ENHANCING RELATION CLASSIFICATION BY USING SHORTEST DEPENDENCY PATHS BETWEEN ENTITIES WITH PRE-TRAINED LANGUAGE MODELS

by

Haluk Alper Karaevli

B.S., Computer Engineering, Boğaziçi University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2022

ENHANCING RELATION CLASSIFICATION BY USING SHORTEST DEPENDENCY PATHS BETWEEN ENTITIES WITH PRE-TRAINED LANGUAGE MODELS

APPROVED BY:

Prof. Dr. Tunga Güngör (Thesis Supervisor)	.....
Prof. Dr. Banu Diri	.....
Assist. Prof. Suzan Üsküdarlı	.....

DATE OF APPROVAL: 27.01.2022

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincerest gratitude to my advisor Tunga Güngör for his guidance, patience, and immerse knowledge. It was a great pleasure for me to be working with him.

I am also grateful to my thesis committee members: Prof. Dr. Banu Diri and Assist. Prof Suzan Üsküdarlı for sparing their time and their insightful questions and comments on my thesis.

I would like to express my deepest gratitude to my lifelong friend Merih Kaner, who singlehandedly saved my thesis by providing me with the resources I require and more at my greatest time of need. I will always be indebted to him.

I am grateful to my mother, Neşe Karaevli, my father, Rafet Karaevli, and my brother İzzet Levent Karaevli who always be there for me.

I would like to give special thanks to my friends Alper Ösün and Eser Murat Kahraman for their invaluable support.

I would like to dedicate this thesis work to my beloved wife, my soulmate Ceren Zeynep Karaevli. Thank you for believing me even the times that I don't.

ABSTRACT

ENHANCING RELATION CLASSIFICATION BY USING SHORTEST DEPENDENCY PATHS BETWEEN ENTITIES WITH PRE-TRAINED LANGUAGE MODELS

Relation Extraction (RE) is the task of finding the relation between entities from a plain text. As the length of the text increases, finding the relation becomes more challenging. The shortest dependency path (SDP) between two entities, obtained by traversing the terms in the text's dependency tree, provides a view focused on the entities by pruning noisy words. In RE's supervised form Relation Classification, the state-of-the-art methods generally integrate a pre-trained language model (PLM) into their approaches. However, none of them incorporates the shortest dependency paths into their calculations to our knowledge.

In this thesis, we investigate the effects of using shortest dependency paths with pre-trained language models by taking the R-BERT relation classification model as our baseline and building upon it. Our novel approach enhances the baseline model by adding the sequence representation of the shortest dependency path between entities, collected from PLMs, as an additional embedding. In experiments, we have evaluated the proposed model's performance for each combination of SDPs generated from Stanford, HPSG, LAL dependency parsers, and baseline with BERT and XLNet PLMs in two datasets, SemEval-2010 Task 8 and TACRED.

We improve the baseline model by absolute 1.41% and 3.6% scores, increasing the rankings of the model from 8^{th} to 7^{th} and 18^{th} to 7^{th} in SemEval-2010 Task 8 and TACRED, respectively.

ÖZET

ÖN EĞİTİMLİ DİL MODELLERİ İLE VARLIKLAR ARASI EN KISA BAĞLILIK YOLLARINI KULLANARAK İLİŞKİ SINIFLANDIRMASININ GELİŞTİRİLMESİ

İlişki Çıkarma (İÇ), düz bir metinden varlıklar arasındaki ilişkiyi bulma görevidir. Verilen metnin uzunluğu arttıkça ilişkiyi bulmak da gittikçe zorlaşmaktadır. Metnin bağlılık ağacında iki varlık arasındaki terimleri izleyerek oluşturulan en kısa bağlılık yolları, metindeki gürültü yaratan kelimeleri budayarak varlıklara odaklanmış bir gösterim sunar. İlişki Çıkarma konusunun denetimli versiyonu olan İlişki Sınıflandırma'da, çoğu son teknoloji metod yaklaşımlarına ön eğitimli dil modellerini entegre etmektedir. Ancak şu ana kadar ön eğitimli dil modelleri, varlıklar arası en kısa bağlılık yolları ile birlikte kullanılmamıştır.

Bu tez, ön eğitimli modellerin varlıklar arası en kısa bağlılık yolları ile beraber kullanılmasının etkilerini incelemektedir. Bu inceleme için R-BERT ilişki sınıflandırma modeli temel model olarak alınmış ve üzerine geliştirmeler yapılmıştır. Sunduğumuz yeni yaklaşımda, temel modeli geliştirmek amacıyla, iki varlık arasındaki en kısa bağlılık yolunun ön eğitimli dil modellerinden geçirilmesi ile elde edilmiş genel temsili, ek bir vektör olarak temel modele eklenir. Deneylerde, temel model, Stanford, HPSG ve LAL bağlılık ayrıştırıcılarının XLNet ve BERT ön eğitimli dil modelleri ile kombinasyonları SemEval-2010 Task 8 ve TACRED veri kümelerinde değerlendirilmiştir. Deney sonuçlarında, önerilen modelin temel modelden SemEval-2010 Task 8 veri kümesinde 1.41%, TACRED veri kümesinde 3.6% daha iyi sonuç verdiği görülmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A_x	Attention coefficients matrix for the input x
b	Bias vector
c_i	context vector
e_i	i^{th} Entity in a relation sample
$\overrightarrow{h_i}$	forward representation of the i^{th} word
$\overleftarrow{h_i}$	backward representation of the i^{th} word
J_1	loss function of span tree prediction
J_{labels}	loss function of dependency labels
K_x	Query matrices calculated for the input x
N_l	Number of dependency labels
Q_x	Key matrices calculated for the input x
S^w	Set of words
S^t	Set of POS tags
S^l	Set of dependency labels
s_i	i^{th} element in the stack of the arc-standart algorithm
s_{ij}	vector for a span between i^{th} word and j^{th} word
V_x	Value matrices calculated for the input x
V_{e_1}	first entities embedding in proposed model
V_{e_2}	second entities embedding in proposed model
V_{sent}	sentences embedding in proposed model
V_{sdp}	Shortest dependency paths embedding in proposed model
W^K	Key weights of a self-attention layer
W^Q	Query weights of a self-attention layer
W^V	Value weights of a self-attention layer
W^w	Weights matrix for words embeddings vector
W^t	Weights matrix for POS tags embeddings vector
W^l	Weights matrix for dependency labels embeddings vector
X	Input matrix

z	The processing order of the terms in sequence
α_{ij}	child-parent score for dependency prediction between i^{th} and j^{th} terms
Θ	Parameter set
Δ	the hamming loss on the labeled spans

LIST OF ACRONYMS/ABBREVIATIONS

DS	Distantly Supervised
EE	Event Extraction
HPSG	Head-driven Phrase Structure Grammar
LAL	Label Attention Layer
LAS	Labeled Attachment Score
NER	Named Entity Recognition
PLM	Pre-trained Language Model
QA	Question Answering
RE	Relation Extraction
SDP	Shortest Dependency Path
TE	Time Extraction

1. INTRODUCTION

The internet is the most significant information source of our current world. However, most of the data it contains is in an unstructured text form which requires additional processing to be understandable by the machine. Considering this source's growth and its current size, a way to automatically extract information from the web becomes a must.

The process of extracting structured, machine-understandable data from the unstructured text is the main task of Information Extraction (IE), a subdomain of Natural Language Processing. IE plays a key role in applications such as expanding knowledge-bases, augmenting search queries by recognizing entities, understanding a question and generating an answer from a text in question answering (QA) systems, scheduling an event to calendar from a message, and much more. Information Extraction can be divided into smaller tasks, including but not limited to Relation Extraction (RE), Named Entity Recognition (NER), Time Extraction (TE), Event Extraction (EE). Each task specializes in acquiring different information from plain text. The task this thesis focuses on is Relation Extraction.

A relation between entities can be defined in the form of a tuple $t = (e_1, e_2, r, D)$ where the e_i's are entities with relation r within document D [2]. The objective of relation extraction is to find the relation r between entities given the plain text of the document D. The task is assessed in three general manners; unsupervised, distantly-supervised, and supervised. In supervised approaches, the aim is to predict the correct relation between entities from a fixed number of relations using a sentence that contains the entities and the target relation between them. Distantly supervised methods have the same objective as their supervised peers, but a set of sentences is taken as input instead of one sentence. Not all sentences in a particular set represent the relation between entity pairs. In unsupervised approaches, none of the entity tuples, the relations, or the corresponding sentences for the entities are known beforehand.

Dependency parsers have been used in various aspects by the models from all approaches of Relation Extraction [3–9]. By providing a structured representation of the raw text using contextual connections, dependency parsers enable models to focus on the interactions between terms.

Shortest dependency path is defined as the path with minimum contextual connections between two entities in the dependency tree. For example, for the sentence "The house at the end of the street is red." the SDP text constructed by combining the shortest dependency path between the words "house" and "red" would be "house is red.".

Pre-trained Language Models (PLMs) are language models trained on large corpora to learn contextual semantics of words without focusing on a specific task. In the task of relation classification, it has been observed that the state-of-art methods generally integrate a pre-trained language model to their approaches [10–12] or train their own [13].

This thesis investigates the effects of using shortest dependency paths with pre-trained language models in the supervised relation extraction domain. The relation classification model R-BERT [10] is chosen as the baseline for this task. In R-BERT, The sentence and the entities are represented as separate vectors concatenated in the last layer. Because of its compartmentalized architecture, adding a new feature to the model is a considerably straightforward process. We aim to improve R-BERT's performance by integrating pre-trained language model representation of the shortest dependency path between target entities to the concatenated embeddings. Three dependency parsers named Stanford, HPSG, and LAL, have been used to generate the shortest dependency paths. In the pre-trained language models aspect, in addition to the BERT employed in the baseline model, the latest state-of-the-art method, XLNet, is applied.

The experiments are conducted on SemEval-2010 Task 8 and TACRED datasets. SemEval-2010 Task 8 is the most used dataset in relation classification, with nine relation classes, each having two versions for the direction of the relation between entities and no-relation classes for negative samples. TACRED is a dataset constructed by Stanford University with 41 relations consisting of 106,264 samples.

With this thesis, our contribution to the domain is an improved R-BERT model that surpasses the performance of the baseline system in both SemEval and TACRED datasets. According to the results we received, in SemEval, we improve the standing of the model from 8^{th} place to 7^{th} place, and in TACRED, a more significant improvement from 18^{th} place (unofficial) to 7^{th} place has been observed.

The thesis is organised as follows: Chapter 2 gives a literature review on the relation extraction domain. In Chapter 3, details of the datasets are provided. Chapter 4 consists of the structure of the proposed model, shortest dependency path generation algorithm from dependency trees, and methodologies of dependency parsers and pre-trained language models. In Chapter 5 the experiment environment, model parameters and the results of the experiments are given. Lastly, with chapter 6, we summarize the work, provide possible scope extensions and conclude the thesis.

2. LITERATURE REVIEW

This literature review analyzes the domain of relation extraction by its three approaches. The approaches differ from each other in their definition of the objective task. For example, supervised methods predict the relation class between two terms from a sentence. In contrast, distantly supervised ones use a set of sentences instead of one sentence as input for their predictions. In unsupervised approaches, none of the entity tuples, the relations, or the corresponding sentences for the entities are known beforehand.

2.1. Supervised Relation Extraction - Relation Classification

The area that attracts the most attention among the three is Supervised Relation Extraction, also known as Relation Classification (RC). In RC, the entities, the raw text to extract the relation, and the relation class are known beforehand. The goal is to correctly classify the relation among predefined relations using the given raw text and entities. Relation Classification models differ from their peers by having a fixed number of classes. In these approaches, the quality of the instances in the datasets is procured by human annotators.

SemEval 2010 Task 8 dataset is the most common dataset chosen for the task of supervised relation extraction. As it is also one of the datasets we conduct our experiments upon, we skip the details of the dataset here to come back to it in the datasets section.

In the following sections of the chapter, summaries of the relation classification papers investigated can be found.

2.1.1. Semantic Relation Classification via Convolutional Neural Networks with Simple Negative Sampling

The depLCNN model Xu et al. [14] proposed aims to learn more robust relation representations of the entities from feeding their shortest dependency path through a convolutional neural network. In addition, they propose a novel negative sampling strategy that addresses the relation directionality that finds the subject and object entity of the relation. They found that dependency paths show the subjects and objects via path directions. Therefore, the shortest dependency paths from objects to the subjects are proposed as negative samples to make the model learn the assignments of subjects and objects.

The flow of the model is as follows: first, the shortest dependency path between entities is generated, then each item's (word, label, or arrow) embedding has been acquired as 50-dimensional word vectors trained by Turians model [15]. After acquiring the embeddings, a fixed-sized window of items around each node is given as consecutive vectors. This representation is then fed to the convolution layer, where each window vector is multiplied by the convolution matrices. The method of max-pooling is applied to the outputs of the convolution layer to get the most beneficial local features.

As the last layer, the tanh function has been used as the non-linearity. To classify the inputs softmax layer is applied to the outputs of tanh. Cross entropy error between the actual class and the softmax output is used as the loss function. Stochastic gradient descent with the AdaGrad method is applied to train the system.

The most significant contribution to the improvement achieved by the system comes from the negative sampling proposal. By using dependency paths from the object to subject as negative samples (subj $\rightarrow$ obj have relation r but not vice versa), the authors achieved an F1 score of 85.4 in the SemEval 2010 Task 8 dataset.

2.1.2. A Dependency-Based Neural Network for Relation Classification

The paper [8] focuses on using information acquired from the dependency tree of the sentences in determining the relations between entities. Usage of dependency trees as features has been previously explored, but the authors come up with a new representation called Augmented Dependency Path (ADP) and dependency-based neural network (DepNN), a neural network that combines CNN and RNN to model the ADP representation.

The layout of the proposed system is as follows; first, each word w and dependency relation r is associated with vector representation $x_w, x_r \in R^{dim}$. Then, an RNN is developed for each word on the shortest path between the first and second entity, which starts from the leaves of the subtree the word has and goes up to the root (the word itself), generating a subtree embedding c_w which is used with x_w in the final representation of word w. Next, the representations of the words and the relations in the shortest dependency path are fed to a convolutional neural network with a max-pooling layer. The output of the max-pooling layer is then given to a softmax layer whose results show the probability of each relation between entities e_1 and e_2. Finally, cross-entropy error between real class and predicted class is used as the objective function to minimize.

The shortest dependency trees are generated by the Stanford Parser with the "collapsed" option. Two kinds of embeddings are used for acquiring vector representation of words in the augmented dependency path, 50d embeddings provided by SENNA and 200d embeddings trained on Gigaword with Word2Vec.

The F1 score the proposed model achieves is 83.2 at SemEval-2010 Task 8 dataset with the usage of named entity recognition features.

2.1.3. Improved Relation Classification by Deep Recurrent Neural Networks

The proposal of [16] to the domain is the usage of "deep" recurrent neural networks in relation extraction, believing that by using individual RNN's in each depth level of the dependency tree representation of the sentence, one can acquire the granular, "whole" representation of the text.

They define the shortest dependency path between entities which the authors proposed in a previous paper [14] as the backbone of this model. There are two advantages of using the shortest dependency path (SDP). First, it filters the information irrelevant to entities, and second, grammatical relations between words in SDP focus on action and players of that action, representing the relationship between entities clearer than the whole text.

Also, to increase the amount of data -and with it, the performance- the inverse relations between entities (r',e2,e1 is the inverse relation of r,e1,e2) are used. With that, more meaningful data have been generated without using any external data source.

The structure of the paper is as follows. In simple, an RNN keeps a hidden state vector h, changing with the input word at each step accordingly. For example, the hidden state at t depends on the hidden state of t-1 and the input x given at t. Representation of the data consists of four information channels; words themselves, part-of-speech tags, grammatical relations, and wordnet hypernyms. Each channel is trained in parallel with different networks, then the outputs of all RNN's are concatenated together and fed through a softmax layer.

The function to minimize to train the model is given as the standard cross-entropy loss for (r,e1,e2) and (r',e2,e1) summed up with the Frobenius norm of all weight matrices as the regularization term. The proposed model gets an 86.10 F1 score in the common dataset of SemEval 2010 Task 8.

2.1.4. Semantic Relation Classification via Bidirectional LSTM Networks with Entity-aware Attention using Latent Entity Typing

The model in [17] consists of four main components named word representation, self-attention, bidirectional LSTM, and entity-aware attention. In word representation, each word is mapped to a vector representation. Self-attention captures the meaning correlation of the words using vectors with a multi-head attention mechanism. Bidirectional LSTM takes the outputs of the self-attention layer and generates the encodings that also represent the order information of the words in the input sentence. Finally, entity-aware attention adds the attention weights of entity pairs to the representation of each word by using the relative position of the words to the entities and the "latent types" of the entities acquired from Latent Entity Typing.

After these steps, the entity aware attention outputs are fed into a softmax layer to generate the probabilities of relation classes. The cross-entropy is selected as the loss function between calculated predictions and actual relations. Minimization of the loss function is done by AdaDelta optimizer.

The model is evaluated on the SemEval-2010 Task 8 relation classification dataset. The proposed model with the latent entity typing gets the best macro F1 score of 85.2. The proposed model does not improve state-of-the-art but contributes to the domain by proposing the approach of latent entity typing.

2.1.5. Enriching Pre-trained Language Model with Entity Information for Relation Classification

The contributions of the paper [10] to the topic of Relation extraction are as follows: Like [17] they propose a way to incorporate entity-level information into the pre-trained language model to achieve an increase in performance of relation classification. The differences are, one, in this paper, the entities representations are fed to the model explicitly, and two, the architectures of the pre-trained models differ from

each other.

The paper proposes to use a pre-trained BERT model to represent the input string tokens as vectors fed to a simple, fully connected neural network whose outputs are used to predict the probabilities of each class with the softmax layer. A token ['CLS'] is appended to each sequence's beginning to acquire the sentence's embedding from the final layer of the transformer's output. The pre-training of the BERT structure is done with the objective defined by the "masked language model" which masks randomly selected tokens from the input sequence, where the objective is finding the original id of the masked tokens from the context.

The model architecture can be summarized as follows. For a sentence s and two target entities, unique tokens of "$" and "#" are inserted into the input sequence denoting the starting and ending positions of the entities, "$" for first entity positions, "#" for the second. To represent the overall sentence, the token of "[CLS]" is also added to the start of the sequence. The sentence of "[CLS] The $ kitchen $ is the last renovated part of the # house # ." with entities kitchen and house can be given as an example output of the preprocessing.

The vector representation of the relation structure consists of three embeddings: The embedding of the sentence acquired from the token "[CLS]" and the average of the transformer output representations of the words that remain between the entity position tokens for each entity. The final versions of the entity embeddings are acquired by feeding averaged transformer vectors into a fully connected neural network. Finally, the softmax layer, which gets the concatenation of these three embeddings as the input, is constructed to predict the probabilities of the relation classes.

This paper is taken as the baseline model to be improved. At the time of thesis writing, the proposed model is the eighth-best performing model in the SemEval 2010 Task 8 dataset with the macro F1 score of 89.25, omitting the other class.

2.2. Distantly - Weakly Supervised Relation Extraction

The distantly supervised (DS) approaches emerge with the proposal of [18]. In the training process of distant supervision, relation and entity tuples in knowledge-bases (KB) are used as seeds, and sentences that contain the entities given in a particular relation in KB are fetched. The fetched texts are used as the input and the relation as the desired output of the system. The main setback of the DS approaches is the noisy instances which are defined as the sentences that have the correct entities but do not represent the given relation. To solve that problem multi-instance learning method is adopted.

NYT-Freebase aligned dataset is the most frequently used dataset in distantly supervised relation extraction models. It is constructed by "aligning" freebase knowledge-base entity pairs with the text data of New York Times articles in [19]. Freebase is a freely available database that stores structured semantic data. The datasets' relations are acquired from the 2009 December snapshot of the freebase database. Four categories named as people, business, person, and location of freebase relations are used. The reason for selecting these categories is that the newswire corpus has a high number of items for the relation types in these four categories.

To align these relations with their mentions in newswire texts, first, Stanford's named entity recognizer is applied to the corpus to get entity mentions. Second, the entities in the corpus are associated with the ones in the freebase entities by using a simple string match. Finally, for each pair of entities representing a relation in freebase and found in the resulting entity association, sentences containing both entities are extracted from the text corpus. Thus the alignment of knowledge base with NYT corpus is done.

The constructed dataset is then divided into two as train and test. The train part contains the entity pair sentences alignments for the years 2005 and 2006, while the test part consists of the alignments from the texts dated 2007. Training and test

sets have 4700 and 1950 distinct entity pairs, respectively.

2.2.1. Neural Relation Extraction with Selective Attention over Instances

In [20] authors propose a sentence-level attention-based CNN for the task of distantly supervised relation extraction. A sentence-level attention mechanism is built to dynamically reduce the weights of incorrect instances. The proposed method consists of two main parts named sentence encoder and selective attention mechanism over instances. Instances are candidate sentences that possibly indicate a relation r, which is known already from the knowledge-base, between entities e1 and e2.

In the sentence encoder, the first step is constructing words vector representations. A vector representation of a word consists of three parts, a word semantic embedding gathered from word2vec and two positional embeddings denoting the word's position to the target entities. After the vectorization step, a convolution layer takes place with two types of max pooling, piecewise and default. In piecewise, each convolution filter is divided into three segments, and max pooling is applied to each segment separately.

Assume a set s containing the sentences of the entity pair (e1,e2). In predicting the relation r for the entity pair, the proposed model represents the set s with a real-valued vector that is a sum of the vectors of the sentences in the set with their weights. The weights are calculated by multiplying the input sentence vector with a trainable diagonal matrix and a query vector associated with vector r.

In evaluation, as a measure of the system's performance, the curve of precision over recall and Precision@N (P@N) is given for various states.

2.2.2. Hierarchical Relation Extraction with Coarse-to-Fine Grained Attention

In the paper [21] a novel hierarchical attention scheme that utilizes relation hierarchies the entities have instead of directly using hierarchic information as features has been proposed.

The method, given entity-pair and its corresponding entity-pair bag (set of sentences), measures the probability of each relation r between the entities in the pair. This measurement is done by two steps: first sentence encoding and then coarse-to-fine grained hierarchical attention. For each sentence in the entity-pair bag, its encoding is calculated by feeding concatenation of the vector representations the words in the sentence have to a convolutional neural network (CNN) or piecewise CNN. The word vectors are acquired by concatenating context embeddings acquired with the skip-gram approach and positional embeddings of the relative distances to entities.

After representations of the sentences are found, a hierarchical selective attention mechanism is applied to the sentence set to get the weights for each sentence. A query vector q_r is assigned to each relation r that is used in the attention calculation equation:

$$\begin{aligned} e_i &= q_r^T W_s s_i \\ \alpha_i &= \frac{\exp(e_i)}{\sum_{j=1}^{m} \exp(e_j)} \end{aligned}$$

Where W_s is the weight matrix and α_i is the weight of sentence i.

In the Knowledge graphs, the common relations in the high-level set (e.g., location) usually contain several sub-relations in the base-level set. In the paper, it is assumed that the sub-relations of different relations are disjoint.

For evaluation, precision-recall curves are drawn for all models. Besides precision-recall curves, Precision@N results are given as other works. The evaluation results

show that incorporating the inherent hierarchical structure of relations into attention mechanisms can benefit from correlations among relations.

2.2.3. Reinforcement Learning for Relation Classification from Noisy Data

In addressing the noisy data issue, previous studies use multi-instance learning to decrease the effects of noisy instances. The noisy instances are defined as the sentences that have the target entities but represent another relation between the targets. In multi-instance learning, all sentences for an entity pair are considered together as a bag. However, the appliance of the bag approach to extract relations between entities results in two shortcomings; one, sentence-level relation extraction cannot be done in bag representation, and two, the results are sensitive to bags containing only noisy sentences.

In order to solve these two shortcomings, in [22] authors propose a relation classification model with two parts: instance selector and relation classifier. The proposed model filters all the instances in a bag if they are noisy to handle the second limitation. However, the proposed method's difficulty comes from training both parts together since the instance selector does not know which instances are correctly labeled. As a solution to this issue, the instance selection part is formulated as a reinforcement learning problem.

The instance selector is an agent who follows a specific policy for selecting a sentence or not according to the current state the agent is in and gets a reward according to the selection. The state consists of three information; vector representation of the considered sentence, representation of the chosen instances calculated by taking the average of the selected sentences, and the embeddings of the entities acquired from a pre-trained knowledge graph embedding table. The decision of whether selecting an

instance or not is made by a logistic function such that:

$$\begin{aligned}\pi_\Theta(s_i, a_i) &= P_\Theta(a_i|s_i) \\ &= a_i\sigma(W * F(s_i) + b) \\ &+ (1 - a_i)(1 - \sigma(W * F(s_i) + b))\end{aligned}$$

Where $\pi_\Theta(s_i, a_i)$ is the policy function, a_i is the action for i^{th} state s_i which is the state for i^{th} sentence. $F(s_i)$ stands for the vector representation of the state the instance selector is currently in. The reward function indicates the correctness or utility of the chosen sentences. For each bag, the reward is calculated once after all the sentences in a bag are considered, in other words, at the state $S_{|B|+1}$. The reward is calculated as the average log probability of the relation r given the sentence X among all the selected sentences for that particular bag. The optimization task is to maximize the expected total reward in each bag.

2.2.4. Long-tail Relation Extraction via Knowledge Graph Embeddings and Graph Convolution Networks

The long-tailness of the relations in the corpora is the de facto reason for the performance leaks in relation extraction task, whereas even the most recent models fail to provide a solution or completely ignore this problem. For example, the frequency label analysis of the commonly used NYT dataset shows that almost 70% of the relations it contains are long-tailed (have less than 1000 instances).

The long-tail relations are rather hard to deal with since the number of instances for these relations is quite limited. In paper [23], finding semantically similar head relations, which are the relations that have lots of examples, to long-tailed ones and transferring their knowledge onto less sampled ones is found to be beneficial for the performance of relation extraction systems. For instance; If a pair of entities contain the relation /people/deceased person/place of death, it is highly likely to contain /people/

deceased person/place of burial relation.

To learn this relational knowledge and represent the relations similar to each other, the authors propose using class embeddings and graph convolution networks. The proposed model consists of three parts. In the order of their application, these parts are instance encoder, relational knowledge learning, and knowledge-aware attention. Instance encoder generates vector representations of instances by using CNNs which take sentence sequence and entity pairs as inputs. In relational knowledge learning, pre-trained knowledge graph embeddings (TransE) are used as implicit relational knowledge and fed to GCNs to learn explicit embeddings for knowledge relation, whose outputs are concatenated to form final class embeddings. With the knowledge-aware attention mechanism, most informative instances for the relations between entities are selected.

Experimental results on the NYT dataset show that the proposed model is more successful in extracting relation information than other models explained in this report, especially in long-tailed relations in terms of precision over recall and Precision@N (P@N).

2.3. Unsupervised Relation Extraction

In unsupervised relation extraction, the only input is the vast amount of raw text. From this text, words' part of speech (POS) classes and the entities are extracted using POS tagging and Named Entity Recognition approaches. With the assumption that words representing the relation between two entities are positioned between these two entities in text, candidate relation snippets are generated. These snippets are then analyzed to find patterns such as "X born in Y," and resulting patterns are accepted as relations.

2.3.1. Unsupervised Relation Extraction by Mining Wikipedia Texts Using Information from the Web

Authors of the [24] consider integrating linguistic analysis with Web frequency information to improve the performance of unsupervised relation extraction. However, since deep linguistic technologies are problematic when applied in muddy text sources (such as Reddit, Twitter), plain texts from the Web are not considered in this unsupervised relation extraction model. Instead, well-written texts, texts of Wikipedia articles in specific, are examined.

The enumeration of all potential relation types of interest for RE in a corpus as extensive as Wikipedia and labeling the relations found in corpus via extraction are the two most challenging tasks of the selected dataset. The proposed model aims to improve the performance of the previous work of Davidov et al. [25] by combining frequency information from the Web and the "high quality" characteristics of Wikipedia text.

The model proposed in the paper consist of four main modules named as text preprocessor and concept pair collector, web context collector, dependency pattern extractor, and clustering algorithm. Text preprocessor and concept pair collector preprocess Wikipedia articles and give concept pairs accompanied with their related text as output. Web context collector generates relational terms and surfaces for each concept pair. Dependency pattern extractor generates dependency patterns from sentences related to concept pairs in Wikipedia articles. Clustering algorithm clusters concepts according to their contexts with "dependence clustering" and "surface clustering" versions.

2.3.2. Discovering Relations Among Named Entities From Large Corpora

Hasegawa et al.'s work [1] is considered as one of the first unsupervised relation extraction models. The key idea behind their proposal is that clustering pairs of entities according to the similarity of context words between the named entities in the text

results in relation clusters. For this idea, the authors make several assumptions, which also can be used to describe their methodology. These assumptions are;

- Pairs of entities occurring in a similar context can be clustered, and each pair in a cluster is an instance of the same relation
- In cases where the contexts linking a pair of entities express multiple relations, it is expected that the target pair of entities are clustered into the pair's most frequently expressed relation or could not be clustered at all.
- Useful relations' mentions will be frequent in the text data. The relations encountered once or twice are unimportant. Those relations are disregarded.

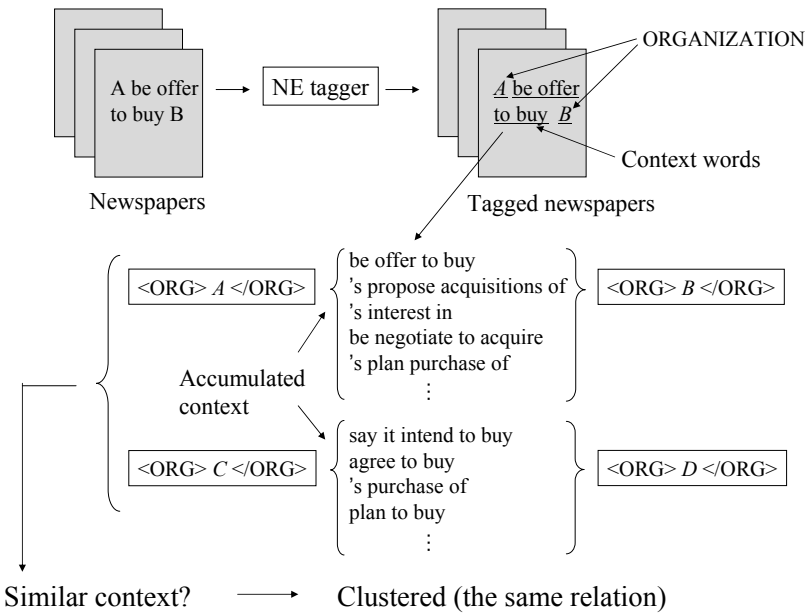


Figure 2.1. Process of Relation Cluster Generation [1]

The approach proposed in the paper can be described by the following steps:

(i) Tagging the named entities in the text corpora.

(ii) Forming of co-occurring named entity pairs and associating their texts to constructed pair.

(iii) Computing context similarities among the pairs found in step (ii).

(iv) Clustering the pairs according to the similarity values computed in the previous step.

(v) Assigning a label to each resulting cluster describing the relation type represented by it.

A representation of the given steps can be found in Figure 2.1. The model requires a named entity tagger to identify named entities in the text so that it focuses only on finding pairs for these named entity mentions.

3. DATASETS

In this thesis work, SemEval-2010 Task 8 [26] and TACRED [27] datasets have been selected to evaluate the performances of the proposed models.

3.1. SemEval-2010 Task 8

SemEval-2010 Task 8 is the most used dataset in the Relation Classification task according to the number of papers associated with it in the Papers with Code website among datasets like DocRED, TACRED, FewREL, and more.

The purpose of generating this dataset is to create a testbed that can be used in automatic semantic relations classification. While constructing the dataset, authors try to achieve two goals; first, all types of relations between two nominals should be represented, and second, each nominal pair should match to one and only one relation in the context it was given. In reality, compromises are required to be made to meet those goals since no dataset is able to fully accomplish both.

The dataset comprises nine relations covering a broad enough area to be considered general and practical. In the process of selecting the relations, semantic overlaps are tried to be avoided as much as possible. Nevertheless, two groups of relations with strong semblances (entity-origin, entity-destination as one group and content-container, component-whole, member-collection as another) are given to analyze proposed models' ability to discern subtle differences.

The annotation process of the dataset samples is composed of three steps. First, 1200 sentences are manually extracted from the web using a unique set of patterns for each relation. The number of patterns in each set ranges between one hundred to several hundred. Second, two annotators label the sentences gathered in the first. Finally, the disagreed candidates are reexamined. If no agreement has been reached by the annotators, the item in consideration is discarded.

The relations available in the dataset are:

- Cause-Effect: An effect occurring because of an object or cause. Ex: Corn pops when heated.
- Instrument-Agency: An instrument used by an agent. Ex: Pilot landed the plane.
- Product-Producer: A product comes to exist by a producer. Ex: the honey acquired from bees nest.
- Content-Container: An physical item stored in a designated place or another object. Ex: cookies are in the jar.
- Entity-Origin: An entity coming or derived from an origin. Ex: This pencil is made in China.
- Entity-Destination: An entity traveling to destination. Ex: The plane will arrive at the airport.
- Component-Whole: Entity is a part of a larger whole. The apartment has three rooms.
- Member-Collection: A member constituting a nonfunctional part of a collection. Ex: There are lots of trees in the forest.
- Message-Topic: The written, spoken, shown message about a topic: The president gives a speech about animal rights.
- Other: The terms that do not fit the classes above.

The frequencies of realtions, the percentages of items selected as the representatives of relation from all candidates of the relation, and inter-annotator agreement values of each relation class can be found in Table 3.1.

Table 3.1. SemEval Dataset Statistics.

Relation	**Freq**	**Pos**	**IAA**
Cause-Effect	1331 (12.4%)	91.2%	79.0%
Component-Whole	1253 (11.7%)	84.3%	70.0%
Entity-Destination	1137 (10.6%)	80.1%	75.2%
Entity-Origin	974 (9.1%)	69.2%	58.2%
Product-Producer	948 (8.8%)	66.3%	84.8%
Member-Collection	923 (8.6%)	74.7%	68.2%
Message-Topic	895 (8.4%)	74.4%	72.4%
Content-Container	732 (6.8%)	59.3%	95.8%
Instrument-Agency	660 (6.2%)	60.8%	65.0%
Other	1864 (17.4%)	N/A	N/A
Total	10717 (100%)		

Each sample consist of three parts:

(i) The sentence text where entities are enclosed with tags [e1], [/e1] and [e2], [/e2].
(ii) The relation type, along with the direction between entities as (e1,e2) or (e2,e1).
(iii) Comments that are given by annotators to explain the reason behind the selection of relation type.

The objective of the dataset is to find the relation and the direction of the relation between entities given the sentence and two entities. The official scoring metric have been chosen as the macro averaged F1 score for all relations in the dataset except Other, considering the directions.

3.2. TACRED

TACRED is a supervised relation extraction dataset constructed by Stanford University to improve the performance of the models in TAC KBP tasks. It focuses on the relations between people, organizations, and locations instead of more general semantic relations in which SemEval-2010 Task 8 concentrates on. In means of amount of data samples TACRED is 7 to 10 times larger than the other common datasets in the domain, Automatic Content Extraction (ACE) and SemEval-2010 Task 8.

The samples in the dataset have been constructed from the source corpuses and query entities of TAC KBP evaluations from years between 2009 and 2015. All entity pairs in the dataset consist of at least one query entity. All the samples are either tagged by a LDC annotator or collected from corpus using Stanford's Statistical coreference system and Illinois Wikifier which is followed with an annotation process done by Mechanical Turk. A total of 10,691 and 110,021 annotations are acquired from two approaches respectively. After duplicates and examples with overlapped entities are discarded 106,264 samples left.

To predict inter-annotator agreement randomly selected 761 samples are shown to five annotators. As a result, all five annotators are agreed on 74.2% of the samples, 90.5% of the samples have been agreed on by at least four of the annotators, and all samples have at least 3 annotators that are in agreement.

In order to overcome the dataset bias between the train, dev and test sets, the samples are stratified by the years of the TAC KBP challenges. Train set is constructed from the years of 2009 to 2012 with the total of 68.124 samples where years and sample counts are 2013, 22.631 and 2014, 15.509 for the dev and test sets respectively.

TACRED consist of 41 relation classes, with no relation as the 42nd class which is used for the entity tuples which don't have a relation between in the related text. 79.5% of the examples are in the class of no relation. All classes with their distribution probabilities for train, dev and test sets can be seen in Table 3.2.

Table 3.2. TACRED Relation Statistics.

Relation	**Total**	**Percentage**	**Train 2009–2012**	**Development 2013**	**Test 2014**
no relation	84491	79.51%	55112	17195	12184
org:alternate names	1359	1.28%	808	338	213
org:city of headquarters	573	0.54%	382	109	82
org:country of headquarters	753	0.71%	468	177	108
org:dissolved	33	0.03%	23	8	2
org:founded	166	0.16%	91	38	37
org:founded by	268	0.25%	124	76	68
org:member of	171	0.16%	122	31	18
org:members	286	0.27%	170	85	31
org:number of employees/members	121	0.11%	75	27	19
org:parents	444	0.42%	286	96	62
org:political/ religious affiliation	125	0.12%	105	10	10
org:shareholders	144	0.14%	76	55	13
org:stateorprovince of headquarters	350	0.33%	229	70	51
org:subsidiaries	453	0.43%	296	113	44
org:top members /employees	2770	2.61%	1890	534	346
org:website	223	0.21%	111	86	26

Table 3.2. TACRED Relation Statistics. (cont.)

Relation	**Total**	**Percentage**	**Train 2009–2012**	**Development 2013**	**Test 2014**
per:age	833	0.78%	390	243	200
per:alternate names	153	0.14%	104	38	11
per:cause of death	337	0.32%	117	168	52
per:charges	280	0.26%	72	105	103
per:children	347	0.33%	211	99	37
per:cities of residence	742	0.70%	374	179	189
per:city of birth	103	0.10%	65	33	5
per:city of death	227	0.21%	81	118	28
per:countries of residence	819	0.77%	445	226	148
per:country of birth	53	0.05%	28	20	5
per:country of death	61	0.06%	6	46	9
per:date of birth	103	0.10%	63	31	9
per:date of death	394	0.37%	134	206	54
per:employee of	2163	2.04%	1524	375	264
per:origin	667	0.63%	325	210	132
per:other family	319	0.30%	179	80	60
per:parents	296	0.28%	152	56	88
per:religion	153	0.14%	53	53	47
per:schools attended	229	0.22%	149	50	30
per:siblings	250	0.24%	165	30	55
per:spouse	483	0.45%	258	159	66
per:stateorprovince of birth	72	0.07%	38	26	8

Table 3.2. TACRED Relation Statistics. (cont.)

Relation	**Total**	**Percentage**	**Train 2009–2012**	**Development 2013**	**Test 2014**
per:stateorprovince of death	104	0.10%	49	41	14
per:stateorprovinces of residence	484	0.46%	331	72	81
per:title	3862	3.63%	2443	919	500
Total	106264	100.00	68124	22631	15509

Compared to SemEval-2010 Task 8 which has the average sentence length of 19.1, TACRED dataset consist of longer sentences with the mean length of 36.4. This has been emphasized by the authors of the dataset as an indicator of the contents complexity. They argue that as the length of a sentence increases the complexity of it also increases, thus a better model for real world scenarios is provided. The length distribution of TACRED and SemEval-2010 task 8 can be seen in Figure 3.1.

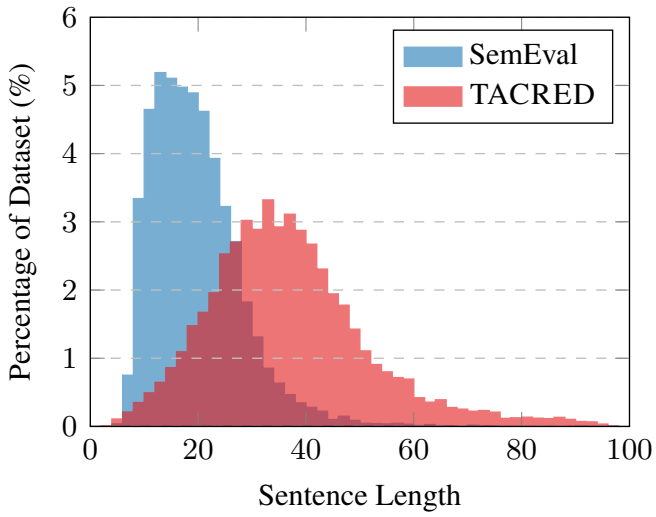


Figure 3.1. Sentence Length Histogram of SemEval-2010 Task 8 and TACRED Datasets.

4. METHODOLOGY

4.1. Overview

R-BERT is the first paper that represents the sentence and the entities separately as explicit embeddings from the pre-trained language model of BERT. With our proposed model, we aim to increase the performance of the R-BERT method by integrating shortest dependency paths between terms and replacing BERT with a novel pre-trained language model XLNet.

Dependency paths of entities derived from the dependency trees have been studied by various papers. [3, 4, 7–9, 28] from supervised and unsupervised subdomains of relation extraction. By providing grammatical connections of terms, dependency parsers define sentences in a structured way.

As the length of the input text increases, finding the relation between the given entities becomes more challenging. The increase in the number of terms results in a muddier representation of the target relation in the sentence embedding. The shortest dependency path between two entities, obtained by traversing the terms in the dependency tree, provides a view focused on the entities by pruning noisy words.

We propose to add the sequence representation of the shortest dependency path, collected from pre-trained language models, as the fourth embedding to the input of the last dense layer in the R-Bert relation classification model. We concatenate the SDP representation to the general embedding instead of replacing the sentence's part to preserve the potential information coming from the noisy data. Additionally we examine the performance change of the system by replacing the pre-trained model with the new state-of-the-art XLNet.

The proposed method's input consists of two parts:

- The raw sentence text where the entities are enclosed with special tokens [E11], [E12], [E21], [E22] which are not available in the text otherwise. In special tokens, the first digit represents the entity number, and the second represents the beginning and end of an entity. Unlike the baseline model, the beginning and end of the entities have separate tokens.
- The SDP text where the first and last terms are first and second entities, respectively, and the terms in between are the tokens in the shortest dependency path.

Sentence text is fed to a transformer-based pre-trained language model, which returns the general embeddings of the sentence and every token individually. The embeddings of the tokens forming the entities, including the entity delimiters, are averaged. The resulting vectors from the average operation are then fed to a fully-connected layer that uses Tanh as the activation function. For both embeddings, the same dense layer is applied.

Along with the sentence, the text for the shortest dependency path is given to the pre-trained language model. The embedding corresponding to the [CLS] token is taken as the SDP's representation. In the version with XLNet, the general embeddings are generated by running an additional layer named 'sequence summary'. It takes the hidden layer output for the <cls> token and passes it from a fully connected layer with the Tanh activation.

Finally, the concatenation of four inputs; sentence's general embedding, first and second entity's embeddings from fully-connected layer, and the general embedding of the SDP are fed to a softmax layer. The dimension of the softmax layer is equal to the total number of relation classes. The loss function is chosen as the negative log-likelihood of the correct embedding.

In the following sections, detailed information on pre-trained models, dependency parsers, the flow of generating shortest dependency paths from samples' respective dependency trees, and the proposed model's comprehensive explanation shall be given.

4.2. Pre-Trained Language Models

A common approach applied by the latest models in representation learning is dividing the training process into two as pretraining and fine-tuning. In pretraining, the model or embeddings at hand are trained using a large corpus (such as Wikipedia, PubMed articles, New York Times articles). In fine-tuning, the model or embeddings are further trained with the samples from the target (downstream) task. The majority of models in the domain fall into two paradigms named autoregressive and autoencoding according to their objective functions for generating representations.

In autoregressive models, the probability of a text sequence $s = [w_0, w_1, w_2 \ldots, w_n]$ is modeled as a forward or backward multiplication of conditional next-term probabilities based on the previous or following terms, respectively.

The main setback of the autoregressive models is their training with only one direction (forward or backward). It hinders their performance in deducting deep bidirectional connections. With the emergence of the XLNet, this setback has been solved.

The objective of pretraining models in autoencoding can be simplified as regenerating the original data from a distorted version. The most popular approach in the pretraining domain based on this idea is BERT. In BERT, the training task is to find the values of the masked terms in a text sequence.

In this thesis work, BERT and XLNet are chosen as examples for autoencoding and autoregressive language models, respectively. In the following subsections, details on the methodologies for the two models can be found.

4.2.1. Bidirectional Encoder Representations from Transformers

Bert is a multilayer bidirectional transformer encoding mechanism whose architecture is based on transformers [29]. In the following paragraphs, input structure, embedding representation, objective functions, and overall structure of BERT are explained.

The model takes a text sequence as input that can consist of one or two sentences, tokenizes them using WordPiece [30] in which rarer words are represented by most frequent subwords, adds [CLS] token to the top of the tokens list and [SEP] token between two sentences if the input consists of two sentences otherwise to the end of the list.

In BERT, the final input embedding of a token is built by summing token, segment, and position embeddings. Token embeddings correspond to the embeddings received from WordPiece; segments embeddings represent whether the term is in the first sentence or in the second, and position embeddings are embeddings associated with the index position of the token in input as its name suggests.

The process of pretraining has two tasks with different objectives. In the first one named Masked LM, a certain percent (15% in default) of the input tokens are selected to be replaced by [MASK] token. The objective is to predict the original token id of the masked tokens by feeding the hidden vectors of the masked tokens into a softmax layer whose output dimension is the number of items in the vocabulary. Unfortunately, replacing all the selected terms with the mask token creates a discrepancy between pretraining and fine-tuning. To solve this discrepancy issue, 80% of the items designated to be masked are replaced with [MASK] token, while 10% of them are replaced with random words, and the remaining 10% of them are left untouched.

The second task is Next Sentence Prediction, a process that focuses on capturing the relationship between two sentences. In this task, the input consists of two sentences

following each other. In the training phase of the model, half of the time second sentence of the input is replaced with a random one. The goal is to predict whether the second sentence is the original one or not. To predict the outcome, the last hidden vector of the [CLS] token is fed to a binary classifier. The output of the token [CLS] can also be used as the general representation of the input text sequence in text classification tasks.

The transformer structure is constructed by concatenating multiple self-attention and position-wise feedforward layers. In a self-attention layer, the output h_x is calculated by:

$$
\begin{aligned}
Q_x &= W^Q * X \\
K_x &= W^K * X \\
A_x &= \text{Softmax}(\frac{q_i * K_i}{\sqrt{d}}) \\
V_x &= W^V * X \\
h_x &= (A_x * V_x) + X
\end{aligned}
$$

Where W^Q, W^K, and W^V are the query, key, value weights of the self-attention layer, Q_x, K_x, V_x are the query, key, value matrices calculated for the input X, and A_x is the attention coefficients matrix. As the next step, the output of the self-attention layer is normalized and fed through the position-wise feedforward system [29].

In the paper of the proposed model, two versions named $BERT_{base}$ and $BERT_{large}$ have been analyzed. The differences between them are in hyperparameters, which are; the sizes of the hidden state outputs (768 in base, 1024 in large), the number of transformer layers (12 in base, 24 in large), and the number of self-attention heads in a transformer layer (12 in base, 16 in large). In the experiments of this thesis, both versions are used in acquiring representations.

4.2.2. Generalized Autoregressive Pretraining for Language Understanding

XLnet [31] is an autoregressive language model that combines the powerful aspects of both autoregressive and autoencoding paradigms. It surpasses the state-of-art pretraining method BERT in various tasks, including but not limited to reading comprehension, text classification, question answering by a margin of 5% to 10% depending on the task. It is constructed with the architecture of transformer-XL [32]. In subsequent paragraphs, the input representation, objective function, and two-stream attention mechanism which enables the model to learn bidirectional relations are explained.

Like BERT, XLNet accepts a text sequence as an input, applies WordPiece algorithm to find the most frequent subwords, and tokenizes the sequence according to them. In the token representation of the input, <cls> token, which corresponds to the [CLS] token in BERT, is added to the end of the tokens list. In cases where the sequence consists of two sentences, <sep> token is added between the sentences. Otherwise, it is added before the <cls>.

The training objective of XLNet calculates the probability of a term t whose conditions are based on all permutations of terms in a sequence, not just in a forward or backward manner like in ELMo. To elaborate on the idea, let us give an example. Assume that we have a sequence of terms $s = [x_1, x_2, x_3, x_4]$. In a common autoregressive language model, the likelihood of sequence s is calculated by:

$$\log(p_\theta(s)) = \sum_{t=1}^{T} (\log(p_\theta(x_t|x_{i<t}))$$

In which the term x in position t is conditional to previous terms in sequence. This likelihood calculation is an example of forward autoregression. In next-token probability calculation of the modified version used by XLNet, conditional probabilities for all permutations of tokens' orders in a sequence are taken into account. For the sequence s some permutation examples are $[1, 2, 3, 4]$, $[3, 2, 4, 1]$, $[1, 4, 2, 3]$.

For the case of order $z = [1, 4, 2, 3]$, the log-likelihood of sequence s becomes:

$$\log(p_\theta(s)) = \log(p_\theta(x_1)) + \log(p_\theta(x_4|x_1)) + \log(p_\theta(x_2|x_4, x_1)) + \log(p_\theta(x_3|x_2, x_4, x_1))$$

In generalized form, the log-likelihood of a sequence for a permutation (factorization) order z is:

$$\log(p_\theta(s)) = \sum_{t=1}^{|z|} (\log(p(x_{z_t}|x_{z_{i<t}})))$$

Where $\log(p(x_{z_t}|x_{z_{i<t}}))$ is the next-token probability for term x_{z_t} given the previous terms in the permutation order z of the input sequence.

In calculating log-likelihoods of sequences, the index information of tokens are needed. Otherwise, the probability of an item x coming after item k would be equal for all x's. For instance, in the scenario without the index information, the sentences "Apple is red, not cyan" and "Apple is cyan, not red" are equally likely. By constructing a proper attention mask of transformers, the permutation order and the index of terms in sequence can be integrated into the next-token distribution. Note that in the creation of proper attention masks, the original sequence of the tokens should be preserved.

In order to calculate the next-item probability of x_{z_t}, the target-aware representations (i.e., representations in which term indexes are known) should use only the position information of z_t, not context. But for the terms coming after the x_{z_t} in the permutation z, i.e. $x_{z_{k>t}}$, the context of x_{z_t} should be known. In the original design of the transformer mechanism, the index information is encoded inside the embeddings of words. Even though the original Transformer can be used for the cases where the context x_{z_t} is needed (in the prediction of terms $x_{z_{k>t}}$), a new representation is required for instances where the context information should not be known.

To solve this issue, two-stream self-attention is proposed. As its name suggests, the proposed structure employs two hidden representations: query and content. The content representation serves a similar function to the hidden states of Transformer. It encodes both the context information X_{z_t} and index information z_t. Denoted by h_z, in the m^{th} layer, it is calculated as:

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(Q = h_{z_t}^{(m-1)}, KV = h_{z_{\leq t}}^{(m-1)}; \theta)$$

The query representation uses the information of index z_t but not the context information to fit the constraints stated above. Denoted by g_z, its calculation in the m^{th} layer is:

$$g_{z_t}^{(m)} \leftarrow \text{Attention}(Q = g_{z_t}^{(m-1)}, KV = h_{z_{<t}}^{(m-1)}; \theta)$$

Q, K, V are the query, key, value inputs in an attention process [29]. The representation update of the two-stream self-attention follows the original attention method. The representation of $g_{z_t}^{(m)}$, where m is the last layer, is used in the next-token distribution calculation. With the integration of two-stream self-attention, the next-token probability for term x_{z_t} would be calculated by:

$$p_\theta(X_{z_t} = x | X_{z_{i<t}}) = \frac{\exp(e(x)^T g_\theta(X_{z_{<t}}, z_t))}{\sum_{x'} \exp(e(x')^T g_\theta(X_{z_{<t}}, z_t))}$$

where $e(x)$ denotes the input embedding of x.

4.3. Dependency Parsing

To analyze the performance of the proposed system with the SDPs generated from different dependency parsers, we have selected Stanford Neural Parser, HPSG Parser, and LAL Parser. Stanford's Neural Parser is one of the most commonly used dependency parsers in the domain. Furthermore, LAL and HPSG parsers are the first and fourth state-of-art in the dependency parsing task on Penn English Treebank, respectively. In the following sub-sections, each one's methodology is examined in detail.

4.3.1. Stanford Neural Dependency Parser

Stanford dependency parser [33] solves the problems of earlier approaches caused by their usage of numerous manually generated sparse features with a transition-based neural network representing features as dense vectors. With dense representations, no manual work for creating feature templates is needed, the data insufficiency is no longer a problem, and the time for feature selection has been significantly decreased.

In transition-based dependency parsing, the objective is to predict the sequence of transitions from a given initial configuration to a desired final configuration, generating the dependency tree from the resulting transition sequence. The paper selects the arc-standard transition algorithm [34] as its dependency parsing system .

In the arc-standard algorithm, a configuration $c = (s, b, A)$ contains three parts; a stack s, a buffer b, and a dependency arcs set A. The starting configuration of a text input $w_1, w_2, \ldots, w_n$ is $s = [\text{ROOT}]$, $b = [w_1, w_2, ..., w_n]$, $A = 0$. A transition from one configuration to another can happen in three ways:

- LEFT-ARC(l): adds $s_1 \rightarrow s_2$ with label l to arcs set. Pops s_2 from the stack. More than 1 item in the stack is required.

- RIGHT-ARC(l): add $s_2 \rightarrow s_1$ with label l to arcs set. Pops s_1 from the stack. More than 1 item in the stack is required.
- SHIFT: Removes b_1 from the buffer and adds it to the top of the stack. More than 1 item in the buffer is required.

Where s_i is the i^{th} item in a stack from the top and b_i is the i^{th} item in a buffer. A configuration is considered final if the only item in the stack is ROOT and the buffer is empty. The parse tree then can be constructed from the arcs in set A.

The objective function of the trained neural network is to predict the correct transition from a given configuration. The number of possible transitions is equal to $2 * N_l + 1$, where N_l corresponds to the number of dependency labels in the system. The configuration representation that is the input of the neural network is generated using embeddings of items from sets of words S^w, POS tags S^t, and dependency labels S^l.

Word set consists of the top three and first three words on the stack and buffer, the first two leftmost/rightmost children of the top two words on the stack, and the leftmost of leftmost/rightmost of rightmost children of the top two words on the stack. The input's word embeddings vector is constructed by concatenating the embeddings of the above words.

The POS set is the corresponding POS tags of the items in the word set. The vector used by the network for POS tags is constructed in the same way as the words vector. The items in the labels set are the labels of the relations between the words. The embeddings of the labels are also concatenated to be used by the network. The word, POS, and label sets have 18, 18, and 12 items. A special token NULL is given to places in sets where an item is unavailable.

The structure of the neural network is quite simple. It consists of only one hidden layer. The three concatenated embedding vectors are given to the hidden layer with d nodes using a cube activation function:

$$h = (W^w x^w + W^t x^t + W^l x^l + b)^3$$

On top of the hidden layer, a softmax layer with N_l dimensions is added. The cross-entropy loss function is used with AdaGrad as the optimizer. Pre-trained embeddings from (Collobert et al. 2011) are chosen for English. A dropout is applied before the softmax layer with a rate of 0.5.

The model achieves a 91.8% accuracy in unlabeled attachment score and 89.6% accuracy in labeled attachment score on English Penn Treebank with a speed of 654 sentences per second.

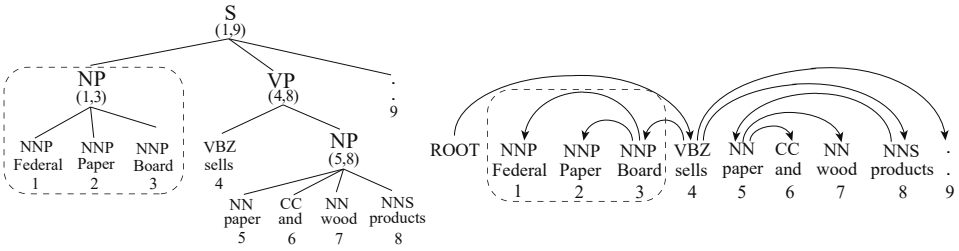
4.3.2. Head-Driven Phrase Structure Grammar Parsing

This parser [35] is the first successful attempt to represent the items in the Penn Treebank as a simplified version of Head Phrase Structure Grammar (HPSG) by combining constituent and dependency parse information. HPSG is a highly lexicalized, constraint-based grammar created by Carl Pollard and Ivan Sang.

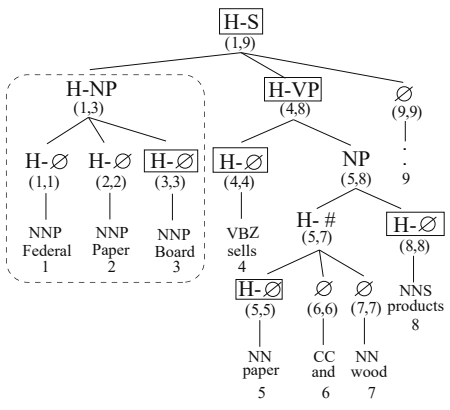
By converting simplified HPSG to a tree representation based on a span structure like a constituent tree, HPSG trees are made compatible with the existing parsers. The authors define two alternative tree structures for this conversion task: division span and joint span.

In the division span, a span is divided into two based on the relative position of its head term. A special token H is added to the categories of sub spans on the left of the head term, including the sub span containing the head as the last item, to differentiate the two parts. This way, head information of a span is incorporated into the standard constituent tree representation. After this operation, a span's head term can be found by traversing its last daughter spans with H.

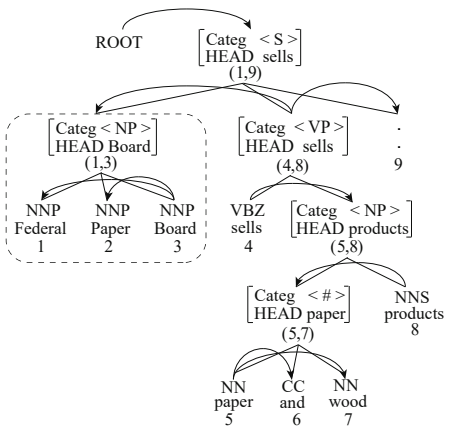
The second structure, Joint span, is a recursive one, consisting of its children phrases (which are also joint spans), category of the span, and dependency relations between heads of those children phrases. A sentence's tree representations of the proposed structures are shown in Figure 4.1.



(a) Constituent and dependency.



(b) Division span structure.



(c) Joint span structure.

Figure 4.1. Constituent, dependency and two simplified HPSG tree representation of the same sentence. The dotted box shows the same phrase. [35]

As both representations generate their outputs in a span-like manner, constituent tree parsers with slight alterations can be used to predict the simplified HPSG tree of a given sentence. For this objective, the authors propose an encoding-decoding model for each representation. The difference between the two is that while the model for division requires only span scores, the joint model needs both span and dependency scores.

The proposed models follow the same skeletal structure, divided into token representation, encoder (self-attention), scoring, and decoder. The tokens are acquired by concatenating character, word, and POS embeddings. CharLSTM [36] is used to generate the character embeddings. Word embeddings are concatenations of random vectors and 100 dimensional GloVe embeddings. Finally, the embeddings for POS tags are initiated randomly.

In the encoder phase, an adaptation of the self-attention mechanism [29] is used. The position information is explicitly given in the input matrices of the encoder by concatenating respective position embeddings to each content embedding (token representation).

Decoders of both representations use the same process to score spans for constituent tree parsing. The span scorer is a layer feed-forward network in which normalization is applied to the results before the nonlinearity operation. For nonlinearity, the Rectified Linear Unit function is chosen. Scorer takes the vector of the span as input. The vector for a span, starting at the i^{th} word and ending at the j^{th} word, denoted as s_{ij}, is computed as:

$$s_{ij} = [\overrightarrow{h_j} - \overrightarrow{h_{i-1}}; \overleftarrow{h_{j+1}} - \overleftarrow{h_i}]$$

where $\overrightarrow{h_i}$ is the forward and $\overleftarrow{h_i}$ is the backward representation of the i^{th} word, constructed by splitting the encoder output for the i^{th} word to half.

$$h_i = [\overrightarrow{h_i} : \overleftarrow{h_i}]$$

The function for generating the span scores S(i,j) is then:

$$S(i,j) = W_2 g(LN(W_1 s_{ij} + b_1)) + b_2$$

Where W_1 and W_2 are learnable weights, b_1 and b_2 are biases, LN is linear normalization, and g is the ReLU function. The score of category l is given as:

$$S_{categ}(i,j,l) = [S(i,j)]_l$$

where $[]_l$ corresponds to the value of the span score vector at the index for category l.

The score of a constituent tree s(T) is calculated by summing up all spans in the tree with category l:

$$s(T) = \sum_{(i,j,l) \in T} S_{categ}(i,j,l)$$

A CKY-style algorithm is applied to find the highest scoring tree using the span predictions. The system's objective is to find the golden parse tree T^*, where for all trees T:

$$s(T^*) >= s(T) + \Delta(T, T^*)$$

condition is fulfilled.

Δ is the hamming loss on the labeled spans. The loss function is defined as the following hinge loss:

$$J_1 = \max(0, \max_T[s(T) + \Delta(T, T^*)] - s(T^*))$$

In the dependency parsing part, division and joint approaches have different methods. Division span trains a multiclass classifier to predict the dependency labels between a head h_i and its children. The classifier is optimized to minimize the cross-entropy loss of negative log probability for the true dependency label of the head child-parent tuple (x_i, h_j).

$$J_{labels}(\theta) = -\log P_\theta(l_i|x_i, h_j)$$

The total loss of the division is defined as the sum of j_labels and j_1.

$$J_{division}(\theta) = J_1(\theta) + J_{labels}(\theta)$$

Joint span handles the dependency parsing by predicting each word's possible head distribution. (Dozat and Manning, 2017)'s biaffine attention mechanism is used to calculate probability distribution. The child-parent score α_{ij} where i^{th} word is the child (dependent) and j^{th} word is the parent (head), is calculated as:

$$\alpha_{ij} = h_i^{(d)^T} W h_j^{(h)} + U^T h_i^{(d)} + V^T h_j(h) + b$$

Where $h_i^{(d)}$ and $h_j^{(h)}$ are the dependent and head representations of the i^{th} and j^{th} word, respectively, acquired by running their encoder outputs through individual one layered perceptrons of dependent and head. W, U, and V are the learnable matrices. The loss function for dependency prediction of the joint span is cross-entropy for the sum of negative log probabilities of head i given dependent j, and dependency label l given the tuple of head i and dependent j.

The approach achieves F1 values of 96.33% for constituent parsing and F1 values of 97.20% and 95.72% on unlabeled attachment scores and labeled attachment scores for dependency in English Penn Treebank. At the writing of the thesis it is the 4^{th} best model.

4.3.3. Head-Driven Phrase Structure Grammar Parsing with Label Attention Layer

LAL parser [37] is the state-of-the-art approach in the domains of constituent and dependency parsing by the time this thesis is written. It is built upon the HPSG parser by constituting self-attention with a new layer named Label Attention Layer into the original system.

The structure of the LAL parser consists of token representation, Label Attention Layer as encoder, scorer, and decoder parts. LAL parser follows the joint span model of the HPSG parser (which can be found in Section 4.3.2.) since it performs better than the division span. Furthermore, considering that token representation, scorer, and decoder mechanisms are the same as the baseline HPSG parser, these parts' explanations are omitted here. Instead, in the following paragraphs, the details of the Label Attention Layer, its differences to self-attention, and the performance of the model shall be investigated.

The authors propose that the word representations can be improved by incorporating each label's attention-weighted interpretation of the sentence into the information gained from self-attention results. The Label Attention Layer is a modified version of self-attention [29], in which only one query vector is available in place of a query matrix that contains a different vector for each term. Additionally, no orthogonality constraint is involved in training the label attention mechanism. Thus in LAL, each dependency label can have several attention heads representing it, and also, an attention head can be used in the representation of several labels.

For attention head i and input matrix X, which consists of word vectors, the corresponding attention weights a_i are calculated as:

$$a_i = \text{Softmax}(\frac{q_i * K_i}{\sqrt{d}})$$

Where q_i is the query vector of the attention head, K_i is the matrix of key vectors, and d is the number of dimensions the query and key vectors have. The K_i is computed as:

$$K_i = W_i^K * X$$

In which W_i^K corresponds to the key matrix of the i^{th} head in label attention. Attention heads in Label Attention do not have query matrices. In lieu, each of them has a query vector. As a consequence, each head provides a single vector instead of a matrix of vectors as output. The output of a head, i.e. context vector c_i is calculated as:

$$c_i = a_i * V_i$$

Where a_i is the attention weights calculated previously and V_i is the matrix of value vectors acquired by multiplying the head's value matrix W_i^V with input matrix X as:

$$V_i = W_i^V * X$$

The resulting context vector c_i is added to each vector in input matrix X. The outcome of the addition is then projected to a lower-dimensional space and normalized. Next, the normalized output is distributed to the input words such that the representation of a word becomes:

$$W_k = [H_k^1 : H_k^2 : \cdots : H_k^n]$$

In which W_k is the k^{th} word in input tokens, H_k^i is the k^{th} vector in the output of the attention head i. After this, the word representations are fed through a position-wise Feed-Forward Layer, which follows [vaswani et al. 2017], like in HPSG to be used in span predictions.

In the dataset of Penn Treebank, the LAL parser achieves the state-of-the-art results with 97.42 for unlabeled attachment score (UAS), 96.26 for labeled attachment score (LAS) in English, 94.56 for UAS, and 89.28 for LAS in Chinese.

4.4. Shortest Dependency Path Generation

The shortest dependency path generation is handled as a preprocess. First, the raw text of each sample in the datasets are fed to dependency parsers. In this thesis, three different dependency parsers named Stanford Neural Dependency parser, HPSG parser, and LAL parser are compared. The details of each parser are given in the Dependency Parsing section. Second, the results of the parsers are processed to generate the shortest dependency trees. The flow of this process is as follows:

(i) Find the tokens that constitute each entity in the dependency tree.
(ii) Generate the list of parents from root to the token for each token t.
(iii) For token pairs (t_1, t_2) where t_1 and t_2 are the tokens of the first and second entity, compare items in the same index in the parents lists of tokens until a mismatch occurs. If no mismatch occurs and a list depletes, go to step vi.
(iv) Revert t_1's remaining parents list, not including the mismatch index.
(v) Append t_2's remaining parents list (mismatch index included) to the output of the v^{th} step. Go to step vii.
(vi) If the parents list of t_1 is emptied, return t_2's remaining terms. If t_2's parents list is emptied, then return the reverse of the remaining items in t_1's list. Include the last compared item in both cases.
(vii) Apply steps iii-vi for each token pair.
(viii) Take the shortest list as the shortest dependency path.

(ix) Replace the first and last items with the first and second entities.

(x) Return the joined string of items with space as the separator.

4.5. Proposed Model

The general flow of the proposed model can be seen in Figure 4.2. Initially, as a pre-training step for acquiring the shortest dependency path, the sentence is given to a dependency parser whose outputs are fed to SDP generator with the target entities. The shortest dependency path of a sentence $s = (w_1, w_2 \dots w_t \dots w_r \dots w_k \dots w_m \dots w_n)$ is generated as $SDP_s = (w_t \dots w_r \dots w_k \dots w_m)$ where w_x is the x^{th} word, t and r are the indexes of the first and last words that constitute the first entity, and k and m are the respective indexes for the second one.

To identify the spans of the entities by the sentence encoder (the pre-trained model that receives the whole sentence), special tokens of [E11],[E12], and [E21], [E22] showing the starting and ending points of the first and second entities respectively, are added to the sentence. With the addition of special tokens sentence s becomes $s' = (w_1, w_2 \dots [\text{E11}], w_t \dots w_r, [\text{E12}] \dots [\text{E21}], w_k \dots w_m, [\text{E22}] \dots w_n)$.

The sentence encoder tokenizes the tagged sentence, generates input mask and entity masks (where the tokens of a particular entity are 1, others are 0), and produces the embeddings H_i for all tokens and [CLS] which corresponds to the sentence embedding. From the SDP encoder, only the [CLS]'s embedding is acquired. Note that in the XLNet version, a module named sequence summary, a fully-connected layer with tanh activation, is applied to the <cls>(the equivalent of [CLS]) token's last hidden state as an additional step.

The final embeddings for entities are produced by multiplying the token embeddings acquired from the last hidden layer of the pre-trained model with the entity masks and dividing each term by the number of ones in the mask, in other words, by averaging the embeddings of the entities tokens. The averaged embedding is then fed to a fully-connected layer that uses tanh as the activation function. With all these, the equations for entity embeddings become:

$$V_{e_1} = W_1 \left[\tanh \left(\frac{1}{r-t+1} \sum_{i=t}^{r} H_i \right) \right] + b_1$$

$$V_{e_2} = W_2 \left[\tanh \left(\frac{1}{m-k+1} \sum_{i=k}^{m} H_i \right) \right] + b_2$$

where W_1, W_2 and b_1, b_2 are equal to each other.

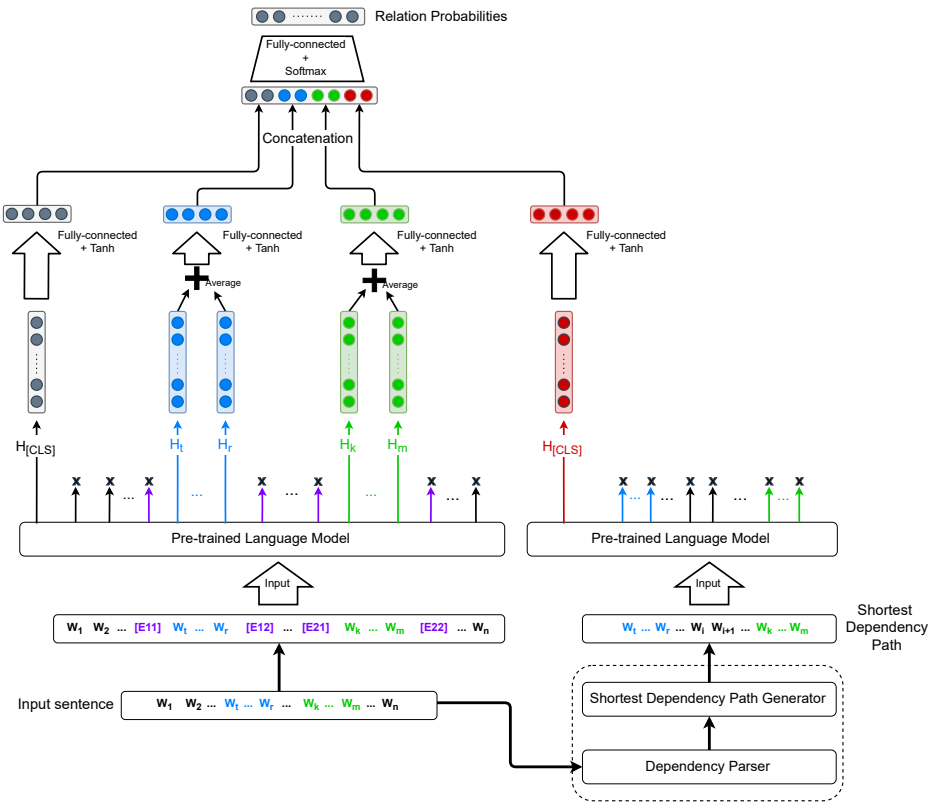


Figure 4.2. The Architecture of the Proposed Model. The Dotted Box Shows the Preprocessing Steps.

The sentence and SDP embeddings also pass through their respective fully-connected layer with tanh activation:

$$V_{sent} = W_0 \left(\tanh\left(H_{[CLS]_{sent}}\right)\right) + b_0$$
$$V_{SDP} = W_3 \left(\tanh\left(H_{[CLS]_{SDP}}\right)\right) + b_3$$

All W_1, W_2, W_0, and W_3 have the same dimension, $W_i \in R^{dxd}$ where d is the dimension of the embeddings received from the pre-trained model. Before each fully connected layer, a dropout with 0.1 probability is applied.

The outputs from the fully-connected layers are concatenated and fed through a softmax layer. i.e:

$$h' = W_4 \left[\text{concat}\left(V_{sent}, V_{e_1}, V_{e_2}, V_{SDP}\right)\right] + b_4$$
$$p = \text{softmax}\left(h'\right)$$

The dimension of the results from the softmax layer is equal to the number of relations in the system. The loss function is chosen as the negative log-likelihood of the correct embedding.

5. EXPERIMENTS AND RESULTS

The shortest dependency path texts are generated beforehand using the stanza library for the Stanford parser and the official GitHub repositories of the parsers for the HPSG and LAL. For the results acquired from HPSG and LAL parsers, the weights of the best-performing models are taken from their respective repositories. Figure 5.1 shows the shortest dependency paths, and dependency tree's of all parsers for a sample text from the SemEval dataset.

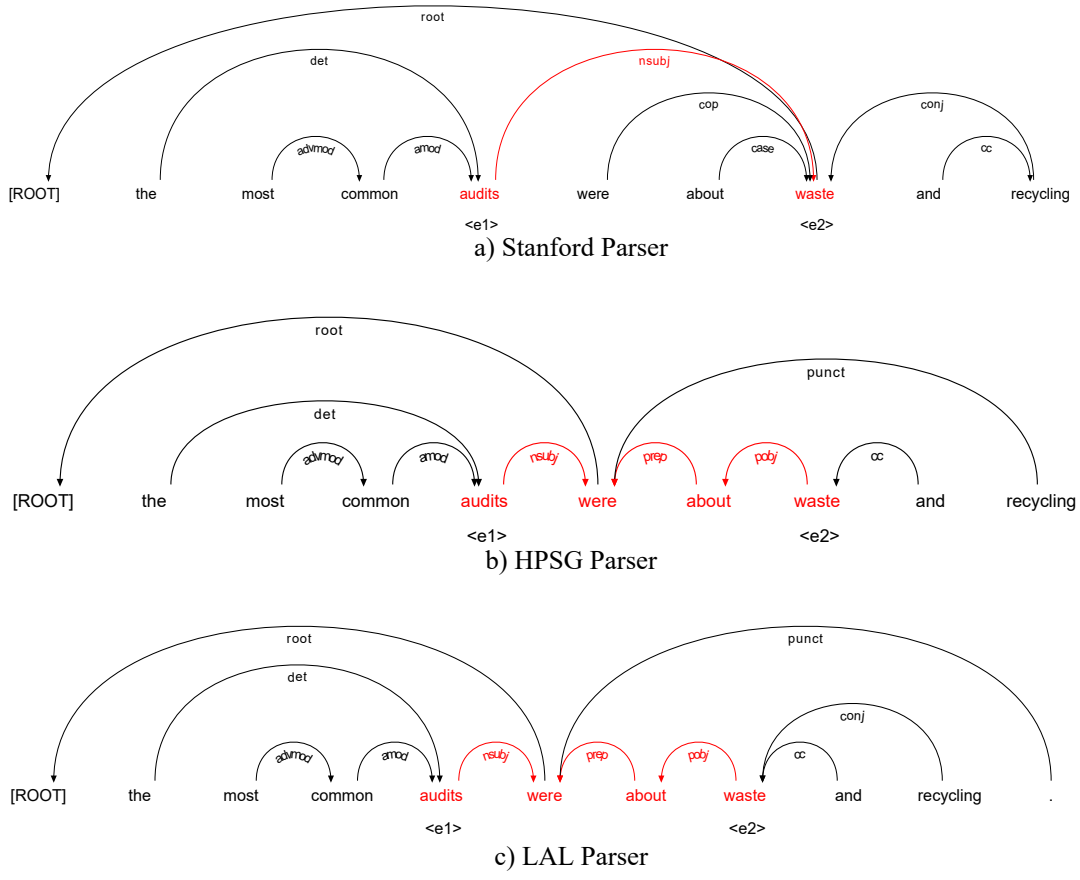


Figure 5.1. Shortest Dependency Path and Dependency Tree Representation of a Sentence for Each Parser

In Figure 5.1, the arrows represent the dependency relation between terms from child to parent, whose class is given just below the arrow. Entities are tagged with <e1> and <e2> under terms. Red items are depicting the shortest dependency path (SDP) between <e1> and <e2>. The SDP representation is constructed by following the red arrows from <e1 > to <e2 > ignoring the directionality, only using the terms.

In our experiments we have compared the performances of each combination of Stanford, HPSG and LAL dependency parsers with Bert and XLNet embeddings in two datasets, SemEval and TACRED. The experiments are conducted on two GPUs. For SemEval results, variants with base pre-trained models are trained with a GTX1060, others are trained with an RTX3090. For TACRED, all training processes are done with the RTX3090.

5.1. Experiments in SemEval-2010 Task 8

In SemEval experiments, both base and large versions of pre-trained language models are used, increasing the total number of compared models to 16. Each model is trained for ten epochs that are further divided into five checkpoints. The performance of the models are evaluated at each checkpoint. The learning rate of 2e-5 is applied to the models using a BERT pre-trained model, while this value is chosen as 1e-5 for the XLNet cases. The rates of dropouts are 0.1 in every model.

Since no official validation dataset is available, the models are compared on the test set. The calculated metrics are micro and macro F1 scores over 19 and 10 classes, respectively. In micro F1, each tuple of relation and its direction is considered as a separate class, increasing the number of classes to 19 (2 * number of classes + no-relation). In the macro version, F1 scores of 9 classes are averaged without the no-relation, taking the directionality into account instead of representing them as separate classes. For example, (r,e2,e1) is considered as a false prediction for the gold relation (r,e1,e2) in F1 calculation of r, even though r is the correct relation.

The official evaluation method of the dataset is chosen as macro F1 excluding no-relation class by the authors of the SemEval-2010 Task 8 dataset [26]. For each model, its best-scoring checkpoint on micro F1 measurements is taken for macro F1 evaluation. In Table 5.1, each model's micro F1 score over 19 classes (where each relation direction is considered as a separate class), the training epoch and the checkpoint of the model (given in parentheses, a value between 0 and 4, inclusive), and the corresponding model's macro F-score is given.

Table 5.1. Evaluation Results of the Trained Models Sorted by Official Macro F1 Scores

Model	(19 Class) Micro F1 with no-relation	Epoch (Checkpoint)	(9 Class) Official Macro F1 without no-relation
stanford_bert_base	85.24%	6 (3)	88.53%
baseline_bert_base	85.13%	8 (1)	88.54%
lal_xlnet_base	85.54%	7 (2)	88.68%
hpsg_bert_large	85.90%	4 (1)	88.75%
stanford_xlnet_base	85.50%	5 (4)	88.75%
hpsg_xlnet_base	85.61%	4 (3)	88.79%
lal_bert_base	85.68%	8 (4)	88.80%
baseline_xlnet_base	85.76%	8 (3)	88.80%
hpsg_xlnet_large	85.72%	8 (1)	88.80%
baseline_bert_large	85.68%	5 (3)	89.02%
lal_xlnet_large	86.27%	8 (1)	89.08%
hpsg_bert_base	85.79%	7 (4)	89.09%
stanford_xlnet_large	86.20%	7 (4)	89.33%
baseline_xlnet_large	86.82%	8 (1)	89.83%
lal_bert_large	86.64%	8 (1)	89.84%
stanford_bert_large	86.12%	4 (1)	89.95%

In baseline R-BERT's paper [10], the proposed model is trained for five epochs. However, in our experiments, among 16 alternatives, 13 of them achieve their best micro F1 scores in later epochs (See Table 5.1.). From this, we deduce that our proposed structure requires more than 5 epochs to reach its best form.

Once we compare the official macro F1 scores given in Table 5.1, we see that the best and worst performances belong to models with Stanford parsers. The most significant performance increase obtained by changing the model size was observed in BERT Stanford parsers. On the other hand, in BERT HPSG, the change of language model size decreases the macro F1 score. Otherwise, increasing the PLMs size from base to large always ensures a performance increase in both measurements.

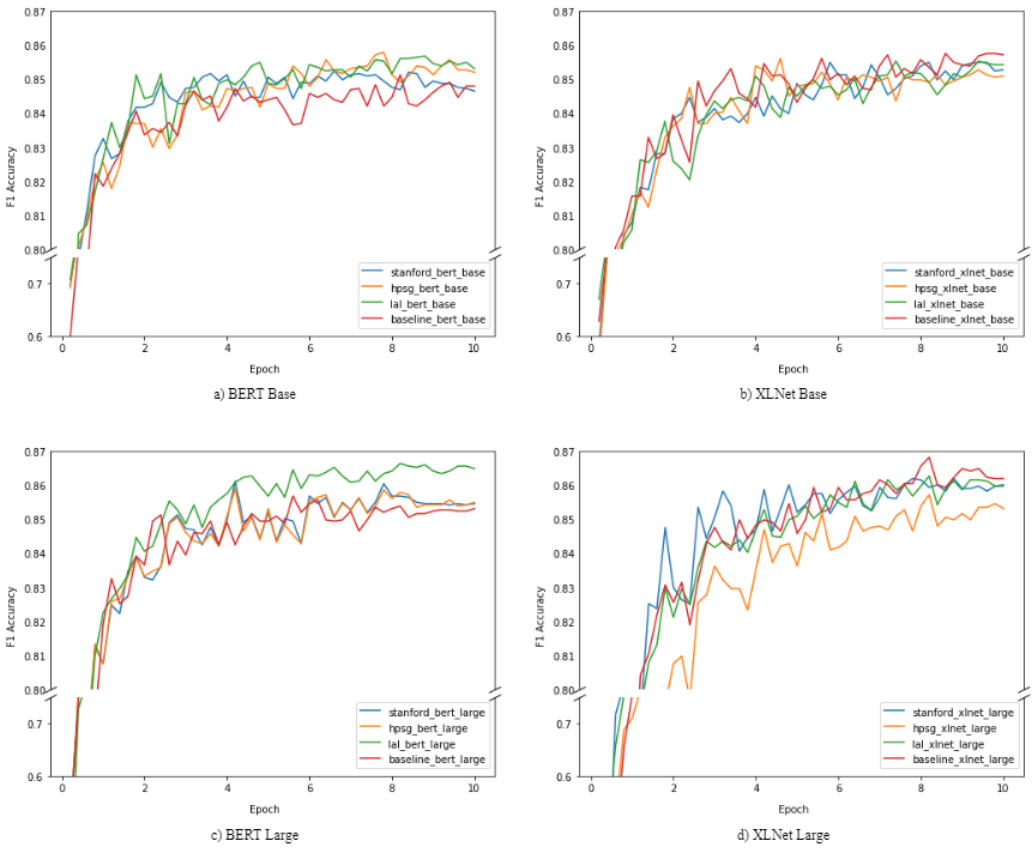


Figure 5.2. The Micro F1 Scores of Baseline and SDP Enhanced Models For Each of the Pre-trained Language Model in SemEval

In Figure 5.2, micro F1 scores of baseline and SDP enhanced models are compared for each pre-trained language model seperately. We observe that among all the models using BERT$_{base}$ and BERT$_{base}$, the ones with the LAL parsers achieve better micro F1 scores than their peers.

Surprisingly, in the XLNet versions, the baselines were found to be the better ones, indicating that SDPs do not increase the performance of XLNet. The reason for that would be the term process orderings of XLNet's transformer heads having the SDP's order for the terms in the SDP. The transformer heads using these orderings incorporate the SDP information into XLNet embeddings.

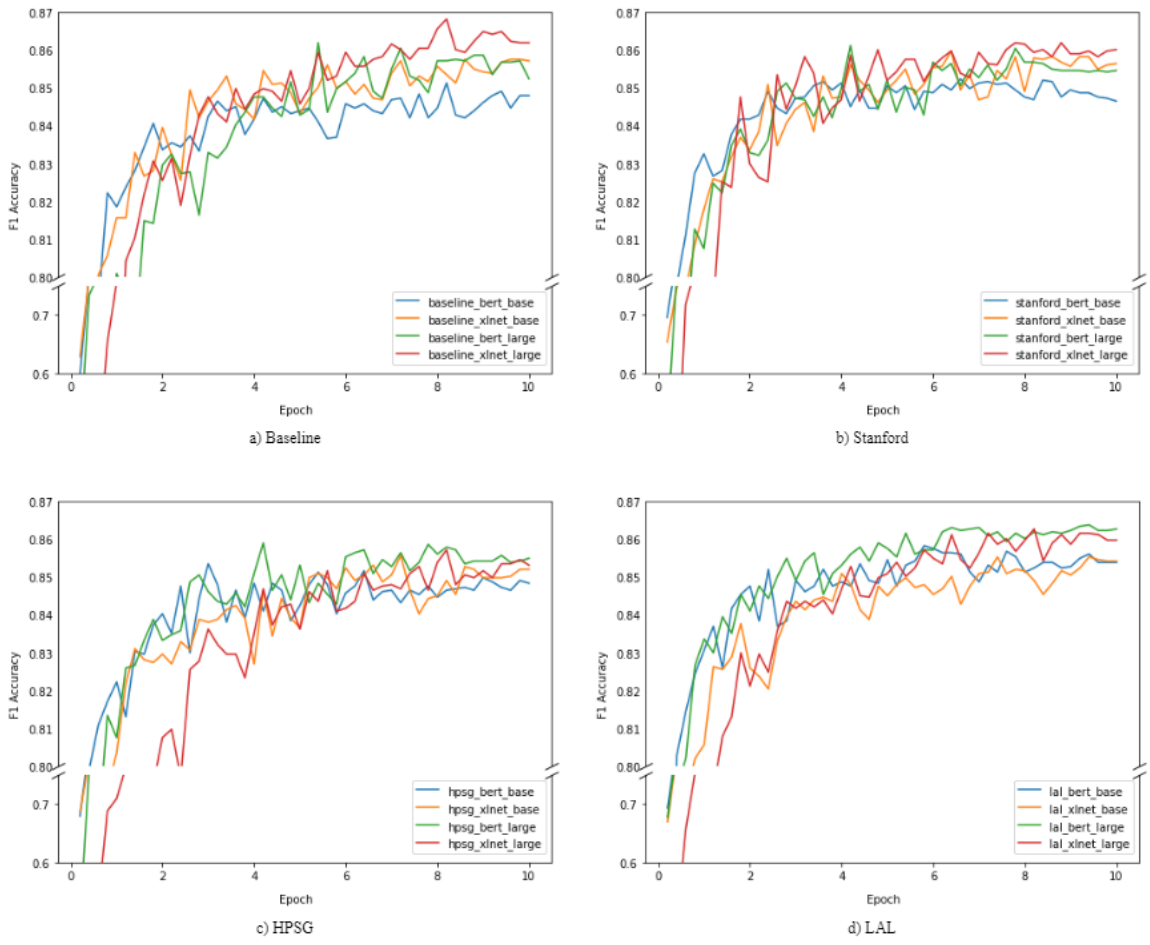


Figure 5.3. The Micro F1 Scores of Pre-trained Language Models For Each of Baseline and SDP Enhanced Models in SemEval-2010 Task 8

In Figure 5.3, micro F1 scores of PLMs are plotted for each SDP algorithm and baseline. Both of Figures 5.2 and 5.3 are depicted using the same information with different groupings of models, each providing specific information about the distinct characteristics of the proposed model.

By comparing the plots of pre-trained language model performances in each dependency parser methodologies given in Figure 5.3, we see that the models using XLNet PLMs give better results than the models with BERT in micro F1 scores. Unfortunately, the same deduction cannot be made from the official macro f1 scorings given in Table 5.1.

Overall, we have found that for the SemEval-2010 Task 8 dataset, combining XLNet PLMs with dependency parsers does not increase the performance of the baseline model with XLNet. However, integrating shortest dependency paths to the models with BERT PLMs increases both macro and micro f1 scores of the baseline up to 1 absolute point.

5.2. Experiments in TACRED

In the experiments of the TACRED dataset, only the large versions of the pre-trained language models are used. The dataset contains three sets to be used in train, dev, and test. All variations are trained with a learning rate of 1e-5 and a dropout rate of 0.1. Each model is trained for ten epochs, each divided into four checkpoints. At each checkpoint, the model's micro accuracy in the dev set is calculated.

The selection of the best checkpoint of the model to be evaluated in test set is made using two methods. The first method selects the checkpoint with the best accuracy score in the dev set among all model checkpoints. As the second method, an early stopping mechanism based on accuracy is applied. If the accuracy of the trained system does not surpass the current best score for five checkpoints, the training is stopped, and the checkpoint with the best score until that moment is selected.

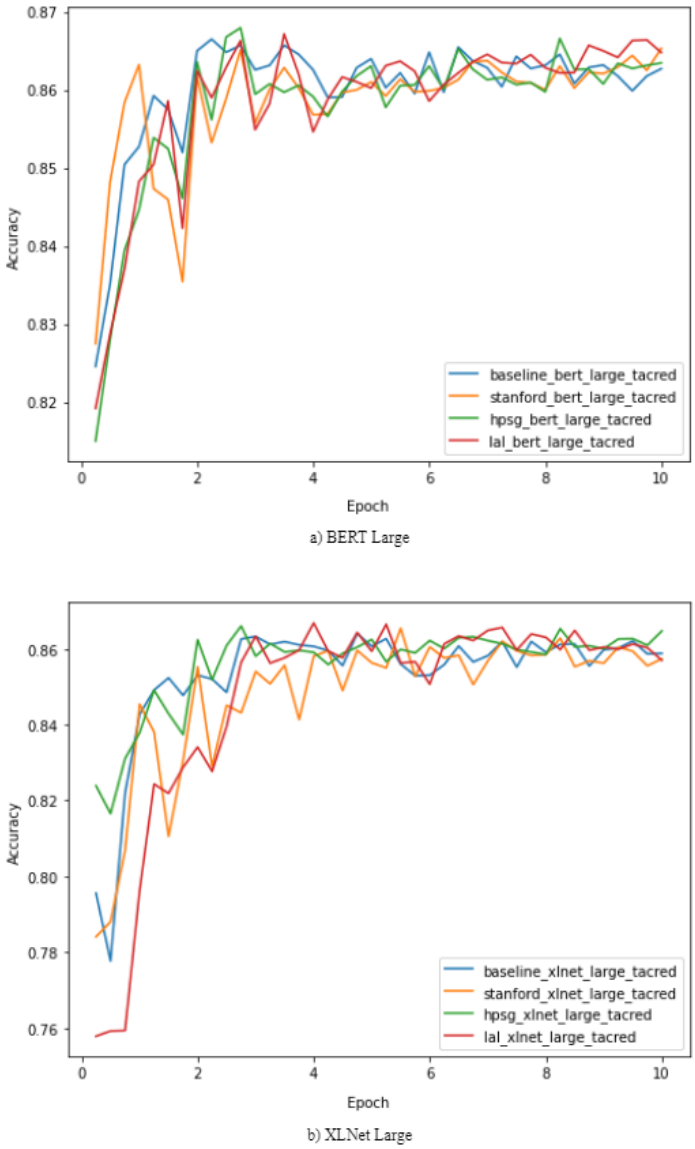


Figure 5.4. The Accuracies of Baseline and SDP Enhanced Models for Each Pre-trained Language Model Used in TACRED Dev Dataset

In Figure 5.4 the baseline and dependency parsers' accuracies are grouped by their pre-trained language models. The accuracies used in plotting the Figure 5.4 and Figure 5.5 are obtained by evaluating the dev set. In both PLMs, the models using SDPs from the Stanford dependency parser perform worse than their peers. The LAL parser versions acquire the lowest scores in the earlier epochs, but by the end of the training, they perform better than the other alternatives in most cases.

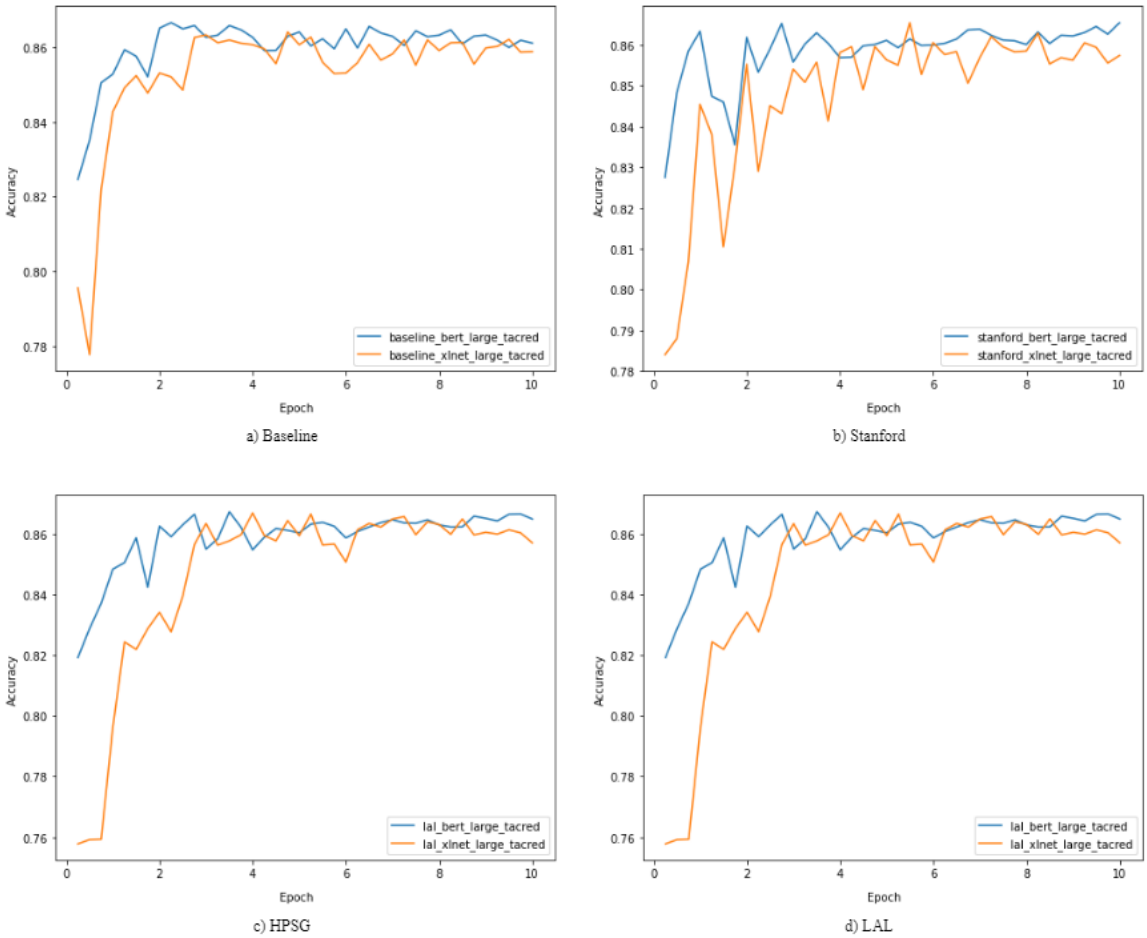


Figure 5.5. The Accuracies of Pre-trained Language Models for Each of Baseline and SDP Enhanced Models Used in TACRED Dev Dataset

The performances of the pre-trained models for each dependency parser version are depicted in Figure 5.5. According to Figure 5.5, the models that apply XLNet learns more slowly than their BERT peers in all cases. Also, the models with BERT PLMs provide better results than their XLNet counterparts in the dev set. However, in the test results (Table 5.2 and Table 5.3), the two of the best performing models are found to be the ones that apply XLNet with LAL and HPSG parsers. The models outperform the closest BERT version by 1.4% and 0.9%, respectively.

In Table 5.2 and Table 5.3 every model's precision, recall, and f1 score on the test set are given with the selected model's epoch information. Each epoch is accompanied by a checkpoint number in a parenthesis which can have a value between 0 and 3.

The results for the models in Table 5.2 are acquired from the best-performing ones in the dev set, while in Table 5.3 the results come from the models selected by applying early stopping on training. The lowest performing models are found to be the ones with the Stanford parser which have performed worse than the baselines. The best performance is obtained by the model that combines XLNet and LAL also achieving the greatest recall score among all. Based on these results, we can deduct that shortest dependency path information does not increase the performance at all times. Nevertheless, the state-of-art approaches enables the system to acquire better results.

Table 5.2. Tacred Evaluation Results of the Trained Models Sorted by Micro F1 Scores Using Best of 10 Epochs

Model	Precision	Recall	F1	Epoch (Checkpoint)
xlnet_large_stanford	77.88%	63.89%	70.19%	5 (2)
bert_large_stanford	78.00%	64.05%	70.34%	2 (3)
bert_large_hpsg	79.45%	65.02%	71.52%	2 (3)
xlnet_large_baseline	75.21%	68.17%	71.52%	4 (3)
bert_large_baseline	75.48%	68.09%	71.60%	3 (0)
bert_large_lal	76.28%	67.46%	71.60%	3 (2)
xlnet_large_hpsg	74.56%	70.59%	72.52%	4 (0)
xlnet_large_lal	74.87%	71.23%	73.01%	4 (0)

Table 5.3. Tacred Evaluation Results of the Trained Models Sorted by Micro F1 Scores Using Early Stopping

Model	Precision	Recall	F1	Epoch (Checkpoint)
bert_large_stanford	77.80%	64.05%	70.14%	1 (0)
xlnet_large_stanford	75.65%	65.58%	70.26%	2 (3)
bert_large_baseline	74.66%	68.10%	71.23%	1 (3)
xlnet_large_baseline	73.32%	69.71%	71.47%	3 (3)
bert_large_hpsg	79.45%	65.02%	71.52%	2 (3)
bert_large_lal	76.28%	67.46%	71.60%	3 (2)
xlnet_large_hpsg	74.56%	70.59%	72.52%	4 (0)
xlnet_large_lal	74.87%	71.23%	73.01%	4 (0)

In the results of Table 5.2, we see the best measurements are acquired from the checkpoints of the fifth epoch or earlier, suggesting that the system is not required to run for ten epochs. Additionally, by comparing the f1 results in Table 5.2 and Table 5.3 we notice that for half of the system variations, the checkpoints acquired with early stopping achieve identical scores from the test set. For the other half, the decrease is also negligible. Thus without a notable decrease in the system's performance, we can significantly lower the time required to train the system by applying early stopping with five checkpoints.

We are unable to determine a method that receives the best results for both Datasets. In SemEval-2010 Task 8, the models that apply Stanford and LAL SDPs with BERT pre-trained language model have received the best evaluation scores, while in TACRED, the best performing approaches are found to be the models using LAL and HPSG parsers with XLNet. Yet, in both datasets, we achieved a better result than the baseline method with our proposed structure. By considering all results, the most effective dependency parser for generating the shortest dependency path is found to be the LAL parser.

6. CONCLUSION

In this thesis work, we proposed an improved version of R-BERT by integrating shortest dependency path embedding acquired from the pre-trained language model to the relation representation embedding of the model.

We evaluate the model combinations of Stanford, HPSG, and LAL dependency parsers with pre-trained language models BERT and XLNet. For each pre-trained language model, base and large versions have been examined. The experiments are done in two commonly used relation extraction datasets, SemEval-2010 Task 8 and TACRED. In SemEval dataset proposed model achieves an F1 score of 89.90%, improving the baseline by 0.65% and 1.41% according to the paper's performance and our calculations. In TACRED, the proposed model achieves a 73.0% F1 score, surpassing the unofficial performance score 69.4% of the baseline by %3.6 percent, increasing the rank of the model from 18^{th} to 7^{th} in papers with code rankings.

In the experiments, we have observed that all versions of the proposed model surpass the performance of the baseline model of BERT_{base} in SemEval-2010 Task 8 dataset except one case. In TACRED, LAL parser increases the performance of both PLMs, HPSG improves the XLNet, and Stanford decreases their performance. We can say that shortest dependency paths obtained from state-of-the-art dependency parsers improve the results of relation extraction.

A promising future work would be the investigation of different dependency representations, such as the augmented shortest dependency path in place of the vanilla version. Additionally, the labels of the dependencies between the terms are not taken into account in the extraction of the shortest dependency paths; embedding that information into the SDP's could further increase the performance of the proposed model. Finally, the current system works only on English texts. Investigation of the model's performance in different languages could be an excellent extension.

REFERENCES

1. Hasegawa, T., S. Sekine and R. Grishman, "Discovering relations among named entities from large corpora", *Proceedings of the 42nd annual meeting on association for computational linguistics*, p. 415, Association for Computational Linguistics, 2004.

2. Bach, N. and S. Badaskar, "A review of relation extraction", .

3. Fundel, K., R. Küffner and R. Zimmer, "RelEx—Relation extraction using dependency parse trees", *Bioinformatics*, Vol. 23, No. 3, pp. 365–371, 2007.

4. Wang, M., "A re-examination of dependency path kernels for relation extraction", *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*, 2008.

5. Liu, C., W. Sun, W. Chao and W. Che, "Convolution neural network for relation extraction", *International Conference on Advanced Data Mining and Applications*, pp. 231–242, Springer, 2013.

6. Miwa, M. and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures", *arXiv preprint arXiv:1601.00770*, 2016.

7. Zhang, Y., P. Qi and C. D. Manning, "Graph convolution over pruned dependency trees improves relation extraction", *arXiv preprint arXiv:1809.10185*, 2018.

8. Liu, Y., F. Wei, S. Li, H. Ji, M. Zhou and H. Wang, "A dependency-based neural network for relation classification", *arXiv preprint arXiv:1507.04646*, 2015.

9. Li, Z., Z. Yang, C. Shen, J. Xu, Y. Zhang and H. Xu, "Integrating shortest dependency path and sentence sequence into a deep learning framework for relation extraction in clinical text", *BMC medical informatics and decision making*, Vol. 19,

No. 1, pp. 1–8, 2019.

10. Wu, S. and Y. He, "Enriching Pre-trained Language Model with Entity Information for Relation Classification", *arXiv preprint arXiv:1905.08284*, 2019.

11. Li, C. and Y. Tian, "Downstream model design of pre-trained language model for relation extraction task", *arXiv preprint arXiv:2004.03786*, 2020.

12. Tao, Q., X. Luo, H. Wang and R. Xu, "Enhancing relation extraction using syntactic indicators and sentential contexts", *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1574–1580, IEEE, 2019.

13. Soares, L. B., N. FitzGerald, J. Ling and T. Kwiatkowski, "Matching the blanks: Distributional similarity for relation learning", *arXiv preprint arXiv:1906.03158*, 2019.

14. Xu, K., Y. Feng, S. Huang and D. Zhao, "Semantic relation classification via convolutional neural networks with simple negative sampling", *arXiv preprint arXiv:1506.07650*, 2015.

15. Turian, J., L.-A. Ratinov and Y. Bengio, "Word Representations: A Simple and General Method for Semi-Supervised Learning", *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 384–394, Association for Computational Linguistics, Uppsala, Sweden, Jul. 2010, `https://aclanthology.org/P10-1040`.

16. Xu, Y., R. Jia, L. Mou, G. Li, Y. Chen, Y. Lu and Z. Jin, "Improved relation classification by deep recurrent neural networks with data augmentation", *arXiv preprint arXiv:1601.03651*, 2016.

17. Lee, J., S. Seo and Y. S. Choi, "Semantic Relation Classification via Bidirectional LSTM Networks with Entity-Aware Attention Using Latent Entity Typing", *Symmetry*, Vol. 11, No. 6, p. 785, 2019.

18. Mintz, M., S. Bills, R. Snow and D. Jurafsky, "Distant supervision for relation extraction without labeled data", *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, Association for Computational Linguistics, Suntec, Singapore, Aug. 2009, `https://www.aclweb.org/anthology/P09-1113`.

19. Riedel, S., L. Yao and A. McCallum, "Modeling relations and their mentions without labeled text", *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 148–163, Springer, 2010.

20. Lin, Y., S. Shen, Z. Liu, H. Luan and M. Sun, "Neural relation extraction with selective attention over instances", *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2124–2133, 2016.

21. Han, X., P. Yu, Z. Liu, M. Sun and P. Li, "Hierarchical relation extraction with coarse-to-fine grained attention", *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2236–2245, 2018.

22. Feng, J., M. Huang, L. Zhao, Y. Yang and X. Zhu, "Reinforcement learning for relation classification from noisy data", *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

23. Zhang, N., S. Deng, Z. Sun, G. Wang, X. Chen, W. Zhang and H. Chen, "via Knowledge Graph Embeddings and Graph Convolution Networks", *arXiv preprint arXiv:1903.01306*, 2019.

24. Yan, Y., N. Okazaki, Y. Matsuo, Z. Yang and M. Ishizuka, "Unsupervised relation extraction by mining Wikipedia texts using information from the web", *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:*

Volume 2-Volume 2, pp. 1021–1029, Association for Computational Linguistics, 2009.

25. Davidov, D. and A. Rappoport, "Classification of semantic relationships between nominals using pattern clusters", *Proceedings of ACL-08: HLT*, pp. 227–235, 2008.

26. Hendrickx, I., S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti, L. Romano and S. Szpakowicz, "SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations between Pairs of Nominals", *Proceedings of the 5th International Workshop on Semantic Evaluation*, pp. 33–38, Association for Computational Linguistics, Uppsala, Sweden, Jul. 2010, `https://aclanthology.org/S10-1006`.

27. Zhang, Y., V. Zhong, D. Chen, G. Angeli and C. D. Manning, "Position-aware Attention and Supervised Data Improve Slot Filling", *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*, pp. 35–45, 2017, `https://nlp.stanford.edu/pubs/zhang2017tacred.pdf`.

28. Bunescu, R. and R. Mooney, "A shortest path dependency kernel for relation extraction", *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pp. 724–731, 2005.

29. Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need", *Advances in neural information processing systems*, pp. 5998–6008, 2017.

30. Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation", *arXiv preprint arXiv:1609.08144*, 2016.

31. Yang, Z., Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov and Q. V. Le, "Xlnet:

Generalized autoregressive pretraining for language understanding", *Advances in neural information processing systems*, Vol. 32, 2019.

32. Dai, Z., Z. Yang, Y. Yang, J. Carbonell, Q. Le and R. Salakhutdinov, "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context", *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, Association for Computational Linguistics, Florence, Italy, Jul. 2019, https://aclanthology.org/P19-1285.

33. Chen, D. and C. D. Manning, "A fast and accurate dependency parser using neural networks", *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 740–750, 2014.

34. Nivre, J., "Incrementality in Deterministic Dependency Parsing", *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pp. 50–57, Association for Computational Linguistics, Barcelona, Spain, Jul. 2004, https://aclanthology.org/W04-0308.

35. Zhou, J. and H. Zhao, "Head-driven phrase structure grammar parsing on Penn treebank", *arXiv preprint arXiv:1907.02684*, 2019.

36. Kitaev, N. and D. Klein, "Constituency Parsing with a Self-Attentive Encoder", *CoRR*, Vol. abs/1805.01052, 2018, http://arxiv.org/abs/1805.01052.

37. Mrini, K., F. Dernoncourt, Q. Tran, T. Bui, W. Chang and N. Nakashole, "Rethinking self-attention: Towards interpretability in neural parsing", *arXiv preprint arXiv:1911.03875*, 2019.