An Empirical Study of Pipeline vs. Joint Approaches to Entity and Relation Extraction

Zhaohui Yan*, Zixia Jia* and Kewei Tu
School of Information Science and Technology, ShanghaiTech University
Shanghai Engineering Research Center of Intelligent Vision and Imaging
{yanzhh, jiazx, tukw}@shanghaitech.edu.cn

Abstract

The Entity and Relation Extraction (ERE) task includes two basic sub-tasks: Named Entity Recognition and Relation Extraction. In the last several years, much work focused on joint approaches for the common perception that the pipeline approach suffers from the error propagation problem. Recent work reconsidered the pipeline scheme and shows that it can produce comparable results. To systematically study the pros and cons of these two schemes, we design and test eight pipeline and joint approaches to the ERE task. We find that with the same span representation methods, the best joint approach still outperforms the best pipeline model, but improperly designed joint approaches may have poor performance. We hope our work could shed some light on the pipeline-vs-joint debate of the ERE task and inspire further research.

1 Introduction

The Entity and Relation Extraction (ERE) task aims to extract entities and their relations from unstructured text and is a fundamental task in the area of information extraction. There are two typical approaches to the ERE task: one is the pipeline approach (Chan and Roth, 2011) consisting of two models for the two sub-tasks, Named Entity Recognition (NER) and Relation Extraction (RE), respectively. Another is the joint approach that models the two sub-tasks jointly (Miwa and Sasaki, 2014; Zheng et al., 2017; Wang and Lu, 2020; Eberts and Ulges, 2020).

Pipeline approaches do not share any parameters between sub-tasks and decode sequentially. For joint approaches, one typical method is to share encoders across sub-tasks and performs pipelined decoding (Miwa and Bansal, 2016). Another method uses joint inference in addition to shared encoders, for example, Wang and Lu (2020) cast the ERE task into a table-filling problem. Among these joint approaches, some span-based joint approaches (Sun et al., 2019) have different task-dividing strategy. Span-based models embed each span for an input sentence and there are $O(n^3)$ possible span pairs. To reduce the high complexity, previous span-based joint approaches pre-identify entity spans and then use a cross-task module for the entity and relation type deduction. To clarify the definitions, we define a purely joint approach as a method with only a cross-task module for sub-tasks, a purely pipelined approach has no cross-task module.

It is generally believed that the pipeline approach suffers from the problem of error propagation, while the joint approach could leverage interactions between sub-tasks. However, recent research from Zhong and Chen (2021) shows that the feature confusion problem of the joint model may negate its benefits. There is also some work (Yan et al., 2021) that disagree with their conclusion and propose a new state-of-the-art approach. However, these studies are based on different settings and hence cannot be directly compared.

The debate on pipeline vs. joint approaches motivates us to perform a systematically empirical study. For a fair comparison, we design pipeline and joint approaches with similar settings. The pipeline works recently (Zhong and Chen, 2021; Ye et al., 2022) use span-based models to better leverage span-level features, we also adopt this setting. For previous span-based joint approaches (Sun et al., 2019) divide the NER task into two sub-tasks, this leads to a second level of pipeline vs. joint dilemma which we also wish to investigate. Specifically, we consider four sub-tasks for ERE: entity identification (Eid), entity classification (Ecls), relation identification (Rid) and relation classification (Rcls). We design ten modules for these sub-tasks and connect them to build eight approaches.
 approaches as shown in Table 1. Following the recent work (Anonymous, 2022), we use high-order inference to better exploit the correlation between sub-tasks. To experiment with the full joint span-based approach, we use span pruner in addition to an entity pre-identifier for our joint approaches. With the high recall of the span pruner, our last approach a8 can be viewed as a full joint approach.

Our empirical study shows that with the same embedding method, the pipeline approach could achieve competitive results even compared to some joint approaches, but the full joint approach still outperforms all the pipeline approaches.

2 Our methods

As mentioned above, there are ten modules solving different sub-task combinations and eight approaches. The former four approaches are fully pipelined, the final one is the fully joint approach and the others are approaches with cross-task modules. We first introduce all the modules and then describe the training and decoding processes of the approaches.

We denote an input sequence with n tokens as \( X = \{x_1, x_2, ..., x_n\} \), \( m \) candidate spans of these tokens can be denoted as \( S = \{s_i | 1 \leq i \leq m\} \). \( \text{START}(i) \) and \( \text{END}(i) \) represent the head and tail token indices of \( s_i \). The gold entity label set and the gold relation label set are represented as \( \mathcal{E} \) and \( \mathcal{R} \) respectively.

2.1 Encoding

For each module, we feed the token sequence into a pre-trained language model. For each token \( x_i \), we use the embedding of the first sub-token from the last layer as the contextualized representations \( x_i \).

Follow Zhong and Chen (2021), for a given span \( s_i \in S \), the span representation \( b_i \) is defined as:

\[
b_i = [x_{\text{START}(i)}; x_{\text{END}(i)}; \phi(s_i)]
\]

where \( \phi(s_i) \in R^{d_i} \) is a learned embedding of the span length. For each module, we feed the span representations into a two-layer MLP to get an \( R^{d_i} \)-dimension hidden vector \( h_i \) and for modules involving RE, we obtain span pair representations in a similar way:

\[
h_i = \text{MLP}_{\text{span}}(b_i), \quad h_{ij} = \text{MLP}_{\text{rel}}([b_i; b_j])
\]

2.2 Single-task modules

For modules 1-6, the span or span pair representations are fed into a linear layer to score the span or span pair for each label.

\[
g_i = \text{Linear}_{\text{ent}}(h_i), \quad g_{ij} = \text{Linear}_{\text{rel}}(h_{ij})
\]

\( g_i \in R^{d_{\text{ent}}} \) and \( g_{ij} \in R^{d_{\text{rel}}} \). For classification modules \( \text{Ec} \) and \( \text{Re} \), \( d_{\text{ent}} = |\mathcal{E}| \) and \( d_{\text{rel}} = |\mathcal{R}| \). For module \( \text{Ner} \) involving both entity identification and classification, we add a \( \text{Null} \) label representing that the span is not an entity, so we have \( d_{\text{ent}} = |\mathcal{E}| + 1 \). Similarly, we have \( d_{\text{rel}} = |\mathcal{R}| + 1 \) for module \( \text{Re} \). For the identification modules \( \text{Ei} \) and \( \text{Ri} \), we set \( d_{\text{ent}} = d_{\text{rel}} = 1 \), meaning that we only score the existence of an entity or relation and fix the non-existence score to zero. The prediction of a span or a span pair is the label with the largest score among the gold label set, or we identify the span or relation with a score larger than 0.

2.3 Cross-task modules

Our cross-task modules adopt high-order inference (Jia et al., 2022). There are two types of scores in the modules: unary scores and ternary scores.

The unary score of a span or a span pair captures the prior distribution information and is computed solely based on the feature of the variable. The unary score \( g_i \) for the span or \( g_{ij} \) for the span pair is the same as defined in single-task modules.

The ternary score is defined cover a span pair that spans or relates \( s_i \) and \( s_j \). For each span pair \( (s_i, s_j) \), we calculate a score tensor \( f_{ij} \in R^{d_{\text{ent}}} \times d_{\text{rel}} \) as follows. First, two separate linear transformations project the head and tail span representations into \( d_{t} \)-dimension hidden space:

\[
h_i^t = \text{Linear}_{\text{head}}^t(b_i), \quad h_j^t = \text{Linear}_{\text{tail}}^t(b_j)
\]
Then a weight tensor $W_t \in \mathbb{R}^{d_h \times (d_{ent})^2 (d_{rel})}$ is used to transform the element-wise product $h_i^j \odot h_j^i$ into the score tensor $f_{ij}$:

$$f_{ij} = (h_i^j \odot h_j^i)W_t$$

**High-order Inference** The first-order inference is based solely on the unary score and the high-order inference is based on both the unary score and the tenary score. We follow Anonymous (2022) and the Mean-field Variational Inference (MFVI) for high-order inference which iteratively updates a factorized variational distribution $Q$ to approximate the posterior label distribution. Specifically, $Q_i(e)$ represents the probability of span $s_i$ having entity type $e$ and $Q_{ij}(r)$ represents the probability of spans $s_i, s_j$ having a relation of type $r$. For simplicity, we use $g_i(a, g_j(b), g_j(r)), f_{ij}(a, b, r)$ to represent the unary and tenary scores of spans $s_i, s_j$ having entity types $a, b$ and a relation of type $r$ between them. Messages delivered for entity and relation types are updated as follows:

$$F_i^T(a) = \sum_j \sum_b \sum_r Q_{ij}^{T-1}(b) \sum_r (Q_{ij}^{T-1}(r) f_{ij}(a, b, r) + Q_{ij}^{T-1}(r) f_{ij}(b, a, r))$$

$$F_{ij}^T(r) = \sum_{e_i, e_j} Q_{ij}^{T-1}(e_i) Q_{ij}^{T-1}(e_j) f_{ij}(e_i, e_j, r)$$

The messages are then used to update the posterior distributions $Q$:

$$Q_i^T(e) \propto \exp(g_i(e) + F_i^T(e))$$

$$Q_{ij}^T(r) \propto \exp(g_j(r) + F_{ij}^T(r))$$

With the distribution $Q$, we choose the label with the highest probability. For EcRi and NerRi, the $Q_{ij} > 0.5$ represents that the relation exists between the span pair $(s_i, s_j)$.

**2.4 Training and decoding**

**Training** With the modules defined above, we build eight approaches as shown in Table 1. a8 is an end-to-end joint model consisting of only NerRe and all the other approaches are pipelines of two or more modules. We train different modules in an approach independently without sharing any parameters. We train module $Ei$ and Ner on all possible O(nL) spans with a span length limit $L$. For a7 and a8, we cannot train the cross-task modules NerRi and NerRe on all spans for the high complexity $O(n^2 L^2 | \mathcal{E}|^2 | \mathcal{R}|)$, so we use a pre-trained pruner (see Appendix A for details) which identifies $O(n)$ most likely spans for both approaches and reduce the computational complexity to $O(n^2 | \mathcal{E}|^2 | \mathcal{R}|)$. For the downstream modules in a1-a7, we train them on the gold entity set or the span pair set built by enumerating all the spans $s_i, s_j$ in the gold entity set following Zhong and Chen (2021). For example, with the span set $S = \{s_1, s_2, ..., s_m\}$, we build the span pair set $\{(s_1, s_2), (s_1, s_3), ..., (s_1, s_{i+1}), ..., (s_i, s_m)\}$. There are two loss functions for these modules:

$$L_{ent} = - \sum_{s_i \in S} \log P_i(e_i^*) \quad L_{rel} = - \sum_{s_i, s_j \in S, i \neq j} \log P_{ij}(r_{ij}^*)$$

$e_i^*$ and $r_{ij}^*$ are the gold labels for span $s_i$ and span pair $(s_i, s_j)$ respectively. For cross-task modules, we have $P_i(e_i) = Q_i(e_i)$ and $P_{ij}(r_{ij}) = Q_{ij}(r_{ij})$; for the other modules, we have $P_i(e_i) = \text{Softmax}(g_i(e_i))$ and $P_{ij}(r_{ij}) = \text{Softmax}(g_{ij}(r_{ij}))$. The training objective of a module is to minimize $L_{ent} + L_{rel}$ where the $L_{ent}$, $L_{rel}$ indicate whether the module predicts entities and relations respectively. High-order inference with the MFVI is end-to-end differentiable.

**Decoding** For the pipeline approaches a1-a7, the decoding is a cascade process. The upstream module is decoded first and each downstream module builds the input using the output of the upstream module.

**3 Experiments**

**3.1 Experimental settings**

**Datasets** We experiment on two popular relation extraction datasets: ACE2005 (Christopher Walker and Maeda, 2006) and SciERC (Luan et al., 2018). We adopt the official training/validation/testing splits.

**Evaluations** We follow previous works and use the F1 scores with micro-averaging as the evaluation metric.

Specifically, for the NER task, a predicted entity is considered correctly identified (Ent-I) if its boundary matches the corresponding gold entity and correctly classified (Ent-C) if its type also matches. For RE tasks, the predicted relation is correctly identified (Rel-I) if the boundaries of its endpoints are correct and correctly classified (Rel-C) if the relation type matches the corresponding gold relation. To evaluate both tasks, the strict evaluation (Rel+ -I and Rel+ -C) requires correctly
The main results of the eight approaches are shown in Table 3. The results of purely pipeline approaches show that \text{a4} is the best pipeline approach for almost all classification evaluations. We can conclude that dividing the NER or RE task into pipelines does not help the entire ERE task. From the results of \text{a2}, \text{a3} and \text{a4}, we could find out that the dividing of RE (\text{a3} vs. \text{a4}) leads to a larger performance drop than the dividing of NER (\text{a2} vs. \text{a4}). To exclude the effect of error propagation from NER task, we do extra experiments with gold entities for \text{Ri-Rc} and \text{Re}. The results are shown in Table 3 and \text{Ri-Rc} has a large performance drop with \text{Re}. Dividing RE task brings a negative effect to the approaches. We guess because the identification and classification are highly correlated sub-tasks, if they are both difficult, then solving them jointly in one module can promote the performance of both. The entity sub-tasks are not so difficult, especially on ACE2005, so the improvement of \text{a3} or \text{a4} over \text{a1} or \text{a2} on Ent-C is not significant.

### The results of approaches with cross-task

We compare the results of all the joint approaches: \text{a5} to \text{a8}. We observe that \text{a8} is better than the other three on almost all the evaluations except for Ent-C on ACE2005. We first compare \text{EcRi} and \text{NerRi} to \text{EcRe} and \text{NerRe}. The Ent-C results of \text{a5} are higher than those of \text{a6} on both datasets and \text{a7} is better than \text{a8} on ACE2005, but for the results of Rel-I and Rel+I, \text{a8} outperforms \text{a7} and \text{a6} outperforms \text{a5} on most evaluations. We can conclude that \text{EcRe} and \text{NerRe} are better than \text{EcRi} and \text{NerRi}. We guess it is the reason that the entity labels of a span pair have a stronger correlation with their relation label than with the existence of their relation. Then from the results of \text{a5} vs. \text{a7} and \text{a6} vs. \text{a8}, we wish to investigate the effect of a separate entity identifier. For \text{a5} and \text{a7}, we cannot clearly judge which is better, but for \text{a6} and \text{a8}, \text{a8} significantly outperforms \text{a6} on most evaluations which shows that the separate \text{Ei} hurt the performance of the cross-task module for the error propagated to the downstream modules. \text{Ei} has much lower performance on SciERC than on ACE2005, so we guess it brings more performance drop on SciERC than on ACE2005. Comparing the results of \text{a1} vs. \text{a5}, \text{a2} vs. \text{a6}, and we can see that the \text{EcRi} and \text{EcRe} modules have lower the evaluation results of Ent-C and Rel-I and the performance gap between \text{a1} and \text{a5}, \text{a2} and \text{a6} of Rel+ on SciERC is also more than on ACE2005. This gives us the insight that the cross-task module
may be more sensitive to input error than the single-task module at least for our high-order inference.

The comparison of pipeline and joint approaches Comparing all the pipeline and joint approaches, the common pipeline structure a4 is comparable to all the other approaches except for a8. In particular, a4 outperforms a6, which has a similar structure to some previous joint models (Sun et al., 2019), on almost all evaluations. It shows that pipeline and joint approaches could have comparable performance with the same embedding. But even with the same embedding method, the fully joint approach a8 with the pruner has significantly better performance than a4 on Rel+.

3.3 Analysis

To further investigate the effect of the input error on the joint modules, we conduct extra experiments on the following approaches:

- **Ei-NerRe**: In this approach, we replace the pruner with a pre-trained Ei module in approach a8. We could treat the Ei module as an entity pruner with lower recall but much higher precision compared to the pruner we used (refer to Table 4). Meanwhile, as the NerRe module could identify the existence of entities, it could fix some input errors compared to EcRe in a6.

- **NerRe***: It is the NerRe module with no joint inference. NerRe* only shares encoders across NER and RE sub-tasks and it could be treated as a less complex joint module compared to NerRe.

From the results of Ei-EcRe vs. Ei-NerRe in Table 5, we observe that, as NerRe could reduce the impact of wrongly predicted entities, the latter approach has a slight but not significant advantage over the former on SciERC. When there are only a small amount of input errors, NerRe has a significant advantage over Ei-EcRe and Ei-NerRe on SciERC. Surprisingly, the result is different on ACE2005. NerRe and Ei-NerRe achieve comparable performance. This may come from the different recalls of Ei on the two datasets. According to Table 4, Ei has a much lower recall on the SciERC test dataset than on ACE2005 in comparison to the pruner.

For the same reason, we also see a similar phenomenon in the results of NerRe vs. NerRe* in Table 6. Replacing the pre-trained Ei with the pruner, we could find large performance improvement on the Rel+ metric for both NerRe and NerRe* on SciERC. On the other hand, the pruner does not show any advantage over Ei on ACE2005. Ei-NerRe performs much worse on SciERC than on ACE2005 in comparison to Ei-NerRe*, which also shows that the NerRe module is more sensitive to the input error than the NerRe* module.

4 Conclusion

In this paper, we empirically study several pipeline and joint approaches of the ERE task. We find that pipeline approaches could achieve quite competitive results with some joint approaches, but with span pruning and high-order inference, the full joint model could still outperform the pipeline approaches. We observe that if the tasks have strong correlations, a properly designed joint approach tends to have higher performance.
5 Acknowledgement

This work was supported by the National Natural Science Foundation of China (61976139).

References

Anonymous. 2022. High-order inference for entity recognition, relation extraction, and event extraction. In ACL.


A Pruner

Pruning strategy For a given token sequence \( X = \{x_1, x_2, ..., x_n\} \), the pruner scores the existence for each possible spans with the length limitation \( L \). We rank the spans by their scores and filter out top \( K \) as the candidate spans. Basically we filter out the spans according to a ratio to the length of the sentence. As the gold span number is not strict linear with the sentence length, there is an upper bound of the gold span number for each sentence. We set a upper limit \( m_u \) for candidate span number of each sentence and a lower limit \( m_l \) to avoid the zero candidate span for very short sentences. So the number of candidate spans of a sentence length \( n \) is \( K = \max(m_l, \min(m_u, \alpha \ast n)) \), where \( \alpha \) is the top-K ratio. For both ACE2005 and SciERC datasets, we take \( \alpha = 0.5 \), \( m_l = 3 \), \( m_u = 18 \).

Span representation and scoring The model first embed each token, then produces the span representations by the tokens inside the spans. The first sub-token embeddings from the last layer of a pre-trained language model is used as the contextualized representation \( x_i \) for each token \( x_i \).

We use two kinds of embedding layers: bi-affine and self-attention pooling for span encoding. For a span \( s_i \) with its tokens \( (x_{\text{START}(i)}, ..., x_{\text{END}(i)}) \), its bi-affine representation is a \( d_{\text{biaf}} \)-dimension vector:

\[
\mathbf{h}_b(s_i) = [x_{\text{START}(i)}; 1]^\top W_b [x_{\text{END}(i)}; 1]
\]

The self-attention pooling function use the span’s token representations as the keys and values, and a linear layer scores the keys to get the weight of the values.

\[
w_j \propto \text{Linear}_{\text{att}}(x_j)
\]

\[
\mathbf{h}_a(s_i) = \sum_{\text{START}(i) \leq j \leq \text{END}(i)} w_j x_j
\]

Then a two-layers MLP projects the concatenation of these representations into a \( d_{\text{att}} \)-dimension hidden space for the final span representation:

\[
\mathbf{h}(s_i) = \text{MLP}([\mathbf{h}_b(s_i); \mathbf{h}_a(s_i)])
\]

The span representation of \( s_i \) is feed into a linear layer to get the score \( g_i \):

\[
g_i = \text{Linear}(\mathbf{h}(s_i))
\]

\[
\text{Table 7: The performance of pruners used for a7 and a8}
\]

Training and evaluation We train the pruner as an identifier, the training loss is the binary cross-entropy:

\[
\text{Loss} = -\sum_i p_i \log(q_i) + (1 - p_i) \log(1 - q_i)
\]

\( p_i = 1 \) if the span \( s_i \) is an entity span otherwise \( p_i = 0 \) and \( q_i = \text{Sigmoid}(g_i) \).

For the evaluation, the pruner produces a candidate span set and calculates the f1 score. We choose the best model on dev sets.

B Hyper-parameters and Implementation Details

We tune the hidden size of MLP\(_{\text{span}}\) and MLP\(_{\text{rel}}\) among \([200, 300, 400]\) for each module. The learning rate is tuned among \([1e-5, 2e-5, 5e-5]\) and dropout rate is tuned among \([0.1, 0.2, 0.3]\).

\[
\text{Table 8: Summary of hyper-parameters}
\]