Improving Relation Extraction through Syntax-induced Pre-training with Dependency Masking

Yuanhe Tian♥, Yan Song♠†, Fei Xia♥
♥University of Washington♠ The Chinese University of Hong Kong (Shenzhen)
♥{yhtian, fxia}@uw.edu ♠songyan@cuhk.edu.cn

Abstract

Relation extraction (RE) is an important natural language processing task that predicts the relation between two given entities, where a good understanding of the contextual information is essential to achieve an outstanding model performance. Among different types of contextual information, the auto-generated syntactic information (namely, word dependencies) has shown its effectiveness for the task. However, most existing studies require modifications to the existing baseline architectures (e.g., adding new components, such as GCN, on the top of an encoder) to leverage the syntactic information. To offer an alternative solution, we propose to leverage syntactic information to improve RE by training a syntax-induced encoder on auto-parsed data through dependency masking. Specifically, the syntax-induced encoder is trained by recovering the masked dependency connections and types in first, second, and third orders, which significantly differs from existing studies that train language models or word embeddings by predicting the context words along the dependency paths. Experimental results on two English benchmark datasets, namely, ACE2005EN and SemEval 2010 Task 8 datasets, demonstrate the effectiveness of our approach for RE, where our approach outperforms strong baselines and achieve state-of-the-art results on both datasets.1

1 Introduction

Relation extraction (RE) provides deep analyses of the input text by extracting the relation between two given entities in the input. Therefore, it is an important task in natural language processing (NLP) and is widely used in many downstream NLP applications such as summarization (Wang and Cardie, 2012), question answering systems (Xu et al., 2016) and text mining (Distiawan et al., 2019). To correctly extract the relation between two entities, it normally requires a good modeling and analysis of the input text. Recent models such as LSTM, Transformers (Vaswani et al., 2017), and pre-trained language models (e.g., BERT (Devlin et al., 2019) and XLNet (Yang et al., 2019)) have significantly improved the performance of RE models with an important reason of their encoding power on contextual information. However, such models still reach a bottleneck because it is hard for them to capture structural information of the running text (which is essential for RE) by modeling the text as a linear sequence of words. To deal with this situation, extra knowledge and features (e.g., syntactic knowledge) are used in many studies, while of all choices, the dependency trees have been widely used and demonstrated to be effective (Xu et al., 2015; Zhang et al., 2018; Guo et al., 2019; Mandya et al., 2020; Sun et al., 2020; Yu et al., 2020b; Tian et al., 2021), for the reason that the dependency trees are able to provide long-distance word-word relations which are important structural complement to existing models for RE.

To leverage dependency information, most existing approaches in NLP either treat it as extra input features (Prokopidis and Papageorgiou, 2014; Kiperwasser and Goldberg, 2015; Yu and Bohnet, 2017), which requires heavy feature engineering, or use complicated architectures (Xu et al., 2015; Roth and Lapata, 2016; Marcheggiani and Titov, 2017; Zhang et al., 2018; Li et al., 2018; Guo et al., 2019; Nie et al., 2020; Li et al., 2020a,b; Chen et al., 2020) to encode it, which suffers from the difficulty of designing an effective model. In addition, these approaches normally require dependency trees as extra input when processing sentences, and thus potentially suffer from noises from the dependency trees because of errors from automatic parsing. Therefore, an alternative is needed to leverage dependency information, especially auto-generated
In this paper, we propose to enhance RE through learning a good encoder equipped with dependency information, where the learning is carried out by a dependency-guided process. In detail, a dependency masking approach is designed to introduce such information, where we firstly apply an off-the-shelf dependency parser to large raw data and extract the dependency connections and types from the auto-parsed dependency trees, and then mask these connections and types so as to pre-train a syntax-induced encoder by recovering (predicting) them, which significantly differs from that of training word embeddings (Levy and Goldberg, 2014; Komninos and Manandhar, 2016) by predicting the context words along the dependency relations. In doing so, the dependency information weakly supervises the encoder and the pre-training on dependency masking ensures a selective learning process on those frequent and important dependency relations, which is more flexible than taking dependency parses (with noises) as fixed knowledge. In addition, by noting that higher order dependency information is beneficial in many cases (Coppola and Steedman, 2013; Kamigaito et al., 2018; Li et al., 2020b), we further enhance our approach by pre-training with masking second and third order word dependencies rather than just doing it on the first order. Once pre-trained, the resulted encoder is applied with ordinary fine-tune procedure for RE. Experimental results on two English benchmark datasets, namely, English ACE2005EN and SemEval 2010 Task 8 (Hendrickx et al., 2010), for RE demonstrate the effectiveness of our approach, which outperforms strong baselines and achieves state-of-the-art results on both of the datasets.

2 The Approach

To learn a text encoder with important structural information for RE, we propose to pre-train it with masking and recovering word-word dependency connections and types that are auto-analyzed from
existing parsers. The resulted syntax-induced encoder is thus weakly supervised by such information and provided with necessary syntax integration. In doing so, the dependency information is introduced during pre-training the encoder, thus no extra input is required in applying it to real applications, avoiding particular design of models to leverage such information during inference. Figure 1 illustrates the architecture of our approach to learn from an input sentence $x = x_1x_2 \cdots x_i \cdots x_j \cdots x_n$ with $n$ words and its dependency tree $T_X$, so that the masking and recovering can be formalized by

$$Y^* \equiv \mathcal{D}\mathcal{M}(\mathcal{D}\mathcal{E}(T_X)) \quad (1)$$

and

$$\hat{Y}_M = f(\mathcal{E}\mathcal{N}(X)) \quad (2)$$

respectively, where $Y^*_M$ is the set of all masked dependency connections and types obtained by dependency extraction ($\mathcal{D}\mathcal{E}$) and dependency masking ($\mathcal{D}\mathcal{M}$), and $f$ the process (with pre-training on it) to recover (predict) $Y^*_M$ to $\hat{Y}_M$, with the base encoder $\mathcal{E}\mathcal{N}$ trained accordingly during the process. In the following text, we firstly illustrate dependency extraction, then the process to integrate syntax information into the encoder with dependency masking, and finally the steps to apply the resulted syntax-induced encoder to RE.

### 2.1 Dependency Extraction

To extract dependency information form the input text, we firstly apply an off-the-shelf dependency parser to the input and obtain its dependency tree $T_X$. Then, we extract first, second, and third order dependency information from $T_X$ and represent them in the form of tuple, i.e., $(x_i, x_j, type)$, where there is a connection between $x_i$ and $x_j$ and the dependency type (which is directional) of $x_j$ towards $x_j$ is type. Specifically, for the first order dependencies, we directly use the dependency connections and types in $T_X$, where we construct a directed connection between $x_i$ and $x_j$ (denoted by $(x_i, x_j)$) if $x_j$ is the head of $x_i$ and the dependency type between them is the syntactic role (e.g., nominal subject) of $x_i$ with respect to $x_j$. For the second order dependencies, we construct a second order dependency connection between $x_i$ and $x_j$ if there is a word $x' \in x_i$ that connects to both $x_i$ and $x_j$ by two connections $(x_i, x')$ and $(x', x_j)$ in $T_X$. In the second order case, we define three types for their connections namely, ancestor, sister, and descendant, according to the position of $x_i$ and $x_j$ in the dependency tree $T_X$, which are illustrated in (I), (II), and (III) in Figure 2 (a), respectively. Similarly, for third order dependencies, we extend the types to four ones, namely, ancestor, uncle, nephew, and descendant, which are illustrated in (I)-(IV) in Figure 2 (b).

### 2.2 Dependency Masking and Prediction

Previous studies leveraging dependency information by pre-training mainly focused on predicting the context words associated through dependency connections. Compared with these approaches, ours focuses on a different direction to leverage auto-parsed dependency information through learning word-word associations (i.e., dependency connections) and their dependency types. In doing so, we propose a weakly supervised learning task, namely, dependency masking (DM) with masked dependency prediction (MDP), to enhance text encoder pre-training, where they are paired processes that DM masks all connections and dependency types associated with each $x_i$ (the masked connections and relations are denoted by $(x_i, [MASK])$ and $(x_i, x_j, [MASK])$ in Figure 1, respectively) and MDP aims to recover them during training.

![Figure 2: An illustration of the three second-order (a) and four third-order (b) dependency types between $x_j$ and $x_i$, based on their positions in the parse tree.](image-url)
Specifically, to recover the masked dependency connections and types, we firstly pass the input \( X \) into the base encoder (shown in the red box in Figure 1) that can be initialized in different ways (e.g., by a pre-trained language model) and obtain the hidden vector \( \mathbf{h}_i \) of the word \( x_i \). Then, we use three modules with the same architecture to recover masked dependency connections and types from first, second, and third order dependencies. Taking the first order dependencies as examples, we compute the connection score \( s_{i,j}^{\text{con},1} \) and type scores \( s_{i,j}^{\text{type},1} \) for each pair of \( x_i \) and \( x_j \) by

\[
\begin{align*}
    s_{i,j}^{\text{con},1} &= \mathbf{h}_j^\intercal \cdot \mathbf{W}_1^{\text{con}} \cdot \mathbf{h}_i \\
    s_{i,j}^{\text{type},1} &= \mathbf{W}_1^{\text{type}} \cdot (\mathbf{h}_i \oplus \mathbf{h}_j)
\end{align*}
\]

where \( \oplus \) denotes vector concatenation; \( \mathbf{W}_1^{\text{con}} \) and \( \mathbf{W}_1^{\text{type}} \) are trainable matrices. Herein, \( s_{i,j}^{\text{con},1} \) is a scalar and \( s_{i,j}^{\text{type},1} \) is a vector with the values representing the scores for all possible types between \( x_i \) and \( x_j \). Similarly, we use the same procedure to obtain the connection scores \( s_{i,j}^{\text{con},2} \), \( s_{i,j}^{\text{con},3} \) and the type score vectors \( s_{i,j}^{\text{type},2} \), \( s_{i,j}^{\text{type},3} \) for second and third order dependencies, respectively. Based on the connection and type scores of the first, second, and third order dependencies, our model recovers the masked connection by treating it as a binary classification using sigmoid function and predicts the masked type by applying \text{softmax} to the type score vectors. As a result, dependency information in different orders is implicitly introduced into the base encoder by the gradients back-propagated from the MDP process.

### 2.3 RE with Syntax-induced Encoder

Once the encoder is trained, we extract the obtained syntax-induced encoder and fine-tune it on RE tasks, where the goal of our RE model is to predict the relation \( \hat{y} \in \mathcal{R} \) (\( \mathcal{R} \) is the set for all relation types) between two given entities \( E_1 \) and \( E_2 \) in the input \( X \), which is formally expressed by

\[
\hat{y} = \arg \max_{rel \in \mathcal{R}} s(rel)(X, E_1, E_2) \tag{5}
\]

where \( s(\cdot) \) computes the score \( s^{\text{rel}} \) for a particular relation type \( rel \in \mathcal{R} \) with the given input \( X \) and entities (i.e., \( E_1 \) and \( E_2 \)). In doing so, we firstly fed \( X \) into the pre-trained syntax-induced encoder and obtain the hidden vectors \( \mathbf{h}_i \) for each \( x_i \). Next, we apply the max pooling operation to the hidden vectors of the words in each entity and obtain the vector representations, namely, \( \mathbf{e}_1 \) and \( \mathbf{e}_1 \), of \( E_1 \) and \( E_2 \). Then, we apply bi-affine attentions (Vaswani et al., 2017) to \( \mathbf{e}_1 \) and \( \mathbf{e}_2 \) to compute the score \( s^{\text{rel}} \) for the particular relationship \( rel \). Specifically, bi-affine attentions pass \( \mathbf{e}_1 \) and \( \mathbf{e}_2 \) into two different multi-layer perceptrons (MLP), namely, \( \mathbf{W}_1^{\text{rel}} \) and \( \mathbf{W}_2^{\text{rel}} \), and use a trainable relationship matrix \( \mathbf{W}_1^{\text{rel}} \) to compute \( s^{\text{rel}} \) via

\[
\begin{align*}
    \mathbf{e}_1' &= \mathbf{MLP}_1(\mathbf{e}_1) \tag{6} \\
    \mathbf{e}_2' &= \mathbf{MLP}_2(\mathbf{e}_2) \tag{7} \\
    s^{\text{rel}} &= (\mathbf{e}_1' \oplus [1])^\intercal \cdot \mathbf{W}_1^{\text{rel}} \cdot (\mathbf{e}_2' \oplus [1]) \tag{8}
\end{align*}
\]

where \([1]\) is a one-dimensional unit vector which is the bias term for \( \mathbf{e}_1' \) and \( \mathbf{e}_2' \). Afterwards, we compute the scores \( s^{\text{rel}} \) for all types of relations and predict the one with the highest score.

### 3 Experimental Settings

#### 3.1 Datasets

We use the newest English Wikipedia dump (Wiki) as the raw data to train the syntax-induced encoder through masked dependency prediction (MDP). We filter out sentences whose lengths are fewer than 10 words and obtain the resulting corpus with 92M sentences and 2,380M tokens. In obtaining dependency relations, we use Berkeley Neural Parser\(^5\)

\[^{5}\text{We obtain their models from https://github.com/nikitakit/self-attentive-parser.}\]
Datasets | Sent. # | Token # | Instance # |
---|---|---|---|
ACE05 | Train | 7K | 145K |
| | Dev | 2K | 36K |
| | Test | 2K | 31K |
SemEval | Train | 8K | 141K |
| | Test | 3K | 48K |

Table 1: The statistics of the two English benchmark datasets used in our experiments for relation extraction, where the number of sentence, tokens, and instances (i.e., entity pairs) are reported.

(Kitaev and Klein, 2018) trained on English Penn Treebank (PTB)\(^6\) (Marcus et al., 1993) to automatically parse the Wiki data into constituency trees and then convert them into dependency trees by Stanford Dependency converter\(^7\) (Manning et al., 2014). For relation extraction, we use English ACE2005EN (ACE05)\(^8\) and SemEval 2010 Task 8 (SemEval)\(^9\) (Hendrickx et al., 2010) with the standard train/dev/test splits\(^10\) and follow previous studies (Christopoulou et al., 2018; Ye et al., 2019; Zhang et al., 2017; Soares et al., 2019) to process them. The statistics, namely, the number of sentences and tokens, as well as the number of instances (i.e., entity pairs), of both datasets are reported in Table 1.

### 3.2 Implementation Details

Since a good text representation plays an important role in achieving outstanding performance in many NLP tasks (Song and Shi, 2018; Devlin et al., 2019; Yang et al., 2019; Liu et al., 2019; Lewis et al., 2020; Song et al., 2021; Sun et al., 2021), in the experiments, we use pre-trained language models, i.e., BERT (Devlin et al., 2019) and XLNet (Yang et al., 2019) that have demonstrate their effectiveness in many NLP tasks (Yan et al., 2020; Tian et al., 2020; Ke et al., 2021; Shi et al., 2020; Du et al., 2020; Qin et al., 2021a) as the base encoder for syntax inducing (pre-training) with dependency masking. For both BERT and XLNet, we try their base and large version following the default hyper-parameter settings, where their base version uses 12 layers of self-attentions with 768 dimensional hidden vectors and the large version uses 24 layers of self-attentions with 1024 dimensional hidden vectors for their large version.\(^11\)

For syntax inducing, we train the model on the auto-parsed English Wiki for 700K steps\(^12\) with the batch size set to 32. It is worth noting that, since English Wiki is used as a part of the data to train BERT and XLNet, it could be considered that we do not use additional data in experiments. For the process of fine-tuning the final RE model, we use the obtained syntax-induced encoder with randomly initialized bi-affine attentions. For other hyper-parameters, Table 2 reports the ones tested in training our models for training the relation extraction models. We test all combinations of them for each model and use the one achieving the highest results (i.e., F1 scores) on the development set. For evaluation, we follow previous studies to use the micro-F1 scores for ACE05 and use the official evaluation script\(^13\) for SemEval.

### 4 Results and Analyses

#### 4.1 Overall Results

Table 3 reports the results of our approach on the test set of ACE05 and SemEval with different encoders trained on first, second, and third order of dependencies (e.g., “+ DM (2nd)” denotes our approach with induced first and second order dependencies), as well as their corresponding baselines with only using the initial encoders (e.g., BERT and XLNet). We also run baselines with the standard graph convolutional networks (GCN) and the standard graph attentive networks (GAT) (Veličković et al., 2017) to leverage the auto-parsed dependency trees obtained in the same process as we obtain the auto-parsed Wiki (i.e., parsing and converting).

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-training Step</td>
<td>700K, 1,400K, 2,100K, 2,800K</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>1e-5, 5e-5</td>
</tr>
<tr>
<td>Warmup Rate</td>
<td>0.08, 0.1, 0.2</td>
</tr>
<tr>
<td>Batch Size</td>
<td>16, 32</td>
</tr>
</tbody>
</table>

Table 2: The hyper-parameters tested in tuning our models for relation extraction. The best ones used in our final experiments are highlighted in boldface.

\(^6\)https://catalog.ldc.upenn.edu/LDC99T42.
\(^7\)We use the converter of version 3.3.0 from https://stanfordnlp.github.io/CoreNLP/index.html.
\(^8\)We obtain the official data (LDC2006T06) from https://catalog.ldc.upenn.edu/LDC2006T06.
\(^9\)The data is downloaded from http://docs.google.com/View?docid=dfVxd49s_36c28v9pmw.
\(^10\)There is no official development set for ACE05.
\(^11\)We download the cased version of BERT from https://github.com/google-research/bert and XLNet from https://github.com/zihangdai/xlnet.
\(^12\)Syntax-induced encoder trained for 700K steps on the auto-parsed data achieves the optimal performance in most cases of the experiments (see more analyses in Section 4.4).
\(^13\)We download the evaluation script from http://semeval2.fbk.eu/scorers/task08/SemEval2010_task8_scorer-v1.2.zip.
<table>
<thead>
<tr>
<th>Models</th>
<th>ACE05</th>
<th>SemEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-Base</td>
<td>73.31</td>
<td>88.41</td>
</tr>
<tr>
<td>+ GCN</td>
<td>73.53</td>
<td>88.51</td>
</tr>
<tr>
<td>+ GAT</td>
<td>73.61</td>
<td>88.59</td>
</tr>
<tr>
<td>+ DM (1st)</td>
<td>73.62</td>
<td>88.65</td>
</tr>
<tr>
<td>+ DM (2nd)</td>
<td><strong>73.76</strong></td>
<td><strong>88.60</strong></td>
</tr>
<tr>
<td>+ DM (3rd)</td>
<td>73.65</td>
<td><strong>88.74</strong></td>
</tr>
<tr>
<td>BERT-Large</td>
<td>73.94</td>
<td>89.03</td>
</tr>
<tr>
<td>+ GCN</td>
<td>74.16</td>
<td>89.23</td>
</tr>
<tr>
<td>+ GAT</td>
<td>74.30</td>
<td>89.42</td>
</tr>
<tr>
<td>+ DM (1st)</td>
<td>74.34</td>
<td>89.42</td>
</tr>
<tr>
<td>+ DM (2nd)</td>
<td><strong>74.47</strong></td>
<td><strong>89.65</strong></td>
</tr>
<tr>
<td>+ DM (3rd)</td>
<td>74.29</td>
<td>89.37</td>
</tr>
</tbody>
</table>

(a) BERT-based Models

<table>
<thead>
<tr>
<th>Models</th>
<th>ACE05</th>
<th>SemEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLNet-Base</td>
<td>73.42</td>
<td>88.78</td>
</tr>
<tr>
<td>+ GCN</td>
<td>73.55</td>
<td>88.84</td>
</tr>
<tr>
<td>+ GAT</td>
<td>73.67</td>
<td>88.90</td>
</tr>
<tr>
<td>+ DM (1st)</td>
<td>73.74</td>
<td>88.93</td>
</tr>
<tr>
<td>+ DM (2nd)</td>
<td><strong>73.86</strong></td>
<td><strong>89.11</strong></td>
</tr>
<tr>
<td>+ DM (3rd)</td>
<td>73.68</td>
<td>89.02</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>74.26</td>
<td>89.47</td>
</tr>
<tr>
<td>+ GCN</td>
<td>74.33</td>
<td>89.56</td>
</tr>
<tr>
<td>+ GAT</td>
<td>74.45</td>
<td>89.62</td>
</tr>
<tr>
<td>+ DM (1st)</td>
<td>74.41</td>
<td>89.60</td>
</tr>
<tr>
<td>+ DM (2nd)</td>
<td><strong>74.60</strong></td>
<td><strong>89.90</strong></td>
</tr>
<tr>
<td>+ DM (3rd)</td>
<td>74.51</td>
<td>89.76</td>
</tr>
</tbody>
</table>

(b) XLNet-based Models

Table 3: Experimental results of different models using base and large version of BERT and XLNet on the test set of ACE05 and SemEval. “+ GCN” and “+ GAT” refer to the models with the standard graph convolutional network and standard graph attentive networks, respectively. “+ DM” denotes our approaches with based encoder trained through dependency masking (DM) on word dependencies of different orders (“2nd” means both first and second order dependencies are masked and learnt, the same for “3rd”).

There are several observations. First, our approach works well with different pre-trained language models (i.e., base and large BERT and XLNet), where the models with syntax-induced encoder outperform the vanilla BERT and XLNet baselines on both datasets, even though the baseline models have already achieved desirable performance. Second, compared with baseline models with standard GCN and GAT to leverage auto-parsed dependencies, our approach with different orders of dependency information consistently outperforms those baselines, which further confirms the effectiveness of our approach to leverage auto-parsed dependency information. Third, among models that leveraging dependency information in different orders, the ones with second order dependencies (i.e., ”+ DM (2nd)”) achieve the best performance in most cases. This observation confirms that RE models can benefit from high-order word dependencies since they provide association information among words with longer syntactic relations so as leading to better structure-aware understanding towards a sentence. However, it is still worth noting that, incorporating further higher order word dependencies (e.g., third order) may introduce noise or task-irrelevant information to the encoder since they are provided with auto-generated parses, which results in inferior performance comparing to using the second order dependencies.

### 4.2 Comparison with Previous Studies

We further compare our best performing model with previous studies on the test set of ACE05 and SemEval and report the results in Table 4. It is observed that, our approach outperforms all previous studies with different settings and encoders and achieves state-of-the-art scores on both datasets, which further confirms the effectiveness of our approach. Particularly, compared with previous studies (Zhang et al., 2018; Guo et al., 2019; Mandya et al., 2020; Sun et al., 2020; Yu et al., 2020b) that leverage the auto-parsed dependency tree of the input sentence through a particular module (e.g., Guo et al. (2019) proposed an graph-based approach with attentions to leverage dependency connections), where such dependency trees are required as extra input in inference, our approach uses an encoder to learn the dependency information through DMP and then fine-tune the obtained syntax-induced encoder on RE task. Such design in our approach allows our final RE model to be used without requiring the dependency tree of the sentence as the extra input in inference, which allows our model to run faster than previous approaches.

### 4.3 The Effect of Encoder Initialization

To explore the effect of encoder initialization with our approach, we run experiments by training our encoder starting from Transformer that uses the same architecture as BERT-base (i.e., 12 layers of
multi-head attentions with 768 dimensional hidden vectors) with random initialization (without using parameters from pre-trained language models or word embeddings). Table 5 reports the results of our approach when using different orders of dependency information, as well as the baseline results from the Transformer. As demonstrated, our approach significantly improves the baseline Transformer on both datasets, where around 30% absolute boost is observed on both ACE05 and SemEval datasets. This observation further confirms not only the effectiveness of our approach in improving base encoder with leveraging dependency information, but also its robustness of being applied to a randomly initialized base encoder.

### 4.4 The Effect of Training Steps

To analyze the performance change of the learned syntax-induced encoder on RE along with the increasing of training steps, we investigate the learned encoder (randomly initialized by a vanilla Transformer or pre-trained BERT-base model) with second order dependencies obtained from different training steps by fine-tuning it on ACE05 and SemEval. The test results (i.e., F1 scores) of our approach based on the vanilla Transformer and the BERT-base model with respect to the training steps (in 100 thousands) are illustrated in Figure 4 (a) and (b), respectively, and the performance of BERT-base baseline on different datasets is illustrated in dashed lines in different colors\(^ {14}\) in Figure 4 (b). In addition, we also evaluate the performance of the learned encoders (i.e., vanilla Transformer and the BERT-base model) trained by MDP on the test set of PTB for dependency parsing to illustrate how intensive of dependency information is introduced during the pre-training process\(^ {15}\), where the labeled attachment score (LAS) curves are presented in Figure 4 (c) for reference.

It is shown that, when the Transformer is used, consistent improvements are observed with more training steps for both datasets. When a pre-trained language model (i.e., BERT-base) is used, it is observed that RE benefits much at the beginning of the pre-training (where the noisy auto-parsed dependency information is not intensively learning) and reach the peak (i.e., 74.11% for ACE05 and 89.02% for SemEval) when the training step reaches around 1,000K (where the syntax-induced encoder does not hurt by the noise in the dependencies). This phenomenon confirms the observations in previous studies (Xu et al., 2015; Zhang et al., 2018; Yu et al., 2020b; Sachan et al., 2021) that intensively leverage dependency information may introduce noise and confusion to relation classification, so that effective dependency pruning and introduce is of great importance. It also shows the effectiveness of our approach to address the noise by controlling the intensity of dependency information learning during pre-training.

\(^{14}\)The performance on ACE05 and SemEval are illustrated in green and orange colors, respectively.

\(^{15}\)Herein, the higher the performance of learned encoders on dependency parsing, the more intensive the dependency information is introduced in pre-training.

---

**Table 4:** The comparison of F1 scores between previous studies and our best model with BERT-large on the test sets of ACE05 and SemEval. Previous studies that leverage syntactic information (e.g., the dependency tree of the input sentence) are marked by “†”.

<table>
<thead>
<tr>
<th>Models</th>
<th>ACE05</th>
<th>SemEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socher et al. (2012)</td>
<td>82.4</td>
<td></td>
</tr>
<tr>
<td>Zeng et al. (2014)</td>
<td>82.7</td>
<td></td>
</tr>
<tr>
<td>Zhang and Wang (2015)</td>
<td>79.6</td>
<td></td>
</tr>
<tr>
<td>Xu et al. (2015)</td>
<td>83.7</td>
<td></td>
</tr>
<tr>
<td>Wang et al. (2016)</td>
<td>88.0</td>
<td></td>
</tr>
<tr>
<td>Zhou et al. (2016)</td>
<td>84.0</td>
<td></td>
</tr>
<tr>
<td>†Zhang et al. (2018)</td>
<td>84.8</td>
<td></td>
</tr>
<tr>
<td>Wu and He (2019)</td>
<td>89.2</td>
<td></td>
</tr>
<tr>
<td>Christopoulou et al. (2018)</td>
<td>64.2</td>
<td>-</td>
</tr>
<tr>
<td>Ye et al. (2019)</td>
<td>68.9</td>
<td>-</td>
</tr>
<tr>
<td>†Guo et al. (2019)</td>
<td>85.4</td>
<td></td>
</tr>
<tr>
<td>Baldini Soares et al. (2019)</td>
<td>89.5</td>
<td></td>
</tr>
<tr>
<td>†Mandya et al. (2020)</td>
<td>85.9</td>
<td></td>
</tr>
<tr>
<td>†Sun et al. (2020)</td>
<td>86.0</td>
<td></td>
</tr>
<tr>
<td>†Yu et al. (2020a)</td>
<td>86.4</td>
<td></td>
</tr>
<tr>
<td>Wang et al. (2020)</td>
<td>66.7</td>
<td>-</td>
</tr>
<tr>
<td>Wang and Lu (2020)</td>
<td>67.6</td>
<td>-</td>
</tr>
<tr>
<td>Wang et al. (2021)</td>
<td>66.0</td>
<td>-</td>
</tr>
</tbody>
</table>

| Ours (BERT)               | 74.47 | 89.65  |
| Ours (XLNet)              | 74.60 | 89.90  |

**Table 5:** Comparisons of RE results from vanilla Transformer and our approach that being applied to a randomly initialized Transformer (without pre-trained language models or word embeddings).

<table>
<thead>
<tr>
<th>Models</th>
<th>ACE05</th>
<th>SemEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>31.85</td>
<td>54.62</td>
</tr>
<tr>
<td>+ DM (1st)</td>
<td>66.79</td>
<td>79.37</td>
</tr>
<tr>
<td>+ DM (2nd)</td>
<td>66.67</td>
<td>80.02</td>
</tr>
<tr>
<td>+ DM (3rd)</td>
<td>64.54</td>
<td>79.95</td>
</tr>
</tbody>
</table>

---

1. \(\text{ACE05}\) and \(\text{SemEval}\) are illustrated in green and orange colors, respectively.

2. Herein, the higher the performance of learned encoders on dependency parsing, the more intensive the dependency information is introduced in pre-training.
Figure 4: Curves of fine-tuning different base encoders (Transformer (a) and BERT-base (b)) on ACE05 and SemEval with respect to the number of training steps (in 100K). For reference, (c) shows the dependency parsing performance (LAS) of the learned encoders (Transformer and BERT-base) on the test set of English Penn Treebank (PTB) against its pre-training steps, where higher scores suggest that more intensive introduction of dependency information.

4.5 The Effect of Learned Representations

In previous results and analysis, we already show that the syntax-induced encoder outperforms baselines on RE with implicit integration of dependency information. Therefore, it is interesting to analyze the encoded word representations by qualitatively investigating their relations, which is similar to what has been done for word embeddings. In doing so, we collect word representations from the last layer of the trained syntax-induced encoder (XLNet-large). Then, for each word, we average its representation vectors under different contexts and use the resulting vector as its final representations. Figure 5 visualizes (by t-SNE) the representations of some example words, where the distance between two words indicates their similarity (closer distances indicate more relevant relations). It is observed that words with relevant syntactic properties (e.g., similar form or part-of-speech role) and semantic meanings are grouped into the same cluster (words in different clusters are represented in different colors). For example, all plural nouns of job names, e.g., “teachers”, “journalists”, “publisher”, “librarians”, “shipowners”, and “supporters”, are in the same cluster (represented in red color), while they are far away from irrelevant words, e.g., “praying”. This finding is inspiring since such representations are automatically generated so that the MDP process shows its validity in learning syntax-aware word representations and ensuring that their relevance in syntax and semantics are appropriately modeled, which allows our model to achieve promising performance.

5 Related Work

Relation extraction is an important task in NLP and it requires deep understanding of the input text to achieve model performance. Therefore, in addition to leveraging advanced text encoders (e.g., bi-LSTM, Transformer (Vaswani et al., 2017), BERT (Devlin et al., 2019)) to capture contextual information, structural information, namely, the dependency information, of the running text has been widely used as an effective resource to improve RE (Xu et al., 2015; Zhang et al., 2018; Guo et al., 2019; Yu et al., 2020b; Chen et al., 2021). In most recent studies in NLP, the dependency information is leveraged either as extra input features (Prokopenko and Papageorgiou, 2014; Kiperwasser and Goldberg, 2015; Yu and Bohnet, 2017) or modeled by complicated graph-based architectures, such as convolutions neural networks (Marcheggiani and Titov, 2017; Zhang et al., 2018) and tree LSTMs (Peng et al., 2017; Li et al., 2018). Previous studies also tried to use attention mechanism to weight different dependency features (Guo et al., 2019; Yu et al., 2020b; Qin et al., 2021b) and LSTM to en-
code linearized dependency path (Xu et al., 2015; Roth and Lapata, 2016). In addition to modeling dependency information, there is another track to leverage it by pre-training dependency-based word embeddings through predicting the context words in auto-parsed dependency trees (Levy and Goldberg, 2014; Komninos and Manandhar, 2016) or designing an auxiliary module to learn the dependency information by treating the dependencies as additional input during pre-training (Xu et al., 2021). This research follows the pre-training paradigm and offers an alternative way to do so.

Specifically, compared with existing studies, our approach leverages the dependency information by inducing it to the pre-training process through masked dependency prediction, whose object is to predict the masked dependencies rather than directly using it as extra fixed input along with the input sentence through an additional module. Also, since the dependency information is learnt by the syntax-induced encoder and the encoder is further fine-tuned on the training data in the same way as general RE model, our approach neither requires any additional input features nor needs complicated architectures to encode them, which allows our model to be efficient in inference.

6 Conclusion

In this paper, we propose to use dependency masking and recovering to improve the text encoder and thus enhance RE that requires deep understanding of the running text, where the encoder is trained on large scaled auto-parsed data. Specifically, we try such masking on first, second, and third order word dependencies from the auto-parsed data, and train a base encoder that is able to recover all the masked dependencies. In doing so, the resulted syntax-induced encoder is integrated with dependency information in a dynamic and flexible manner and it can be directly applied to different downstream tasks requiring no extra input or particular design to accommodate dependency information. Experimental results and analyses on two English benchmark datasets (i.e., ACE05 and SemEval) for RE show the effectiveness of our approach, where our approach outperforms strong baselines and achieve state-of-the-art on both datasets.

Acknowledgements

This work is supported by Shenzhen Science and Technology Program under the project “Fundamental Algorithms of Natural Language Understanding for Chinese Medical Text Processing” (JCYJ20210324130208022) and the Natural Science Foundation of Guangdong Province, China, under the project “Deep Learning based Chinese Combination Category Grammar Parsing and its Application in Relation Extraction”. It is also supported by Shenzhen Institute of Artificial Intelligence and Robotics for Society under the project “Automatic Knowledge Enhanced Natural Language Understanding and Its Applications” (AC01202101001).

References


