by preserving word to word dependencies (dependency tree) and phrase level information (constituency tree). To utilize these structural syntactic information sources, Tai, Socher, and Manning (2015) introduced dependency (child sum tree-LSTM) and constituency (N-ary tree-LSTM) tree-LSTMs.

For the child sum tree, the internal gates of a component node are updated by the summed hidden state values of its child nodes. Then, using this updated hidden state value the other intermediate gates are updated as follows:

$$\tilde{h}_j = \sum_{\kappa \in C_j} h_{j\kappa} \tag{1}$$

$$i_j = \sigma(W_i x_j + U_i \tilde{h}_j + b_i) \tag{2}$$

$$o_j = \sigma(W_o x_j + U_o \tilde{h}_j + b_o) \tag{3}$$

$$\tilde{c}_j = \tanh(W_c x_j + U_c \tilde{h}_j + b_c) \tag{4}$$

Here, $W$’s and $b$’s are weights and bias values, and $C_j$ is the set of child nodes. In the child sum tree-LSTM, for each child node, there is a separate forget gate ($f_{j\kappa}$) which allows the model to selectively incorporate information for the parent node from the child nodes. For each child node, the corresponding cell state and forget gate values are then multiplied and finally all of these values are combined together to compute the forget gate value of the parent node. Then, the cell state ($c_j$) and hidden state ($h_j$) values of the parent node are computed using this forget gate value as follows:

$$f_{j\kappa} = \sigma(W_f x_j + U_f \tilde{h}_j + b_f) \tag{5}$$

$$\tilde{f}_j = \sum_{\kappa \in C_j} f_{j\kappa} \cdot c_{\kappa} \tag{6}$$

$$c_j = i_j \cdot \tilde{c}_j + \tilde{f}_j \tag{7}$$

$$h_j = o_j \cdot \tanh(c_j) \tag{8}$$

In the N-ary tree-LSTM, each parent node contains identical cell and hidden states for each of its children. The internal gate values and forget gates are computed as follows:

$$i_j = \sigma(W_i x_j + \sum_{i=1}^{N} U_{i,j} \tilde{h}_j + b_i) \tag{9}$$

$$o_j = \sigma(W_o x_j + \sum_{i=1}^{N} U_{o,j} \tilde{h}_j + b_o) \tag{10}$$

$$\tilde{c}_j = \tanh(W_c x_j + \sum_{i=1}^{N} U_{c,j} \tilde{h}_j + b_c) \tag{11}$$

$$f_{j\kappa} = \sigma(W_f x_j + \sum_{i=1}^{N} U_{f,j\kappa} \tilde{h}_j + b_f) \tag{12}$$

Just like in the child sum tree-LSTM, the final forget gate of the parent node is computed by multiplying the corresponding forget gate and cell state values and then summing them

**The Model**

This section describes our work in detail. Our investigation of the PPI relation extraction task examines four tree-structured neural network models: dependency and constituency tree-LSTMs with a self attention mechanism and tree-transformers. Their working principles are discussed first. Two ensembles of these models are then discussed.
(Eq. 13). The cell state (Eq. 7) and new hidden state (Eq. 8) values are computed as before.

\[
\hat{f}_j = \sum_{i=1}^{N} f_{ji} \cdot c_{ji}
\]

Ahmed, Samee, and Mercer (2019a) introduced self attention for such tree structured recursive neural networks. It incorporates three matrices: query, key, and value. They are calculated as follows:

\[
\text{key} = \omega_k M_k \quad \text{s.t.} \quad \omega_k \in \mathbb{R}^{d \times d} \tag{14}
\]

\[
\text{value} = \omega_v M_v \quad \text{s.t.} \quad \omega_v \in \mathbb{R}^{d \times d} \tag{15}
\]

\[
\text{query} = \omega_q M_q \quad \text{s.t.} \quad \omega_q \in \mathbb{R}^{d \times d} \tag{16}
\]

For the child sum tree, the \( M \)s are the concatenations of all of the child nodes’ word vectors for a corresponding parent node, whereas in the N-ary tree-LSTM the word vectors under a constituent are concatenated. Then these \( \text{key, value, and query} \) matrices are aligned considering the representation’s dimension (Eq. 17).

\[
\text{align} \in \mathbb{R}^{n \times n} = (\text{query})^T \text{key} \cdot (1/\sqrt{d}) \tag{17}
\]

where \( n \) is the number of offspring nodes under any particular parent node and \( d \) is the normalizing factor. Then, \( \text{softmax} \) is applied over this \( \text{align} \) matrix to compute the attention probability matrix \( \alpha \in \mathbb{R}^{n \times n} \). Finally, batch-wise matrix multiplication is applied between the attention matrix \( \alpha \) and the matrix \( \text{value} \) to compute the attentive hidden states \( \tilde{h}_i \in \mathbb{R}^{n \times d} \). Rows of this matrix are concatenated to produce the final hidden representation of the parent node for both the child sum and N-ary tree-LSTM.

**Tree-Transformers**

Ahmed, Samee, and Mercer (2019b) applied the concept of transformer (Vaswani et al. 2017) over the constituency and dependency trees, and introduced two tree-transformer models: constituency tree-transformer and dependency tree-transformer. Both of these models apply multi-branch attention over the child nodes’ representations. Just like the self attention mechanism, this approach also uses \( \text{key, query, and value} \) matrices as follows:

\[
\alpha = \text{softmax} \left( \frac{\text{query} \cdot \text{key}^T}{\sqrt{d_k}} \right) \text{value} \tag{18}
\]

where \( d_k \) is the dimension of the key. For the multi-branch attention \( (\beta_i) \), \( n \) copies of \( \text{key, query, and value} \) matrices are created with the appropriate weight matrices \( \omega_i \), where \( n \) is the number of branches, and finally a scaled dot product attention (Eq. 18) is applied over each branch (Eq. 19).

\[
\beta_i = \alpha_{i \in [1, n]}(\text{query}_i, \omega_1^{\text{query}}, \text{key}_i, \omega_1^{\text{key}}, \text{value}_i, \omega_1^{\text{value}}) \tag{19}
\]

A residual connection is then employed over these tensors followed by a layer-wise batch normalization layer. A scaling factor \( \tau \) is applied in the end to produce the branch representation (Eq. 20). Following this, position-wise CNN (PCNN) is applied over every \( \beta_i \) (Eq. 21). The attention-encoded representations of these semantic subspaces are computed by applying weighted summation where each \( \gamma_i \in \mathbb{R}^{n} \) is a hyperparameter (Eq. 22). In the end, with BranchAttn, another residual connection is employed. This is then fed to a tanh layer and an element-wise summation (EWS) is performed to generate the parent node representation (Eq. 23). Here, \( \chi \) and \( \chi \) represent the input and the outcome of the attention module, respectively.

\[
\tilde{h}_i = \text{LayerNorm}(\beta_i \omega_i^b + \beta_i) \times \tau_i \tag{20}
\]

\[
\text{PCNN}(x) = \text{Conv}(\text{Relu}(\text{Conv}(x) + b_1)) + b_2 \tag{21}
\]

\[
\text{BranchAttn} = \sum_{i=1}^{n} \gamma_i \text{PCNN}(\tilde{h}_i) \tag{22}
\]

\[
\text{ParentNodeRep} = \text{EWS}(\text{tanh}((\chi + \chi)\omega + b)) \tag{23}
\]

**Ensemble Architecture**

After exploring the tree-structured LSTMs and transformer models, we investigated two ensemble models. In the first approach, we train all the models, and then, when testing, each sentence is fed to all of the models. All of the models predict the class label individually. Finally, a winner takes all method (Roy et al. 2018) is applied over these individual models’ selections for the final class prediction. In our second approach, we utilize only the dependency and constituency tree-transformers. Each sentence is fed to both of the tree-transformers and then the sentence representations are concatenated and then fed to the following MLP for class label prediction. The intention behind investigating this model is to find out what happens if features containing both word-level dependencies and phrase-level information are used for the PPI relation extraction task. Figure 1 provides a sketch of this ensemble architecture.

**Experiments and Performance Analysis**

This section reports the results found for the tree-structured neural network models and the ensemble architectures with F1-score as the performance evaluation metric. It also provides a statistical description of the five standard benchmark PPI corpora for this task along with the pre-processing steps. The PPI problem has been formulated as a classification task. Finally, the performance of the tree-structured models are compared against the most prominent sequential and the previous tree-structured architectures used to solve this task.

For evaluating the investigated tree-structured neural networks, the models are tested on the five standard PPI corpora: AIMed (Bunescu et al. 2005), BioInfer (Pyysalo et al. 2007), IEPA (Ding et al. 2001), HPRD50 (Fundel, Küffner, and Zimmer 2007), and LLL (Nédélec 2005). For all the experiments, the converted version of the corpora is used as mentioned by Ahmed et al. (2019). All of the protein names in all five corpora are substituted with three special symbols: PROT0, PROT1 and PROT2. If any two proteins in a sentence are being considered as interacting with each other, they are replaced with PROT1 and PROT2. All the mentioned proteins in the sentence which are not being considered for interaction identification are replaced with PROT0. As an example, the sentence “LEC induced maximal migration of CCR1 and CCR8 transfected cells at 89.3 nmol/L and
For the tree-transformer models, the initial learning rate is 0.1 and the same learning rate decay approach is used. Six PCNN layers are used in the multi-branch attention block. For the experiments, six branches of attention layer are used. Each PCNN layer is composed of 2 CNNs. The first CNN layer employs 341-dimension kernels without any dropout and 300-dimension kernels are used in the second layer of the CNN. In the second layer, dropout 0.1 is used in all cases just like Ahmed, Samee, and Mercer (2019b). The hyperparameters of the tree-transformers are updated using the ‘Adagrad’ optimizer. All of the models (both tree-LSTMs and tree-transformers) are fed with Bio-RoBERTa (Gururangan et al. 2020) word embeddings which are not updated during training. We also tried fasttext (Bojanowski et al. 2017) and Bio-WordVec (Zhang et al. 2019) embeddings. However, the best results are with the Bio-RoBERTa embeddings.

Table 2 shows the performance of the tree-structured LSTMs, transformers, and the ensemble architectures over the five benchmark PPI corpora and some prominent sequential and tree-structured models for comparability. Among these five, AIMed contains many erroneous annotations. In addition, having nested named entities makes it more difficult to work with (Ahmed et al. 2019).

From this table it is clearly visible that all of the tree-structured models outperform the sequential models for this task. Among the two investigated tree-LSTM models (child sum and N-ary treeLSTM), the child sum tree-LSTM with self-attention performs slightly better than the N-ary tree-LSTM with self attention (average F1-scores are 84.67% and 84.28% accordingly). Overall, both of the investigated tree-transformer models perform better than the tree-LSTM models. However, among the tree-transformer architectures, the dependency tree-transformer (DT-transformer) performs better than the constituency tree-transformer (CT-transformer) and it happens for all five corpora. So, these two observations suggest that neural networks based on dependency trees perform slightly better than the models built on constituency trees. The reason behind this may be that because the sentences here are quite complex in nature, word-level dependency provides more useful information.

For each dataset among these four standalone models, DT-Transformer has the highest F1-scores (87.88%, 95.37%, 82.56%, 88.01%, and 91.46% for the AIMed, BioInfer, IEPA, HPRD50, LLL datasets, accordingly). For the Ensemble - Winner Takes All model, a little performance boost is achieved for four datasets. For HPRD50, the F1-score is a bit less than the two transformer models. It achieves a better average F1-score compared to all of the standalone mod-
els. The DT+CT-Transformer model combines features from both of the constituency and dependency tree transformers. The reasons behind choosing only the tree-transformer-based models are that both of the transformer-based models perform better than the LSTM-based models and both the word dependency-level and phrase-level information have already been provided by the transformer-based models. The DT+CT-Transformer outperforms all other models for every dataset with an average F1-score 89.70%. Furthermore, this approach is computationally less expensive compared to the previously mentioned ensemble model as that method requires four models to be trained whereas for the DT+CT-Transformer only two models and an additional MLP are required to be trained. Additionally, the results can be explained by means of the attention value on each node as presented by Ahmed, Samee, and Mercer (2019b).

**Conclusions**

In this work, we have explored various tree-structured neural network models for the PPI relation extraction task. The experimental results show that the tree-structured models, because of having additional syntactical information at word dependency and phrase-level, perform better than the sequential models. Among all of the explored models, the combined model with both the dependency and constituency tree-transformers performs the best as it utilizes both the word dependency and constituency information. However, opportunities for improvement in this field remain. In the future we want to explore graph-based neural network models with attention mechanisms, and to leverage additional features for this task. Further analysis of results based on AUC and ROC curves can be performed.

**Acknowledgments**

We thank the reviewers for their constructive comments. This research is partially funded by The Natural Sciences and Engineering Research Council of Canada (NSERC) through a Discovery Grant to R. E. Mercer.

### References


