RELATION EXTRACTION BASED ON DEEP LEARNING

Zihao Chu

ABSTRACT

Relation Extraction (RE) is a vital task in natural language processing (NLP). In this task, it aims to distinguish the relation type between two nominals given in the text and this technology has been widely applied in a large variety of fields, such as constructing Knowledge Base (KB), biomedical data mining, Question Answering, etc. Meanwhile, utilizing dependency tree with its rich information is one of the most common technics to improve NLP model's performance. Thus, in this paper, we introduce a joint model combining Recurrent Neural Networks (RNN) and Tree-Structured Long Short-Term Memory Networks (Tree-LSTM). This model allows us to do both relation extraction and named entity recognition (NER) at the same time, when the subtask NER could assist to improve the model's performance on RE and accelerate the convergence speed of the model. Meanwhile, by gathering semantic related words together and abandon those irrelated words in Tree-LSTM, we could extract the valuable semantic information from the text and reduce the impact of the noise of useless word vectors. Furthermore, in this paper, we have conducted a comprehensive investigation on specific datasets and network mechanisms. Carried out detailed amendment of hyper-parameters in the model. By analyzing the performance of the model in different cases, we completed a relation extraction system with competitive performance based on deep neural networks.

Key words: Deep Learning, Relation Extraction, SP-Tree LSTM

Contents

Chapter 1 Introduction

The relation classification task is to identify semantic mentions between a pair of nominals entities in text. It is an important NLP task, which plays a vital role in various scenarios, e.g., information extraction ([1], Wu and Weld, 2010), question answering ([2] Yao and Van Durme, 2014), gene-disease relationships ([3] Chun et al., 2006) and protein-protein interaction ([4] Huang et al., 2004). For instance, given a sentence input:

“The [burst]$_{e1}$ has been caused by water hammer [pressure]$_{e2}$.”

with annotated target entity e1 = “burst” and e2 = “diabetes”, our goal is to recognize the entities *burst* and *diabetes* are of relation *Cause-Effect (e2, e1)*.

Traditional approaches for relation classification mainly rely on feature representation ([5] Kambhatla, 2004) with a large set of features. However, it is difficult to improve the performance of the model if the feature sets aren’t well-chosen. Later, researchers focus on kernel design ([6] Bunescu and Mooney, 2005), when the model summarizes all data information, especially those tokens on the shortest path of the dependency tree. Deep learning, which applies neural networks, appeared recently with a conspicuous improvement over traditional approach. It provides a way of highly automatic feature learning, which does not require people to design specific features. However, incorporating human knowledge to the computer network’s architecture still remains beneficial and important.

In general, there are two ways to predict relations between an entity pair: convolutional neural networks (CNNs) and recurrent neural networks (RNNs). CNNs are regularized versions of multilayer perceptron, where each neuron in one layer is connected to all neurons in the next layer. This “fully-connectedness” of CNNs make them sensitive to capture most of those unaware features in a sentence, but on the other hand, it makes them easy to overfitting. CNNs has been proved to be efficient for relation classification ([7] Zeng et al., 2014) for its high performance and less preprocessing. On the other side, neurons in RNNs form a directed graph along a temporal sequence, allowing feedforward neural networks, capturing sequential features, and directly representing essential linguistic structures. Besides, there is no requirement for input sentence length, which makes them applicable for tasks in natural language processing. RNNs have also been applied in relation classification in recent years ([8] Xu et al., 2015a), even at first, they performed worse than CNNs despite their representation ability. Meanwhile, another type of RNNs, which represents dependency trees ([9] Tai et al., 2015) shows that the tree-structured network outperforms linear chain structured RNNs in sentiment classification.

Recent studies show that end-to-end modeling of both entity and relation could lead to better performance ([10] Li and Ji, 2014) since relations interact closely with entity detections. Later, a joint model combing entity detection and relation classification using LSTMs on both sequences and tree structures ([11] Miwa and Bansal, 2016). In this model, the author proposes a novel tree-structure LSTM capturing information of nodes out of the shortest dependency paths (SDP) between two entities. SDPs are informative since they concentrate on the most relevant information while reducing less relevant noise. In this paper, we would reproduce the tree-structured LSTM-RNNs (Miwa and Bansal, 2016) and testing different factors that would possibly improve the model's performance. The model basically could be divided into three parts. At first, we use a bidirectional sequential LSTM to get all states of the hidden layer. Then we detect entities using a single incrementally-decoded NN structure and attach an entity label to each token. The third part is a bidirectional Tree-LSTM based on sentences' linguistic structure. Several technics, such as scheduled sampling, negative sampling, etc., are added to reach a higher performance. All experiments will be conducted on datasets SemEval-2010 Task 8, and finally, we will complete a relation extraction system with satisfying performance.

Chapter 2 Related Work

Relation classification is a broadly studied task in the NLP community. Various methods have been proved to be efficient since the last decade. At first, relation classification usually is a pipeline model with four steps: part-of-speech tagging, entity recognition, syntactic analysis, and semantic analysis. But once an error occurs, it will make an impact on the next step, which will cause chaos and confusion for the machine to learn.

To handle such a problem, a joint model was proposed to train every step at the same time ([12] Miller et al., 2000). This model applies an augmented dependency tree to extract syntactic relations and extract relations through statistical methods. However, to strengthen training effectiveness, it needs a large corpus with man-made annotations, which limits its possible application scenarios. In 2004, the application of maximum entropy has reached a satisfying effect in relation extraction ([13] Kambhatla, 2004). Several features including words, entity type, mention level, overlap and parse tree, allow the relation classifier to work well without relying too much on the parse tree. In contrast, another group of researchers designed a tree kernel to map the features into a high dimension space and used the support vector machine (SVM) to improve the performance of the augmented dependency tree ([13] Culotta and Sorensen, 2004). Then the model is updated to a new version with three kernels corresponding to tokenization, sentence parsing, and deep dependency analysis respectively ([14] Zhao and Grishman, 2005). Another improvement is dividing the sentence into three parts: words on the left of entity e_1, words on the right of entity e_2, and words between e_1 and e_2 ([15] Mooney and Bunescu, 2006). The systematic analysis of several kernels ([16] Wang, 2008) shows that relation extraction can take advantage of combining the convolution kernel as well as syntactic features. However, most of the kernel-based models deeply depend on the kernel's performance since all data information is acquired through kernels, and thus designing an effective kernel can be crucial.

Before deep neural networks populated, distant supervision methods for relation extraction are proved to be beneficial ([17] Mintz et al. 2009). Mintz assumes that if there is any relation between two entities in the knowledge base, then when these two entities appear in one sentence at the same time, then the sentence should express the same relation. By such a strategy, we could produce positive samples and we annotate random entity in negative samples, dealing with small datasets without enough relation labels. But there might be more than one relationship between two entities, to solve the problem during extracting overlapping relations as well as the noise during distant supervision, we could learn multiple samples of the two same entities ([18] Hoffmann et al., 2011). On the other side, the difficulty of learning multi-relations between two entities could be eliminated by a novel graphical model for multi-instance multi-label (MIML) learning ([19] Surdeanu et al., 2012)

Since the population of deep neural networks, the performance of the relation classifier has been enhanced significantly. Zeng first applies CNNs onto relation classification. Compared with traditional machine learning methods, which usually need designing features and where the errors can propagate along with the NLP tools, CNNs can extract high dimensional features automatically. By extracting positional features and lexical features laying under the sentence, entity information could be better utilized for relation classification. An optimization of Zeng's CNNs is adding n-gram features and using other features rather than word features, such as WordNet ([20] Nguyen and Grishman, 2015). Another optimization is similar to what Mooney has done in 2006. They divided the sentence into three parts according to two target entities' position and each part will be sent to a max-pooling layer, called precise wise max-pooling ([21] Zeng et al., 2015). Later, more complex deep neural networks appear, such as multi-level attention CNNs ([22] Wang et al., 2016). Using multi-level attention mechanism can capture both relation-specific and entity-specific attention, it allows us to detect more elusive cues despite the heterogeneous structure of the input sentences, enabling it to automatically pay attention to parts that are more relevant for a given classification.

Meanwhile, another type of deep neural network, LSTM-RNNs begin to be used for relation extraction ([23] Xu et al., 2015b). In this model, the importance of the sentence's dependency tree has been emphasized. By focusing on the shortest dependency path between two target entities, we could eliminate the influence of irrelated information. Meanwhile, this model own different channels for POS-tags, dependency labels, and WordNet respectively, which helps to find out that the WordNet channel has the most conspicuous effect on the performance. Recently, a combination of both CNNs and RNNs also outperform other models by utilizing the shortest dependency path and multi-level attention mechanism ([24] Guo et al., 2019). It also inspires that gated recurrent unit (GRU), compared with the most commonly used LSTM, could also achieve a satisfied or even better score in specific models.

Some researchers turn to look up a new way by joining different tasks together. For instance, Miwa and Bansal (2016) propose to let entity detection be a subtask of relation classification. The same idea is shared to be conducted on the biomedical corpus ([25] Li et al., 2017). Another innovation is combining the knowledge base (KB) with relation classification ([26] Xu and Barbosa, 2019). That is, we update the KB and local classifier model at the same time and learn to decrease the dissimilarity between language level representation and knowledge level representation. Our model presents a fine amended joint model close to Miwa and Bansal (2016) model, where takes entity recognition as a subtask for the end-to-end relation extraction.

Chapter 3 Model

3.1 Overview

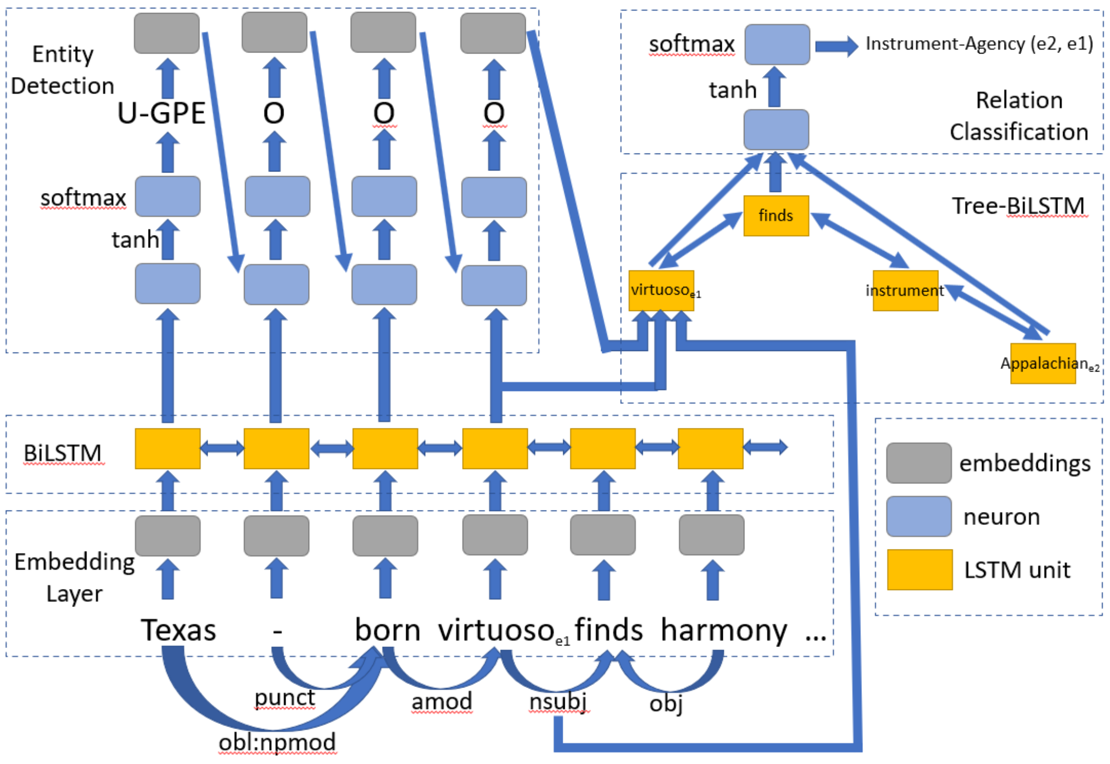

Figure 1: model

Figure 1 depicts the overall architecture of our neural network. First, a sentence is parsed by the Stanford parser. Each word is represented by a vector in the embedding layer and we could receive a two-dimension matrix after embedding the sentence. Then, we feed a sequential bidirectional LSTM with this matrix and we collect the states of LSTM unit in this layer for two usages: entity detection and relation classification. For the first task, we use a greedy left-to-right entity detection. Then we apply a bidirectional tree-structured LSTM based on the sentence's dependency tree to realize the relation classification. Here, we choose the subtree corresponding to the shortest dependency path between two entities. After decoding the entire model structure, we train the parameters via back-propagation through time (BPTT) along with other enhancements. The model is a joint model since both entity detection and relation classification will affect each other, so we set coefficients to each loss function corresponding to these two tasks.

3.2 Embedding Layer

The embedding layer turns words (positive indexes) into dense vectors of fixed size, stop people from using one-hot encoded vectors, which are high-dimensional and sparse. For example, in our experiment, the total number of words is 400,000, as a result, the one-hot vector could be really redundant with 399,999 zeros. Thus, embedding layer is indispensable and in our model, the embedding layer transforms words in a parsed sentence along with other lexical information into n_w, n_p, n_d and n_e- dimensional vectors: $(v^{(w)}, v^{(p)}, v^{(d)}, v^{(e)})$, which corresponding to words embedding, part-of-speech (POS) tags embedding, dependency labels embedding and entity labels embedding, respectively.

3.2.1 Word Representations

Each word in a given sentence could be represented by a one-dimension real-valued vector. These vectors can be regarded as features of words in a large variety of applications, such as information extraction. Compared to random initiated word vectors, word embeddings trained on a large corpus without supervision could well capture words' syntactic and semantic information. Since recent years, there have been several successful word representation models, such as word2vec, GloVe, ELMo and BERT. Despite BERT outperform all other models, in this paper, we choose GloVe ([27] Pennington and Manning, 2014) for our word embedding layer because of its easy-to-use and relatively faster embedding process. By using the co-occurrence matrix, the GloVe model takes both local and global information into consideration, which performs relatively good enough for our experiment.

3.2.2 Part-of-Speech Tags

Part-of-speech tagging, also known as grammatical tagging, is the process of attaching a particular part of speech to each word in a sentence, based on both its definition and its context. This is a common technic in natural language processing since it could distinguish the different meanings of the same word in a different context. For instance, in the sentence “I have to wait for the train”, the POS-tag of the word “train” is NOUN. However, it could be explained as VERB in the sentence “I will train the model”. Thus, correct grammatical tagging could reflect more accurate grammatical information and compensate for the inaccuracy of word embeddings and improve the total performance of the model.

3.2.3 Dependency Labels

Dependency is the notion that words are connected by directed links and dependency parsing is the task of extracting a dependency parsing tree of a sentence that represents its grammatical structure and defines the relationship between every two connected nodes. For instance, given a sentence as follows:

“Texas-born virtuoso finds harmony, sophistication in Appalachian instrument.”

Then, we apply Stanford parser and get the dependency tree as shown in Figure 2.

In Figure 2, we could observe that for each input, there is only one root even there are several sentences and the number of each node’s children is various from each other. Those light blue labels are dependency labels, which will be embedded into vectors for our model. It should be noticed that the same word pair may have different relation labels according to its context, thus it is necessary to capture such kind of grammatical information in the shortest dependency paths.

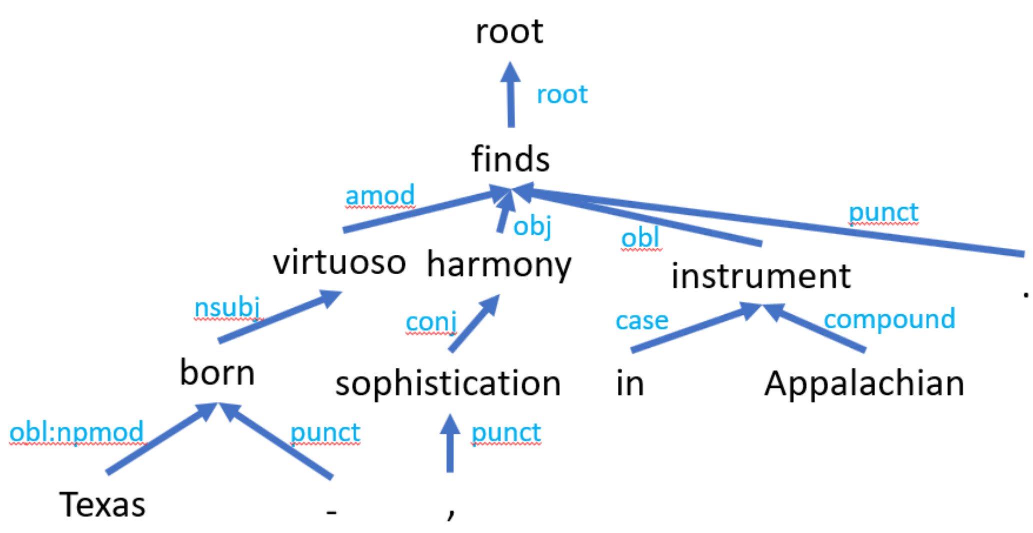

Figure 2: Dependency Tree

3.2.4 Entity Labels

Entity detection, also known as named-entity recognition (NER) is one of the subtasks of information extraction, which requires to locate and classify named entities mentioned in a sentence. There are pre-defined categories, such as organization (ORG), date (DATE), person (PERSON), etc. In our model, we assign an entity label to each word using a common encoding scheme BILOU ([28] Ratinov and Roth, 2009). BILOU is the abbreviation of begin, inside, last, outside, and unit. For each full entity, if it is constituted of one word, we assign U in front of its entity label. For instance, the word “Einstein” has an entity label: U-PERSON. For those entities made up of more than two words, it should be assigned the entity labels obeying the following rules:

New | York

B-GPE | L-GPE

or:

16 | th | July | 2020

B-DATE | I-DATE | I-DATE | L-DATE

By assigning such kind of entity labels to each word, we could express more accurate information hidden in the sentence during the embedding process.

3.3 BiLSTM

Long short-term memory (LSTM) networks are a type of recurrent neural network (RNN) architecture applied in deep learning, which can learn and remember over long sequences of input sentences through the use of "gates". Since most RNNs usually encounter several common problems, such as short-term memory, vanishing gradient, and exploding gradient, lexical and semantic information could easily be lost during forwarding propagation and the gradient could not contribute too much to better performance during backpropagation. Thus, in LSTM we have several gates to control data flow for transferring information over a long distance. The LSTM unit is composed of a cell (a cell state and a hidden state), an input gate, a forget gate, and an output gate.

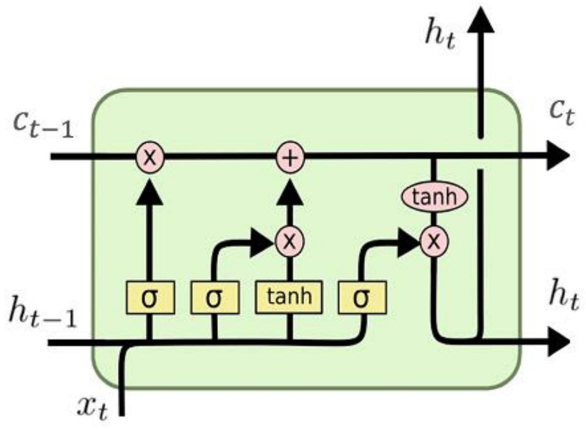

Figure 3: LSTM unit

where c_t is the cell state, which acts as a highway that transports relative information along the sequence chain, σ is the logistic function (known as sigmoid) with its expression:

$$\sigma\ (x) = \frac{1}{1+e^{-x}} \tag{1}$$

The LSTM unit at t-th word receives the input vector x_t and the hidden state h_{t-1} of previous unit, and the cell state c_{t-1} and calculate the new vectors using following equations:

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}\right) \qquad (2)\\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\right)\\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}\right)\\
u_t &= \tanh\left(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}\right)\\
c_t &= i_t \odot u_t + f_t \odot c_{t-1}\\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

where $\odot$ denotes Hadamard product, W and U are regular weight matrices and b are bias vectors.

A bidirectional LSTM is a combination of two LSTM with n_{h_s}-dimension hidden layer while one propagates from the start of the sentence to the end of the sentence, another one run in parallel following the opposite direction. To feed this layer, we choose to concatenate word embeddings with POS-tag embeddings together: $x_t^{(s)} = \left[v_t^w; v_t^p\right]$. We also collect all the hidden state vectors of these two directions networks $(h^{\rightarrow}, h^{\leftarrow})$ corresponding to each word. And then we concatenate them together as the output of this BiLSTM layer, $s_t = [h_t^{\rightarrow}; h_t^{\leftarrow}]$, and send it to entity detection and dependency layer (Tree-BiLSTM).

3.4 Entity Detection

In this section, we will try to predict entity labels for each word based on their input s_t received from the BiLSTM layer. Although multiple choices are available, we prefer relatively simple and light entity recognition models since our major task is relation extraction. Thus, we apply a left-to-right greedy entity prediction, which is a two-layer neural network with a n_{h_e}-dimension hidden layer and a softmax layer for entity detection. (Figure 1)

$$h_t^{(e)} = \tanh\left(W^{(eh)}\left[s_t; \hat{v}_{t-1}^{(e)}\right] + b^{(eh)}\right) \tag{3}$$

$$y_t^{(e)} = \mathrm{softmax}(W^{(ey)} h_t^{(e)} + b^{(ey)})$$

where W are weight matrices and b are bias vectors.

After softmax layer, we will get the estimated entity label for t-th word by argmax function, and then transform this estimated entity label into an entity embedding vector and send it to the next unit to predict the next entity label. Thus, each predicted entity label relies intensely on the previous one. Besides, those predicted entity labels will also be sent to the Tree-BiLSTM layer as features of each word.

3.5 Tree-BiLSTM

As shown in Figure 1, Tree-BiLSTM is a bidirectional LSTM layer built on the dependency tree, especially on the shortest dependency path (SDP) between two target words, since this method has been proved to be effective in relation classification. (Xu, Y. 2015). Each Tree-LSTM unit has the same components as standard LSTM unit, input gates, output gates, forget gates, memory cells, and hidden states. The difference is that gating vectors and memory cell updates are based on dependent on the states of their child units and each Tree-LSTM unit may contain several forgets gates corresponding to its child units.

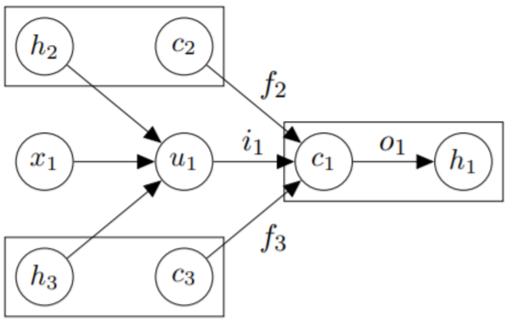

Figure 4: A Tree-LSTM unit with two child units

As shown in Figure 4, labeled edges correspond to gating by the previous gating vector with dependent relations. A Tree-LSTM unit receives an input vector x_1, to calculate its memory cell, we need to combine the hidden states and memory cells of its child units with designed forget gates attached to each child unit. This allows the Tree-LSTM able to learn and grasp information of emphasized semantic heads in a semantic relatedness task and ignore those words less important.

Thus, the structure of units in Tree-LSTM plays an essential role in this layer and we will take the sentence in Figure 1 and Figure 2 as examples. Given the sentence: "Texas-born [virtuoso]$_{e1}$ finds harmony, sophistication in [Appalachian]$_{e2}$ instrument." A dependency tree could be extracted as shown in Figure 2. The least common ancestor of target word e1 end e2 is "finds". Thus, we focus on the path between "virtuoso" and "finds" and the path between "Appalachian" and "finds" and meanwhile, we keep those children of nodes on these two paths. That means, the words "born", "harmony", "in", "." will also be kept as input as shown in Figure 5.

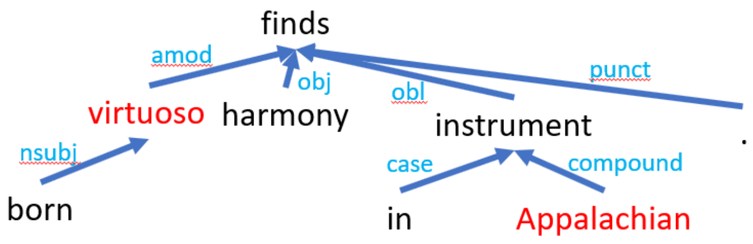

Figure 5: SubTree

We employ bidirectional tree-structured LSTM to output the relation candidate by capturing the dependency around two target words. This neural network propagates both from target words to the root (bottom-up) and from root to leaves (top-down). There are two major tree-structured LSTM: Child-Sum Tree-LSTM and N-ary Tree-LSTM. We will introduce each one of these two models and then explain why none of them fit our needs in this model.

3.5.1 Child-Sum Tree-LSTM

Given a tree, and let $C(t)$ denote the set children of node t, the equations of Child-Sum Tree-LSTM are shown as following:

$$\tilde{h}_t = \sum_{k \in C(t)} h_k$$

$$i_t = \sigma\left(W^{(i)} x_t + U^{(i)} \tilde{h}_t + b^{(i)}\right)$$

$$f_{tk} = \sigma\left(W^{(f)} x_t + U^{(i)} h_k + b^{(f)}\right)$$

$$o_t = \sigma\left(W^{(o)} x_t + U^{(o)} \tilde{h}_t + b^{(o)}\right)$$

$$u_t = \tanh\left(W^{(u)} x_t + U^{(u)} \tilde{h}_t + b^{(u)}\right)$$

$$c_t = i_t \odot u_t + \sum_{k \in C(t)} f_{tk} \odot c_k$$

$$h_t = o_t \odot \tanh(c_t)$$

where in Eq. (4) line 3, $k \in C(j)$.

By these equations, we could get those important children with close relationship have more influence during recurrent propagation, by learning weight parameter $W^{(i)}$. For instance, when a semantically important content word j is given as input, we prefer to have the input gate i_t close to 1, and vice versa.

Since the Child-Sum Tree-LSTM unit determines its mechanisms on the sum of child units' states, it is well suitable for the tree with a high branching factor or the tree with children without order. For example, in Figure 2, the number of children of Tree-LSTM units varies from 0 to 4, and the tree is more likely to be unordered since the dependent relations are the more vital elements.

3.5.2 N-ary Tree-LSTM

Compared with Child-Sum Tree-LSTM, the N-ary Tree-LSTM could be used when the branching factor of a tree is at most N and the children are ordered. That means, each child is indexed with a number t, the weight matrices and bias vectors could vary from each other respectively according to their different indexes. The N-ary Tree-LSTM equations are listed as following:

$$i_t = \sigma\left(W^{(i)}x_t + \sum_{l=1}^{N} U_l^{(o)} h_{tl} + b^{(i)}\right) \tag{5}$$

$$f_{tk} = \sigma\left(W^{(f)}x_t + \sum_{l=1}^{N} U_{kl}^{(f)} h_{tl} + b^{(f)}\right)$$

$$o_t = \sigma\left(W^{(o)}x_t + \sum_{l=1}^{N} U_l^{(o)} h_{tl} + b^{(o)}\right)$$

$$u_t = \tanh\left(W^{(u)}x_t + \sum_{l=1}^{N} U_l^{(u)} h_{tl} + b^{(u)}\right)$$

$$c_t = i_t \odot u_t + \sum_{l=1}^{N} f_{tl} \odot c_l$$

$$h_t = o_t \odot \tanh(c_t)$$

where in Eq. (5) line 2, $k \in [1, 2, \ldots \ , N]$.

These equations introduce a series of separate parameter matrices for each child k, which allows that N-ary Tree-LSTM to learn much more fine-grained conditioning on the states of a unit's children compared with Child-Sum Tree-LSTM. For instance, we could arrange the data and set the first child with a noun word, the second child with a verb, and the third child with an adverb. Suppose we are in a case that it could be much beneficial to emphasize the verb and adverb. Then the $U_{kl}^{(f)}$ would be trained such that the forget gate for the first child f_{t1} is close to 0, while f_{t2} and f_{t3} are close to 1. That would eliminate the impact of noise data and increase the performance.

3.5.3 Our Tree-LSTM

Both of these two variants of Tree-LSTM allow for richer network topologies and each of them is able to incorporate information from multiple child units. However, they don't meet our needs in this model. Child-Sum Tree-LSTM doesn't deal with different children's type and N-ary Tree-LSTM is not able to handle a variable number of children. As a result, we modified Tree-LSTM a little, and this variant of tree-structured LSTM-RNN shares the same weight matrices for the same type children and is able to receive a variable number of children. Denote that $C(t)$ is a group of t-th node's children. For this variant, we calculate the states of t-th node with following equations:

$$i_t = \sigma\left(W^{(i)} x_t + \sum_{l \in C(t)} U_{m(l)}^{(i)} h_{tl} + b^{(i)}\right) \qquad (6)$$

$$f_{tk} = \sigma\left(W^{(f)} x_t + \sum_{l \in C(t)} U_{m(k)m(l)}^{(f)} h_{tl} + b^{(f)}\right)$$

$$o_t = \sigma\left(W^{(o)}x_t + \sum_{l \in C(t)} U^{(o)}_{m(l)} h_{tl} + b^{(o)}\right)$$

$$u_t = \tanh\left(W^{(u)}x_t + \sum_{l \in C(t)} U^{(u)}_{m(l)} h_{tl} + b^{(u)}\right)$$

$$c_t = i_t \odot u_t + \sum_{l \in C(t)} f_{tl} \odot c_{tl}$$

$$h_t = o_t \odot \tanh(c_t)$$

where in Eq. (6) line 2, $k \in C(t)$, $m(\cdot)$ is a type mapping function and the hidden layer is of n_{h_d}-dimension.

In our experiments, we test two types of tree. Primarily, we use the shortest path structure (known as SP-Tree), in which we only care about nodes on the shortest path and all other nodes. (Figure 6) In this case, we can just ignore $m(\cdot)$ since all the nodes in the neural network are of the same type. Then, we construct the SubTree (for example, Figure 5), where two types of node appear at the same time, one for nodes on the shortest paths and one for other child nodes outside the shortest paths. Thus, mapping function $m(\cdot)$ has two output value and it can be used to distinguish two nodes types.

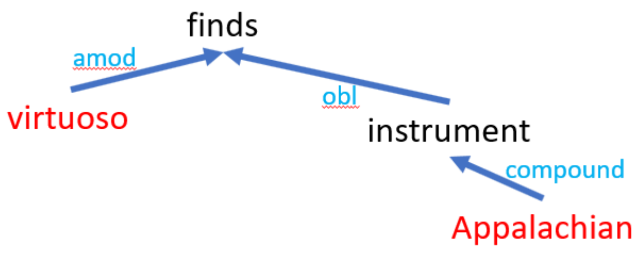

Figure 6: SP-Tree

For the training, we first concatenate the hidden states of BiLSTM layer with the dependency label embeddings (denote the dependency relations to the parents) and the entity label embeddings (correspond to the predicted entity labels) such as $x_t^{(d)} = \left[s_t; v_t^{(d)}; v_t^{(e)}\right]$ where $s_t = [h_t^{\rightarrow}; h_t^{\leftarrow}]$. After feeding the Tree-BiLSTM with $x^{(d)}$, we collect the hidden states vector $\uparrow h^{(p)}$ of top Tree-LSTM unit (the least common ancestor, noted as LCA) in the bottom-up Tree-LSTM (expressions are shown in Eq. (6)), and the hidden states vectors $\downarrow h_1^{(p)}$ and $\downarrow h_2^{(p)}$ of the end Tree-LSTM unit (two target entites) in the top-down Tree-LSTM (simply LSTM-RNNs running on the SDP started at LCA for two times).

3.6 Relation Classification

In this layer, we will finally get the relation prediction vector $y^{(p)}$ from the input $x^{(p)}$. We employ a two-layer NN with the same structure as which we have employed in entity detection. Denote that the dimension of hidden layer is n_{h_p}.

$$h^{(p)} = \tanh\left(W^{(ph)}x^{(p)} + b^{(ph)}\right) \qquad (7)$$

$$y^{(p)} = \text{softmax}(W^{(py)}h^{(p)} + b^{(py)})$$

We construct the input vector $x^{(p)}$ for relation classification from the previous layer as shown: $x^{(p)} = [\uparrow h^{(p)}; \downarrow h_1^{(p)}; \downarrow h_2^{(p)}]$. Moreover, since the contribution of target words to the final relation classification layer is indirect, so we add two extraneous terms to $x^{(p)}$: $x^{(p)} = [\uparrow h^{(p)}; \downarrow h_1^{(p)}; \downarrow h_2^{(p)}; s_1; s_2]$ where s_1 is the hidden state of the first target word in BiLSTM layer and s_2 is of the second target word. Thus, the model can fully take advantage of the information of target words.

3.7 Dropout Strategies

With the increasement of the model's complexity and the number of training epochs, overfitting usually occurs and becomes one of the troublesome problems. One proper explanation is that during the training process, neurons develop strong co-dependency amongst each other, which might lead to suppression of the individual power of each neuron and finally cause over-fitting. To alleviate such kind of situation, we usually apply dropout as a kind of regularization approach ([29] Srivastava et al., 2014). This method, dropout, simply refers to ignoring certain neurons randomly with a predefined hyper-parameter dropout rate p during the feed-forward networks. However, the dropout strategy mentioned by Srivastava (2014) actually doesn't work fine in our recurrent networks, since the forget gates have already played the similar role in LSTM units and randomly dropout might cause the problem when the units are transporting some vital information from one side to the other side of the sentence. Thus, we employ dropout on those entity label embeddings and dependency label embeddings sent to the Tree-BiLSTM layer. That is, $x_t^{(d)} = [s_t; D(v_t^{(d)}); D(v_t^{(e)})]$, where $D(\cdot)$ denotes the dropout function with the probability p. We also apply the dropout method in entity detection layer when one predicted entity label embedding is sent as input of the next unit. Recall the Eq. (3) line 1: $h_t^{(e)} = \tanh\left(W^{(eh)}\left[s_t; v_{t-1}^{(e)}\right] + b^{(eh)}\right)$, we then make a little change to this equation: $h_t^{(e)} = \tanh\left(W^{(eh)}\left[s_t; D(v_{t-1}^{(e)})\right] + b^{(eh)}\right)$. Since most of entity labels are "O", which refers to none type, in such condition, dropping out the previous label should be inimical to our model and alleviate error propagation to some extent.

3.8 Negative Sampling

Negative sampling in relation classification was proposed by Xu (2015a), which is the strategy to help simplifying the $(2 \times K + 1)$-relation classification task into a $(K + 1)$-relation classification task and an assignment for (subject, object) task. This has been proved to be beneficial for model performance and adopted by later researchers.

Let's start with the same example, given the sentence:

"Texas-born [virtuoso]$_{e1}$ finds harmony, sophistication in [Appalachian]$_{e2}$ instrument."

we put this sentence into the model and expect to get the predicted relation label: *Instrument-Agency (e2, e1)* instead of *Instrument-Agency (e1, e2)* or any other labels. Usually, people pretend treat two relation labels in opposite direction (for example, *Instrument-Agency (e2, e1)* and *Instrument-Agency (e1, e2)*) as two different irrelated labels. Thus, the total number of relation labels should be $(2 \times K + 1)$, where 1 is the no-relation type (*Other*) with no direction. However, the shortest dependency path could offer the relative positions of both subjects and objects. Thus, we could simplify the $(2 \times K + 1)$-relation task into a $(K + 1)$-relation task. We would explain it with our model in the following.

In our Tree-BiLSTM layer, when we employ a top-down Tree-LSTM, we actually do employ 2 LSTM-RNNs with different weight matrices and bias vectors: one designed for the shortest dependency path (SDP) from the least common ancestor (LCA) to subject entity (noted as subj-LSTM), another designed for object entity (noted as obj-LSTM). For the upper example sentence, the true label for entity pair (virtuoso, Appalachian) is *Instrument-Agency (e2, e1)*, where e_2 is the subject entity. To get the positive sample, we feed (subj-LSTM, obj-LSTM) with $(SDP_{LCA \to e2}, SDP_{LCA \to e1})$. Correspondingly, we could construct negative sample by simply feeding (subj-LSTM, obj-LSTM) with $(SDP_{LCA \to e1}, SDP_{LCA \to e2})$. For entity pairs in negative samples, we consider them as the non-relation type *Other*. Finally, instead of receiving the predicted probability distribution of relation label $y^{(p)}$, we could obtain two output vectors: $y_{+}^{(p)}$ for positive sample and $y_{-}^{(p)}$ for negative sample.

3.9 Training Objective

In the Section 3.4 (Entity Detection) and Section 3.6 (Relation Classification), we employ the softmax classifier to predict entity label distribution $y_t^{(e)}$ for t-th word and relation label distribution $y^{(p)}$. Since they are two separate tasks, we calculate a loss function for each of them. In our model, we use the cross-entropy error function as the criterion.

Thus, we have loss function for entity prediction as:

$$Loss_t^{(e)} = -\sum_{i=1}^{N_{ent}} \gamma_i^{(e)} z_i^{(e)} \log y_{ti}^{(e)} \qquad (8)$$

Where N_{ent} is the total number of entity labels, $z^{(e)} \in \mathbb{R}^{N_{ent}}$ denotes the one-hot representation of true entity labels, $y_{ti}^{(e)}$ is the i-th value of vector $y_t^{(e)}$ and $\gamma^{(e)}$ is the normalized weight vector obtained according to each entity label's appearance frequency in whole dataset.

When considering the loss function for relation prediction, since for each input sentence, we could receive two output predictions: $y_+^{(p)}$ for positive sample and $y_-^{(p)}$ for negative sample. Thus, we take both of them into account and the loss function should be:

$$Loss^{(p)} = -\sum_{i=1}^{N_{class}} \gamma_i^{(p)} z_i^{(p)} \log y_{+,i}^{(p)} - \gamma_{N_{class}}^{(p)} \log y_{-,N_{class}}^{(p)} \qquad (9)$$

where N_{class} denotes the number of relations labels, $z^{(p)} \in \mathbb{R}^{N_{class}}$ denotes the one-hot representation of true relation labels and $\gamma^{(p)}$ is the normalized weight vector obtained according to each relation label's appearance frequency in whole dataset. Since all entity pairs in negative samples are assumed to own no relationship, so the loss for negative samples is simply written as $-\log y_{-,N_{class}}^{(p)}$.

Meanwhile, L1 and L2 regularization are the most common technics to reduce overfitting. This function updates the general cost function with an additional term known as the regularization term. Such term decreases values of weight matrices and leads to a simpler model. Suppose we have two extremely correlated features, L2 regularization could get better understandable results since the coefficients will be quite evenly distributed among the features, while L1 regularization could get coefficients differed greatly in magnitude even though they will be directionally the same. Thus, in this model we choose L2 regularization and our object function is written as:

$$J(\theta) = \alpha \sum_t Loss_t^{(e)} + Loss^{(p)} + \lambda \, ||\theta||_2^2 \qquad (10)$$

where α and λ are two hyper-parameters and θ is the ensemble of all weight matrices in model. Model's parameters could be efficiently trained via backpropagation through time (BPTT) ([30] Werbos, 1990). We also accelerate the training process by applying AdamW optimizer ([31] Kingma and Ba 2014; [32] Loshchilov and Hutter 2017) with gradient clipping, where we note the hyper-parameter clipping norm as η.

Chapter 4 Experiments

In this section, we will show our experiments in detail. Section 4.1 in traduces the dataset; Section 4.2 describes those hyper-parameters and other training details; In Section 4.3, we will introduce some variation of model and in Section 4.4, we will show our experiment results.

4.1 Dataset

We evaluate our model on dataset: SemEval-2010 Task 8 ([33] Hendrickx et al., 2009) for end-to-end relation extraction. The dataset focuses on semantic relations between pairs of nominals and contains 8,000 training sentences and 2,717 testing sentences. For validation, we randomly choose 1,000 samples from training set. This dataset defines 9 relation labels between nominals (each relation has a direction) as well as the tenth type *Other*, which refers to no relation. Note that this dataset has avoided semantic overlapping as much as possible, but it still keeps two ensembles of strongly related relations: *Entity-Origin* / *Entity-Destination* and *Content-Container* / *Component-Whole* / *Member-Collection*, which helps to exam models' ability of fine-grained distinctions. The definitions of each kind of relation are presented below (Table 1):

Table 1 Relation Description

Relation	Description
Cause-Effect	An object or event yields an effect
Instrument-Agency	An agent uses an instrument
Product-Producer	A producer causes a product to exist
Content-Container	An object is physically stored in defined area of space
Entity-Origin	An entity is coming or is derived from an origin
Entity-Destination	An entity is moving towards a destination
Component-Whole	An object is a component of a large whole
Member-Collection	A member forms a nonfunctional part of a collection
Communication-Topic	An act of statement, written or spoken, about a topic

Table 2 Relation Distribution

Relation	Train Dataset	Test Dataset	All
Cause-Effect	1003	328	1331
Instrument-Agency	941	312	1253
Product-Producer	845	292	1137
Content-Container	717	231	948
Entity-Origin	716	258	974
Entity-Destination	690	233	923
Component-Whole	634	261	895
Member-Collection	540	192	732
Communication-Topic	504	156	660
Other	1410	454	1864
Total	8000	2717	10717

Table 2 shows that the distribution of relation reaches an acceptable balance and as mentioned in Section 3.9, we could obtain the weight vector for relation labels: $\gamma^{(p)} = [0.1242, 0.1169, 0.1061, \ldots, 0.1739] \in \mathbb{R}^{N_{class}}$, where $\gamma_1^{(p)} =$ Count(*Cause-Effect*) / Count(Total) = 1331 / 10717 = 0.1242.

For other details, we have $N_{class} = 10$ (Table 2), $N_{ent} = 66$, $N_{pos} = 17$ and $N_{dep} = 44$. (Table 3)

Table 3 Relation Distribution

Group	Elements
Entity Label	B-CARDINAL, B-DATE, B-EVENT, B-FAC, B-GPE, B-LAW, B-LOC, B-MONEY, B-NORP, B-ORG, B-PERCENT, B-PERSON, B-PRODUCT, B-QUANTITY, B-TIME, B-WORK_OF_ART, I-CARDINAL, I-DATE, I-EVENT, I-FAC, I-GPE, I-LAW, I-LOC, I-MONEY, I-NORP, I-ORG, I-PERCENT, I-PERSON, I-PRODUCT, I-QUANTITY, I-TIME, I-WORK_OF_ART, L-CARDINAL, L-DATE, L-EVENT, L-FAC, L-GPE, L-LAW, L-LOC, L-MONEY, L-NORP, L-ORG, L-PERCENT, L-PERSON, L-PRODUCT, L-QUANTITY, L-TIME, L-WORK_OF_ART, O, U-CARDINAL, U-DATE, U-EVENT, U-FAC, U-GPE, U-LANGUAGE, U-LAW, U-LOC, U-MONEY, U-NORP, U-ORDINAL, U-ORG, U-PERSON, U-PRODUCT, U-QUANTITY, U-TIME, U-WORK_OF_ART

POS-tag	ADJ, ADP, ADV, AUX, CCONJ, DET, INTJ, NOUN, NUM, PART, PRON, PROPN, PUNCT, SCONJ, SYM, VERB, X
Dependecy Label	acl, acl:relcl, advcl, advmod, amod, appos, aux, aux:pass, case, cc, cc:preconj, ccomp, compound, compound:prt, conj, cop, csubj, csubj:pass, det, det:predet, discourse, expl, fixed, flat, goeswith, iobj, list, mark, nmod, nmod:npmod, nmod:poss, nmod:tmod, nsubj, nsubj:pass, nummod, obj, obl, obl:npmod, obl:tmod, parataxis, punct, root, vocative, xcomp

4.2 Hyper-Parameters and Training Details

4.2.1 Preprocessing

In the preprocessing part, we utilize Stanford Parser to tokenize input sentences and to assign POS-tag to each token and we could also obtain the dependency information (for instance, SDP for two nominals, dependency type to parent) amongst tokens in sentence. In certain cases, target nominal is constituted of several tokens. In our experiments, we choose the token belongs to target nominal in dependency tree with the smallest depth. For example, given target nominal “cold air”, we could obtain that the token “cold” is the child of the token “air”. So, the token “air” owns smaller depth in dependency tree than “cold” and we then choose “air” as entity e_1/e_2. Although Stanford Parser could finish the entity labeling task, we choose spaCy as the tool to do such job, which has a faster and better performance. We also add a prefix based on BILOU to each entity label.

In the embedding layer, we use pre-trained word embeddings trained by GloVe on Wikipedia 2014 and Gigaword 5. It contains 6B tokens, 400K vocabulary of 200-dimension vectors. We also mark those words not in dictionary as “unknown”, corresponding to a trainable vector of the same dimension. We set the dimension of embeddings as: $n_w = 200, n_p = 25, n_d = 25$ and $n_e = 25$. For all hidden layer in each layer, we set $n_{h_e} = n_{h_s} = n_{h_d} = n_{h_p} = 100$.

4.2.2 Hyper-Parameter Sets

In Eq. (10), we mention two hyper-parameters: α for loss of entity prediction and λ for L2 regularization. Here, we set $\alpha = 1$ and $\lambda = $ 1e-5 as default. For all dropout layers, we set the dropout rate $p = 0.3$ as default. And we set $\eta = 10$ as default for gradient clipping process. Besides, we set the batch size to 20. To investigate influence of hyper-parameters, we tried different hyper-parameters separately, including the entity prediction loss parameter $\alpha \in [0, 0.1, 0.5, 1]$, the regularization parameter $\lambda \in$ [1e-3, 1e-4, 1e-5, 1e-6], the dropout out rate $p \in [0, 0.1, 0.2, 0.3, 0.4, 0.5]$ and the size of gradient clipping $\eta \in [1, 5, 10]$. For each case, we train 3 times and calculate the average of their performance.

4.2.3 Scheduled Learning Rate

Each epoch contains 350 training batches (7000 in total). Then we decide to use scheduled learning rate to update model's parameters by:

$$lr = lr_0 \times \beta^{\left\lfloor \frac{i}{k_1} \right\rfloor} \qquad (11)$$

where i denotes the training step and $\beta \leq 1, k_1 \geq 1, lr_0$ are three hyper-parameters. In our model, we set $\beta = 0.92,\ k_1 = 50$ and $lr_0 =$ 1e-3 as default.

4.2.4 Negative Sampling

When we feed the model with test dataset, we feed (subj-LSTM, obj-LSTM) in relation classification layer with both $(SDP_{LCA \to e1}, SDP_{LCA \to e2})$ and $(SDP_{LCA \to e2}, SDP_{LCA \to e1})$, and receive $y_1^{(p)}$ and $y_2^{(p)}$ as outputs. We choose the relation *Other* if and only if both predictions are *Other*: $\text{argmax}_i\ y_{1,i}^{(p)} = \text{argmax}_i\ y_{2,i}^{(p)} = N_{class}$, where *Other* is the

N_{class}-th relation type. Otherwise, we choose the non-other relation with the highest confidence as the output prediction. Then, we evaluate our model by P (Precision), R (Recall), macro-averaged F_1 for non-other relations.

$$\begin{aligned} P &= \frac{TP}{TP + FP} \\ R &= \frac{TP}{TP + FN} \\ F_1 &= \frac{2PR}{P + R} \\ f_1 &= \frac{1}{N}\sum_{n=1}^{N} F_{1,n} \end{aligned} \tag{12}$$

where TP, FP, FN refer to true positives, false positives and false negatives respectively, P, R, F_1 are vectors of dimension N (number of prediction class, equal to $2 \times (N_{class} - 1)$). Note that in this part, the same relation label with two opposite directions are considered as two different types. In other words, we employ F1 function over $2 \times (N_{class} - 1)$ relations.

4.2.5 Scheduled Sampling

In the entity detection layer, it is designed that when we predicting entity label $\hat{v}_t$, it uses the estimate entity label of previous word $\hat{v}_{t-1}$. This might cause chaos at the beginning of training process. Thus, scheduled sampling is proposed as one of the solutions. ([34] Bengio et al., 2015) The main change on training strategy is that when we predicting the entity label $\hat{v}_t$, we could use the true entity label v_{t-1} along with $\hat{v}_{t-1}$ to guide the entity classifier to the correct direction with faster convergence and reduce the problem caused by unreliable prediction of entities at the early stage of training. In scheduled sampling, we use the true entity label v_{t-1} with probability ε_i, which is decreased along with the training step i. In our model, we take the inverse sigmoid decay:

$$\varepsilon_i = \frac{k_2}{k_2 + \exp\left(\frac{i}{k_2}\right)} \tag{13}$$

where $k_2 \geq 1$ is the hyper-parameter and in our model, we set $k_2 = 100$.

4.3 Variation of Model

4.3.1 SP-Tree

In the first case, we use SP-Tree instead of SubTree in dependency layer. As we known, SP-Tree only carry the information of nodes on the shortest dependency path between two entities, this reduces the complexity of the model with less information might be both an advantage and a disadvantage for the final performance. Thus, it is valuable to check whether this variation of model could make some progress.

4.3.2 CNN

Secondly, we replace the BiLSTM layer by a convolutional layer with $n_{h_c} = 2 \times n_{h_s} = 200$ channels. The kernel (W_c) size we used to be written as: $w \times (n_w + n_p)$ with $w = 7$ the window size. To obtain the same size output as the BiLSTM, we use following equation:

$$s_c(C) = W_c(C) ※ x^{(s)} + b_c(C) \tag{14}$$

where ※ is the valid 2D cross-correlation operator, W_c is the kernel, b_c is the bias, and C denotes the channel. Then, we have $s = s_c^{\mathrm{T}}$ and we could send s to entity detection layer and Tree-LSTM layer as we used to do.

4.3.3 Att-CNN

Attention-based CNN (Att-CNN) have been demonstrated success in a wide range of tasks including machine translations, question answering, etc. In this section, we simply add attention mechanism to word-level representation. In detail, words in the shortest dependency paths (SDP) are assigned with a higher weight value μ_{high} while the rests are assigned with μ_{low}. The modified word embedded vectors are calculated as:

$$v_w^{modified} = v_w \times \mu \qquad (15)$$

In our model, we choose $\mu_{high} = 1$ and $\mu_{low} = 0.5$.

4.4 Experiment Results

First, we investigate on the influence of hyper-parameters to our model. The results are shown in Table 4.1, for each case, we run at least 3 times and measure the macro-average F_1 score over the validation dataset. The baseline model refers to the hyper-parameter set: $\alpha = 1, \lambda = 1e\text{-}5, p = 0.3, \eta = 10$ (BiLSTM+SubTree-LSTM). And then we choose hyper-parameter set: $\alpha = 0.1, \lambda = 1e\text{-}5, p = 0.3, \eta = 10$ as baseline model and compare its performance with other existing models (Zeng et al., 2014; [35] Zhou et al., 2016). The results are shown in Table 4.2.

Table 4.1 Results of different Hyper-parameters

Hyper-Parameter	Value	F_1 (%)
Baseline	-	74.55
α	0	76.20
	0.1	**77.08**
	0.5	75.10

	1	-
λ	1e-3	71.96
	1e-4	73.45
	1e-5	-
	1e-6	72.46
p	0	73.72
	0.1	72.83
	0.2	73.20
	0.3	-
	0.4	74.40
	0.5	74.03
η	1	72.69
	5	73.95
	10	-

Table 4.2 Results of different Models

Model	P (%)	R (%)	F_1 (%)
Our Model	**76.43**	79.45	**77.91**
+SPTree (-SubTree)	74.36	**79.82**	76.98
+CNN (-BiLSTM)	58.38	74.87	71.48
+Att-CNN (-BiLSTM)	70.03	77.02	73.36
CNN (Zeng, 2014)	71.07	74.22	72.61
BiLSTM (Zhou, 2016)	69.95	70.16	69.55

In table 4.2, cases +SPTree, +CNN and +Att-CNN correspond to variations in Section 4.3. CNN (Zeng, 2014) is a multi-layer CNNs with different kinds of embeddings. BiLSTM (Zhou, 2016) contains a BiLSTM layer, a max-pooling layer and a full connected layer. These two models are all markable neural networks, so we choose to compare our model with theirs.

Chapter 5 Analysis

5.1 Hyper-Parameter Analysis

We will investigate hyper-parameters' influence on our model based on results shown in Table 4.1. First, coefficient of entity detection loss α, it has a great impact on our model. When $\alpha = 1$ (baseline), which means in case of considering entity detection and relation classification as the same importance, the model's performance is even worse than those conditions with smaller α. Thus, we could assume that entity detection task should not be evaluated too much and it should be just a mean of slight improvement. The condition in which $\alpha = 0$ verifies our assumption since the case $\alpha = 0.1$ outperforms the case $\alpha = 0$.

Secondly, λ is also a vital factor to the training effects. As we could observed, $\lambda = 1e$-5 outperforms other cases with a conspicuous difference. When λ is too large, the effect of λ on avoiding overfitting is not obvious and in contrast, the L2 norm is too tiny and the model can't be trained fluently. This remind us how important it is to choose to proper hyper-parameters.

Considering the dropout rate, we could observe that it makes a little difference in the final version model's performance. Actually, dropout do alleviate the overfitting problem, but for our model with not-low complexity, we stop the training process earlier before it begins to present its effects.

The norm clipping method is designed to accelerate training by limiting gradient L2 norm to save time, but too small η would have some negative impacts on the performance, so we keep $\eta = 10$ in the next stage.

5.2 Model Analysis

In this section, since we already studied influence on of hyper-parameter on our models, we choose the set best fit our model: $\alpha = 0.1, \lambda = 1e\text{-}5, p = 0.3, \eta = 10$. And then we measure the P, R and macro F_1 on each of these models shown in Table 4.2.

As mentioned before, we assume that Sp-Tree might bring both advantages and disadvantages due to its simple construction. By comparing the performance of the SubTree (baseline) and SP-Tree, we then found that the disadvantage of the simple structure of Sp-Tree affects more. That means, the children of words on the SDP also contribute a lot to our model. Then, changing the BiLSTM layer to CNN layer is not a good choice, even with the attention mechanism. This might be caused because the CNN layer could not provide enough valuable sequential features for the recurrent neural network in the entity detection layer as well as the Tree-LSTM layer. Secondly, we only added a lexical attention mechanism to the CNN layer. If we could spend more time and concentrate on designing a more complicated, multi-attention CNN layer, the final result might reach the same or even better than the current model.

Comparing with other existing models shows the proper strength of LSTM on SDP over basic deep neural networks. For instance, extracting valuable hidden information and reconstruct it into a new input to feed Tree-LSTM extremely fit the requirement and feature of relation classification. Besides, the joint model could also make a difference in relation classification. This model inspires us such a way for relation extraction in future work and relation extraction itself could also be a subtask in other NLP tasks.

Chapter 6 Conclusion

We have conducted researches on the joint model for relation extraction by utilizing entity detection information. Through changing different hyper-parameters and slightly modifying model, we could achieve optimized performance and gain a deeper understanding for relation extraction. This allows us to presents the model with favorably comparable performance on SemEval-2010 Task 8 compared with other state-of-the-art models.

Our evaluation led to several major findings. First, the coefficient of the loss for subtask entity detection should be limited to small, and unproper the coefficient for L2 norm regularization could also make huge negative impacts over performance. Secondly, SubTree-LSTM, which combines more nodes' information outperforms SpTree-LSTM. Last but not least, joint model indeed makes a satisfying improvement over the original model. By the analysis in the previous section, we have finished this relation extraction model with competitive performance.

REFERENCE

[1] Wu, F., & Weld, D. S. (2010, July). Open information extraction using Wikipedia. In Proceedings of the 48th annual meeting of the association for computational linguistics (pp. 118-127).

[2] Yao, X., & Van Durme, B. (2014, June). Information extraction over structured data: Question answering with freebase. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 956-966).

[3] Chun, H. W., Tsuruoka, Y., Kim, J. D., Shiba, R., Nagata, N., Hishiki, T., & Tsujii, J. I. (2006). Extraction of gene-disease relations from Medline using domain dictionaries and machine learning. In Biocomputing 2006 (pp. 4-15).

[4] Huang, M., Zhu, X., Hao, Y., Payan, D. G., Qu, K., & Li, M. (2004). Discovering patterns to extract protein–protein interactions from full texts. Bioinformatics, 20(18), 3604-3612.

[5] Kambhatla, N. (2004, July). Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In Proceedings of the ACL 2004 on Interactive poster and demonstration sessions (pp. 22-es).

[6] Bunescu, R., & Mooney, R. (2005, October). A shortest path dependency kernel for relation extraction. In Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (pp. 724-731).

[7] Zeng, D., Liu, K., Lai, S., Zhou, G., & Zhao, J. (2014, August). Relation classification via convolutional deep neural network. In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers (pp. 2335-2344).

[8] Xu, K., Feng, Y., Huang, S., & Zhao, D. (2015a). Semantic relation classification via convolutional neural networks with simple negative sampling. arXiv preprint arXiv:1506.07650.

[9] Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. arXiv preprint arXiv:1503.00075.

[10] Li, Q., & Ji, H. (2014, June). Incremental joint extraction of entity mentions and relations. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 402-412).

[11] Miwa, M., & Bansal, M. (2016). End-to-end relation extraction using lstms on sequences and tree structures. arXiv preprint arXiv:1601.00770.

[12] Miller, S., Fox, H., Ramshaw, L., & Weischedel, R. (2000). A novel use of statistical parsing to extract information from text. In 1st Meeting of the North American Chapter of the Association for Computational Linguistics.

[13] Culotta, A., & Sorensen, J. (2004, July). Dependency tree kernels for relation extraction. In Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04) (pp. 423-429).

[14] Zhao, S., & Grishman, R. (2005, June). Extracting relations with integrated information using kernel methods. In Proceedings of the 43rd annual meeting of the association for computational linguistics (acl'05) (pp. 419-426).

[15] Mooney, R. J., & Bunescu, R. C. (2006). Subsequence kernels for relation extraction. In Advances in neural information processing systems (pp. 171-178).

[16] Wang, M. (2008). A re-examination of dependency path kernels for relation extraction. In Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II.

[17] Mintz, M., Bills, S., Snow, R., & Jurafsky, D. (2009, August). Distant supervision for relation extraction without labeled data. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (pp. 1003-1011).

[18] Hoffmann, R., Zhang, C., Ling, X., Zettlemoyer, L., & Weld, D. S. (2011, June). Knowledge-based weak supervision for information extraction of overlapping relations. In Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies (pp. 541-550).

[19] Surdeanu, M., Tibshirani, J., Nallapati, R., & Manning, C. D. (2012, July). Multi-instance multi-label learning for relation extraction. In Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning (pp. 455-465).

[20] Nguyen, T. H., & Grishman, R. (2015, June). Relation extraction: Perspective from convolutional neural networks. In Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing (pp. 39-48).

[21] Zeng, D., Liu, K., Chen, Y., & Zhao, J. (2015, September). Distant supervision for relation extraction via piecewise convolutional neural networks. In Proceedings of the 2015 conference on empirical methods in natural language processing (pp. 1753-1762).

[22] Wang, L., Cao, Z., De Melo, G., & Liu, Z. (2016, August). Relation classification via multi-level attention cnns. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 1298-1307).

[23] Xu, Y., Mou, L., Li, G., Chen, Y., Peng, H., & Jin, Z. (2015b, September). Classifying relations via long short term memory networks along shortest dependency paths. In Proceedings of the 2015 conference on empirical methods in natural language processing (pp. 1785-1794).

[24] Guo, X., Zhang, H., Yang, H., Xu, L., & Ye, Z. (2019). A single attention-based combination of CNN and RNN for relation classification. IEEE Access, 7, 12467-12475.

[25] Li, F., Zhang, M., Fu, G., & Ji, D. (2017). A neural joint model for entity and relation extraction from biomedical text. BMC bioinformatics, 18(1), 1-11.

[26] Xu, P., & Barbosa, D. (2019). Connecting language and knowledge with

heterogeneous representations for neural relation extraction. arXiv preprint arXiv:1903.10126.

[27] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[28] Ratinov, L., & Roth, D. (2009, June). Design challenges and misconceptions in named entity recognition. In Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009) (pp. 147-155).

[29] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

[30] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10), 1550-1560.

[31] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[32] Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101.

[33] Hendrickx, I., Kim, S. N., Kozareva, Z., Nakov, P., Séaghdha, D. O., Padó, S., ... & Szpakowicz, S. (2009). SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations Between Pairs of Nominals. SEW-2009 Semantic Evaluations: Recent Achievements and Future Directions, 94.

[34] Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In Advances in Neural Information Processing Systems (pp. 1171-1179).

[35] Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., & Xu, B. (2016, August). Attention-based bidirectional long short-term memory networks for relation classification. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (pp. 207-212).