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.[1] For exam-

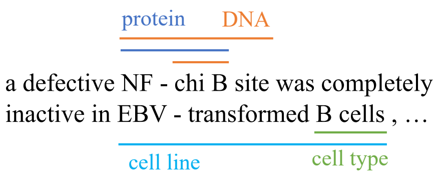
protein
DNA
a defective NF - chi B site was completely
inactive in EBV - transformed B cells , …
cell line
cell type

Figure 1: An example sentence with nested entities from the GENIA dataset.

[1]Label is the perdition object that we cannot touch in representation learning. Here, leveraging label information only means we need label-aware representation learning.

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 F_1, 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., 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 $\mathcal{W} \in \mathbb{R}^{d+1} \times \mathbb{R}^d \times \mathbb{R}^{d+1}$ which outputs a scalar by applying distinct MLP transformations on input vectors and calculating tensor vector multiplications.

$$\mathbf{u}' = \begin{bmatrix} \mathrm{MLP}(\mathbf{u}) \\ 1 \end{bmatrix}, \mathbf{v}' = \begin{bmatrix} \mathrm{MLP}(\mathbf{v}) \\ 1 \end{bmatrix} \quad (1)$$

$$\mathbf{w}' = \mathrm{MLP}(\mathbf{w}) \quad (2)$$

$$\mathrm{TriAff}(\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathcal{W}) = \mathcal{W} \times_1 \mathbf{u}' \times_2 \mathbf{w}' \times_3 \mathbf{v}' \quad (3)$$

where $\times_n$ is the mode-n tensor vector multiplication. A constant 1 is concatenated with inputs to retain the biaffine transformation. The tensor $\mathcal{W}$ 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 $\{\mathcal{W}_r\}$ and triaffine scoring as $\{\mathcal{V}_r\}$, 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 $\mathbf{x}_i^c$ with the pre-trained language model,

$$\mathbf{x}_1^c, \mathbf{x}_2^c, ..., \mathbf{x}_N^c = \mathrm{PLM}(x_1, x_2, ..., x_N) \quad (4)$$

Then, we concatenate $\mathbf{x}_i^c$ with word embedding $\mathbf{x}_i^w$, part-of-speech embedding $\mathbf{x}_i^p$ and character embedding $\mathbf{x}_i^{ch}$, 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):

$$s_{i,j,k,r} = \mathrm{TriAff}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_k, \mathcal{W}_r) \quad (5)$$

$$\alpha_{i,j,k,r} = \frac{\exp(s_{i,j,k,r})}{\sum_{k'=i}^{j} \exp(s_{i,j,k',r})} \quad (6)$$

$$\mathbf{h}_{i,j,r} = \sum_{k=i}^{j} \alpha_{i,j,k,r} \mathrm{MLP}(\mathbf{h}_k) \quad (7)$$

Boundary representations ($\mathbf{h}_i$, $\mathbf{h}_j$) and the label-wise parameters ($\mathcal{W}_r$) 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 $\mathbf{h}_{i,j,r}^c$ for

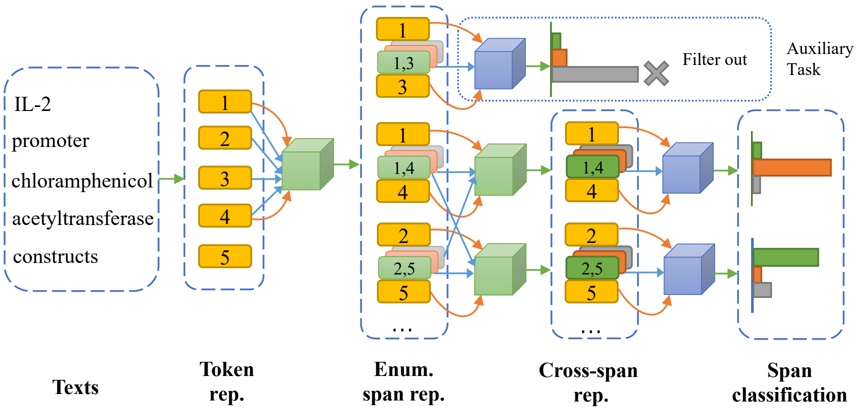

Figure 2: The architecture of our method. Green cubes indicate triaffine attention. Blue cubes indicate triaffine scoring. Orange arrows mean boundary information. Blue arrows mean inside tokens or related spans information. For each span, we have head and tail representations in yellow and label-wise span representations in different colors. The grey color indicates None class.

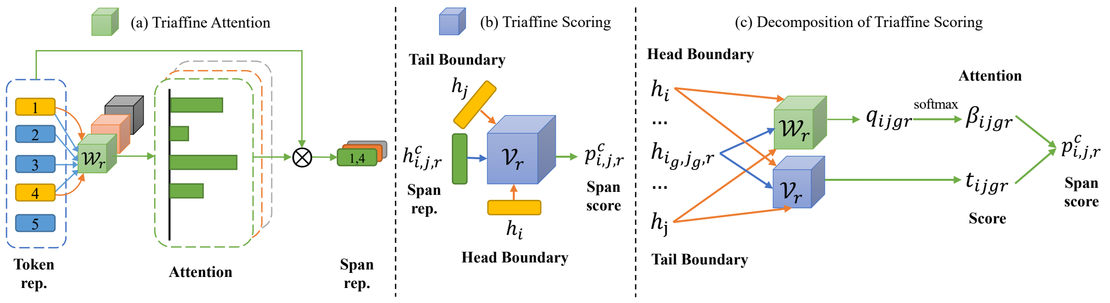

Figure 3: Visualization of triaffine attention, triaffine scoring, and the decomposition of triaffine scoring.

the span (i, j) based on triaffine attention $\beta_{i,j,g,r}$.

$$q_{i,j,g,r} = \text{TriAff}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{i_g,j_g,r}, \mathcal{W}_r) \quad (8)$$

$$\beta_{i,j,g,r} = \frac{\exp(q_{i,j,g,r})}{\sum_{g'} \exp(q_{i,j,g',r})} \quad (9)$$

$$\mathbf{h}^c_{i,j,r} = \sum_g \beta_{i,j,g,r} \text{MLP}(\mathbf{h}_{i_g,j_g,r}) \quad (10)$$

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.

3.5 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 $p^c_{i,j,r}$ of the span (i, j) for label r using boundaries $\mathbf{h}_i, \mathbf{h}_j$ and cross-span representations $\mathbf{h}^c_{i,j,r}$.

$$p^c_{i,j,r} = \text{TriAff}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}^c_{i,j,r}, \mathcal{V}_r) \quad (11)$$

Since $\mathbf{h}^c_{i,j,r}$ are composed by $\mathbf{h}_{i_g,j_g,r}$, we can decompose Equation 11 into:

$$t_{i,j,g,r} = \text{TriAff}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{i_g,j_g,r}, \mathcal{V}_r) \quad (12)$$

$$p^c_{i,j,r} = \sum_g \beta_{i,j,g,r} t_{i,j,g,r} \quad (13)$$

Figure 3b 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 $\mathbf{h}_{i,j,r}$ to estimate log probabilities $p_{i,j,r}$ as intermediate predictions.

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 $\{(i_l, j_l)\}_{l=1}^m$ as candidates. We calculate cross-span representations $\mathbf{h}^c_{i_l,j_l,r}$ for retained spans by considering the full interactions among them, and estimate the classification logits $p^c_{i_l,j_l,r}$. Thus, we have two groups of predictions in our model $\{p_{i,j,r}\}_{1\leq i\leq j\leq N}$ and $\{p^c_{i_l,j_l,r}\}_{1\leq l\leq m}$. $\{p_{i,j,r}\}$ are calculated for every possible span, and $\{p^c_{i_l,j_l,r}\}$ are calculated only on top-m spans.

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

$$\mathcal{L}_{aux} = -\frac{2}{N(N+1)}\sum_{i,j}\log\frac{\exp(p_{i,j,r_{ij}})}{\sum_r \exp(p_{i,j,r})} \tag{14}$$

$$\mathcal{L}_{main} = -\frac{1}{m}\sum_{1\leq l\leq m}\log\frac{\exp(p^c_{i_l,j_l,r_{i_l,j_l}})}{\sum_r \exp(p^c_{i_l,j_l,r})} \tag{15}$$

$$\mathcal{L} = \mu_{aux}\mathcal{L}_{aux} + \mathcal{L}_{main} \tag{16}$$

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 $\{p^c_{i_l,j_l,r}\}$ by assigning label $\tilde{r}_{i_l,j_l} = \arg_r \max p^c_{i_l,j_l,r}$, 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[4] (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 `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 μ_{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

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

[2] https://catalog.ldc.upenn.edu/LDC2005T09
[3] https://catalog.ldc.upenn.edu/LDC2006T06
[4] https://catalog.ldc.upenn.edu/LDC2019T12

	ACE2004			ACE2005			GENIA		KBP2017		
	Train	Dev	Test	Train	Dev	Test	Train	Test	Train	Dev	Test
# sentences	6,200	745	812	7,194	969	1,047	16,692	1,854	10,546	545	4,267
# entities	22,204	2,514	3,035	24,411	3,200	2,993	50,509	5,506	31,236	1,879	12,601
# nested entities	10,149	1,092	1,417	9,389	1,112	1,118	9,064	1,199	8,773	605	3,707
max entity count	28	22	20	27	23	17	25	14	58	15	21

Table 1: Statistics of nested NER datasets ACE2004, ACE2005, GENIA, and KBP2017.

Model + Encoder	ACE2004			ACE2005			GENIA		
	P	R	F_1	P	R	F_1	P	R	F_1
Span-based Methods									
DYGIE (Luan et al., 2019) + LSTM	-	-	84.7	-	-	82.9	-	-	76.2
MGNER (Xia et al., 2019) + ELMo	81.7	77.4	79.5	79.0	77.3	78.2	-	-	-
BENSC (Tan et al., 2020)	85.8	84.8	85.3	83.8	83.9	83.9	79.2	77.4	78.3
TreeCRF (Fu et al., 2021)	86.7	86.5	86.6	84.5	86.4	85.4	78.2	78.2	78.2
Biaffine (Yu et al., 2020)	87.3	86.0	86.7	85.2	85.6	85.4	81.8	79.3	80.5
Pyramid (Wang et al., 2020)	86.08	86.48	86.28	83.95	85.39	84.66	79.45	78.94	79.19
Pyramid (Wang et al., 2020) + ALBERT	87.71	87.78	87.74	85.30	87.40	86.34	80.33	78.31	79.31
Others									
SH (Wang and Lu, 2018) + LSTM	78.0	72.4	75.1	76.8	72.3	74.5	77.0	73.3	75.1
ARN (Lin et al., 2019) + LSTM	76.2	73.6	74.9	75.8	73.9	74.8	-	-	-
BiFlag (Luo and Zhao, 2020) + LSTM	-	-	-	75.0	75.2	75.1	77.4	74.6	76.0
Merge Label (Fisher and Vlachos, 2019)	-	-	-	82.7	82.1	82.4	-	-	-
Seq2seq (Straková et al., 2019)	-	-	84.40	-	-	84.33	-	-	78.31
Second-best (Shibuya and Hovy, 2020)	85.94	85.69	85.82	83.83	84.87	84.34	77.81	76.94	77.36
BartNER (Yan et al., 2021) + BART	87.27	86.41	86.84	83.16	86.38	84.74	78.87	79.60	79.23
Sequence to Set (Tan et al., 2021)	88.46	86.10	87.26	87.48	86.63	87.05	82.31	78.66	80.44
Locate and Label (Shen et al., 2021)	87.44	87.38	87.41	86.09	87.27	86.67	80.19	80.89	80.54
Triaffine (Ours)	87.13	87.68	87.40	86.70	86.94	86.82	80.42	82.06	**81.23**
Triaffine (Ours) + ALBERT	88.88	88.24	**88.56**	87.39	90.31	**88.83**	-	-	-

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

Model + Encoder	KBP2017		
	P	R	F_1
ARN + LSTM	77.7	71.8	74.6
BiFlag + LSTM	77.1	74.3	75.6
Sequence to Set	84.91	83.04	83.96
Locate and Label	85.46	82.67	84.05
Triaffine (Ours)	86.50	83.65	85.05
Triaffine (Ours) + ALBERT	**89.42**	**85.22**	**87.27**

Table 3: Results on the KBP2017 dataset. BERT is the default encoder if not specified.

paradigms, our model also achieves the state-of-the-art results on the GENIA (+0.69 vs. Locate and Label) and KBP2017 dataset (+1.00 vs. Locate and Label) and shows comparable performances on ACE2004 (-0.01 vs. Locate and Label) and ACE2005 (-0.23 vs. Sequence to Set). With a stronger encoder ALBERT, our model achieves 88.56, 88.83, and 87.27 scores in terms of F_1 on ACE2004, ACE2005, and KBP2017 respectively, which exceeds all existing baselines including the Pyramid model with ALBERT (+0.82 on ACE2004, +2.49 on ACE2005) and the previous state-of-the-art method on KBP2017 dataset (+3.22 vs. Locate and Label).

4.5 Ablation Study

Considering we leverage multiple factors in multiple parts of the model, we design the following ablation settings to validate the effectiveness of each factor and the proposed triaffine mechanism. (a) To show the effectiveness of triaffine mechanism, we use a baseline biaffine model with the combination of boundary representations:

$$p_{i,j,r} = \begin{bmatrix} \mathbf{h}_i \\ 1 \end{bmatrix}^T \mathbf{V}_r \begin{bmatrix} \mathbf{h}_j \\ 1 \end{bmatrix} \tag{17}$$

(b) To show the effectiveness of boundaries in scoring, we remove boundaries factor from scoring:

$$p_{i,j,r} = \mathbf{V}_r \mathbf{h}_{i,j,r} + \mathbf{b}_r \tag{18}$$

(c) To show the effectiveness of labels in representation, we remove label factor in attention:

$$s_{i,j,k,r} = \mathrm{TriAff}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_k, \mathcal{W}) \tag{19}$$

	Setting							Datasets	
	Span Representation			Span Classification				ACE2004	GENIA
Setting	Label	Boundary	Function	Boundary	Attention	Cross	Function	F_1	
(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.	√	√	×	lin.	87.49	80.70
(g)	√	√	tri.	√	√	×	tri.	87.54	80.84
(h)	√	√	tri.	√	√	√	tri.	**87.82**	**81.23**

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.

(d) To show the effectiveness of boundaries in representation, we remove boundaries factor in attention:

$$s_{i,j,k,r} = s_{k,r} = \mathbf{q}_r \cdot \mathbf{h}_k \tag{20}$$

(e) To show the effectiveness of the triaffine mechanism in representations, we replace triaffine attention with linear attention:

$$s_{i,j,k,r} = \mathbf{W}_r(\mathbf{h}_i \parallel \mathbf{h}_j \parallel \mathbf{h}_k) + \mathbf{c}_r \tag{21}$$

(f) To show the effectiveness of triaffine scoring, we replace triaffine scoring to linear scoring:

$$p_{i,j,r} = \mathbf{V}_r(\mathbf{h}_i \parallel \mathbf{h}_j \parallel \mathbf{h}_{i,j,r}) + \mathbf{b}_r \tag{22}$$

(g) To show the effectiveness of cross-span interactions, we use our partial model with intermediate predictions (model (a)-(g) use $p_{i,j,r}$).
(h) Our full model (i.e, use $p^c_{i_l,j_l,r}$ 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

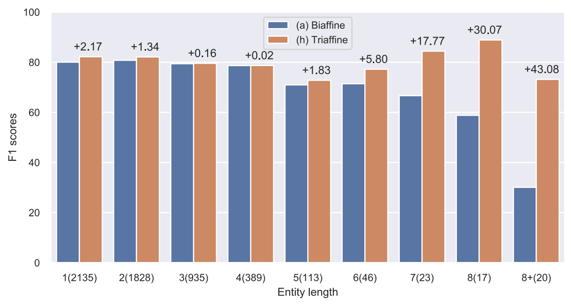

Figure 4: Comparison between triaffine and biaffine models on GENIA with different lengths of entities. Entity counts are in the parentheses.

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

Span	Type	$p_{i,j,r}$ Probability	Rank	Type	$p^{c}_{i,j,r}$ Probability
... [Cisco]$_{\text{ORG}}$'s been slammed, but once [they]$_{\text{ORG}}$'re exposed to [the rest of [the trading population]$_{\text{PER}}$]$_{\text{PER}}$ ...					
Cisco	ORG	1.00	1	ORG	1.00
they	ORG	1.00	2	ORG	1.00
the rest of the trading population	PER	1.00	3	PER	1.00
the trading population	GPE	0.50	4	PER	0.68
population	None	1.00	5	None	1.00
... simian virus 40 enhancer activity was blocked by the [MnlI-AluI fragment]$_{\text{DNA}}$ in [HeLa cells]$_{\text{cl}}$ but not in [B cells]$_{\text{ct}}$.					
HeLa cells	cell line	0.99	1	cell line	0.99
B cells	cell type	0.97	2	cell type	0.88
MnlI-AluI fragment	DNA	0.96	3	DNA	0.95
simian virus 40 enhancer	DNA	0.90	4	DNA	0.89
MnlI-AluI	protein	0.43	5	None	0.41
40 enhancer	None	0.99	6	None	1.00

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.

	ACE2004		GENIA	
	Flat (1,422)	Nested (1,092)	Flat (4,307)	Nested (1,199)
(a)	88.51	84.19	80.09	74.23
(h)	89.54	85.45	82.18	77.24
Δ	+1.03	+1.26	+2.09	+ 3.01

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

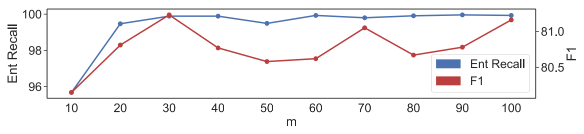

Figure 5: Recall for entity spans and F_1 scores with different numbers of candidate spans in GENIA dataset.

the recall of entity spans. We observe that taking top-30 spans achieves a recall of 99.89, which means it covers almost all entities. As the maximum number of entities is 25, we believe it is enough to select top-30 spans. Secondly, we check the model performance. With top-30 spans, the model achieves 81.23 scores in terms of F_1 and there is no obvious performance improvement with more candidates. Based on two above observations, we choose $m = 30$, which can well balance the performance and efficiency.

Finally, we test the efficiency of the decomposition. Compared with the naive triaffine scoring that takes 638.1ms (509.4ms in GPU + 128.7ms in CPU), the decomposed triaffine scoring takes 432.7ms (330.5ms in GPU + 102.2ms in CPU) for 10 iterations, which leads to approximately 32% speedup (details are shown in Appendix B).

4.7 Case Study

To analyze the effect of fusing information from related spans with the cross-span interaction, we show two examples from ACE2004 and GENIA datasets in Table 5. In the first example, the model first predicts "*the trading population*" as "GPE", however, it revises to "PER" correctly by considering span interactions with the outer span "*the rest of the trading population*". In the second example, it first predicts "*MnlI-AluI*" as "protein". By interacting with surrounding entities "*MnlI-AluI fragment*", the model corrects its label to None.

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.

References

Beatrice Alex, Barry Haddow, and Claire Grover. 2007. Recognising nested named entities in biomedical text. In *Biological, translational, and clinical language processing*, pages 65–72.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

George R Doddington, Alexis Mitchell, Mark A Przybocki, Lance A Ramshaw, Stephanie M Strassel, and Ralph M Weischedel. 2004. The automatic content extraction (ace) program-tasks, data, and evaluation. In *Lrec*, volume 2, pages 837–840. Lisbon.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Jenny Rose Finkel and Christopher D Manning. 2009. Nested named entity recognition. In *Proceedings of the 2009 conference on empirical methods in natural language processing*, pages 141–150.

Joseph Fisher and Andreas Vlachos. 2019. Merge and label: A novel neural network architecture for nested NER. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5840–5850, Florence, Italy. Association for Computational Linguistics.

Yao Fu, Chuanqi Tan, Mosha Chen, Songfang Huang, and Fei Huang. 2021. Nested named entity recognition with partially-observed treecrfs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12839–12847.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Heng Ji, Xiaoman Pan, Boliang Zhang, Joel Nothman, James Mayfield, Paul McNamee, and Cash Costello. 2017. Overview of tac-kbp2017 13 languages entity discovery and linking. *Theory and Applications of Categories*.

Arzoo Katiyar and Claire Cardie. 2018. Nested named entity recognition revisited. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 861–871.

J-D Kim, Tomoko Ohta, Yuka Tateisi, and Jun'ichi Tsujii. 2003. Genia corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl_1):i180–i182.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2020. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Xiaoya Li, Jingrong Feng, Yuxian Meng, Qinghong Han, Fei Wu, and Jiwei Li. 2020a. A unified MRC framework for named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5849–5859, Online. Association for Computational Linguistics.

Zuchao Li, Shexia He, Hai Zhao, Yiqing Zhang, Zhuosheng Zhang, Xi Zhou, and Xiang Zhou. 2019. Dependency or span, end-to-end uniform semantic role labeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6730–6737.

Zuchao Li, Hai Zhao, Rui Wang, and Kevin Parnow. 2020b. High-order semantic role labeling. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1134–1151, Online. Association for Computational Linguistics.

Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. 2019. Sequence-to-nuggets: Nested entity mention detection via anchor-region networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5182–5192, Florence, Italy. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Wei Lu and Dan Roth. 2015. Joint mention extraction and classification with mention hypergraphs. In *Proceedings of the 2015 Conference on Empirical*

Methods in Natural Language Processing, pages 857–867.

Yi Luan, Dave Wadden, Luheng He, Amy Shah, Mari Ostendorf, and Hannaneh Hajishirzi. 2019. A general framework for information extraction using dynamic span graphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3036–3046, Minneapolis, Minnesota. Association for Computational Linguistics.

Ying Luo and Hai Zhao. 2020. Bipartite flat-graph network for nested named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6408–6418, Online. Association for Computational Linguistics.

Yongliang Shen, Xinyin Ma, Zeqi Tan, Shuai Zhang, Wen Wang, and Weiming Lu. 2021. Locate and label: A two-stage identifier for nested named entity recognition. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*.

Takashi Shibuya and Eduard Hovy. 2020. Nested named entity recognition via second-best sequence learning and decoding. *Transactions of the Association for Computational Linguistics*, 8:605–620.

Jana Straková, Milan Straka, and Jan Hajic. 2019. Neural architectures for nested NER through linearization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5326–5331, Florence, Italy. Association for Computational Linguistics.

Chuanqi Tan, Wei Qiu, Mosha Chen, Rui Wang, and Fei Huang. 2020. Boundary enhanced neural span classification for nested named entity recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9016–9023.

Zeqi Tan, Yongliang Shen, Shuai Zhang, Weiming Lu, and Yueting Zhuang. 2021. A sequence-to-set network for nested named entity recognition. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI-21*.

David Wadden, Ulme Wennberg, Yi Luan, and Hannaneh Hajishirzi. 2019. Entity, relation, and event extraction with contextualized span representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5784–5789, Hong Kong, China. Association for Computational Linguistics.

Bailin Wang and Wei Lu. 2018. Neural segmental hypergraphs for overlapping mention recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 204–214, Brussels, Belgium. Association for Computational Linguistics.

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, Qiuyang 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*, pages 3295–3305, Online. Association for Computational Linguistics.

Changmeng Zheng, Yi Cai, Jingyun Xu, HF Leung, and Guandong Xu. 2019. A boundary-aware neural model for nested named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics.

Zexuan Zhong and Danqi Chen. 2021. A frustratingly easy approach for entity and relation extraction. In *North American Association for Computational Linguistics (NAACL)*.

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.

Parameters	ACE04	ACE05	KBP17	GENIA
Epoch	50	50	50	15
PLM lr	1e-5	3e-5	1e-5	3e-5
Static emb. lr	1e-4	5e-4	1e-4	5e-4
Task lr	1e-5	3e-5	1e-5	3e-5
σ	0.01	0.01	0.01	0.01
Batch size	8	72	8	48
d	256	256	256	320
m	30	30	30	30
Adam ϵ	1e-8	1e-8	1e-8	1e-8
Warmup ratio	0.0	0.0	0.0	0.0
Emb. dropout	0.2	0.2	0.2	0.2
MLP dropout	0.1	0.1	0.1	0.2
Weight decay	0.01	0.01	0.01	0.01
Clipping grad	0.1	0.1	0.1	0.1

Table 7: Hyper-parameters for using BERT encoder.

B The Decomposition of Triaffine Scoring

We introduce the decomposition of triaffine scoring in calculating $p_{i,j,r}$ and $p^c_{i,j,r}$.

Parameters	ACE04	ACE05	KBP17
Epoch	10	10	10
PLM lr	1e-5	1e-5	3e-5
Static emb. lr	1e-4	1e-4	5e-4
Task lr	1e-5	1e-5	3e-5
σ	0.01	0.01	0.01
Batch size	8	8	72
d	256	256	256
m	30	30	30
Adam ϵ	1e-8	1e-8	1e-8
Warmup ratio	0.0	0.0	0.0
Emb. dropout	0.2	0.2	0.2
MLP dropout	0.1	0.1	0.2
Weight decay	0.01	0.01	0.01
Clipping grad	0.1	0.1	0.1

Table 8: Hyper-parameters for using ALBERT encoder.

The naive calculation procedure of $p_{i,j,r}$ is:

$$s_{i,j,k,r} = \mathrm{TriAff}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_k, \mathcal{W}_r) \tag{23}$$

$$\alpha_{i,j,k,r} = \frac{\exp(s_{i,j,k,r})}{\sum_{k'=i}^{j} \exp(s_{i,j,k',r})} \tag{24}$$

$$\mathbf{h}_{i,j,r} = \sum_{k=i}^{j} \alpha_{i,j,k,r}\mathrm{MLP}(\mathbf{h}_k) \tag{25}$$

$$p_{i,j,r} = \mathrm{TriAff}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{i,j,r}, \mathcal{V}_r) \tag{26}$$

For our proposed decomposition of $p_{i,j,r}$, we first calculate $\alpha_{i,j,k,r}$ as equations 23 and 24. And we calculate:

$$o_{i,j,k,r} = \mathrm{TriAff}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_k, \mathcal{V}_r) \tag{27}$$

$$p_{i,j,r} = \sum_{k=i}^{j} \alpha_{i,j,k,r} o_{i,j,k,r} \tag{28}$$

The main difference between naive calculation and decomposition calculation is between Equation 26 and Equation 27.

We suppose our batch size as B, sequence count as N, output dimensions of MLP layers as d, the count of spans for calculating cross span representations as m, and label count as R (including None class). The shapes of tensors $[\mathbf{h}_i]$, $[\mathbf{h}_j]$, $[\mathbf{h}_k]$ are $B \times N \times d$. The shape of tensor $[\mathbf{h}_{i,j,r}]$ is $B \times N \times N \times R \times d$.

We benchmark the performances of Equation 26 and Equation 27 in PyTorch for 10 iterations. We use the same hyper-parameters and devices as our main experiments. We levearge opt_einsum[5] to calculate triaffine transformations in both equations.

Table 9 shows the time usage comparison between Equation 26 and Equation 27. Equation 26

[5]https://github.com/dgasmith/opt_einsum

Method	Function	CPU Time		GPU Time	
		Usage	Percentage	Usage	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 $[\mathbf{h}_{i,j,r}]$ 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.

Futhermore, we also test the time usage of $p^c_{i,j,r}$ 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 in CPU) regardless the relatively small size of $\mathbf{h}^c_{i,j,r}$ (The shape of tensor $[\mathbf{h}^c_{i,j,r}]$ is $B \times m \times R \times d$).