Learning Structured Representations of Entity Names using Active Learning and Weak Supervision

(Supplementary Materials)

Kun Qian, Poornima Chozhiyath Raman, Lucian Popa, Yunyao Li
IBM Research – Almaden
qian.kun@ibm.com, {pchozhi, lpopa, yunyaoli}@us.ibm.com

In this supplementary material, we provide more details about the datasets we used, our model implementation, and our experimental evaluation presented in the EMNLP 2020 paper: "Learning Structured Representations of Entity Names using Active Learning and Weak Supervision".

1 Video demo of the System

We included a short video demo in the uploaded supplementary materials, in which we showcase the system that implements the learning framework described in this paper. We believe active learning systems or human-in-the-loop systems should have intuitive graphic user interfaces so that they are truly usable, especially for the tasks that require non-trivial labeling effort from the user. We developed an end-to-end system that supports scenario creation, data uploading, active learning, normalization and variant generation, and many other user-friendly features. The system was implemented using purely open source packages (e.g., Angular 7, Django, pytorch, pytorch-transformer, typescript). The source code of the system will also be released after the paper is published.

We strongly encourage and sincerely hope the reviewers can watch the video demo (about 3 minutes) to better understand our proposed learning framework.

2 Datasets

We also include the four datasets used in our experiments. Each dataset contains a training set and a held-out test set, which we will release. The datasets are manually annotated (with the help of the tool showed in the video demo), and we adopted the 70%-30% splitting convention. Concretely, we first label all examples, and then 70% of them became the "unlabeled" data that is going to be provided to our system, and the rest 30% became a held-out test set.

We consider four different types of entities (datasets can be found in a zipped file in the uploaded supplementary materials. Table 1 lists some sample entity mentions for the four datasets and their expected structured representations. Note that there is an implicit component for all the data types: ⟨nonessential⟩, which basically means that the token in consideration is not an essential semantic component.

2.1 Dictionaries for LUSTRE

One of our baselines, LUSTRE, requires dictionaries as part of the input. In our uploaded zip file, it also lists the dictionaries we provided to LUSTRE.

3 Model Implementation

Table 2 lists the hyperparameters of the best-performing model that we reported in the main paper. As mentioned earlier, we used huggineFace pytorch-transformer (Wolf et al., 2019) to implement our BERT-CRF model. In particular, we used the pretrained `BertModel`, and we used the corresponding tokenizer and pretrained weights using the `bert-base-cased` configuration.

The Bert embedding size (i.e., 768) is predefined by the Bert model, and the size of predicates for structural vectors are predefined by us. The input-output dimension size of MLP-1 (bounds are determined by the embedding size of Bert and number of labels) and learning rate are determined using random sampling (using the DATE dataset).

4 Evaluation Metrics

We used three metrics in the main paper: *entity-level*, *token-level*, *component-level*. Here we give a concrete example for computing the three metrics. Consider a test set consisting of a single DATE entity string: **June 3rd, 2020**. Assume that we have

Type	Sample Mentions	Structured Representation
PER	*Prof Liat Sossove*	⟨title⟩⟨first⟩⟨last⟩
	Hagop Youssoufia, B.S.	⟨first⟩⟨last⟩,⟨degree⟩
	ElmeDxsna Adzemovi Sr.	⟨first⟩⟨last⟩⟨suffix⟩
ORG	*SONY CORP.*	⟨corename⟩⟨suffix⟩
	JONES APPAREL GROUP INC	⟨corename⟩⟨corename⟩⟨type⟩⟨suffix⟩
	STAPLES, INC.	⟨corename⟩,⟨suffix⟩
DATE	*February 2, 2019*	⟨MonthOfYear⟩⟨Day⟩,⟨Year⟩
	6/13/2012	⟨MonthOfYear⟩/⟨Day⟩/⟨Year⟩
	1st day of April 2019	⟨Day⟩⟨tok⟩{2}⟨MonthOfYear⟩⟨Year⟩
LOG	*719+1: Tue Aug 22 08:26:41 1995 (/wow/wow-mbos.gif): Sent binary: GET /wow/wow-mbos.gif HTTP/1.0*	⟨host⟩⟨timestamp⟩(⟨filename⟩)⟨operation⟩: ⟨requestType⟩ ⟨filename⟩ ⟨remainder⟩

Table 1: Sample entity mentions and their expected structured representations from each dataset

Input Representation	
BERT-CRF word embeddings size	768
Input dropout rate	0
Structure Vectors	
size of predicates	15
Multilayer Perceptron Layer	
MLP-1 (after BERT) (input, output)	(768,50)
MLP-2 (after Concatenation) (input, output)	(50, # labels)
Activation function	relu
Training	
Optimization	SGD
# epochs	30
Learning rate	0.01
Learning rate decay	1×10^{-4}
Loss function	Negative log likelihood
Active Learning & Weak Supervision Parameters	
# structurally similar examples	50
# high-confidence examples	15
# low-confidence examples	15

Table 2: Architecture, hyperparameters, and training

three models: m_1, m_2, and m_3. The predictions of the three models over the single test date entity is shown in Table 3, and the F1-scores of these models at entity-level, token-level, and component-level are shown Table 4.

	June	**3rd**	**2020**
Ground truth	**month**	**day**	**year**
m_1's predictions	month	day	year
m_2's predictions	month	month	year
m_3's predictions	none	none	none

Table 3: Predictions of three dummy models (none means the model does not make a prediction)

Given that m_1 correctly predict all the tokens (thus the entire entity), it is easy to see that it's F1-scores are all 1.0's. Similar argument can also show that m_3's F1-scores are all 0's. For m_2, since it does not correctly label all the three tokens of the entity string, so the entity-level is 0. However, it does correctly labeled two tokens (i.e., "June" and "3rd") out of the three tokens, so the precision is $0.67 = 2/3$. Moreover, since m_2 makes predictions for all the three tokens, thus the recall is trivially 100%, which means it's token-level F1 is 0.80.

	Entity-level	**Token-level**	**Component-level**	
m_1	1.0	1.0	month	1.0
			day	1.0
			year	1.0
m_2	0.0 (precision = 0) (recall = 1.0)	0.80 (precision = 0.67) (recall = 1.0)	month	0.67 (precision=0.5) (recall = 1.0)
			day	0 (precision=1.0) (recall = 0.0)
			year	1.0
m_3	0.0	0.0	month	0.0
			day	0.0
			year	0.0

Table 4: F1-scores of the three dummy models at entity-level, token-level, and component-level

In fact, DL-based approaches such as ours that takes a sequence of tokens as input, will trivially achieve 100% recall. For component-level evaluation, m_2 predicted two tokens as **month**, where only one of them is correct, so the precision for month component is 50%. Since m_2 identified all the true **month** tokens, the recall is 100%. For **day** component, m_2 does not predict any token as **day**, but the second token "3rd" has true label **day**, so the recall is 0%, which leads to an F1-score of 0. We do not discuss its **year** component performance because it is obvious 100%.

5 Training

Unlike typical deep learning-based supervised learning approaches, where there are a lot of labeled examples, we have limited training data in each active learning iteration. It does not make sense to split the limited number of labeled examples (e.g., about 30) into a training set and a development set, and use the development set to choose the best-performing model. First, the number of examples is too small to make the splitting meaningful. Second, we could potentially perform

k-fold cross validation, but that would require much more time, which makes the user experience bad (i.e., the user has to wait for a long time before the training is done).

To make the system as interactive as possible, we used a simple heuristic, that is, we simply train the model with a fixed number of epoch (with random shuffling after each epoch), in our experiments, we set the number to 30 epochs. However, we would terminate the training early if the difference between the total loss of two consecutive epochs is less than a certain threshold, which is set to 10^{-3} in our experiments. The main intuition is that we want to let the model somewhat overfit the training data as they are considered to be "informative" based on our active learning strategy, but we do need to avoid "extreme" overfitting.

6 Bi-LSTM-CRF Model

Although not stated in the paper, our system can also learn Bi-LSTM-CRF-based models in addition to the BERT-CRF models. During the early stage of the work, Bi-LSTM-CRF based models gave slightly better performance compared to an earlier version of the BERT-CRF models reported in this paper. After we made several major changes to the implementation, now BERT-CRF models outperform Bi-LSTM-CRF models, so we only reported the results of BERT-CRF models in this paper. However, Bi-LSTM-CRF models have the benefit that it can learn even more light-weight deep learning models (because it uses fastText as embedding models, which have smaller embedding size that leads to smaller neural nets), so the runtime efficiency of Bi-LSTM-CRF models is slightly better.

7 Environment and Runtime

We have run our experiments both on a CPU machine (Apple Macbook Pro 2019 model) and on a GPU machine (with 1 Tesla V100 GPU). Recall that our framework is active learning-based, where each active learning iteration contains three steps: (1) user labeling, (2) model updating, and (3) user feedback. The most time-consuming part is the model updating phase. Depending on the sizes of labeled data, the model updating phase takes 10 to 90 seconds with the GPU machine, and 30 seconds to 10 mins with the CPU machine.

References

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.