Fusing Heterogeneous Factors with Triaffine Mechanism for Nested Named Entity Recognition

Anonymous ACL submission

Abstract

Nested entities are observed in many domains due to their compositionality, which cannot be easily recognized by the widely-used sequence labeling framework. A natural solution is to treat the task as a span classification problem. To learn better span representation and increase classification performance, it is crucial to effectively integrate heterogeneous factors including inside tokens, boundaries, labels, and related spans which could be contributing to nested entities recognition. To fuse these heterogeneous factors, we propose a novel triaffine mechanism including triaffine attention and scoring. Triaffine attention uses boundaries and labels as queries, and uses inside tokens and related spans as keys and values for span representations. Triaffine scoring interacts with boundaries and span representations for classification. Experiments show that our proposed method achieves the state-of-the-art $F_1$ scores on four nested NER datasets: ACE2004, ACE2005, GENIA, and KBP2017.

1 Introduction

Named entity recognition (NER) is a fundamental natural language processing task that extracts entities from texts. Flat NER has been well studied and is usually viewed as a sequence labeling problem (Lample et al., 2016). However, nested entities also widely exist in real-world applications due to their multi-granularity semantic meaning (Alex et al., 2007; Yuan et al., 2020), which cannot be solved by the sequence labeling framework since tokens have multiple labels (Finkel and Manning, 2009).

Various paradigms for nested NER have been proposed in recent years. A representative direction is the span-based approach that learns deep representation for every possible span and then classifies it to the corresponding type (Zheng et al., 2019; Xia et al., 2019; Wadden et al., 2019; Tan et al., 2020; Wang et al., 2020; Yu et al., 2020). By leveraging the large-scale pretrained language model, several works show that the simple model structure for span representation and classification can achieve satisfactory results (Luan et al., 2019; Zhong and Chen, 2021). However, we still believe that explicit modeling of some relevant features will further benefit the span representation and classification under the complex nested setting. Taking Figure 1 as an example, we claim that the following factors are critical for recognizing whether a span is an entity. (1) Tokens: It is obvious that tokens of the given span contribute to the recognition. (2) Boundaries: We emphasize boundaries (or boundary tokens) because they are special tokens with rich semantics. Works with simple structure may just produce the span representation based on the concatenation or biaffine transformation of boundary representation (Yu et al., 2020; Fu et al., 2021). Some other works take boundary detection as additional supervision for better representation learning (Zheng et al., 2019; Tan et al., 2020). More importantly, a unilateral boundary cannot determine the entity type since it can exist in multiple entities with different labels (e.g., “NF”, “B”, and “cells”) under the nested setting. (3) Labels: As mentioned above, tokens could belong to entities with different labels. Therefore, we propose that the model should learn label-aware span representation to take into consideration of the different token contributions at the label level. However, we still believe that explicit modeling of some relevant features will further benefit the span representation and classification under the complex nested setting. Taking Figure 1 as an example, we claim that the following factors are critical for recognizing whether a span is an entity. (1) Tokens: It is obvious that tokens of the given span contribute to the recognition. (2) Boundaries: We emphasize boundaries (or boundary tokens) because they are special tokens with rich semantics. Works with simple structure may just produce the span representation based on the concatenation or biaffine transformation of boundary representation (Yu et al., 2020; Fu et al., 2021). Some other works take boundary detection as additional supervision for better representation learning (Zheng et al., 2019; Tan et al., 2020). More importantly, a unilateral boundary cannot determine the entity type since it can exist in multiple entities with different labels (e.g., “NF”, “B”, and “cells”) under the nested setting. (3) Labels: As mentioned above, tokens could belong to entities with different labels. Therefore, we propose that the model should learn label-aware span representation to take into consideration of the different token contributions at the label level. For example, labels are the prediction objects that we cannot touch in representation learning. Here, leveraging label information only means we need label-aware representation learning.

Figure 1: An example sentence with nested entities from the GENIA dataset.
ple, “NF” may contribute more to “protein” type when classifying the span “NF - chi B”, as well as “chi B” and “site” contribute more to “DNA” type when classifying the span “NF - chi B site”.

(4) Related spans: Interactions among spans are important in nested entities (Luo and Zhao, 2020; Wang et al., 2020; Fu et al., 2021). The insider and outsider entities may hint at each other’s types. For example, entities inside “EBV-transformed B cells” have more possibilities to be cell-related entities. Interactions can also help the non-entity span like “transformed B cells” to validate its partialness by looking at outer entity “EBV-transformed B cells”.

Although some of the factors may be explored in previous works, to the best of our knowledge, it is the first work to fuse all these heterogeneous factors into a unified network. As the traditional additive, multiplicative attention, or biaffine transformation cannot interact with such multiple heterogeneous factors simultaneously, we propose a novel triaffine mechanism as the tensor multiplication with three rank-1 tensors (vectors) and a rank-3 tensor, which makes it possible to jointly consider high-order interactions among multiple factors. Specifically, our method follows the pipeline of span representation learning and classification. At the stage of span representation learning, we apply the triaffine attention to aggregate the label-wise span representations by considering boundaries and labels as queries as well as inside tokens as keys and values. Then, a similar triaffine attention is applied to produce the label-wise cross-span representations by querying boundaries and labels with related spans. At the stage of span classification, we fuse the span representations and boundaries for label-wise classification with a triaffine score function. In practice, we add an auxiliary object function to classify spans without the cross-span interaction, which benefits learning robust span representation and can be used as a span filter to speed up both training and inference without performance degradation.

We conduct experiments on four nested NER datasets: ACE2004, ACE2005, GENIA, and KBP2017. Our model achieves 88.56, 88.83, 81.23, and 87.27 scores in terms of F1, respectively, outperforming state-of-the-art methods. Ablation studies show the effectiveness of each factor and the superiority of the triaffine mechanism. We will release our codes and models for further research.

Our contributions are summarized as:

- We propose that heterogeneous factors (i.e., tokens, boundaries, labels, related spans) should be taken into consideration in the span-based methods for nested NER.
- We propose a span-based method with a novel triaffine mechanism including triaffine attention and scoring to fuse the above-mentioned heterogeneous factors for span representations and classification.
- Experiments show that our proposed method performs better than existing span-based methods and achieves state-of-the-arts performances on four nested NER datasets.

2 Related Work

2.1 Nested NER

Nested NER approaches do not have a unified paradigm. Here we mainly focus on span-based methods since they are close to our work.

The span-based methods are one of the most mainstream ways for the nested NER. With the development of pre-training, it is easy to obtain the span representation by the concatenation of boundary representation (Luan et al., 2019; Zhong and Chen, 2021) or the aggregated representation of tokens (Zheng et al., 2019; Wadden et al., 2019), and then follow a linear layer (Xia et al., 2019) or biaffine transformation (Yu et al., 2020) for classification. Several works improve the span-based methods with additional features or supervision. Zheng et al. (2019); Tan et al. (2020) point out the importance of boundaries and therefore introduce the boundary detection task. Wang et al. (2020) propose Pyramid to allow interactions between spans from different layers. Fu et al. (2021) adopt TreeCRF to model interactions between nested spans. Compared with previous methods, our method can jointly fuse multiple heterogeneous factors with the proposed triaffine mechanism.

Other methods for nested NER vary greatly. Earlier research on nested NER is rule-based (Zhang et al., 2004). Lu and Roth (2015); Katiyar and Cardie (2018); Wang and Lu (2018) leverage the hypergraph to represent all possible nested structures, which needs to be carefully designed to avoid spurious structures and structural ambiguities. Wang et al. (2018); Fisher and Vlachos (2019) predict the transition actions to construct nested entities. Lin et al. (2019) propose an anchor-based method to recognize entities. There are other works that recognize entities in a generative fashion (Yan et al., 2020; Fu et al., 2021).
et al., 2021; Shen et al., 2021; Tan et al., 2021). Generally, it is not a unified framework for nested NER, and we model it with a span-based method since it is most straightforward.

2.2 Affine Transformations in NLP

Dozat and Manning (2017) introduce the biaffine transformation in the dependency parsing task for arc classification. Later, it is widely used in many tasks that need to model bilateral representations (Li et al., 2019; Yu et al., 2020). The triaffine transformation is further introduced to extend biaffine transformation for high-order interaction in the field of dependency parsing (Wang et al., 2019; Zhang et al., 2020) and semantic role labeling (Li et al., 2020b). There are two key differences between our triaffine transformation and theirs. Firstly, they only model the homogeneous features such as three tokens, but our triaffine transformation can model heterogeneous factors. Secondly, they usually leverage triaffine transformation to obtain log potentials for CRFs, but we apply it for span representation and classification.

3 Approach

Figure 2 shows an overview of our method. We will first introduce the triaffine transformations, which lie in the heart of our model to fuse heterogeneous factors. Then, we will introduce our model based on the proposed triaffine transformations.

3.1 Deep Triaffine Transformation

We define the deep triaffine transformation with vectors \( \mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^d \) and a tensor \( \mathbf{W} \in \mathbb{R}^{d \times d+1} \times \mathbb{R}^d \times \mathbb{R}^{d+1} \) which outputs a scalar by applying distinct MLP transformations on input vectors and calculating tensor vector multiplications.

\[
\begin{align*}
\mathbf{u}' &= \left[ \text{MLP}(\mathbf{u}) \right]_1, \\
\mathbf{v}' &= \left[ \text{MLP}(\mathbf{v}) \right]_1 \\
\mathbf{w}' &= \text{MLP}(\mathbf{w}) \\
\text{TriAff}(\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{W}) &= \mathbf{W} \times_1 \mathbf{u}' \times_2 \mathbf{w}' \times_3 \mathbf{v}'
\end{align*}
\]

where \( \times_n \) is the mode-\( n \) tensor vector multiplication. A constant 1 is concatenated with inputs to retain the biaffine transformation. The tensor \( \mathbf{W} \) is initialized using \( \mathcal{N}(0, \sigma^2) \). In our approach, we use boundary representations as \( \mathbf{u} \) and \( \mathbf{v} \). Inside tokens or span representations are used as \( \mathbf{w} \). We denote the tensors in the triaffine attention as \( \{ \mathbf{W}_r \} \) and triaffine scoring as \( \{ \lambda_r \} \), which decouples attention weights and scores for different labels.

3.2 Text Encoding

We follow Shen et al. (2021) and Tan et al. (2021) to encode the text. For text \( X = [x_1, x_2, ..., x_N] \) with \( N \) tokens, we first generate the contextual embedding \( \mathbf{x}_i \) with the pre-trained language model,

\[
\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N = \text{PLM}(x_1, x_2, ..., x_N) \tag{4}
\]

Then, we concatenate \( \mathbf{x}_i \) with word embedding \( \mathbf{x}_w^i \), part-of-speech embedding \( \mathbf{x}_p^i \) and character embedding \( \mathbf{x}_c^i \), and feed the concatenated embedding \( \mathbf{x}_i \) into a BiLSTM (Hochreiter and Schmidhuber, 1997) to obtain the token representations \( \{ \mathbf{h}_i \} \).

3.3 Triaffine Attention for Span Representations

To fuse heterogeneous factors for better span representation, we propose a triaffine attention mechanism shown in Figure 3a. To interact tokens with labels and boundaries, we learn the label-wise span representation \( \mathbf{h}_{i,j,r} \) with the triaffine attention \( \alpha_{i,j,k,r} \) for the span \((i, j)\):

\[
s_{i,j,k,r} = \text{TriAff}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_k, \mathbf{W}_r) \tag{5}
\]

\[
\alpha_{i,j,k,r} = \frac{\exp(s_{i,j,k,r})}{\sum_{k'=1}^{R} \exp(s_{i,j,k',r})} \tag{6}
\]

\[
\mathbf{h}_{i,j,r} = \sum_{k=1}^{R} \alpha_{i,j,k,r} \text{MLP}(\mathbf{h}_k) \tag{7}
\]

Boundary representations \( \{ \mathbf{h}_i, \mathbf{h}_j \} \) and the label-wise parameters \( \{ \mathbf{W}_r \} \) can be viewed as attention queries, and tokens \( \{ \mathbf{h}_k \} \) can be viewed as keys and values. Compared with the general attention framework (additive or multiplicative attention), our triaffine attention permits all high-order interactions between heterogeneous queries and keys.

3.4 Triaffine Attention for Cross-span Representations

Motivated by the span-level interactions in the nested setting, we fuse related spans information into cross-span representations. We view the boundaries of the span and labels as attention queries, related spans (containing the span itself) as attention keys and values to obtain cross-span representations. Similar to the Equation 7, we obtain label-wise cross-span representations \( \mathbf{h}_{i,j,r} \) for
3.5 Triaffine Scoring for Span Classification

To classify the entity type of the span, we calculate label-wise scores based on cross-span representations. Since boundary information has been proved effective in previous works (Yu et al., 2020; Fu et al., 2021), we leverage the boundaries information and cross-span representations for span classification via triaffine scoring. Specifically, we estimate the log probabilities $p_{i,j,r}$ of the span $(i, j)$ for label $r$ using boundaries $h_i, h_j$ and cross-span representations $h_{i,j,r}$.

$$p_{i,j,r} = \text{TriAff}(h_i, h_j, h_{i,j,r}, \mathcal{V}_r)$$ (11)

where $\{(i_g, j_g)\}$ are the related spans. One can treat all enumerated spans as related spans, and we will introduce how we select them in Section 3.6.

3.6 Training and Inference

In practice, it is expensive and non-informative to consider interactions between all spans. Therefore, we propose an auxiliary task to classify spans with intermediate span representations. Then, we can rank all spans based on the maximum of log probabilities (except None) from the intermediate predictions $p_{i,j} = \max_{r=1}^{R} p_{i,j,r}$, and retain top-$m$ spans $\{(i_t, j_t)\}_{t=1}^m$ as candidates. We calculate cross-span representations $h_{i,j,r}$ for re-

Figure 3 and 3c show the mechanism of triaffine scoring and the decomposition. We also apply the similar decomposition functions in the auxiliary span classification task, which applies the triaffine scoring on boundary representations and intermediate span representations $h_{i,j,r}$ to estimate log probabilities $p_{i,j,r}$ as intermediate predictions.
trained spans by considering the full interactions among them, and estimate the classification logits 
\( p_{i,j,r}^{c} \). Thus, we have two groups of predictions in our model \( \{p_{i,j,r}\}_{1 \leq i,j \leq N} \) and \( \{p_{i,j,r}^{c}\}_{1 \leq i,j \leq m} \). \( \{p_{i,j,r}\} \) are calculated for every possible span, and \( \{p_{i,j,r}^{c}\} \) are calculated only on top-\( m \) spans.

In the training phase, we jointly minimize two groups of cross-entropy losses:

\[
\mathcal{L}_{aux} = -\frac{2}{N(N+1)} \sum_{i,j} \log \frac{\exp(p_{i,j,r_{ij}})}{\sum_{r} \exp(p_{i,j,r})}
\]

(14)

\[
\mathcal{L}_{main} = -\frac{1}{m} \sum_{l=1}^{N} \log \frac{\exp(p_{l,j,r_{l,j}}^{c})}{\sum_{r} \exp(p_{l,j,r}^{c})}
\]

(15)

\[
\mathcal{L} = \mu_{aux} \mathcal{L}_{aux} + \mathcal{L}_{main}
\]

(16)

where \( r_{ij} \) is the label of span \( (i,j) \).

In both the training and inference phase, \( \{p_{i,j,r}\} \) are used to select spans with high possibilities based on the supervision from \( \mathcal{L}_{aux} \). We inference the labels of selected spans using \( \{p_{i,j,r}^{c}\} \) by assigning label \( \tilde{r}_{i,j} = \arg\max_{r} p_{i,j,r}^{c} \), and we assign None class for others.

4 Experiments

4.1 Datasets

We conduct our experiments on the ACE2004\(^2\), ACE2005\(^3\) (Doddington et al., 2004), GENIA (Kim et al., 2003) and KBP2017\(^7\) (Ji et al., 2017) datasets. To fairly compare with previous works, we follow the same dataset split with Lu and Roth (2015) for ACE2004 and ACE2005 datasets and use the split from Lin et al. (2019) for GENIA and KBP2017 datasets. The statistics of all datasets are listed in Table 1. Following previous work, we measure the results using span-level precision, recall, and \( F_1 \) scores.

4.2 Implementation Details

We use BERT-large-cased (Devlin et al., 2019) and albert-xxlarge-v2 (Lan et al., 2020) as the contextual embedding, fastText (Bojanowski et al., 2017) as the word embedding in ACE2004, ACE2005 and KBP2017 dataset. We use BioBERT-v1.1 (Lee et al., 2020) and

\( \text{BioWordVec} \) (Zhang et al., 2019) as the contextual and word embedding in the GENIA dataset respectively. We truncate the input texts with context at length 192. The part-of-speech embeddings are initialized with dimension 50. The char embeddings are generated by a one-layer BiLSTM with hidden size 50. The two-layers BiLSTM with a hidden size of 1,024 is used for the token representations. For triaffine transformations, we use \( d = 256 \) for the ACE2004, ACE2005, and KBP2017 dataset, and \( d = 320 \) for the GENIA dataset, respectively. We set \( \mu_{aux} \) to 1.0, and select \( m = 30 \) in both training and inference. We use AdamW (Loshchilov and Hutter, 2019) to optimize our models with a linear learning rate decay. Detailed training parameters are presented in Appendix A.

4.3 Baselines

\( \text{DYGIE} \) (Luan et al., 2019) uses multi-task learning to extract entities, relations, and coreferences. \( \text{MGNER} \) (Xia et al., 2019) uses a detector to find span candidates and a classifier for categorization. \( \text{BENSC} \) (Tan et al., 2020) trains the boundary detection and span classification tasks jointly. \( \text{TreeCRF} \) (Fu et al., 2021) views entities as nodes in a constituency tree and decodes them with a Masked Inside algorithm. \( \text{Biaffine} \) (Yu et al., 2020) classifies spans by a bi-affine function between boundary representations. \( \text{Pyramid} \) (Wang et al., 2020) designs pyramid layer and inverse pyramid layer to decode nested entities.

We also report the results of models with other paradigms, including hypergraph-based methods (Wang and Lu, 2018), transition-based methods (Fisher and Vlachos, 2019), generative methods (Yan et al., 2021; Tan et al., 2021; Shen et al., 2021), and so on. We do not compare to BERT-MRC (Li et al., 2020a) since they use additional resources as queries. DYGIE++ (Wadden et al., 2019) and PURE (Zhong and Chen, 2021) use different splits of the ACE datasets which are not comparable.

4.4 Results

We compare our method with baseline methods in Table 2 for the ACE2004, ACE2005, and GENIA datasets and Table 3 for the KBP2017 dataset, respectively. With BERT as the encoder, our model achieves 87.40, 86.82, 81.23, and 85.05 scores in terms of \( F_1 \), outperforming all other span-based methods such as BENSC, Pyramid, TreeCRF, and Biaffine (+0.70 on ACE2004, +1.42 on ACE2005, +0.73 on GENIA). Compared with methods in other
Table 1: Statistics of nested NER datasets ACE2004, ACE2005, GENIA, and KBP2017.

| Model + Encoder | ACE2004 | ACE2005 | GENIA | KBP2017 |
|-----------------|---------|---------|-------|---------|
|                 | Train   | Dev     | Test  | Train   | Dev     | Test  | Train   | Dev     | Test  |
| # sentences     | 6,200   | 745     | 812   | 7,194   | 969     | 1,047 | 16,692  | 1,854   | 10,546|
| # entities      | 22,204  | 2,514   | 3,035 | 24,411  | 3,200   | 2,993 | 50,509  | 5,506   | 31,236|
| # nested entities | 10,149  | 1,092   | 1,417 | 9,389   | 1,112   | 1,118 | 9,064   | 1,199   | 8,773 |
| max entity count | 28      | 22      | 20    | 27      | 23      | 17    | 25      | 14      | 58    |

Table 2: Results on the ACE2004, ACE2005, and GENIA datasets. BERT is the default encoder if not specified.

Table 3: Results on the KBP2017 dataset. BERT is the default encoder if not specified.
Table 4: Ablation tests on ACE2004 and GENIA datasets. Cross means using cross attention for span classification. Lin. means linear transformation, bi. means biaffine transformation, and tri. means triaffine transformation.

| Setting | Span Representation | Span Classification | Datasets |
|---------|---------------------|---------------------|----------|
|         | Label | Boundary | Function | Boundary | Attention | Cross | Function | ACE2004 | GENIA |
| (a)     | ×     | ×       | ×        | √       | ×         | ×     | bi.      | 86.71   | 78.97 |
| (b)     | √     | √       | tri.     | ×        | √         | ×     | lin.     | 87.36   | 80.50 |
| (c)     | ×     | √       | tri.     | √        | √         | ×     | tri.     | 87.17   | 80.49 |
| (d)     | √     | ×       | lin.     | √        | √         | ×     | tri.     | 87.14   | 80.50 |
| (e)     | √     | √       | lin.     | √        | ×         | ×     | tri.     | 87.35   | 80.63 |
| (f)     | √     | √       | tri.     | √        | ×         | ×     | tri.     | 87.49   | 80.70 |
| (g)     | √     | √       | tri.     | √        | √         | ×     | tri.     | 87.54   | 80.84 |
| (h)     | √     | √       | tri.     | √        | √         | √     | tri.     | 87.82   | 81.23 |

(d) To show the effectiveness of boundaries in representation, we remove boundaries factor in attention:
\[ s_{i,j,k,r} = s_{k,r} = q_r \cdot h_k \quad (20) \]

(e) To show the effectiveness of the triaffine mechanism in representations, we replace triaffine attention with linear attention:
\[ s_{i,j,k,r} = W_r (h_i \parallel h_j \parallel h_k) + c_r \quad (21) \]

(f) To show the effectiveness of triaffine scoring, we replace triaffine scoring to linear scoring:
\[ p_{i,j,r} = V_r (h_i \parallel h_j \parallel h_{i,j,r}) + b_r \quad (22) \]

(g) To show the effectiveness of cross-span interactions, we use our partial model with intermediate predictions (model (a)-(g) use \( p_{i,j,r} \) as predictions).

(h) Our full model (i.e, use \( p_{i,j,r}^{c} \) as predictions).

Table 4 shows the results of ablation studies on ACE2004 and GENIA datasets. We use BERT-large-cased as the backbone encoder on ACE2004 and BioBERT-v1.1 on GENIA, respectively. By comparing (a) with (g), we observe significant performances drop (-0.87 on ACE2004, -1.87 on GENIA), which indicates that our proposed triaffine mechanism with multiple heterogeneous factors performs better than the biaffine baseline. Comparing (b) with (g), we find that the boundary information contributes to span classification. Comparing (c) and (d) with (g) supports that either label or boundary in the triaffine attention improves the performance. The setting (g) performs better than (e) and (f), which shows the superiority of the triaffine transformation over the linear function. We observe that (h) performs better than (g) (+0.28 on ACE2004, +0.39 on GENIA), proving the strength of triaffine attention with interactions among related spans. The above studies support that our proposed triaffine mechanism with associated heterogeneous factors is effective for span representation and classification.

4.6 Discussion

We compare the \( F_1 \) scores of GENIA between triaffine model (g) and biaffine model (a) grouped by entity lengths in Figure 4. In all columns, the \( F_1 \) score of our method is better than the baseline. Furthermore, the right columns show that the \( F_1 \) score of the baseline gradually decreases with the incremental entity lengths. However, our method based on the triaffine mechanism with heterogeneous factors takes advantage of the interaction from boundaries and related spans, which keeps consistent results and outperforms the baseline.

The results grouped by flat or nested entities are shown in Table 6. Our method has consistent improvements than the baseline, especially for the nested setting. Based on the above observations, our method is good at solving long entities that are more likely to be nested, which supports our model is built upon the characteristics of nested NER.

At the stage of cross-span interactions, we only select top-\( m \) spans in practice. In Figure 5, we analyze the number \( m \) in two aspects. Firstly, we check
Table 5: Case study on ACE2004 and GENIA dataset. Colored brackets indicate the boundaries and semantic types of entities in true labels. “cl” and “ct” is the abbreviation of cell line and cell type, respectively.

Table 6: Comparison between triaffine and biaffine models on ACE2004 and GENIA grouped by flat or nested entities. Entity counts are in the parentheses.

Figure 5: Recall for entity spans and $F_1$ scores with different numbers of candidate spans in GENIA dataset.

5 Conclusion

In this paper, we propose a span-based method for nested NER. Heterogeneous factors including tokens, boundaries, labels, and related spans are introduced to improve span classification with a novel triaffine mechanism. Experiments show our method outperforms all span-based methods and achieves state-of-the-art performance on four nested NER datasets. Ablation studies show the introduced heterogeneous factors and triaffine mechanism are helpful for nested setting. Despite that large-scale pretrained language models have shown consistent improvement over many NLP tasks, we argue that the well-designed features and model structures are still useful for complex tasks like nested NER. Furthermore, although we only verify our triaffine mechanism in nested NER, we believe it can also be useful in tasks requiring high order interactions like parsing and semantic role labeling.
Bailin Wang, Wei Lu, Yu Wang, and Hongxia Jin. 2018. A neural transition-based model for nested mention recognition. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 1011–1017, Brussels, Belgium. Association for Computational Linguistics.

Jue Wang, Lidan Shou, Ke Chen, and Gang Chen. 2020. Pyramid: A layered model for nested named entity recognition. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 5918–5928.

Xinyu Wang, Jingxian Huang, and Kewei Tu. 2019. Second-order semantic dependency parsing with end-to-end neural networks. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4609–4618, Florence, Italy. Association for Computational Linguistics.

Congying Xia, Chenwei Zhang, Tao Yang, Yaliang Li, Nan Du, Xian Wu, Wei Fan, Fenglong Ma, and Philip Yu. 2019. Multi-grained named entity recognition. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 1430–1440, Florence, Italy. Association for Computational Linguistics.

Hang Yan, Tao Gui, Junqi Dai, Qipeng Guo, Zheng Zhang, and Xipeng Qiu. 2021. A unified generative framework for various NER subtasks. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 5808–5822, Online. Association for Computational Linguistics.

Juntao Yu, Bernd Bohnet, and Massimo Poesio. 2020. Named entity recognition as dependency parsing. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 6470–6476, Online. Association for Computational Linguistics.

Zheng Yuan, Yuanhao Liu, Qiyang Yin, Boyao Li, Xiaobin Feng, Guoming Zhang, and Sheng Yu. 2020. Unsupervised multi-granular chinese word segmentation and term discovery via graph partition. Journal of Biomedical Informatics, 110:103542.

Jie Zhang, Dan Shen, Guodong Zhou, Jian Su, and Chew-Lim Tan. 2004. Enhancing hmm-based biomedical named entity recognition by studying special phenomena. Journal of biomedical informatics, 37(6):411–422.

Yijia Zhang, Qingyu Chen, Zhihao Yang, Hongfei Lin, and Zhiyong Lu. 2019. Biowordvec, improving biomedical word embeddings with subword information and mesh. Scientific data, 6(1):1–9.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020. Efficient second-order TreeCRF for neural dependency parsing. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics,
A Reproducibility Checklist

We specific seeds of torch, torch.cuda, numpy, and random in Python to ensure reproducibility. We use a grid search to find the best hyperparameters depending on development set performances. We search contextual embedding learning rate among \{1e-5,3e-5\}. If the contextual embedding learning rate is 1e-5, we use static embedding learning rate and task learning rate as 1e-4 and 1e-5. If the contextual embedding learning rate is 3e-5, we use static embedding learning rate and task learning rate as 5e-4 and 3e-5. We search batch size among \{8,48,72\}. We search MLP dropout ratio among \{0.1,0.2\}. The final hyperparameters we used for four datasets are listed in Table 7 and Table 8.

B The Decomposition of Triaffine Scoring

We introduce the decomposition of triaffine scoring in calculating $p_{i,j,r}$ and $p'_{i,j,r}$.
| Method       | Function | CPU Time | Percentage | GPU Time | Percentage |
|--------------|----------|----------|------------|----------|------------|
| Equation 26  | aten::copy_ | 0.5ms    | 5.9%       | 223.7ms  | 74.5%      |
|              | aten::bmm  | 0.5ms    | 5.0%       | 38.2ms   | 12.7%      |
|              | aten::mm   | 1.5ms    | 15.7%      | 37.1ms   | 12.3%      |
|              | Total      | 9.2ms    | 100.0%     | 300.5ms  | 100.0%     |
| Equation 27  | aten::copy_ | 0.2ms    | 4.7%       | 62.5ms   | 42.9%      |
|              | aten::bmm  | 0.4ms    | 10.0%      | 47.4ms   | 32.6%      |
|              | aten::mm   | 0.3ms    | 6.0%       | 34.4ms   | 23.7%      |
|              | Total      | 4.4ms    | 100.0%     | 145.6ms  | 100.0%     |
| Naive        | aten::copy_ | 7.3ms    | 5.7%       | 302.3ms  | 59.3%      |
|              | aten::bmm  | 1.2ms    | 0.9%       | 109.3ms  | 21.5%      |
|              | aten::mm   | 1.7ms    | 1.4%       | 74.4ms   | 14.6%      |
|              | aten::einsum | 61.8ms  | 48.0%      | 1.1ms    | 0.2%       |
|              | aten::permute | 36.7ms | 28.5%     | 0.8ms    | 0.2%       |
|              | aten::reshape | 1.3ms  | 3.1%       | 0.5ms    | 0.1%       |
|              | Total      | 128.7ms  | 100.0%     | 509.4ms  | 100.0%     |
| Decompose    | aten::copy_ | 0.7ms    | 0.8%       | 136.7ms  | 41.4%      |
|              | aten::bmm  | 1.2ms    | 1.2%       | 102.6ms  | 31.0%      |
|              | aten::mm   | 5.4ms    | 5.3%       | 69.0ms   | 20.9%      |
|              | aten::einsum | 32.0ms  | 31.3%      | 1.1ms    | 0.3%       |
|              | aten::permute | 15.4ms | 15.1%     | 0.7ms    | 0.2%       |
|              | aten::reshape | 37.4ms | 36.6%     | 0.5ms    | 0.2%       |
|              | Total      | 102.2ms  | 100.0%     | 330.5ms  | 100.0%     |

Table 9: Time usage compared with naive triaffine scoring and decomposed triaffine scoring.

uses 309.7ms (300.5ms in GPU + 9.2ms in CPU) and Equation 27 uses 150.1ms (145.6ms in GPU + 4.4ms in CPU). The larger tensor size and higher rank of $[h_{i,j,r}]$ results in slower calculations of aten::bmm, aten::copy_ and aten::permute in Equation 26. The time usage differences are clearly dominated by the function aten::copy_, which is optimized by our decomposition.

We also compare the time usage between the naive triaffine scoring and the decomposed triaffine scoring in Table 9. The naive triaffine scoring takes 638.1ms (509.4ms in GPU + 128.7ms in CPU), and the decomposed triaffine scoring takes 432.7ms (330.5ms in GPU + 102.2ms in CPU) for 10 iterations, which leads to approximately 32% speedup. The GPU time usages are reasonable since they both need to calculate two triaffine transformations. The CPU time usages increase for both naive and decomposition triaffine scoring. Additional CPU time usages come from function aten::einsum, aten::permute, and aten::reshape, and the naive calculation increases more due to slower aten::einsum. Overall, the decomposition triaffine scoring uses less time on both GPU and CPU than the naive triaffine scoring.

Furthermore, we also test the time usage of $p_{i,j,r}^{c}$ using two calculation procedures. We find using the decomposition triaffine scoring still has about 6% speed up (naive:125.8ms in GPU + 15.0ms in CPU vs. decomposition:115.5ms in GPU + 16.8ms) regardless the relatively small size of $[h_{i,j,r}^{c}]$ (The shape of tensor $[h_{i,j,r}^{c}]$ is $B \times m \times R \times d$).