Two are Better than One: Joint Entity and Relation Extraction with Table-Sequence Encoders

Jue Wang[1] and **Wei Lu**[2]
[1]College of Computer Science and Technology, Zhejiang University
[2]StatNLP Research Group, Singapore University of Technology and Design
zjuwangjue@zju.edu.cn, luwei@sutd.edu.sg

Abstract

Named entity recognition and relation extraction are two important fundamental problems. Joint learning algorithms have been proposed to solve both tasks simultaneously, and many of them cast the joint task as a table-filling problem. However, they typically focused on learning a single encoder (usually learning representation in the form of a table) to capture information required for both tasks within the same space. We argue that it can be beneficial to design two distinct encoders to capture such two different types of information in the learning process. In this work, we propose the novel *table-sequence encoders* where two different encoders – a table encoder and a sequence encoder are designed to help each other in the representation learning process. Our experiments confirm the advantages of having *two* encoders over *one* encoder. On several standard datasets, our model shows significant improvements over existing approaches.[1]

1 Introduction

Named Entity Recognition (NER, Florian et al. 2006, 2010) and Relation Extraction (RE, Zhao and Grishman 2005; Jiang and Zhai 2007; Sun et al. 2011; Plank and Moschitti 2013) are two fundamental tasks in Information Extraction (IE). Both tasks aim to extract structured information from unstructured texts. One typical approach is to first identify entity mentions, and next perform classification between every two mentions to extract relations, forming a pipeline (Zelenko et al., 2002; Chan and Roth, 2011). An alternative and more recent approach is to perform these two tasks jointly (Li and Ji, 2014; Miwa and Sasaki, 2014; Miwa and Bansal, 2016), which mitigates the error propagation issue associated with the pipeline approach and leverages the interaction between tasks, resulting in improved performance.

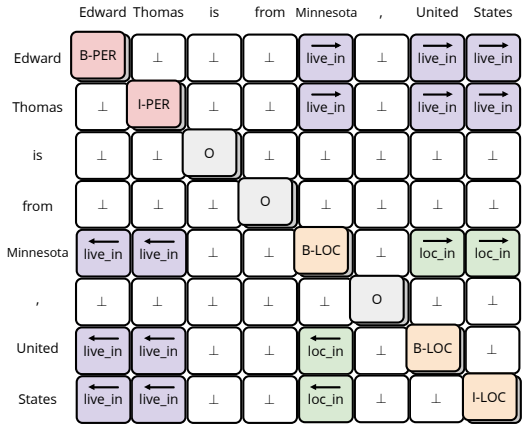


Figure 1: An example of table filling for NER and RE.

Among several joint approaches, one popular idea is to cast NER and RE as a table filling problem (Miwa and Sasaki, 2014; Gupta et al., 2016; Zhang et al., 2017). Typically, a two-dimensional (2D) table is formed where each entry captures the interaction between two individual words within a sentence. NER is then regarded as a sequence labeling problem where tags are assigned along the diagonal entries of the table. RE is regarded as the problem of labeling other entries within the table. Such an approach allows NER and RE to be performed using a single model, enabling the potentially useful interaction between these two tasks. One example[2] is illustrated in Figure 1.

Unfortunately, there are limitations with the existing joint methods. First, these methods typically suffer from *feature confusion* as they use a single representation for the two tasks – NER and RE. As a result, features extracted for one task may

[1]Our code is available at https://github.com/LorrinWWW/two-are-better-than-one.

[2]The exact settings for table filling may be different for different papers. Here we fill the entire table (rather than the lower half of the table), and assign relation tags to cells involving two complete entity spans (rather than part of such spans). We also preserve the direction of the relations.

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 1706–1721,
November 16–20, 2020.

coincide or conflict with those for the other, thus confusing the learning model. Second, these methods *underutilize* the table structure as they usually convert it to a sequence and then use a sequence labeling approach to fill the table. However, crucial structural information (e.g., the 4 entries at the bottom-left corner of Figure 1 share the same label) in the 2D table might be lost during such conversions.

In this paper, we present a novel approach to address the above limitations. Instead of predicting entities and relations with a single representation, we focus on learning two types of representations, namely *sequence representations* and *table representations*, for NER and RE respectively. On one hand, the two separate representations can be used to capture task-specific information. On the other hand, we design a mechanism to allow them to interact with each other, in order to take advantage of the inherent association underlying the NER and RE tasks. In addition, we employ neural network architectures that can better capture the structural information within the 2D table representation. As we will see, such structural information (in particular the context of neighboring entries in the table) is essential in achieving better performance.

The recent prevalence of BERT (Devlin et al., 2019) has led to great performance gains on various NLP tasks. However, we believe that the previous use of BERT, i.e., employing the contextualized word embeddings, does not fully exploit its potential. One important observation here is that the pairwise self-attention weights maintained by BERT carry knowledge of *word-word interactions*. Our model can effectively use such knowledge, which helps to better learn table representations. To the best of our knowledge, this is the first work to use the attention weights of BERT for learning table representations.

We summarize our contributions as follows:

- We propose to learn two separate encoders – a table encoder and a sequence encoder. They interact with each other, and can capture task-specific information for the NER and RE tasks;
- We propose to use multidimensional recurrent neural networks to better exploit the structural information of the table representation;
- We effectively leverage the word-word interaction information carried in the attention weights from BERT, which further improves the performance.

Our proposed method achieves the state-of-the-art performance on four datasets, namely ACE04, ACE05, CoNLL04, and ADE. We also conduct further experiments to confirm the effectiveness of our proposed approach.

2 Related Work

NER and RE can be tackled by using separate models. By assuming gold entity mentions are given as inputs, RE can be regarded as a classification task. Such models include kernel methods (Zelenko et al., 2002), RNNs (Zhang and Wang, 2015), recursive neural networks (Socher et al., 2012), CNNs (Zeng et al., 2014), and Transformer models (Verga et al., 2018; Wang et al., 2019). Another branch is to detect cross-sentence level relations (Peng et al., 2017; Gupta et al., 2019), and even document-level relations (Yao et al., 2019; Nan et al., 2020). However, entities are usually not directly available in practice, so these approaches may require an additional entity recognizer to form a pipeline.

Joint learning has been shown effective since it can alleviate the error propagation issue and benefit from exploiting the interrelation between NER and RE. Many studies address the joint problem through a cascade approach, i.e., performing NER first followed by RE. Miwa and Bansal (2016) use bi-LSTM (Graves et al., 2013) and tree-LSTM (Tai et al., 2015) for the joint task. Bekoulis et al. (2018a,b) formulate it as a head selection problem. Nguyen and Verspoor (2019) apply biaffine attention (Dozat and Manning, 2017) for RE. Luan et al. (2019), Dixit and Al (2019), and Wadden et al. (2019) use span representations to predict relations.

Miwa and Sasaki (2014) tackle joint NER and RE as from a table filling perspective, where the entry at row i and column j of the table corresponds to the pair of i-th and j-th word of the input sentence. The diagonal of the table is filled with the entity tags and the rest with the relation tags indicating possible relations between word pairs. Similarly, Gupta et al. (2016) employ a bi-RNN structure to label each word pair. Zhang et al. (2017) propose a global optimization method to fill the table. Tran and Kavuluru (2019) investigate CNNs on this task.

Recent work (Luan et al., 2019; Dixit and Al, 2019; Wadden et al., 2019; Li et al., 2019; Eberts and Ulges, 2019) usually leverages pre-trained language models such as ELMo (Peters et al., 2018), BERT (Devlin et al., 2019), RoBERTa (Liu et al.,

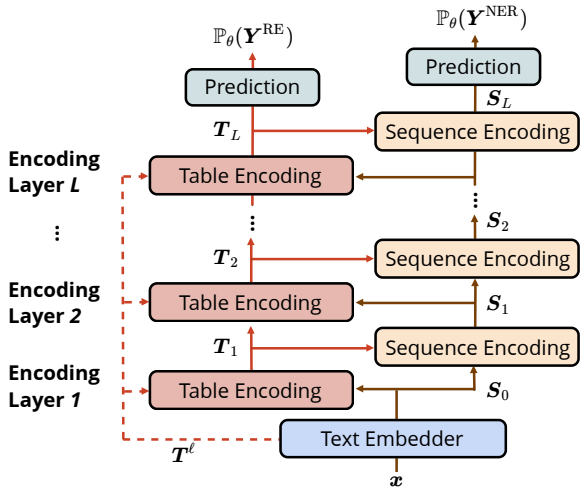


Figure 2: Overview of the table-sequence encoders. Dashed lines are for optional components ($\boldsymbol{T}^\ell$).

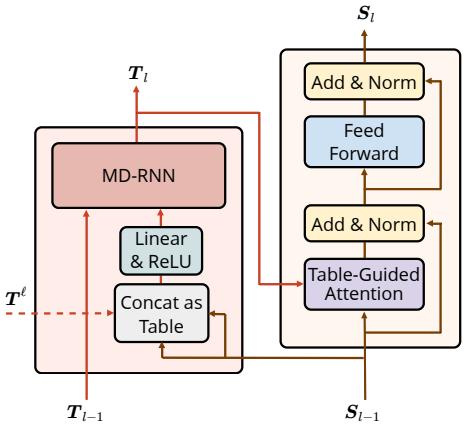


Figure 3: A layer in the table-sequence encoders.

2019), and ALBERT (Lan et al., 2019). However, none of them use pre-trained attention weights, which convey rich relational information between words. We believe it can be useful for learning better table representations for RE.

3 Problem Formulation

In this section, we formally formulate the NER and RE tasks. We regard NER as a sequence labeling problem, where the gold entity tags $\boldsymbol{y}^{\text{NER}}$ are in the standard BIO (Begin, Inside, Outside) scheme (Sang and Veenstra, 1999; Ratinov and Roth, 2009). For the RE task, we mainly follow the work of Miwa and Sasaki (2014) to formulate it as a table filling problem. Formally, given an input sentence $\boldsymbol{x} = [x_i]_{1 \le i \le N}$, we maintain a tag table $\boldsymbol{y}^{\text{RE}} = [y^{\text{RE}}_{i,j}]_{1 \le i,j \le N}$. Suppose there is a relation with type r pointing from mention $x_{i^b}, .., x_{i^e}$ to mention $x_{j^b}, .., x_{j^e}$, we have $y^{\text{RE}}_{i,j} = \overrightarrow{r}$ and $y^{\text{RE}}_{j,i} = \overleftarrow{r}$ for all $i \in [i^b, i^e] \wedge j \in [j^b, j^e]$. We use $\perp$ for word pairs with no relation. An example was given earlier in Figure 1.

4 Model

We describe the model in this section. The model consists of two types of interconnected encoders, a table encoder for table representation and a sequence encoder for sequence representation, as shown in Figure 2. Collectively, we call them *table-sequence encoders*. Figure 3 presents the details of each layer of the two encoders, and how they interact with each other. In each layer, the table encoder uses the sequence representation to construct the table representation; and then the sequence encoder uses the table representation to contextualize the sequence representation. With multiple layers, we incrementally improve the quality of both representations.

4.1 Text Embedder

For a sentence containing N words $\boldsymbol{x} = [x_i]_{1 \le i \le N}$, we define the word embeddings $\boldsymbol{x}^w \in \mathbb{R}^{N \times d_1}$, as well as character embeddings $\boldsymbol{x}^c \in \mathbb{R}^{N \times d_2}$ computed by an LSTM (Lample et al., 2016). We also consider the contextualized word embeddings $\boldsymbol{x}^\ell \in \mathbb{R}^{N \times d_3}$, which can be produced from language models such as BERT.

We concatenate those embeddings for each word and use a linear projection to form the initial sequence representation $\boldsymbol{S}_0 \in \mathbb{R}^{N \times H}$:

$$\boldsymbol{S}_0 = \text{Linear}([\boldsymbol{x}^c; \boldsymbol{x}^w; \boldsymbol{x}^\ell]) \quad (1)$$

where each word is represented as an H dimensional vector.

4.2 Table Encoder

The table encoder, shown in the left part of Figure 3, is a neural network used to learn a table representation, an $N \times N$ table of vectors, where the vector at row i and column j corresponds to the i-th and j-th word of the input sentence.

We first construct a non-contextualized table by concatenating every two vectors of the sequence representation followed by a fully-connected layer to halve the hidden size. Formally, for the l-th layer, we have $\boldsymbol{X}_l \in \mathbb{R}^{N \times N \times H}$, where:

$$X_{l,i,j} = \text{ReLU}(\text{Linear}([S_{l-1,i}; S_{l-1,j}])) \quad (2)$$

Next, we use the Multi-Dimensional Recurrent Neural Networks (MD-RNN, Graves et al. 2007)

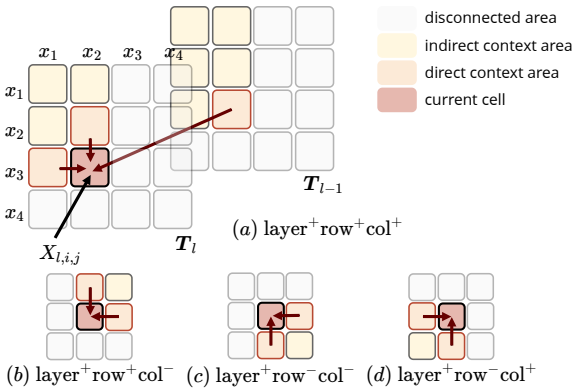


Figure 4: How the hidden states are computed in MD-RNN with 4 directions. We use D^+ or D^- to indicate the direction that the hidden states flow between cells at the D dimension (where D can be *layer*, *row* or *col*). For brevity, we omit the input and the *layer* dimension for cases (b), (c) and (d), as they are the same as (a).

with Gated Recurrent Unit (GRU, Cho et al. 2014) to contextualize $\boldsymbol{X}_l$. We iteratively compute the hidden states of each cell to form the contextualized table representation $\boldsymbol{T}_l$, where:

$$T_{l,i,j} = \text{GRU}(X_{l,i,j}, T_{l-1,i,j}, T_{l,i-1,j}, T_{l,i,j-1}) \quad (3)$$

We provide the multi-dimensional adaptations of GRU in Appendix A to avoid excessive formulas here.

Generally, it exploits the context along *layer*, *row*, and *column* dimensions. That is, it does not consider only the cells at neighbouring rows and columns, but also those of the previous layer.

The time complexity of the naive implementation (i.e., two for-loops) for each layer is $O(N \times N)$ for a sentence with length N. However, antidiagonal entries[3] can be calculated at the same time as they do not depend on each other. Therefore, we can optimize it through parallelization and reduce the effective time complexity to $O(N)$.

The above illustration describes a unidirectional RNN, corresponding to Figure 4(a). Intuitively, we would prefer the network to have access to the surrounding context in all directions. However, this could not be done by one single RNN. For the case of 1D sequence modeling, this problem is resolved by introducing bidirectional RNNs. Graves et al. (2007) discussed quaddirectional RNNs to access the context from four directions for modeling 2D data. Therefore, similar to 2D-RNN, we also need

[3]We define antidiagonal entries to be entries at position (i, j) such that $i+j = N+1+\Delta$, where $\Delta \in [-N+1, N-1]$ is the offset to the main antidiagonal entries.

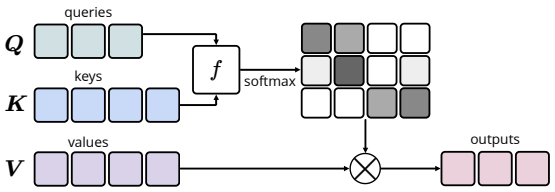


Figure 5: The generalized form of attention. The softmax function is used to normalize the weights of values $\boldsymbol{V}$ for each query Q_i.

to consider RNNs in four directions[4]. We visualize them in Figure 4.

Empirically, we found the setting only considering cases (a) and (c) in Figure 4 achieves no worse performance than considering four cases altogether. Therefore, to reduce the amount of computation, we use such a setting as default. The final table representation is then the concatenation of the hidden states of the two RNNs:

$$T^{(a)}_{l,i,j} = \text{GRU}^{(a)}(X_{l,i,j}, T^{(a)}_{l-1,i,j}, T^{(a)}_{l,i-1,j}, T^{(a)}_{l,i,j-1}) \quad (4)$$

$$T^{(c)}_{l,i,j} = \text{GRU}^{(c)}(X_{l,i,j}, T^{(c)}_{l-1,i,j}, T^{(c)}_{l,i+1,j}, T^{(c)}_{l,i,j+1}) \quad (5)$$

$$T_{l,i,j} = [T^{(a)}_{l,i,j}; T^{(c)}_{l,i,j}] \quad (6)$$

4.3 Sequence Encoder

The sequence encoder is used to learn the sequence representation – a sequence of vectors, where the i-th vector corresponds to the i-th word of the input sentence. The architecture is similar to Transformer (Vaswani et al., 2017), shown in the right portion of Figure 3. However, we replace the scaled dot-product attention with our proposed *table-guided attention*. Here, we mainly illustrate why and how the table representation can be used to compute attention weights.

First of all, given $\boldsymbol{Q}$ (queries), $\boldsymbol{K}$ (keys) and $\boldsymbol{V}$ (values), a generalized form of attention is defined in Figure 5. For each query, the output is a weighted sum of the values, where the weight assigned to each value is determined by the relevance (given by score function f) of the query with all the keys.

For each query Q_i and key K_j, Bahdanau et al. (2015) define f in the form of:

$$f(Q_i, K_j) = U \cdot g(Q_i, K_j) \quad (7)$$

where U is a learnable vector and g is the function to map each query-key pair to a vector. Specifically,

[4]In our scenario, there is an additional layer dimension. However, as the model always traverses from the first layer to the last layer, only one direction shall be considered for the layer dimension.

they define $g(Q_i, K_j) = \tanh(Q_i W_0 + K_j W_1)$, where W_0, W_1 are learnable parameters.

Our attention mechanism is essentially a self-attention mechanism, where the queries, keys and values are exactly the same. In our case, they are essentially sequence representation S_{l-1} of the previous layer (i.e., $\boldsymbol{Q} = \boldsymbol{K} = \boldsymbol{V} = \boldsymbol{S}_{l-1}$). The attention weights (i.e., the output from the function f in Figure 5) are essentially constructed from both queries and keys (which are the same in our case). On the other hand, we also notice the table representation $\boldsymbol{T}_l$ is also constructed from $\boldsymbol{S}_{l-1}$. So we can consider $\boldsymbol{T}_l$ to be a function of queries and keys, such that $T_{l,i,j} = g(S_{l-1,i}, S_{l-1,j}) = g(Q_i, K_j)$. Then we put back this g function to Equation 7, and get the proposed table-guided attention, whose score function is:

$$f(Q_i, K_j) = U \cdot T_{l,i,j} \tag{8}$$

We show the advantages of using this table-guided attention: (1) we do not have to calculate g function since $\boldsymbol{T}_l$ is already obtained from the table encoder; (2) $\boldsymbol{T}_l$ is contextualized along the *row*, *column*, and *layer* dimensions, which corresponds to queries, keys, and queries and keys in the previous layer, respectively. Such contextual information allows the network to better capture more difficult word-word dependencies; (3) it allows the table encoder to participate in the sequence representation learning process, thereby forming the bidirectional interaction between the two encoders.

The table-guided attention can be extended to have multiple heads (Vaswani et al., 2017), where each head is an attention with independent parameters. We concatenate their outputs and use a fully-connected layer to get the final attention outputs.

The remaining parts are similar to Transformer. For layer l, we use position-wise feedforward neural networks (FFNN) after self-attention, and wrap attention and FFNN with a residual connection (He et al., 2016) and layer normalization (Ba et al. 2016), to get the output sequence representation:

$$\tilde{\boldsymbol{S}}_l = \text{LayerNorm}(\boldsymbol{S}_{l-1} + \text{SelfAttn}(\boldsymbol{S}_{l-1})) \tag{9}$$

$$\boldsymbol{S}_l = \text{LayerNorm}(\tilde{\boldsymbol{S}}_l + \text{FFNN}(\tilde{\boldsymbol{S}}_l)) \tag{10}$$

4.4 Exploit Pre-trained Attention Weights

In this section, we describe the dashed lines in Figures 2 and 3, which we ignored in the previous discussions. Essentially, they exploit information in the form of attention weights from a pre-trained language model such as BERT.

We stack the attention weights of all heads and all layers to form $\boldsymbol{T}^{\ell} \in \mathbb{R}^{N \times N \times (L^{\ell} \times A^{\ell})}$, where L^{ℓ} is the number of stacked Transformer layers, and A^{ℓ} is the number of heads in each layer. We leverage $\boldsymbol{T}^{\ell}$ to form the inputs of MD-RNNs in the table encoder. Equation 2 is now replaced with:

$$X_{l,i,j} = \text{ReLU}(\text{Linear}([S_{l-1,i}; S_{l-1,j}; T^{\ell}_{i,j}])) \tag{11}$$

We keep the rest unchanged. We believe this simple yet novel use of the attention weights allows us to effectively incorporate the useful word-word interaction information captured by pre-trained models such as BERT into our table-sequence encoders for improved performance.

5 Training and Evaluation

We use $\boldsymbol{S}_L$ and $\boldsymbol{T}_L$ to predict the probability distribution of the entity and relation tags:

$$P_\theta(\boldsymbol{Y}^{\text{NER}}) = \text{softmax}(\text{Linear}(\boldsymbol{S}_L)) \tag{12}$$

$$P_\theta(\boldsymbol{Y}^{\text{RE}}) = \text{softmax}(\text{Linear}(\boldsymbol{T}_L)) \tag{13}$$

where $\boldsymbol{Y}^{\text{NER}}$ and $\boldsymbol{Y}^{\text{RE}}$ are random variables of the predicted tags, and P_θ is the estimated probability function with θ being our model parameters.

For training, both NER and RE adopt the prevalent cross-entropy loss. Given the input text $\boldsymbol{x}$ and its gold tag sequence $\boldsymbol{y}^{\text{NER}}$ and tag table $\boldsymbol{y}^{\text{RE}}$, we then calculate the following two losses:

$$\mathcal{L}^{\text{NER}} = \sum_{i \in [1,N]} -\log P_\theta(Y_i^{\text{NER}} = y_i^{\text{NER}}) \tag{14}$$

$$\mathcal{L}^{\text{RE}} = \sum_{i,j \in [1,N]; i \neq j} -\log P_\theta(Y_{i,j}^{\text{RE}} = y_{i,j}^{\text{RE}}) \tag{15}$$

The goal is to minimize both losses $\mathcal{L}^{\text{NER}} + \mathcal{L}^{\text{RE}}$.

During evaluation, the prediction of relations relies on the prediction of entities, so we first predict the entities, and then look up the relation probability table $P_\theta(\boldsymbol{Y}^{\text{RE}})$ to see if there exists a valid relation between predicted entities.

Specifically, we predict the entity tag of each word by choosing the class with the highest probability:

$$\underset{e}{\text{argmax}}\, P_\theta(Y_i^{\text{NER}} = e) \tag{16}$$

The whole tag sequence can be transformed into entities with their boundaries and types.

Relations on entities are mapped to relation classes with highest probabilities on words of the entities. We also consider the two directed tags

for each relation. Therefore, for two entity spans (i^b, i^e) and (j^b, j^e), their relation is given by:

$$\underset{\overrightarrow{r}}{\text{argmax}} \sum_{i \in [i^b, i^e], j \in [j^b, j^e]} P_\theta(Y^{\text{RE}}_{i,j} = \overrightarrow{r}) + P_\theta(Y^{\text{RE}}_{j,i} = \overleftarrow{r}) \quad (17)$$

where the no-relation type $\perp$ has no direction, so if $\overrightarrow{r} = \perp$, we have $\overleftarrow{r} = \perp$ as well.

6 Experiments

6.1 Data

We evaluate our model on four datasets, namely ACE04 (Doddington et al., 2004), ACE05 (Walker et al., 2006), CoNLL04 (Roth and tau Yih, 2004) and ADE (Gurulingappa et al., 2012). More details could be found in Appendix B.

Following the established line of work, we use the F1 measure to evaluate the performance of NER and RE. For NER, an entity prediction is correct if and only if its type and boundaries both match with those of a gold entity.[5] For RE, a relation prediction is considered correct if its relation type and the boundaries of the two entities match with those in the gold data. We also report the strict relation F1 (denoted RE+), where a relation prediction is considered correct if its relation type as well as the boundaries and types of the two entities all match with those in the gold data. Relations are asymmetric, so the order of the two entities in a relation matters.

6.2 Model Setup

We tune hyperparameters based on results on the development set of ACE05 and use the same setting for other datasets. GloVe vectors (Pennington et al., 2014) are used to initialize word embeddings. We also use the BERT variant – ALBERT as the default pre-trained language model. Both pre-trained word embeddings and language model are fixed without fine-tuning. In addition, we stack three encoding layers ($L = 3$) with independent parameters including the GRU cell in each layer. For the table encoder, we use two separate MD-RNNs with the directions of "layer$^+$row$^+$col$^+$" and "layer$^+$row$^-$col$^-$" respectively. For the sequence encoder, we use eight attention heads to attend to different representation subspaces. We report the averaged F1 scores of 5 runs for our models. For each run, we keep the model that achieves the highest averaged entity F1 and relation F1 on the development set, and evaluate and report its score on the test set. Other hyperparameters could be found in Appendix C.

[5] Follow Li and Ji (2014); Miwa and Bansal (2016), we use head spans for entities in ACE. And we keep the full mention boundary for other corpora.

Data	Model	NER	RE	RE+
ACE04	Li and Ji (2014) ▽	79.7	48.3	45.3
	Katiyar and Cardie (2017) ▽	79.6	49.3	45.7
	Bekoulis et al. (2018b) ▽	81.2	-	47.1
	Bekoulis et al. (2018a) ▽	81.6	-	47.5
	Miwa and Bansal (2016) ▽	81.8	-	48.4
	Li et al. (2019) ▽	83.6	-	49.4
	Luan et al. (2019) ▽	87.4	59.7	-
	Ours ▽	**88.6**	**63.3**	**59.6**
ACE05	Li and Ji (2014) ▽	80.8	52.1	49.5
	Miwa and Bansal (2016) ▽	83.4	-	55.6
	Katiyar and Cardie (2017) ▽	82.6	55.9	53.6
	Zhang et al. (2017) ▽	83.6	-	57.5
	Sun et al. (2018) ▽	83.6	-	59.6
	Li et al. (2019) ▽	84.8	-	60.2
	Dixit and Al (2019) ▽	86.0	62.8	-
	Luan et al. (2019) ▽	88.4	63.2	-
	Wadden et al. (2019) ▽	88.6	63.4	-
	Ours ▽	**89.5**	**67.6**	**64.3**
CoNLL04	Miwa and Sasaki (2014)▽	80.7	-	61.0
	Bekoulis et al. (2018a)▲	83.6	-	62.0
	Bekoulis et al. (2018b)▲	83.9	-	62.0
	Tran and Kavuluru (2019)▲	84.2	-	62.3
	Nguyen and Verspoor (2019)▲	86.2	-	64.4
	Zhang et al. (2017)▽	85.6	-	67.8
	Li et al. (2019)▽	87.8	-	68.9
	Eberts and Ulges (2019)▽	88.9	-	71.5
	Eberts and Ulges (2019)▲	86.3	-	72.9
	Ours▽	**90.1**	**73.8**	**73.6**
	Ours▲	**86.9**	**75.8**	**75.4**
ADE	Li et al. (2016) ▲	79.5	-	63.4
	Li et al. (2017) ▲	84.6	-	71.4
	Bekoulis et al. (2018b) ▲	86.4	-	74.6
	Bekoulis et al. (2018a) ▲	86.7	-	75.5
	Tran and Kavuluru (2019) ▲	87.1	-	77.3
	Eberts and Ulges (2019) ▲	89.3	-	79.2
	Ours ▲	**89.7**	**80.1**	**80.1**

Table 1: Main results. ▽: micro-averaged F1; ▲: macro-averaged F1.

6.3 Comparison with Other Models

Table 1 presents the comparison of our model with previous methods on four datasets. Our NER performance is increased by 1.2, 0.9, 1.2/0.6 and 0.4 absolute F1 points over the previous best results. Besides, we observe even stronger performance gains in the RE task, which are 3.6, 4.2, 2.1/2.5 (RE+) and 0.9 (RE+) absolute F1 points, respectively. This indicates the effectiveness of our model

LM	$+\boldsymbol{x}^\ell$		$+\boldsymbol{x}^\ell$ $+\boldsymbol{T}^\ell$	
	NER	RE	NER	RE
ELMo	86.4	64.3	-	-
BERT	87.8	64.8	88.2	67.4
RoBERTa	88.9	66.2	89.3	67.6
ALBERT	89.4	66.0	89.5	67.6

Table 2: Using different pre-trained language models on ACE05. $+\boldsymbol{x}^\ell$ uses the contextualized word embeddings; $+\boldsymbol{T}^\ell$ uses the attention weights.

for jointly extracting entities and their relations. Since our reported numbers are the average of 5 runs, we can consider our model to be achieving new state-of-the-art results.

6.4 Comparison of Pre-trained Models

In this section, we evaluate our method with different pre-trained language models, including ELMo, BERT, RoBERTa and ALBERT, with and without attention weights, to see their individual contribution to the final performance.

Table 2 shows that, even using the relatively earlier contextualized embeddings without attention weights (ELMo $+\boldsymbol{x}^\ell$), our system is still comparable to the state-of-the-art approach (Wadden et al., 2019), which was based on BERT and achieved F1 scores of 88.6 and 63.4 for NER and RE respectively. It is important to note that the model of Wadden et al. (2019) was trained on the additional coreference annotations from OntoNotes (Weischedel et al., 2011) before fine-tuning on ACE05. Nevertheless, our system still achieves comparable results, showing the effectiveness of the table-sequence encoding architecture.

The overall results reported in Table 2 confirm the importance of leveraging the attention weights, which bring improvements for both NER and RE tasks. This allows the system using vanilla BERT to obtain results no worse than RoBERTa and ALBERT in relation extraction.

6.5 Ablation Study

We design several additional experiments to understand the effectiveness of components in our system. The experiments are conducted on ACE05.

We also compare different table filling settings, which are included in Appendix E.

6.5.1 Bidirectional Interaction

We first focus on the understanding of the necessity of modeling the bidirectional interaction between the two encoders. Results are presented in Table 3. "RE (gold)" is presented so as to compare with settings that do not predict entities, where the gold entity spans are used in the evaluation.

Setting	NER	RE	RE (gold)
Default	89.5	67.6	70.4
w/o Relation Loss	89.4	-	-
w/o Table Encoder	88.4	-	-
w/o Entity Loss	-	-	69.8
w/o Sequence Encoder	-	-	69.2
w/o Bi-Interaction	88.2	66.3	69.2
NER on diagonal	89.4	67.1	70.2
w/o Sequence Encoder	88.6	67.0	70.2

Table 3: Ablation of the two encoders on ACE05. Gold entity spans are given in RE (gold).

We first try optimizing the NER and RE objectives separately, corresponding to "w/o Relation Loss" and "w/o Entity Loss". Compared with learning with a joint objective, the results of these two settings are slightly worse, which indicates that learning better representations for one task not only is helpful for the corresponding task, but also can be beneficial for the other task.

Next, we investigate the individual sequence and table encoder, corresponding to "w/o Table Encoder" and "w/o Sequence Encoder". We also try jointly training the two encoders but cut off the interaction between them, which is "w/o Bi-Interaction". Since no interaction is allowed in the above three settings, the table-guided attention is changed to conventional multi-head scaled dot-product attention, and the table encoding layer always uses the initial sequence representation $\boldsymbol{S}_0$ to enrich the table representation. The results of these settings are all significantly worse than the default one, which indicates the importance of the bidirectional interaction between sequence and table representation in our table-sequence encoders.

We also experiment the use of the main diagonal entries of the table representation to tag entities, with results reported under "NER on diagonal". This setup attempts to address NER and RE in the same encoding space, in line with the original intention of Miwa and Sasaki (2014). By exploiting the interrelation between NER and RE, it achieves better performance compared with models without such information. However, it is worse than our default setting. We ascribe this to the potential incompatibility of the desired encoding space of entities and relations. Finally, although it does not

# Layers	Shared			Non-shared		
	# params	NER	RE	# params	NER	RE
$L = 1$	2.2M	89.2	66.0	1.9M	89.2	66.0
$L = 2$	2.2M	89.5	67.0	3.2M	89.5	67.1
$L = 3$	2.2M	89.3	67.3	4.5M	89.5	67.6
$L = 4$	2.2M	89.7	67.6	5.7M	89.6	67.7
$L = 5$	2.2M	89.6	67.6	7.0M	89.6	67.7

Table 4: The performance on ACE05 with different number of layers. Pre-trained word embeddings and language models are not counted to the number of parameters. The underlined ones are from our default setting.

Setting	NER	RE
Unidirectional	89.6	66.9
Bidirectional	89.5	67.6
Quaddirectional	89.7	67.6
Layer-wise only	89.3	63.9
Bidirectional w/o column	89.5	67.2
Bidirectional w/o row	89.3	67.4
Bidirectional w/o layer	89.3	66.7

Table 5: The effect of the dimensions and directions of MD-RNNs. Experiments are conducted on ACE05. The underlined ones are from our default setting.

directly use the sequence representation, removing the sequence encoder will lead to performance drop for NER, which indicates the sequence encoder can help improve the table encoder by better capturing the structured information within the sequence.

6.5.2 Encoding Layers

Table 4 shows the effect of the number of encoding layers, which is also the number of bidirectional interactions involved. We conduct one set of experiments with shared parameters for the encoding layers and another set with independent parameters. In general, the performance increases when we gradually enlarge the number of layers L. Specifically, since the shared model does not introduce more parameters when tuning L, we consider that our model benefits from the mutual interaction inside table-sequence encoders. Typically, under the same value L, the non-shared model employs more parameters than the shared one to enhance its modeling capability, leading to better performance. However, when $L > 3$, there is no significant improvement by using non-shared model. We believe that increasing the number of layers may bring the risk of over-fitting, which limits the performance of the network. We choose to adopt the non-shared model with $L = 3$ as our default setting.

6.5.3 Settings of MD-RNN

Table 5 presents the comparisons of using different dimensions and directions to learn the table representation, based on MD-RNN. Among those settings, "Unidirectional" refers to an MD-RNN with direction "layer$^+$row$^+$col$^+$"; "Bidirectional" uses two MD-RNNs with directions "layer$^+$row$^+$col$^+$" and "layer$^+$row$^-$col$^-$" respectively; "Quaddirectional" uses MD-RNNs in four directions, illustrated in Figure 4. Their results are improved when adding more directions, showing richer contextual information is beneficial. Since the bidirectional model is almost as good as the quaddirectional one, we leave the former as the default setting.

In addition, we are also curious about the contribution of *layer*, *row*, and *column* dimensions for MD-RNNs. We separately removed the *layer*, *row*, and *column* dimension. As we can see, the results are all lower than the original model without removal of any dimension. "Layer-wise only" removed *row* and *col* dimensions, and is worse than others as it does not exploit the sentential context.

More experiments with more settings are presented in Appendix D. Specifically, all unidirectional RNNs are consistently worse than others, while bidirectional RNNs are usually on-par with quaddirectional RNNs. Besides, we also tried to use CNNs to implement the table encoder. However, since it is usually difficult for CNNs to learn long-range dependencies, we found the performance was worse than the RNN-based models.

6.6 Attention Visualization

We visualize the table-guided attention with bertviz (Vig, 2019)[6] for a better understanding of how the network works. We compare it with pre-trained Transformers (ALBERT) and human-defined ground truth, as presented in Figure 6.

Our discovery is similar to Clark et al. (2019). Most attention heads in the table-guided attention and ALBERT show simple patterns. As shown in the left part of Figure 6, these patterns include attending to the word itself, the next word, the last word, and the punctuation.

The right part of Figure 6 also shows task-related patterns, i.e., entities and relations. For a relation, we connect words from the head entity to the tail entity; For an entity, we connect every two words inside this entity mention. We can find that our pro-

[6] https://github.com/jessevig/bertviz

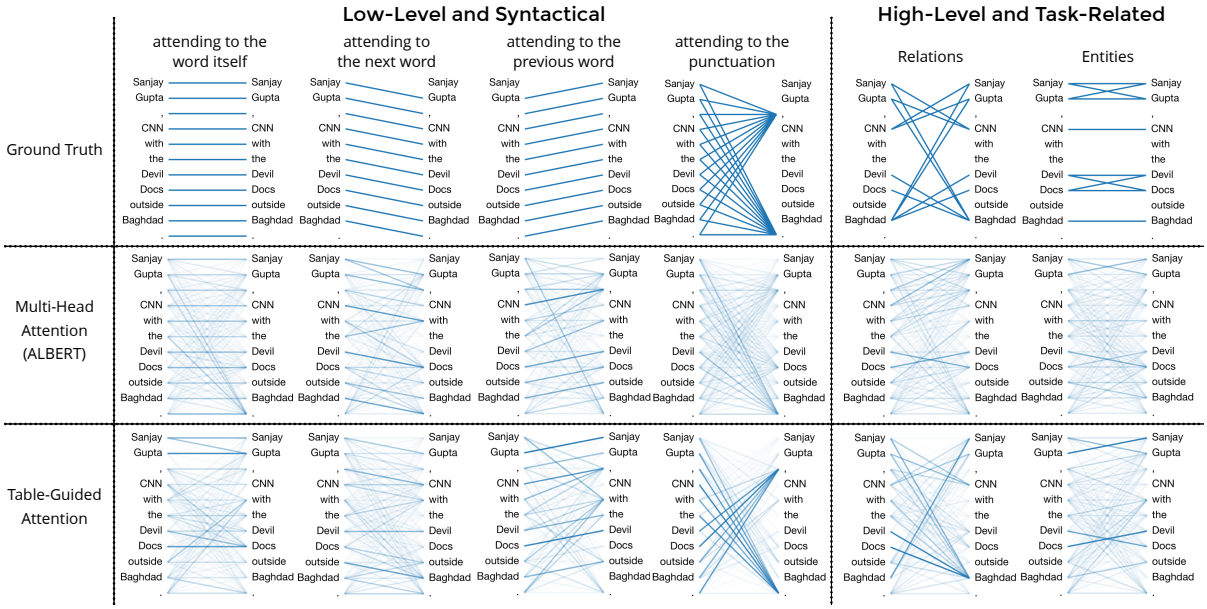


Figure 6: Comparison between ground truth and selected heads of ALBERT and table-guided attention. The sentence is randomly selected from the development set of ACE05.

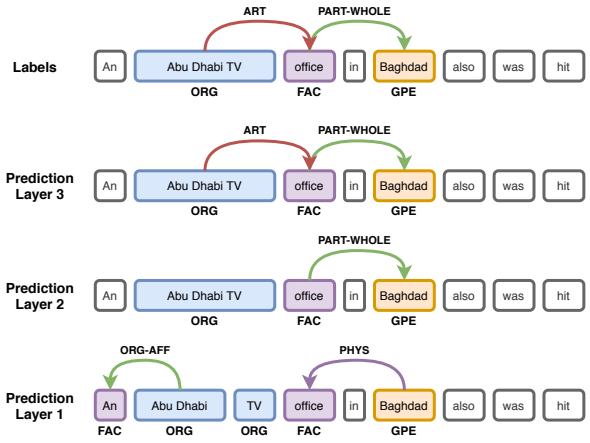


Figure 7: Probing intermediate states

posed table-guided attention has learned more task-related knowledge compared to ALBERT. In fact, not only does it capture the entities and their relations that ALBERT failed to capture, but it also has higher confidence. This indicates that our model has a stronger ability to capture complex patterns other than simple ones.

6.7 Probing Intermediate States

Figure 7 presents an example picked from the development set of ACE05. The prediction layer after training (a linear layer) is used as a probe to display the intermediate state of the model, so we can interpret how the model improves both representations from stacking multiple layers and thus from the bidirectional interaction. Such probing is valid since we use skip connection between two adjacent encoding layers, so the encoding spaces of the outputs of different encoding layers are consistent and therefore compatible with the prediction layer.

In Figure 7, the model made many wrong predictions in the first layer, which were gradually corrected in the next layers. Therefore, we can see that more layers allow more interaction and thus make the model better at capturing entities or relations, especially difficult ones. More cases are presented in Appendix F.

7 Conclusion

In this paper, we introduce the novel *table-sequence encoders* architecture for joint extraction of entities and their relations. It learns two separate encoders rather than one – a sequence encoder and a table encoder where explicit interactions exist between the two encoders. We also introduce a new method to effectively employ useful information captured by the pre-trained language models for such a joint learning task where a table representation is involved. We achieved state-of-the-art F1 scores for both NER and RE tasks across four standard datasets, which confirm the effectiveness of our approach. In the future, we would like to investigate how the table representation may be applied to other tasks. Another direction is to generalize the way in which the table and sequence interact to other types of representations.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments and Lidan Shou for his suggestions and support on this work. This work was done during the first author's remote internship with the StatNLP Group in Singapore University of Technology and Design. This research is supported by Ministry of Education, Singapore, under its Academic Research Fund (AcRF) Tier 2 Programme (MOE AcRF Tier 2 Award No: MOE2017-T2-1-156). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the Ministry of Education, Singapore.

References

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. Flair: An easy-to-use framework for state-of-the-art nlp. In *Proc. of NAACL-HLT*.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *arXiv preprint*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

Giannis Bekoulis, Johannes Deleu, Thomas Demeester, and Chris Develder. 2018a. Adversarial training for multi-context joint entity and relation extraction. In *Proc. of EMNLP*.

Giannis Bekoulis, Johannes Deleu, Thomas Demeester, and Chris Develder. 2018b. Joint entity recognition and relation extraction as a multi-head selection problem. *Expert Systems with Applications*.

Yee Seng Chan and Dan Roth. 2011. Exploiting syntactico-semantic structures for relation extraction. In *Proc. of NAACL-HLT*.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proc. of EMNLP*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does bert look at? an analysis of bert's attention. *arXiv preprint*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL-HLT*.

Kalpit Dixit and Yaser Al. 2019. Span-level model for relation extraction. In *Proc. of ACL*.

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 *Proc. of LREC*.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proc. of ICLR*.

Markus Eberts and Adrian Ulges. 2019. Span-based joint entity and relation extraction with transformer pre-training. *arXiv preprint*.

Radu Florian, Hongyan Jing, Nanda Kambhatla, and Imed Zitouni. 2006. Factorizing complex models: A case study in mention detection. In *Proc. of ACL*.

Radu Florian, John F. Pitrelli, Salim Roukos, and Imed Zitouni. 2010. Improving mention detection robustness to noisy input. In *Proc. of EMNLP*.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2007. Multi-dimensional recurrent neural networks. In *Proc. of ICANN*.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proc. of ICASSP*.

Pankaj Gupta, Subburam Rajaram, Hinrich Schütze, and Thomas Runkler. 2019. Neural relation extraction within and across sentence boundaries. In *Proc. of AAAI*.

Pankaj Gupta, Hinrich Schütze, and Bernt Andrassy. 2016. Table filling multi-task recurrent neural network for joint entity and relation extraction. In *Proc. of COLING*.

Harsha Gurulingappa, Abdul Mateen Rajput, Angus Roberts, Juliane Fluck, Martin Hofmann-Apitius, and Luca Toldo. 2012. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. *Journal of biomedical informatics*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. of CVPR*.

Jing Jiang and ChengXiang Zhai. 2007. A systematic exploration of the feature space for relation extraction. In *Proc. of HLT-NAACL*.

Arzoo Katiyar and Claire Cardie. 2017. Going out on a limb: Joint extraction of entity mentions and relations without dependency trees. In *Proc. of ACL*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proc. of HLT-NAACL*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. In *Proc. of ICLR*.

Fei Li, Meishan Zhang, Guohong Fu, and Donghong Ji. 2017. A neural joint model for entity and relation extraction from biomedical text. *BMC bioinformatics*.

Fei Li, Yue Zhang, Meishan Zhang, and Donghong Ji. 2016. Joint models for extracting adverse drug events from biomedical text. In *Proc. of IJCAI*.

Qi Li and Heng Ji. 2014. Incremental joint extraction of entity mentions and relations. In *Proc. of ACL*.

Xiaoya Li, Fan Yin, Zijun Sun, Xiayu Li, Arianna Yuan, Duo Chai, Mingxin Zhou, and Jiwei Li. 2019. Entity-relation extraction as multi-turn question answering. In *Proc. of ACL*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint*.

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 *Proc. of NAACL-HLT*.

Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using lstms on sequences and tree structures. In *Proc. of ACL*.

Makoto Miwa and Yutaka Sasaki. 2014. Modeling joint entity and relation extraction with table representation. In *Proc. of EMNLP*.

Guoshun Nan, Zhijiang Guo, Ivan Sekulic, and Wei Lu. 2020. Reasoning with latent structure refinement for document-level relation extraction. In *Proc. of ACL*.

Dat Quoc Nguyen and Karin Verspoor. 2019. End-to-end neural relation extraction using deep biaffine attention. In *Proc. of ECIR*.

Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. 2017. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proc. of EMNLP*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Barbara Plank and Alessandro Moschitti. 2013. Embedding semantic similarity in tree kernels for domain adaptation of relation extraction. In *Proc. of ACL*.

Lev-Arie Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proc. of CoNLL*.

Dan Roth and Wen tau Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proc. of CoNLL*.

Erik F. Sang and Jorn Veenstra. 1999. Representing text chunks. In *Proc. of EACL*.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proc. of EMNLP*.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. *arXiv preprint*.

Ang Sun, Ralph Grishman, and Satoshi Sekine. 2011. Semi-supervised relation extraction with large-scale word clustering. In *Proc. of NAACL-HLT*.

Changzhi Sun, Yuanbin Wu, Man Lan, Shiliang Sun, Wenting Wang, Kuang-Chih Lee, and Kewen Wu. 2018. Extracting entities and relations with joint minimum risk training. In *Proc. of EMNLP*.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proc. of ACL*.

Tung Tran and Ramakanth Kavuluru. 2019. Neural metric learning for fast end-to-end relation extraction. *arXiv preprint*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of NIPS*.

Patrick Verga, Emma Strubell, and Andrew McCallum. 2018. Simultaneously self-attending to all mentions for full-abstract biological relation extraction. In *Proc. of NAACL-HLT*.

Jesse Vig. 2019. A multiscale visualization of attention in the transformer model. In *Proc. of ACL*.

David Wadden, Ulme Wennberg, Yi Luan, and Hannaneh Hajishirzi. 2019. Entity, relation, and event extraction with contextualized span representations. In *Proc. of EMNLP/IJCNLP*.

Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. 2006. Ace 2005 multilingual training corpus. *Linguistic Data Consortium*.

Haoyu Wang, Ming Tan, Mo Yu, Shiyu Chang, Dakuo Wang, Kun Xu, Xiaoxiao Guo, and Saloni Potdar. 2019. Extracting multiple-relations in one-pass with pre-trained transformers. In *Proc. of ACL*.

Ralph Weischedel, Eduard Hovy, Mitchell Marcus, Martha Palmer, Robert Belvin, Sameer Pradhan, Lance Ramshaw, and Nianwen Xue. 2011. Ontonotes: A large training corpus for enhanced processing. *Handbook of Natural Language Processing and Machine Translation*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint*.

Yuan Yao, Deming Ye, Peng Li, Xu Han, Yankai Lin, Zhenghao Liu, Zhiyuan Liu, Lixin Huang, Jie Zhou, and Maosong Sun. 2019. Docred: A large-scale document-level relation extraction dataset. In *Proc. of ACL*.

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. 2002. Kernel methods for relation extraction. In *Proc. of EMNLP*.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. 2014. Relation classification via convolutional deep neural network. In *Proc. of COLING*.

Dongxu Zhang and Dong Wang. 2015. Relation classification via recurrent neural network. *arXiv preprint*.

Meishan Zhang, Yue Zhang, and Guohong Fu. 2017. End-to-end neural relation extraction with global optimization. In *Proc. of EMNLP*.

Shubin Zhao and Ralph Grishman. 2005. Extracting relations with integrated information using kernel methods. In *Proc. of ACL*.

A MD-RNN

In this section we present the detailed implementation of MD-RNN with GRU.

Formally, at the time-step layer l, row i, and column j, with the input $X_{l,i,j}$, the cell at layer l, row i and column j calculates the gates as follows:

$$T^{prev}_{l,i,j} = [T_{l-1,i,j}; T_{l,i-1,j}; T_{l,i,j-1}], \in \mathbb{R}^{3H} \tag{18}$$

$$r_{l,i,j} = \sigma([X_{l,i,j}; T^{prev}_{l,i,j}]W^r + b^r)), \in \mathbb{R}^H \tag{19}$$

$$z_{l,i,j} = \sigma([X_{l,i,j}; T^{prev}_{l,i,j}]W^z + b^z)), \in \mathbb{R}^H \tag{20}$$

$$\tilde{\lambda}_{l,i,j,m} = [X_{l,i,j}; T^{prev}_{l,i,j}]W^\lambda_m + b^\lambda_m, \in \mathbb{R}^H \tag{21}$$

$$\lambda_{l,i,j,0}, \lambda_{l,i,j,1}, \lambda_{l,i,j,2} = \mathrm{softmax}(\tilde{\lambda}_{l,i,j,0}, \tilde{\lambda}_{l,i,j,1}, \tilde{\lambda}_{l,i,j,2}) \tag{22}$$

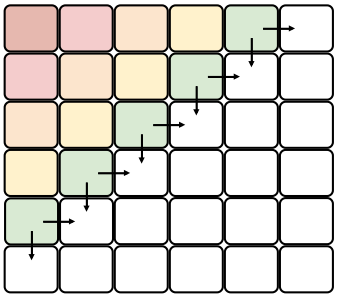


Figure 8: For 2D-RNNs, cells in the same color can be computed in parallel.

And then calculate the hidden states:

$$\tilde{T}_{l,i,j} = \tanh(X_{l,i,j}W^x + r_{l,i,j} \odot (T^{prev}_{l,i,j}W^p) + b^h), \in \mathbb{R}^H \tag{23}$$

$$\tilde{T}^{prev}_{l,i,j} = \lambda_{l,i,j,0} \odot T_{l-1,i,j} + \lambda_{l,i,j,1} \odot T_{l,i-1,j} + \lambda_{l,i,j,2} \odot T_{l,i,j-1}, \in \mathbb{R}^H \tag{24}$$

$$T_{l,i,j} = z_{l,i,j} \odot \tilde{T}_{l,i,j} + (1 - z_{l,i,j}) \odot \tilde{T}^{prev}_{l,i,j}, \in \mathbb{R}^H \tag{25}$$

where W and b are trainable parameters and please note that they share parameters in different rows and columns but not necessarily in different layers. Besides, $\odot$ is the element-wise product, and σ is the sigmoid function.

As in GRU, r is the *reset* gate controlling whether to forget previous hidden states, and z is the *update* gate, selecting whether the hidden states are to be updated with new hidden states. In addition, we employ a *lambda* gate λ, which is used to weight the predecessor cells before passing them through the *update* gate.

There are two slightly different ways to compute the candidate activation $\tilde{T}_{l,i,j}$, namely

$$\tilde{T}_{l,i,j} = \tanh(X_{l,i,j}W^x + r_{l,i,j} \odot (T^{prev}_{l,i,j}W^p) + b^h_l) \tag{26}$$

and

$$\tilde{T}_{l,i,j} = \tanh(W^x_l X_{l,i,j} + (r_{l,i,j} \odot T^{prev}_{l,i,j})W^p + b^h_l) \tag{27}$$

And we found in our preliminary experiments that both of them performed as well as each other, and we choose the former, which saves some computation.

The time complexity of the naive implementation (i.e., two for-loops in each layer) is $O(L \times N \times$

	# sentences	# entities (types)	# relations (types)
ACE04	8.7k	22.5k (7)	4.0k (6)
ACE05	14.5k	38.3k (7)	7.1k (6)
CoNLL04	1.4k	5.3k (4)	2.0k (5)
ADE	4.2k	10.5k (2)	6.6k (1)

Table 6: Dataset statistics

N) for a sentence with length N and the number of encoding layer L. However, antidiagonal entries can be calculated at the same time because their values do not depend on each other, shown in the same color in Figure 8. Therefore, we can optimize it through parallelization and reduce the effective time complexity to $O(L \times N)$.

B Data

Table 6 shows the dataset statistics after pre-processing. We keep the same pre-processing and evaluation standards used by most previous works.

The ACE04 and ACE05 corpora are collected from a variety of domains, such as newswire and online forums. We use the same entity and relation types, data splits, and pre-processing as Li and Ji (2014) and Miwa and Bansal (2016)[7]. Specifically, they use head spans for entities but not use the full mention boundary.

The CoNLL04 dataset provides entity and relation labels. We use the same train-test split as Gupta et al. (2016)[8], and we use the same 20% train set as development set as Eberts and Ulges (2019)[9]. Both micro and macro average F1 are used in previous work, so we will specify this while comparing with other systems.

The ADE dataset is constructed from medical reports that describe the adverse effects arising from drug use. It contains a single relation type "Adverse-Effect" and the two entity types "Adverse-Effect" and "Drug". Similar to previous work, we filter out instances containing overlapping entities, only accounting for 2.8% of total.

Following prior work, we perform 5-fold cross-validation for ACE04 and 10-fold for ADE. Besides, we use 15% of the training set as the development set. We report the average score of 5 runs for every dataset. For each run, we use the model that achieves the best performance (averaged entity metric score and relation metric score) on the development set, and evaluate and report its score on the test set.

[7]We use the prepocess script provided by Luan et al. (2019): https://github.com/luanyi/DyGIE/tree/master/preprocessing

[8]https://github.com/pgcool/TF-MTRNN/tree/master/data/CoNLL04

[9]http://lavis.cs.hs-rm.de/storage/spert/public/datasets/conll04/

Setting	Value
batch size	24
optimizer	Adam
learning rate (lr)	1e-3
warm-up steps	1000
dropout rate	0.5
# layers (L)	3
# attention heads (A)	8
hidden dim (H)	200
token emb dim	100
char emb dim	30
gradient clipping	5.0

Table 7: Hyperparameters used in our experiments.

C Hyperparameters and Pre-trained Language Models

The detailed hyperparameters are present in Table 7. For the word embeddings, we use 100-dimensional GloVe word embeddings trained on 6B tokens[10] as initialization. We disable updating the word embeddings during training. We set the hidden size to 200, and since we use bidirectional MD-RNNs, the hidden size for each MD-RNN is 100. We use inverse time learning rate decay: $\hat{lr} = lr/(1 + \text{decay\_rate} \times \text{steps}/\text{decay\_steps})$, with decay rate 0.05 and decay steps 1000.

Besides, the tested pre-trained language models are shown as follows:

- **[ELMo]** (Peters et al., 2018): Character-based pre-trained language model. We use the `large` checkpoint, with embeddings of dimension 3072.
- **[BERT]** (Devlin et al., 2019): Pre-trained Transformer. We use the `bert-large-uncased` checkpoint, with embeddings of dimension 1024 and attention weight feature of dimension 384 (24 layers $\times$ 16 heads).
- **[RoBERTa]** (Liu et al., 2019): Pre-trained Transformer. We use the `roberta-large` checkpoint, with embeddings of dimension

[10]https://nlp.stanford.edu/projects/glove/

Setting	NER	RE
MD-RNN		
layer$^+$~~row~~ ~~col~~	89.3	63.9
layer$^+$row$^+$col$^+$	89.6	66.9
layer$^+$row$^+$col$^-$	89.4	66.3
layer$^+$row$^-$col$^-$	89.6	66.9
layer$^+$row$^-$col$^+$	89.4	66.7
layer$^+$row$^+$~~col~~ ; layer$^+$row$^-$~~col~~	89.5	67.2
layer$^+$~~row~~ col$^+$; layer$^+$~~row~~ col$^-$	89.3	67.4
~~layer~~ row$^+$col$^+$; ~~layer~~ row$^-$col$^-$	89.3	66.7
layer$^+$row$^+$col$^+$; layer$^+$row$^-$col$^-$	89.5	67.6
layer$^+$row$^+$col$^-$; layer$^+$row$^-$col$^+$	89.7	67.4
layer$^+$row$^+$col$^+$; layer$^+$row$^-$col$^-$; layer$^+$row$^+$col$^-$; layer$^+$row$^-$col$^+$	89.7	67.6
CNN		
kernel size 1×1	89.3	64.7
kernel size 3×3	89.3	66.2
kernel size 5×5	89.3	65.8

Table 8: Comparisons with different methods to learn the table representation. For MD-RNN, D^+, D^- and ~~D~~ are indicators representing the direction, in which the hidden state flows forward, backward, or unable to flow at dimension D (D could be layer, row, or col). When using multiple MD-RNNs, we separate the indicators by ";".

1024 and attention weight feature of dimension 384 (24 layers $\times$ 16 heads).

- **[ALBERT]** (Lan et al., 2019): A lite version of BERT with shared layer parameters. We use the `albert-xxlarge-v1` checkpoint, with embeddings of dimension 4096 and attention weight feature of dimension 768 (12 layers $\times$ 64 heads). **We by default use this pre-trained model.**

We use the implementation provided by Wolf et al. (2019)[11] and Akbik et al. (2019)[12] to generate contextualized embeddings and attention weights. Specifically, we generate the contextualized word embedding by averaging all sub-word embeddings in the last four layers; we generate the attention weight feature (if available) by summing all sub-word attention weights for each word, which are then concatenated for all layers and all heads. Both of them are fixed without fine-tuning.

D Ways to Leverage the Table Context

Table 8 presents the comparisons of different ways to learn the table representation.

[11] https://github.com/huggingface/Transformers

[12] https://github.com/flairNLP/flair

Importance of context Setting "layer$^+$~~row~~ ~~col~~" does not exploit the table context when learning the table representation, instead, only layer-wise operations are used. As a result, it performs much worse than the ones exploiting the context, confirming the importance to leverage the context information.

Context along row and column Neighbors along both the *row* and *column* dimensions are important. setting "layer$^+$row$^+$~~col~~ ; layer$^+$row$^-$~~col~~" and "layer$^+$~~row~~ col$^+$; layer$^+$~~row~~ col$^-$" remove the *row* and *column* dimensions respectively, and their performance is though better than "layer$^+$~~row~~ ~~col~~", but worse than setting "layer$^+$row$^+$col$^+$; layer$^+$row$^-$col$^-$".

Multiple dimensions Since in setting "layer$^+$row$^+$col$^+$", the cell at row i and column j only knows the information before the i-th and j-th word, causing worse performance than bidirectional ("layer$^+$row$^+$col$^+$; layer$^+$row$^-$col$^-$" and "layer$^+$row$^+$col$^-$; layer$^+$row$^-$col$^+$") and quad-directional ("layer$^+$row$^+$col$^+$; layer$^+$row$^-$col$^-$; layer$^+$row$^+$col$^-$; layer$^+$row$^-$col$^+$") settings. Besides, the quaddirectional model does not show superior performance than bidirectional ones, so we use the latter by default.

Layer dimension Different from the *row* and *column* dimensions, the *layer* dimension does not carry more sentential context information. Instead, it carries the information from previous layers, so the model can reason high-level relations based on low-level dependencies captured by predecessor layers, which may help recognize syntactically and semantically complex relations. Moreover, recurring along the *layer* dimension can also be viewed as a layer-wise short-cut, serving similarly to high way (Srivastava et al., 2015) and residual connection (He et al., 2016) and making it possible for the networks to be very deep. By removing it (results under "~~layer~~ row$^+$col$^+$; ~~layer~~ row$^-$col$^-$"), the performance is harmed.

Other network Our model architecture can be adapted to other table encoders. We try CNN to encode the table representation. For each layer l, given inputs $\boldsymbol{X}_l$, we have:

$$\boldsymbol{T}_l^0 = \mathrm{ReLU}(\mathrm{Linear}([\boldsymbol{X}_l; \boldsymbol{T}_{l-1}])) \quad (28)$$

$$\boldsymbol{T}_l^1 = \mathrm{ReLU}(\mathrm{LayerNorm}(\mathrm{CNN}(\boldsymbol{T}_l^0))) \quad (29)$$

$$\boldsymbol{T}_l = \mathrm{ReLU}(\boldsymbol{T}_{l-1} + \mathrm{LayerNorm}(\mathrm{CNN}(\boldsymbol{T}_l^1))) \quad (30)$$

We also try different kernel sizes for CNN. How-

entire table?	entire entity?	directed relation tag?	NER	RE
✗(L)	✗	✓	89.2	65.9
✗(U)	✗	✓	89.2	65.8
✓	✗	✗	89.4	65.1
✓	✓	✗	89.3	65.8
✓	✗	✓	89.6	67.1
✓	✓	✓	89.5	67.6

Table 9: Comparisons of different table filling formulations. When not filling the entire table, L only fills the lower-triangular part, and U fills the upper-triangular part.

ever, despite its advantages in training time, its performance is worse than the MD-RNN based ones.

E Table Filling Formulations

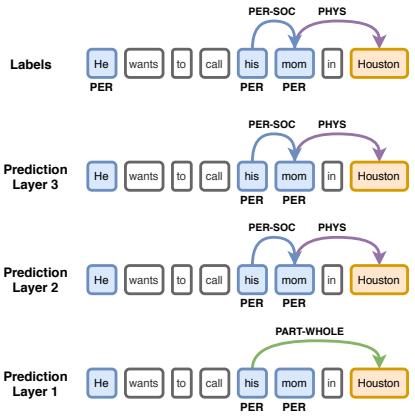
Our table filling formulation does not exactly follow Miwa and Sasaki (2014). Specifically, we fill the entire table instead of only the lower (or higger) triangular part, and we assign relation tags to cells where entity spans intersect instead of where last words intersect. To maintain the ratio of positive instances to negative instances, although the entire table can express directed relations by undirected tags, we still keep the directed relation tags. I.e, if $y^{\text{RE}}_{i,j} = \overrightarrow{r}$ then $y^{\text{RE}}_{j,i} = \overleftarrow{r}$, and vice versa. Table 9 ablates our formulation (last row), and compares it with the original one (Miwa and Sasaki, 2014) (first row).

F Probing Intermediate States

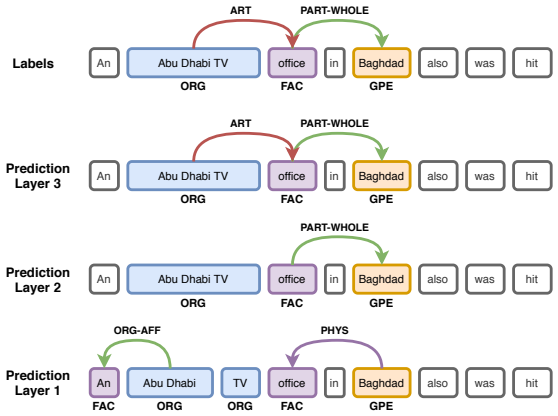
Figure 9 presents examples picked from the development set of ACE05. The prediction layer (a linear layer) after training is used as a probe to display the intermediate state of the model, so we can interpret how the model improves both representations from stacking multiple layers and thus from the bidirectional interaction.

Such probing is valid since for the table encoder, the encoding spaces of different cells are consistent as they are connected through gate mechanism, including cells in different encoding layers; for the sequence encoder, we used residual connection so the encoding spaces of the inputs and outputs are consistent. Therefore, they are all compatible with the prediction layer. Empirically, the intermediate layers did give valid predictions, although they are not directly trained for prediction.

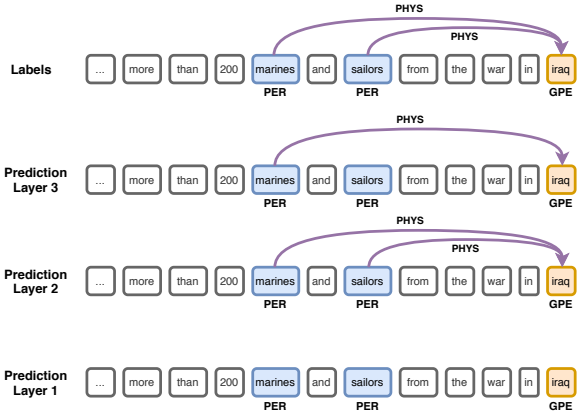
In Figure 9a, the model made a wrong prediction



(a) Correct the prediction at the 2nd layer



(b) Correct the prediction at the 3rd layer



(c) A mistake at the last layer

Figure 9: Comparisons of predictions by different encoding layers. We predict relations and entities with the intermediate sequence and table representation, so that we can figure out how the model improves both representations by stacking multiple encoding layers.

with the representation learned by the first encoding layer. But after the second encoding layer, this mistake has been corrected by the model. This is also the case that happens most frequently, indicating that two encoding layers are already good enough for most situations. For some more compli-

cated cases, the model needs three encoding layers to determine the final decision, shown in Figure 9b. Nevertheless, more layers do not always push the prediction towards the correct direction, and Figure 9c shows a negative example, where the model made a correct prediction in the second encoding layer, but in the end it decided not to output one relation, resulting in a false-negative error. But we must note that such errors rarely occur, and the more common errors are that entities or relationships are not properly captured at all encoding layers.