In-Memory Distributed Training of Linear-Chain Conditional Random Fields, with an Application to Fine-Grained Named Entity Recognition

Robert Schwarzenberg and **Leonhard Hennig** and **Holmer Hemsen**
Deutsches Forschungszentrum für Künstliche Intelligenz, Germany
`firstname.lastname@dfki.de`

Abstract

Recognizing fine-grained named entities, i.e. *street* and *city* instead of just the coarse type *location*, has been shown to increase task performance in several contexts. Fine-grained types, however, amplify the problem of data sparsity during training, which is why larger amounts of training data are needed. In this contribution we address scalability issues caused by the larger training sets. We distribute and parallelize feature extraction and parameter estimation in linear-chain conditional random fields, which are a popular choice for sequence labeling tasks such as named entity recognition (NER) and part of speech (POS) tagging. To this end, we employ the parallel stream processing framework Apache Flink which supports in-memory distributed iterations. Due to this feature, contrary to prior approaches, our system becomes iteration-aware during gradient descent. We experimentally demonstrate the scalability of our approach and also validate the parameters learned during distributed training in a fine-grained NER task.

1 Introduction

Fine-grained named entity recognition and typing has recently attracted much interest, as NLP applications increasingly require domain- and topic-specific entity recognition beyond standard, coarse types such as persons, organizations and locations (Ling and Weld, 2012; Del Corro et al., 2015; Abhishek et al., 2017). In NLP tasks such as relation extraction or question answering, using fine-grained types for entities can significantly increase task performance (Ling and Weld, 2012; Koch et al., 2014; Dong et al., 2015). At the same time, freely-available, large-scale knowledge bases, such as Freebase (Bollacker et al., 2008), DBpedia (Auer et al., 2007) and Microsoft's Concept Graph (Wang et al., 2015) provide rich entity type taxonomies for labeling entities. However, training models for fine-grained NER requires large amounts of training data in order to overcome data sparsity issues (e.g. for low-frequency categories or features), as well as labeling noise, e.g. as introduced by training datasets created with distant supervision (Plank et al., 2014; Abhishek et al., 2017). Furthermore, the diversity of entity type taxonomies and application scenarios often requires the frequent adaptation or re-training of models. The speed and efficiency with which we can (re-)train models thus becomes a major criterion for selecting learning algorithms, if we want to fully make use of these larger datasets and richer type taxonomies.

Linear-chain CRFs (Lafferty et al., 2001) are a very popular approach to solve sequence labeling tasks such as NER (Strauss et al., 2016). Parameter estimation in CRFs is typically performed in a supervised manner. Training, however, is time-consuming with larger datasets and many features or labels. For instance, it took more than three days to train a part-of-speech tagging model (45 labels, around 500k parameters) with less than 1 million training tokens on a 2.4 GHz Intel Xeon machine, Sutton and McCallum (2011) report. This is due to the fact that during training, linear-chain CRFs require to perform inference for each training sequence at each iteration.

Fortunately, linear-chain CRFs hold potential for parallelization. During gradient descent optimization it is possible to compute local gradients on subsets of the training data which then need to be accumulated into a global gradient. Li et al. (2015) recently demonstrated this approach by parallelizing model training within the MapReduce framework (Dean and Ghemawat, 2008). The authors distributed subsets of the training data among the mappers of their cluster, which computed local gradients in a *map* phase. The local gradients were then accumulated into a global gradient in a subsequent *reduce* step. The *map* and *reduce* steps can be repeated until convergence, using the global gradient to update the model parameters at each iteration step. For large data sets, NER experiments showed that their approach improves performance in terms of run times. However, for each learning step, their system invokes a new Hadoop job, which is very time-consuming due to JVM startup times and disk IO for re-reading the training data. As the authors themselves point out, in-memory strategies would be much more efficient.

In this paper, we employ a very similar parallelization approach as Li et al., but implement the training within an efficient, iteration-aware distributed processing framework. The framework we choose allows us to efficiently store model parameters and other pre-computed data in memory, in order to keep the de/serialization overhead across iterations to a minimum (Alexandrov et al., 2014; Ewen et al., 2013).

Our contributions in this paper are:

- a proof-of-concept implementation of a distributed, iteration-aware linear-chain CRF training (Section 3),
- the experimental verification of the scalability of our approach, including an analysis of the communication overhead trade-offs (Sections 4, 5), and
- the experimental validation of the parameters learned during distributed training in a fine-grained NER and typing task for German geolocations (Sections 6, 7).

In what follows, we first define linear-chain CRFs more formally and explain in detail how parameter estimation can be parallelized. We then discuss the details of our implementation, followed by several experimental evaluations.

2 Parallelization of Conditional Random Fields

This section closely follows Sutton and McCallum (2011) and Li et al. (2015). Assume $O = o_1 \dots o_T$ is a sequence of observations (i.e. tokens) and $L = l_1 \dots l_T$ is a sequence of labels (i.e. NE tags). Formally, a linear-chain CRF can then be defined as

$$p(L|O) = \frac{1}{Z(O)} \prod_{t=1}^{T} \exp\left(\sum_{k}^{K} \theta_k f_k(l_{t-1}, l_t, o_t)\right) \tag{1}$$

where f_k denotes one of K binary indicator – or feature – functions, each weighted by $\theta_k \in \mathbb{R}$, and Z is a normalization term, which iterates over all possible assignments

$$Z(O) = \sum_{L'} \prod_{t=1}^{T} \exp\left(\sum_{k}^{K} \theta_k f_k(l'_{t-1}, l'_t, o_t)\right). \tag{2}$$

The parameters θ_k are estimated in a way such that the conditional log-likelihood of the label sequences in the training data, denoted by $\mathbf{L}$ in the following, is maximized. This can be achieved with gradient descent routines.

Partially deriving $\mathbf{L}$ by θ_k yields

$$\frac{\partial \mathbf{L}}{\partial \theta_k} = \mathbf{E}(f_k) - \mathbf{E}_\theta(f_k) \tag{3}$$

where

$$\mathbf{E}(f_k) = \sum_{i=1}^{N} \sum_{t=1}^{T} f_k(l_{t-1}^{(i)}, l_t^{(i)}, o_t^{(i)}) \tag{4}$$

is the expected value of feature k in the training data $D = \{O^{(i)}, L^{(i)}\}_{i=1}^{N}$, and

$$\mathbf{E}_\theta(f_k) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{l,l'} f_k(l, l', o_t^{(i)}) p(l, l'|O^{(i)}; \theta) \tag{5}$$

is the expected value of the feature according to the model with parameter tensor θ. The inconvenience with Eq. 5 is that it requires us to perform marginal inference at each iteration, for each training sequence.

Fortunately, according to Eqs. 4 and 5, Eq. 3 can be computed in a data parallel fashion since

$$\begin{aligned} \frac{\partial \mathbf{L}}{\partial \theta_k} = \sum_{i=1}^{N} \Big(& \sum_{t=1}^{T} f_k(l_{t-1}^{(i)}, l_t^{(i)}, o_t^{(i)}) \\ & - \sum_{t=1}^{T} \sum_{l,l'} f_k(l, l', o_t^{(i)}) p(l, l'|O^{(i)}; \theta) \Big) \end{aligned} \tag{6}$$

The next section explains how we distributed and parallelized the training phase.

3 Implementation

We partitioned the data into disjoint chunks of size p which we distributed among the mappers in a

Flink cluster. Each mapper computed a local gradient on the chunk it received. In a subsequent *reduce* job, the local gradients were accumulated into a global one:

$$\left.\begin{array}{l}\left.\sum_{i=1}^{p}(E^{(i)}(f_k)-E_{\theta}^{(i)}(f_k))\right\}\text{ map}\\ \left.\sum_{i=p+1}^{2p}(E^{(i)}(f_k)-E_{\theta}^{(i)}(f_k))\right\}\text{ map}\end{array}\right\}(+)\text{ reduce}$$

$$\vdots$$

We used the global gradient to update the current model parameters at each iteration. The information flow is depicted in Fig. 1. As can be seen, before the first iteration, we also distributed feature extraction among the mappers.

Our system marries two powerful tools, the probabilistic modeling library FACTORIE[1] (McCallum et al., 2009) and the parallel processing engine Apache Flink[2] (Alexandrov et al., 2014). It inherits features and functions from both tools.

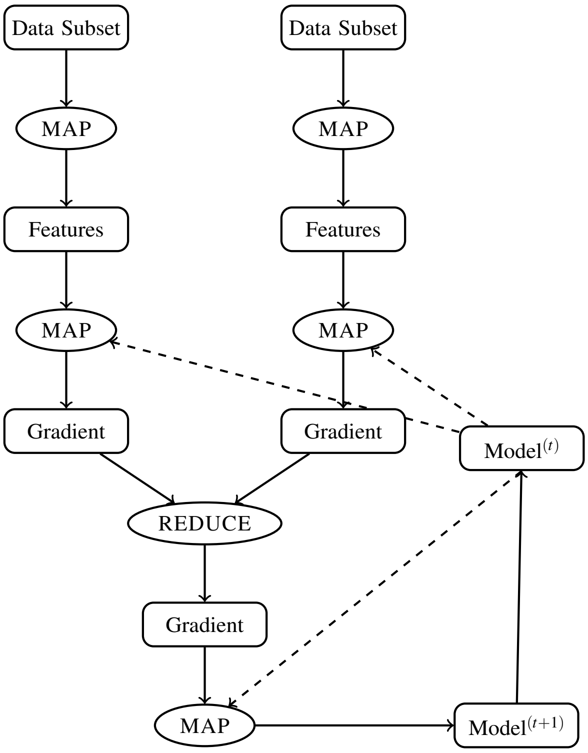

Figure 1: Distributed iteration step. The dashed lines represent Flink broadcasts.

The authors of FACTORIE convincingly promote it as a tool which preserves the 'traditional, declarative, statistical semantics of factor graphs while allowing imperative definitions of the model structure and operation.' Furthermore, Passos et al. (2013) compared FACTORIE's performance with established libraries such as scikit-learn, MALLET and CRFSuite and found that it is competitive in terms of accuracy and efficiency.

We distributed the model we implemented in FACTORIE with the help of Apache Flink. Flink provides primitives for massively parallel iterations and when compiling a distributed program which contains iterations, it analyses the data flow, identifies iteration-invariant parts and caches them to prevent unnecessary recomputations, Ewen et al. (2013) explain. Thus, contrary to prior approaches, due to Flink, our distributed system becomes 'iteration-aware'.

FACTORIE already supported local thread-level parallelism as well as distributed hyper-parameter optimization. Nonetheless, we had to overcome several obstacles when we ported the library into the cluster. For instance, in FACTORIE, object hash identities are used to map gradient tensors onto corresponding weight tensors during training. These identities get lost when an object is serialized in one JVM and deserialized in another JVM. To preserve identities throughout de/serialization among the virtual machines within the cluster, we cached relevant object hashes. We thus ended up using a slightly modified library.

4 Scalability Experiments

We tested our system on a NER task with seven types (including the default type). We compared our distributed parallel system with a local sequential counterpart in which we removed all Flink directives. In both versions our model consisted of a label-label factor and an observation-label factor[3]. During training, we used a likelihood objective, a belief propagation inference method which was tailored to linear chains and a constant step-size optimizer; all of which FACTORIE's modular design allows to plug in easily.

To evaluate performance, we varied three values

1. the level of parallelism,
2. the amount of training instances, and
3. the number of parameters, K.

Points 2) - 3) were varied for both the local version and the distributed version. When we tested

[1] Version 1.2 (modified).

[2] Version 1.3.

[3] We refer to a token's features as *observations*.

the local version we kept the number of participating computational nodes constant at one. In particular, no local thread parallelism was allowed, which is why point one does not apply to the local version.

All distributed experiments were conducted on an Apache Hadoop YARN cluster consisting of four computers (+ 1 master node). The local experiments were carried out using the master node. Two of the computers were running Intel Xeon CPUs E5-2630L v3 @ 1.80GHz with 8 cores, 16 threads and 20 MB cache (as was the master node), while the other two computers were running Intel Xeon CPUs E5-2630 v3 @ 2.40GHz again with 8 cores, 16 threads and 20 MB cache.

Each yarn task manager was assigned 8 GB of memory, of which a fraction of 30% was reserved for Flink's internal memory management. We used the Xmx option on the master node (with a total of 32 GB RAM). The nodes in the cluster thus had slightly less RAM available for the actual task than the master node. However, as a general purpose computer, the master node was also carrying out other tasks. We observed a maximal fluctuation of 1.7% (470 seconds vs. 478 seconds) for the same task carried out on different days. Loading the data from local files and balancing it between the mappers in the cluster was considered part of the training, as was feature extraction.

5 Scalability Evaluation

We first performed several sanity checks. For example, we made sure that multiple physical machines were involved during parameter estimation and that disjoint chunks of the training data reached the different machines. We also checked that the gradients computed by the mappers differed and that they were accumulated correctly during the reduce phase.

The most convincing fact that led us to believe that we correctly distributed feature extraction and parameter estimation was that after we trained the local version and the distributed version using the same training set - with just a few parameters and for just a few iterations - extremely similar parameters were in place. Consequently, the two models predicted identical labels on the same test set containing 5k tokens. The parameters diverge the more features are used and the more training steps are taken. We suspect that this is due to floating point imprecisions that result in different gradients at some point.

The first two experiments we conducted addressed the scalability of our distributed implementation. The results are summarized in Figs. 2 and 3. Fig. 2 shows that our distributed implementation managed to outperform its sequential counterpart after a certain level of parallelism was reached. The level of parallelism required to beat the local version increased with the amount of parameters.

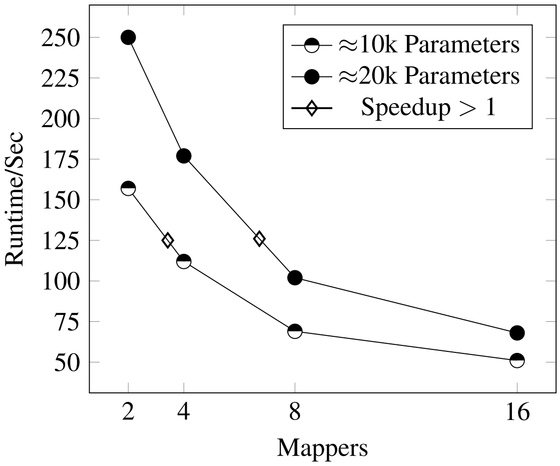

Figure 2: Execution times for increasing numbers of mappers (M). Each training involved around 100k tokens (numbers rounded for better readability) and 25 iterations. The diamonds mark the points from which on the distributed version needed less time than its local counterpart. The sequential version needed 125 seconds for around 10k parameters and 126 seconds for twice as many parameters.

Fig. 3 shows to what extent we were able to counterbalance an increase in training size with an increase in parallelism. The results suggest that our model was indeed able to dampen the effect of increasing amounts of training examples. The average rate of change in execution times was higher when we kept the number of nodes constant. As we doubled the level of parallelism along with the training size, the rate of change reduced significantly. We also compared the distributed implementation with the local implementation in Fig. 3. As can be seen, the average rate of change is higher for the local version than for the distributed version with an increasing level of parallelism. However, it is still much lower when compared to the distributed runs with a fixed level of parallelism.

We conducted a third experiment to address the effect of communication overhead. Thus far, we

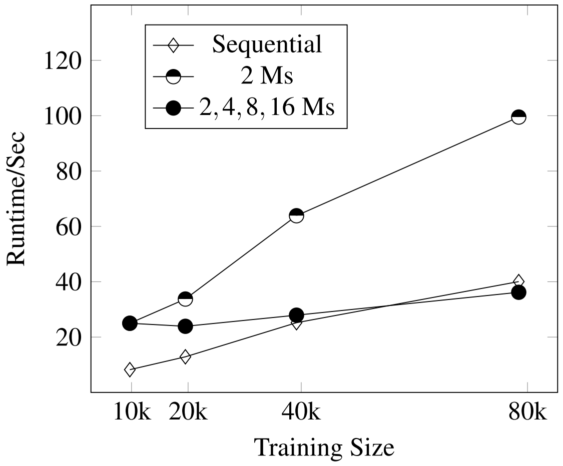

Figure 3: Scalability of the distributed model. The figure offers a comparison between the execution times required by the local version and the distributed version to process an increasing (doubling) amount of training data. The distributed version was tested with a fixed number of mappers (M) and with an increasing (doubling) number of mappers (starting with two mappers at around 10k training instances). For each run, around 20k parameters were considered and the number of iterations was fixed at ten.

have worked with a relatively low number of parameters. This was to ensure that the execution times of the distributed version were falling within the execution time range of the local version. The reason for why the low number was necessary is evident in Fig. 4: an increase in the number of parameters had a significant effect on the distributed runs. This is due to the fact that it is θ which needs to be communicated during MapReduce and it is also the size of θ which co-determines how much data needs to be cached. By contrast, it had little effect on the local version when we increased the size of θ. The execution times increase linearly for the distributed version, while locally they stay at a constant rate. In our cluster, around 40k parameters require more than eight mappers to outperform the local implementation in a distributed run.

6 Accuracy Experiments

The sections above address the scalability of our approach. In this section we report on experiments which demonstrate that our distributed linear-chain CRF learns meaningful parameters. We tested our model in a NER task.

The task was to recognize six fine-grained geospatial concepts in German texts, namely streets (‘Berliner Straße’), cities (‘Berlin’), public transportation hubs (‘Berliner Straße’), routes (‘U6’), distances (‘5 km’) and the super-type location (‘Germany’). The task involves the typical challenges of NER, such as disambiguation. Furthermore, the training sets (which were annotated by trained linguists) contained user-generated content, which is why noise was also an issue.

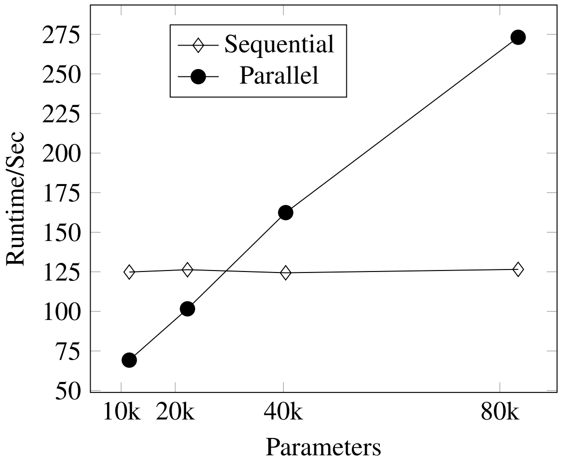

Figure 4: Execution times for increasing numbers of parameters. Each run involved around 100k tokens and 25 iterations. During the distributed runs, the level of parallelism was fixed at eight.

Table 1 characterizes the two datasets and explains what we refer to as *noise*. The RSS dataset consists of a sample of traffic reports crawled from more than 100 RSS feeds that provide traffic and transportation information about road blocks, construction sites, traffic jams, or rail replacement services. Feed sources include federal and state police, radio stations, Deutsche Bahn, and air travel sources. Traffic reports are typically very brief, may be semi-structured (e.g. location, cause and length of a traffic jam), and often contain telegraph-style sentences or phrases. The Twitter dataset consists of a sample of German-language tweets that were retrieved via the Twitter search API using a list of approximately 150 domain-relevant users/channels and 300 search terms. Channels include e.g. airline companies, traffic information sources, and railway companies. Search terms comprise event-related keywords such as “traffic jam” or “roadworks”, but also major highway names, railway route identifiers, and airport codes. Both datasets therefore consist of documents which contain traffic- and mobility-related information that refer to the fine-grained location types defined previously.

Besides the well-established features in NER (e.g. word shape, affixes) our application ('Locator') also considered task specific features and took measures towards text normalization. In the end, a larger number of parameters (100k-150k) was in place than during the scalability experiments.

We again used the FACTORIE components listed in Section 4, such as the BP method for chains and a constant step size optimizer. FACTORIE provides more sophisticated optimizers such as LBFGS or Adagrad. In our current system, however, only the model parameters survive a Flink-iteration step but methods like LBFGS and Adagrad need further information about past update steps.

We conducted a ten-fold cross-validation on the datasets. Feature extraction and parameter estimation were performed in parallel in the way described above. The level of parallelism was fixed at four, for all experiments. After training, the models were serialized and saved for the test phase. The test runs took place on a single machine.

To put the performance of our model into perspective, we also conducted a ten-fold cross-validation using the Stanford NER (v. 3.6.0) in its standard configuration[4]. The Stanford NER used the same tokenizer as our system.

7 Accuracy Evaluation

The results of our accuracy experiments are summarized in Table 2. The F-scores achieved on the Twitter dataset and the scores achieved on the RSS dataset reveal similar trends for both systems: In both cases, the RSS-score is higher than the Twitter-score.

Our distributed model slightly outperforms the Stanford NER on the Twitter dataset but is beaten on the RSS dataset. Since the Twitter dataset is noisier than the RSS dataset, we suspect that the task-specific features and text normalization methods of our system have a greater impact in this case.

Overall, we conclude that the experiments provide sufficient proof that during distributed training our system indeed learns meaningful parameters. It achieves comparable scores.

Dataset	Tokens	Noise
RSS	20152	35.6%
Twitter	12606	45.3%

Table 1: Datasets. Size and noise. We refer to noise as the percentage of tokens that the Enchant v 1.6.0 Myspell de_DE dictionary did not recognize.

System	Dataset	P	R	F1
Locator	RSS	80.7	75.8	75.2
Stanford	RSS	82.8	78.8	**80.5**
Locator	Twitter	57.0	50.4	**51.7**
Stanford	Twitter	79.0	35.9	47.2

Table 2: Results of 10-fold NER experiments. Classification performance was evaluated on token level so that multiple-token spans resulted in multiple true positives or false negatives, for instance. To compensate class imbalances, for each fold, we weighted the fine-grained scores (i.e. precision (P), recall (R) and F1-score (F1) of the entity 'street') by the support of the entity in the test set and averaged over all fine-grained scores. The listed scores are averages over the ten fold scores.

8 Discussion & Conclusion

We distributed and parallelized feature extraction and parameter estimation in linear-chain CRFs. The sequence labeling experiments we conducted suggest that our system learns meaningful parameters and is able to counterbalance growing amounts of training data with an increase in the level of parallelism (see Table 2 and Figs. 2 and 3). We reached speedups greater than one and F-scores comparable to the ones produced by a state-of-the-art approach.

To achieve this, we combined the parallel processing engine Apache Flink with the probabilistic modeling library FACTORIE. Our proof-of-concept implementation now inherits functions and features from both tools.

Contrary to prior approaches, for instance, it is iteration-aware during distributed gradient descent. In addition, our system also benefits from FACTORIE's modular design and rich pool of functions. With little programming effort it is possible to plug in alternative choices for an optimizer or an inference method.

There is, however, room for improvement. The choice for an optimizer, for instance, is restricted by the fact that currently, only the model param-

[4] The configuration file we used can be found in the appendix.

eters survive an iteration. But some optimization procedures that FACTORIE provides, like LBFGS, require additional information about past updates. Enhancing the system to provide this feature remains future work.

Furthermore, the increase in runtime in Fig. 4 seems disproportionate. Working with sparse vectors to reduce the amount of data that needs to be cached will most likely reduce runtime. There might also be a serialization bottleneck. Registering customized serializers for FACTORIE's types with Flink may thus also improve performance. Fortunately, the number of features is typically fixed at some point in most settings. At this point the amount of available training data and the number of mappers in the cluster determine from when on our approach pays off.

Acknowledgments

This research was partially supported by the German Federal Ministry of Economics and Energy (BMWi) through the projects SD4M (01MD15007B) and SDW (01MD15010A), and by the German Federal Ministry of Education and Research (BMBF) through the project BBDC (01IS14013E).

References

[Abhishek et al.2017] Abhishek Abhishek, Ashish Anand, and Amit Awekar. 2017. Fine-Grained Entity Type Classification by Jointly Learning Representations and Label Embeddings. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 797–807, Valencia, Spain, April. Association for Computational Linguistics.

[Alexandrov et al.2014] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, et al. 2014. The stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939–964.

[Auer et al.2007] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg. Springer-Verlag.

[Bollacker et al.2008] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA. ACM.

[Dean and Ghemawat2008] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[Del Corro et al.2015] Luciano Del Corro, Abdalghani Abujabal, Rainer Gemulla, and Gerhard Weikum. 2015. Finet: Context-aware fine-grained named entity typing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 868–878, Lisbon, Portugal, September. Association for Computational Linguistics.

[Dong et al.2015] Li Dong, Furu Wei, Hong Sun, Ming Zhou, and Ke Xu. 2015. A Hybrid Neural Model for Type Classification of Entity Mentions. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1243–1249, Buenos Aires, Argentina. AAAI Press.

[Ewen et al.2013] Stephan Ewen, Sebastian Schelter, Kostas Tzoumas, Daniel Warneke, and Volker Markl. 2013. Iterative parallel data processing with stratosphere: an inside look. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1053–1056. ACM.

[Koch et al.2014] Mitchell Koch, John Gilmer, Stephen Soderland, and Daniel S. Weld. 2014. Type-Aware Distantly Supervised Relation Extraction with Linked Arguments. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1891–1901, Doha, Qatar, October. Association for Computational Linguistics.

[Lafferty et al.2001] John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of the ICML*, volume 1, pages 282–289.

[Li et al.2015] Kenli Li, Wei Ai, Zhuo Tang, Fan Zhang, Lingang Jiang, Keqin Li, and Kai Hwang. 2015. Hadoop recognition of biomedical named entity using conditional random fields. *IEEE Transactions on Parallel and Distributed Systems*, 26(11):3040–3051.

[Ling and Weld2012] Xiao Ling and Daniel Weld. 2012. Fine-Grained Entity Recognition. In *Proc. of AAAI '12*.

[McCallum et al.2009] Andrew McCallum, Karl Schultz, and Sameer Singh. 2009. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Advances in Neural Information Processing Systems*, pages 1249–1257.

[Passos et al.2013] Alexandre Passos, Luke Vilnis, and Andrew McCallum. 2013. Optimization and learning in FACTORIE. In *NIPS Workshop on Optimization for Machine Learning (OPT)*.

[Plank et al.2014] Barbara Plank, Dirk Hovy, Ryan T McDonald, and Anders Søgaard. 2014. Adapting taggers to twitter with not-so-distant supervision. In *COLING*, pages 1783–1792.

[Strauss et al.2016] Benjamin Strauss, Bethany E Toma, Alan Ritter, Marie-Catherine de Marneffe, and Wei Xu. 2016. Results of the wnut16 named entity recognition shared task. In *Proceedings of the 2nd Workshop on Noisy User-generated Text*, pages 138–144.

[Sutton and McCallum2011] Charles Sutton and Andrew McCallum. 2011. An introduction to conditional random fields. *Foundation and Trends in Machine Learning*, 4(4):267–373.

[Wang et al.2015] Zhongyuan Wang, Haixun Wang, Ji-Rong Wen, and Yanghua Xiao. 2015. An Inference Approach to Basic Level of Categorization. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 653–662, New York, NY, USA. ACM.

Appendix A. Stanford NER Properties File

```
trainFile=path/to/training_file
serializeTo=path/to/model
map=word=0,answer=1

useClassFeature=true
useWord=true
useNGrams=true
noMidNGrams=true
maxNGramLeng=6
usePrev=true
useNext=true
useSequences=true
usePrevSequences=true
maxLeft=1
useTypeSeqs=true
useTypeSeqs2=true
useTypeySequences=true
wordShape=chris2useLC
useDisjunctive=true
```