IBM Submission for TAC KBP:EDL 2019
Cascaded Fine-Grained Named Entity Recognition

Parul Awasthy and **Taesun Moon** and **Jian Ni** and **Radu Florian**
IBM Research AI
Yorktown Heights, NY 10598
{awasthyp, tsmoon, nij, raduf}@us.ibm.com

Abstract

In this paper we describe our submission to the TAC KBP: Entity Linking and Discovery 2019 task. The task aims at extracting fine-grained mentions of 190 types from text where no training data is available for these types. Our submissions are based on a two pass approach: first detect nine basic NER types, followed by a second pass of refining these types into the destination fine-grained types.

1 Introduction

Named entity recognition (NER) is an important task in information extraction and natural language processing. Its main goal is to identify, in unstructured text, contiguous typed references to real-world entities, such as persons, organizations, facilities, and locations. It is very useful as a precursor to identifying semantic relations between entities (to fill relational databases), and events (where the entities are the events arguments).

Traditional NER work has focused on coarse-grained entity types, e.g., 4 entity types in CoNLL'02 data (Tjong Kim Sang, 2002) and 7 entity types in ACE'05 data (Walker et al., 2006). However, many real-world applications (e.g., disaster relief, technical support, cybersecurity) require a wider variety of fine-grained entity types. Building fine-grained NER models with no or a limited amount of annotated data is the focus of this paper.

2 Prior Work

Named entity recognition (NER) is a subfield of NLP with a long, established history and a vast literature. For a good overview of the problem and the main corpora involved from a classical perspective, see Nadeau and Sekine (2007). Hammerton (2003) was one of the earliest attempts at using a neural network (specifically, an LSTM) for NER, though with performance marginally above baseline. Collobert et al. (2011) and Lample et al. (2016) were more successful and influential approaches to using neural networks for NER.

Pretrained word-embeddings called word2vec (Mikolov et al., 2013) proved critical in helping neural architectures achieve state-of-the-art results across a variety of tasks in NLP including NER (Lample et al., 2016). The development of contextual or context aware pretrained word-embeddings such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2018) pushed the SOTA frontier for virtually all NLP tasks even further. For example, merely using BERT with a final feed-forward layer and supervised fine-tuning achieved SOTA for a short-period in English CoNLL NER.

There is a wide and disparate literature for information extraction when supervised data is either restricted or unavailable and the type system is either large, unbounded or unspecified. Banko et al. introduces the idea of open information extraction, where relation triples are discovered from the general web without any labeled data. Cimiano and Völker (2005) is related in spirit to this study in that it attempts to classify named entities according to a large ontology with no training examples. Theirs is a fully unsupervised approach using a classical vector space model where they assign a label from the ontology to a named entity by measuring similarity between the label and named entities through a context vector computed from a large corpus. Evans and Street (2003) is a thoughtful attempt at the even harder problem of deriving a type system *de novo* from an unlabeled corpus by using a set of heuristics. Brambilla et al. (2017) attempts to detect emerging entities in social media.

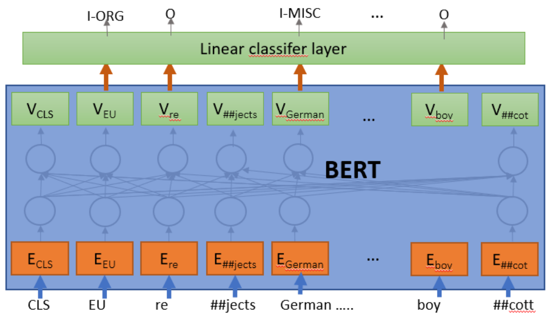

Figure 1: BERT Architecture

3 Large Space Mention Detection

In order to balance the large number of entity types (190) with the need for sufficient training data to perform sequential tagging for mention detection, we have decided to solve the problem with the following two steps:

- First, perform coarse-grained mention detection in the usual fashion, by converting it to an IOB sequential token prediction task (Tjong Kim Sang and Veenstra, 1999) – the type system we used here is composed of 9 entity types (CRM, FAC, GPE, LAW, LOC, ORG, PER, VEH, WEA).
- Second, classify each mention obtained in the previous step with a subtype.subsubtype label - an example-based classification task that takes the full sentence of the mention into account.

We will describe the salient characteristics of these two models.

3.1 Coarse-Grained Mention Detection

The coarse-grained mention detector is a BERT-based mention detector, as described below.

We approach NER in a standard fashion: a sequence labelling task which assigns a tag to each word based on its context. Given a sentence $\{w_1, w_2, ...w_n\}$, we feed it to the BERT model to obtain contextual BERT embeddings for each word as $\{v_1, v_2, ...v_n\}$, capturing each word's context via many attention heads in each of its layer. These embeddings are then fed to a linear feed forward layer to obtain labels $\{y_1, y_2, ...y_n\}$ corresponding to each word piece. The network architecture is shown in Figure 1. The entire network is trained with each epoch thereby fine-tuning the BERT embeddings for the NER task. We are using an IOB1 encoding of the entities (Tjong Kim Sang and Veenstra, 1999), as it performed best in preliminary results.

We use the HuggingFace PyTorch implementation of BERT (HuggingFace github, 2019) and the BERT WordPiece Tokenizer. We follow the recipe in (Devlin et al., 2018) for building named entity taggers: to convert the NER tags from tokens to word pieces, we assign the tag of the token to its first piece, then assign the special tag 'X' to all other pieces. No prediction is made for 'X' tokens during training and testing. Figure 1 shows both the architecture of the proposed model, and the NER annotation style.

We have tried many BERT architectures, including bert-base-uncased, bert-large-cased, bert-large-cased-whole-word-masking. Given the lack of actual training data provided specifically for this evaluation, we have investigated the previously released datasets – specifically ACE'05 and TAC'17. The TAC17 dataset was more recently released, but had fewer types (as it does not include VEHICLE and WEAPON), while at the same time matches more closely the annotation guidelines. In the end, after initial experimentation, we have decided to use an ACE'05 trained system to add silver mentions to the TAC17 dataset. In addition, we have also used an in-house mention detection system to add two other types - COM and LAW - by running the training and dev set through the SIRE classifier (citation) and retaining only mentions of the two types that do not overlap with any existing entities.

3.2 Fine-Grained Mention Detection

As not much gold data is available for the Fine-Grained mention detection task, we decided to approach this as a classification task where given a sentence and the mention boundary in the sentence, the task is to classify the sentence and the mention boundary with a fine-grained type. The architecture is showin in Figure 2

To specialize each mention into subtype.subsubtype label, we use the standard example based classification approach for the specialization task, which assigns a tag to each sentence based on its context, as described below:

Given a sentence in the form of $\{w_1, w_2, ...w_n\}$, with k mentions at $\{(w_{i_1} : w_{j_1}), (w_{i_2} : w_{j_2}), ..., (w_{i_k} : w_{j_k})\}$, where mention span $(w_{i_x} : w_{j_x})$ means mention begins at token position i_x and ends at j_x, a fine-grained (type.subtype.subsubtype) label y_x for each of these mentions and a coarse-grained label (type

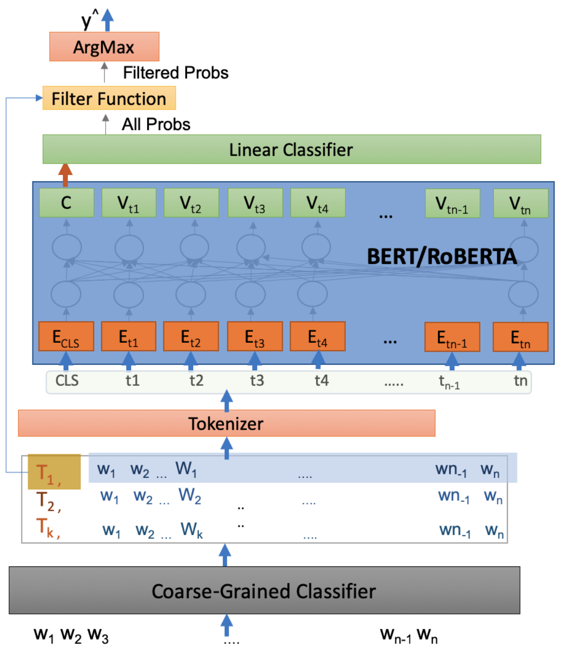

Figure 2: Fine-Grained Classifier Architecture

only) z_x for each of the mentions. We create k examples from the sentence with a special representation W_x for $w_{i_x} : w_{j_x}$ for the x_{th} example, thereby getting the below representations for the example:

$$\{w_1, w_2, ..w_{i_1-1}, W_1, w_{j_1+1}...w_n\}$$
$$\{w_1, w_2...w_{i_2-1}, W_2, w_{j_2+1}...w_n\}$$
$$...$$
$$\{w_1, w_2...w_{i_k-1}, W_k, w_{j_k+1}...w_n\}$$

These examples are then fed to a transformer-based model to obtain a contextual representation for each sentence. This representation is fed to a linear layer to get logits for each possible l fine-grained classes. At training time we compute a standard cross-entropy loss over all these probabilities. But at decoding time we do something different to restrict the output space of the probabilities as described below.

Let the $\{p_{i_0}, p_{i_1}....p_{i_l}\}$ be the representation of the probabilities produced by the classifier for example i where p_{i_k} is the probability for label k. Then we produce y_i as below:

$$y_i = \arg\max(F(\{p_{i_0}, p_{i_1}....p_{i_l}\}, z_i))$$

$F()$ is a function that given l probabilities and a coarse label z_i filters out some of the l probabilities and produces an array of probabilities with size less than or equal to l so that the possible output space is reduced. We play with different functions for $F()$ in our submissions. We also try different representations for W_i in our experiments. These variations are described below.

3.2.1 Mention Representations

Here we describe the representations we tried for the mention text (W).

Masking the mention: We replace the mention tokens in the sentence with a tokenizer specific mask token, and then add the mention token to the end of the sentence after a tokenizer specific separator.E.g.

Alice *was beginning to get very tired of sitting by her sister on the bank .*

<**MASK**> was beginning to get very tired of sitting by her sister on the bank . <SEP> **Alice**

Embedding with Coarse-Grained Type: We surround the mention tokens in the sentence with a special token, the coarse-grained mention type of the mention. To do this we add all the coarse grained types to the tokenizer vocabulary, so they are not split. E.g.

Alice *was beginning to get very tired of sitting by her sister on the bank .*

PER Alice PER was beginning to get very tired of sitting by her sister on the bank .

3.2.2 Filter Functions

To restrict the label space to the most likely, while decoding, not while training, we use a variety of filter functions to arrive at the best possible class. We describe the various filter function ($F()$) we tried here. The filter functions mainly use the coarse-grained label and the probabilities produced by the classifier to narrow down the output space of the labels. As mentioned earlier, for this task each class is represented as $type.subtype.subsubtype$. Let the input to this function be a set of probabilities $p_1, p_2, ...p_l$ where p_i is the probability of the i_{th} class, and a coarse-grained type t. The output is again a set of probabilities $\{p_x, p_y...p_z\}$ such that $|output| <= l$.

Dummy Filter Function: This function returns the same input it received with no filtering.

$$F(\{p_1, ...p_l\}, t) = \{p_1, ...p_l\}$$

Coarse-Grained Type based Filter Function This function returns probabilities of only those classes for which the type part of the class is same as coarse-grained type. All other probabilities are filtered out. E.g. if coarse-grain type for a mentions is PER then the output of F() would be only classes with PER type and no classes with other types line GPE, FAC,etc. irrespective of their probabilities.

$F(\{p_1, ...p_l\}, t) = \{p_x$ s.t. EType(x) $== t\}$

Threshold based Filter Function This function is similar to the above Coarse-Grained Type based filter with a small difference. It still returns probabilities of those classes for which the type part of the class is same as coarse-grained type, but it also returns probabilities of classes of other types, if the probability of that class is greater than the threshold.

$F(\{p_1, ...p_l\}, t, thresh) = \{p_x$ s.t. EType(x) $== t$ or $p_x >= thres\}$

We use Huggingface pytorch Transformers for the fine grained annotation task. We train the model on the community annotated data shared by RPI and we test on AIDA phase 1 Eval data. At training time we train on gold coarse-grained labels but at decode time we use the system coarse grained labels produced by the NER model.

4 Experiments

4.1 Data and Experiments

We use Huggingface pytorch Transformers to train both NER and Classification model (Wolf et al., 2019). For NER we experimented with various kinds of transformer architectures like RoBERTa and BERT. We also experiment with different models of these transformers like *bert-large-cased, bert-large-uncased,bert-large-cased,whole-word-masking, RoBERTa-large*.

For NER, we use the TAC data, with silver ACE types (VEH[ICLE],WEA[PON]) and silver in-house SIRE types (LAW,CRM). We train the NER models for 20 epochs.

For Classifier model we use the community annotated data shared by RPI for training and we develop on AIDA phase 1 Eval data. Later once, we got the feedback, we used the feedback data as the test.The RPI data contains labels for 122 classes, and our model produces only those classes. We train these models for 20 epochs with a variety of learning rates, and observe learning rate of 3e-5 does well usually.

Though, while developing the models we used a different test set, here for clarity we report numbers on the TAC KBP:EDL 2019 eval corpus that comprises of 404 documents.

4.2 Results

Model	P	R	F_1
BERT large_uncased	77.1	79.8	78.4
BERT large_cased_wwm	79.0	81.4	80.2

Table 1: Performance of various BERT models.

Type	Count	P	R	F_1
COM	5	0.00	0.00	0.00
FAC	322	53.63	52.80	53.21
GPE	6004	79.39	88.56	83.73
LAW	33	19.61	30.30	23.81
LOC	703	56.70	49.36	52.78
ORG	3256	64.78	66.55	65.66
PER	4546	94.74	91.49	93.08
SID	58	0.00	0.00	0.00
VAL	2	0.00	0.00	0.00
VEH	19	16.67	5.26	8.00

Table 2: Performance break-down by types.

Coarse-grained NER results obtained by running two BERT models - *bert_large_uncased* and *bert_large_cased_whole_word_masking* are shown in Tables 1 and 2 (for the *bert_large_cased_whole_word_masking* model). As you can see, the cased, whole word model BERT behaved better on the actual evaluation data; however, during our development runs, the large uncased model was more robust to different types of input (such as all-case words, etc), and we have decided to use that model for all evaluation runs.

The results for fine-grained mention detection are given in Table 3. These results are on the gold EDL 2019 data for coarse and fine-grained types. Though, the accuracy for bert-large-uncased model is higher on the EDL2019 eval set, we selected RoBERTa based model for our submission as it performed better on AIDA eval sets. The results in Table 3 use the Coarse-Grained-Type based filtering.

To see the benefits of the Threshold-based filtering, we need to decode the fine-grained model on system coarse grained mentions. These numbers are shown in Table 4. We show numbers for robert-large model that our submissions were based of 3 in this table. The table shows numbers on both coarse-grained models we had. At threshold 1, the filter is behaving like the Coarse-Grained filter, filtering out all probabilities that

Model	Mention Representation	Acc-SST	Acc-ST	Acc-T
BERT large uncased	Masked Mention	68.40	78.96	99.76
BERT large cased	Masked Mention	67.76	79.86	99.73
RoBERTa large	Masked Mention	70.06	78.22	99.75
BERT large uncased	Coarse-Type Boundary	71.09	80.37	99.85
BERT large cased	Coarse-Type Boundary	69.50	78.30	99.83
RoBERTa large*	Coarse-Type Boundary	69.58	78.90	99.84

Table 3: Performance of fine-grained models across different mention representations. Acc-SST is the accuracy at the full type, e.g. per-professionalpoision-minister, Acc-ST is the accuracy whenever the model gets the type and subtype right, e.g. per-professionalpoision. Acc-T is the accuracy whenever the model gets the type right. These models use Coarse-Grained-Type based Filtering. *Model selected for submission.

Coarse-G System	Filter Threshold	Acc-SST	Acc-ST	Acc-T
BERT large_uncased	1	64.93	73.05	88.31
BERT large_uncased	0.9	67.0	75.13	90.24
BERT large_uncased	0.8	67.01	75.14	90.21
BERT large_cased_wwm	1	64.40	72.72	88.26
BERT large_cased_wwm	0.9	66.35	74.72	90.12
BERT large_cased_wwm	0.8	66.38	74.76	90.12

Table 4: Performance of *roberta-large* model using different filter functions across different coarse-grained system models. *Model selected for submission.

belong to labels of types other than the course-grained type it is fed. It seen, when the Fine-Grained model is really sure of a label, with probability of 0.9 or 0.8, the performance improved. The improvement in the AIDA dev set between no filtering and filtering at 0.9 was about 3 f1 points.

Run	P	R	F_1
Run2	59.1	61.5	60.3
Run3	60.2	62.7	61.4

Table 5: Performance of various BERT models.

Finally we show the eval results for our three runs in Table 5. Both submissions used the bert-large-uncased coarse-grained NER model and roberta-large fine-grained classifier. Run 2 used coarse-grained-type filtering and run3 used threshold-based filtering with threshold set at 0.9. As the community annotated training data contains examples for only 122 types, our model can make predictions only for those types and always misses the other types.

We also investigated building ensemble systems, but the performance improvements we obtained with voting was minimal (less than .2F). The voting scheme is a good approach for hedging against bad models (models that overtrain on the development dataset), as the combined model will tend to be more robust, but the large data size (300,000 documents) that needed to be processed during the evaluation was a concern, so we decided to use the best performing model from the development set, instead of an ensemble classifier for the final evaluation.

5 Conclusion

We present in this paper a method of performing fine-grained named entity recognition in two stages: first perform coarse-grained NER using a BERT-based token classification model, followed by an instance-level fine-grained classification. The model is trained on English data reused from previous evaluations (TAC'17), augmented with labels from ACE'05 and an in-house dataset (KLUE) to add four types - vehicle, weapon, crime, and law. The coarse-grained model is based on a BERT (large_uncased) model, while the course-grained model is based on a RoBERTa (large_cased) model. This setup allows the system to pool data in the first step to predict the coarse-type, and making one classification decision per coarse mention in the second step.

References

Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI*.

Marco Brambilla, Stefano Ceri, Emanuele Della Valle, Riccardo Volonterio, and Felix Xavier Acero Salazar. 2017. Extracting emerging knowledge from social media. In *Proceedings of the 26th International Conference on World Wide Web*, pages 795–804. International World Wide Web Conferences Steering Committee.

Philipp Cimiano and Johanna Völker. 2005. Towards large-scale, open-domain and ontology-based named entity classification. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP)*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Richard Evans and Stafford Street. 2003. A framework for named entity recognition in the open domain. *Recent Advances in Natural Language Processing III: Selected Papers from RANLP*, 260(267-274):110.

HuggingFace github. 2019. The Big-&-Extending-Repository-of-Transformers: Pretrained PyTorch models for Google's BERT, OpenAI GPT & GPT-2, Google/CMU Transformer-XL. https://github.com/huggingface/pytorch-pretrained-BERT.

James Hammerton. 2003. Named entity recognition with long short-term memory. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 172–175. Association for Computational Linguistics.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

E. F. Tjong Kim Sang and J. Veenstra. 1999. Representing text chunks. In *Proceedings of EACL'99*.

Erik F. Tjong Kim Sang. 2002. Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Taipei, Taiwan.

Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. 2006. ACE 2005 multilingual training corpus. Philadelphia: Linguistic Data Consortium.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.