Boosting Information Extraction Systems with Character-level Neural Networks and Free Noisy Supervision

Philipp Meerkamp
Bloomberg LP
731 Lexington Avenue
New York, NY 10022
USA
pmeerkamp@bloomberg.net

Zhengyi Zhou
AT&T Labs Research
33 Thomas Street
New York, NY 10007
USA
zzhou@research.att.com

Abstract

We present an architecture to boost the precision of existing information extraction systems. This is achieved by augmenting the existing parser, which may be constraint-based or hybrid statistical, with a character-level neural network. Our architecture combines the ability of constraint-based or hybrid extraction systems to easily incorporate domain knowledge with the ability of deep neural networks to leverage large amounts of data to learn complex features. The network is trained using a measure of consistency between extracted data and existing databases as a form of cheap, noisy supervision. Our architecture does not require large scale manual annotation or a system rewrite. It has led to large precision improvements over an existing, highly-tuned production information extraction system used at Bloomberg LP for financial language text.

1 Introduction

1.1 Information extraction in finance

Unstructured textual data is abundant in the financial domain (see Figure 1). This type of text is usually not in a format that lends itself to immediate processing. Hence, information extraction is an essential step in many business applications that wish to use the financial text data. Examples of such business applications include creating time series databases for macroeconomic forecasting, or real-time extraction of time series data to inform algorithmic trading strategies. For example, consider extracting data from Figure 1 into numerical relations of the form
ts_tick_abs (TS symbol, numerical value),

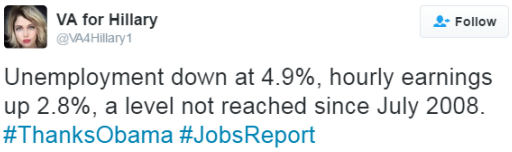

Figure 1: Tweet containing financial data.

e.g. *ts_tick_abs (US_Unemployment, 4.9%)*, or
ts_tick_rel (TS symbol, change in num. value),
e.g. *ts_tick_abs (US_Hourly_Earnings, 2.8%)*.
For these business applications, the extraction needs to be fast, accurate, and low-cost.

There are several challenges to extracting information from financial language text.

- Financial text can be very ambiguous. Language in the financial domain often trades grammatical correctness for brevity. A multitude of numeric tokens need to be disambiguated into entities such as prices, rates, percentage changes, quantities, dates, times, and others. Finally, many words could be company names, stock symbols, domain-specific terms, or have entirely different meanings.

- Large-scale annotated data with ground truths is typically not available. Domain expert manual annotations are expensive and limited in scale. This is especially a problem because the size of the problem domain is large, with many types of financial instruments, language conventions, and text formats.

These two challenges lead to a high risk of extracting false positives.

Bloomberg has mature information extraction systems for financial language text, that are the result of nearly a decade of efforts. When improving

Proceedings of the 2nd Workshop on Structured Prediction for Natural Language Processing, pages 44–51

upon such large industrial extraction systems, it is invaluable if the proposed improvement does not require a complete system rewrite.

The existing extraction systems leverage both constraint-based and statistical methods. Constraint-based methods can effectively incorporate domain knowledge (e.g. unemployment numbers cannot be negative numbers, while changes in unemployment numbers *can* be negative). These constraints can be complex, and may involve multiple entities within an extraction candidate. Adding a single constraint can in many cases vastly improve the system's accuracy. In a purely statistical system, achieving the equivalent behavior would require large amounts of labeled training data.

This existing systems generate extractions with high recall, and in this work, we propose to boost the precision of the existing systems using a deep neural network.

1.2 Our contribution

We present an information extraction architecture that boosts the precision of an existing parser using a deep neural network. The architecture gains the neural network's ability to leverage large amounts of data to learn complex features specific to the application at hand. At the same time, the existing parser may leverage constraints to easily incorporate domain knowledge. Our method uses potentially noisy but cheaply available source of supervision, e.g. via consistency checks of extracted data against existing databases (e.g. an extraction *ts_tick_abs (US_Unemployment, 49%)* would be implausible given recent US employment history), or via human interaction (e.g., clicks on online advertisements).

Our extraction system has two main advantages

- We improve the existing extraction systems cheaply using large amounts of free noisy labels, without the need for manual annotation. This is particularly valuable in large application domains, where manual annotations covering the entire domain would be prohibitively expensive.

- Our method leverages existing codebases fully, and requires no system rewrite. This is critical in large industrial extraction systems, where a rewrite would take many man-years. Even for new systems, the decoupling of candidate-generation and the neural network offers advantages: the candidate-generating parser can easily enforce constraints that would be difficult to incorporate in a system relying entirely on neural networks.

We are not aware of alternative approaches that achieve the above: purely neural network or purely statistical approaches require substantial amounts of human annotation, while our method does not. Constraint-based or hybrid statistical approaches are competitors to the existing extraction systems rather than our work; our work could also be used to boost the precision of other state-of-the-art constraint or hybrid methods. We believe that our approach, with small modifications, can be applied to extraction systems in many other applications.

Compared to the existing, client-serving extraction engine, our system reduced the number of false positive extractions by $> 90\%$. Our system constituted a substantial improvement and is being deployed to production.

We review some related work in Section 1.3. Section 2 details the design, training, and method supervision of our system. We present results in Section 3 and conclude with some discussions in Section 4.

1.3 Related Work

Deep neural networks have been applied to several problems in natural language processing recently. Mikolov introduced the recurrent network language model (Kombrink et al., 2011), which can for instance consume the beginning of a sentence word by word, encoding the sentence in its state vector, and predict a likely continuation of the sentence. Long Short Term Memory (LSTM) architectures (Gehrs, 2001; Hochreiter and Schmidhuber, 1997) have resulted in large improvements in machine translation (Sutskever et al., 2014). There is a growing literature of LSTMs and deep convolutional architectures being used for text classification (e.g. (Zhang et al., 2015; Kim, 2014; dos Santos and Gatti, 2014)) and language modeling problems (Kim et al., 2016; Karpathy, 2015).

Several studies have considered using deep neural networks for information extraction problems. Socher et al introduce compositional vector grammars, which combine probabilistic context free grammars with a recursive neural network (2013). Nguyen et al use convolutional neural networks

(CNN) and recurrent neural networks with word- and entity-position-embeddings for relation extraction and event detection (Nguyen et al., 2016; Nguyen and Grishman, 2015). Zhou and Xu use a bidirectional (Schuster and Paliwal, 1997) word-level LSTM combined with a conditional random field (CRF) (Lafferty et al., 2001) for semantic role labeling (2015). In some cases, providing a neural network character-level rather than word-level input can be beneficial. Chiu and Nichols use a bidirectional LSTM-CNN with character embeddings for named entity recognition (2015), and Ballesteros et al found that character-level LSTM improve dependency parsing accuracy in many languages, relative to word-level approaches (2015). Recently, Miwa and Bansal presented an end-to-end LSTM-based architecture for relation extraction, achieving state-of-the-art results on SemEval-2010 Task 8 (2016).

While much of the recent work on information extraction focuses on statistical methods and neural networks, constraint-based information extraction systems are still commonly used in industry applications. Chiticariu et al suggest that this might be because constraint-based systems can easily incorporate domain knowledge (2013). The need to incorporate constraints into information extraction systems, as well as the difficulty of doing this efficiently, have been recognized for some time.

Several hybrid approaches combine constraint-based and statistical methods. Toutanova et al model dependencies between semantic frame arguments with a CRF (2008). Chang et al incorporate declarative constraints into statistical sequence-based models to obtain constrained conditional models (2012). When making a prediction (inference step), their model solves an integer linear program. This typically involves maximizing a scoring function, based e.g. on a CRF, over all outputs that satisfy the constraints, resulting in high computational costs. Täckström et al recently proposed a more computationally efficient approach to semantic role labeling with constraints, introducing a dynamic programming approach for inference for log-linear models with constraints (2015).

There are several purely statistical approaches, such as those using hidden Markov models (Baum and Petrie, 1966), CRFs (Lafferty et al., 2001) or structured support vector machines (Tsochantaridis et al., 2004).

Our approach is fundamentally different from all of the above. We propose to boost the precision of an existing information extraction system using a neural network trained with free noisy supervision. Our method does not require large amounts of human annotations as purely statistical or neural network-based approaches would. Our work could also be used to boost the precision of other state-of-the-art constraint or hybrid methods.

2 Design

2.1 Overview

The information extraction pipeline we developed consists of four stages (see right pane "execution" in Figure 2).

1. **Generate candidate extractions**: The document is parsed using a potentially constraint-based or hybrid statistical parser, which outputs a set of candidate extractions. Each candidate extraction consists of the character offsets of all extracted constituent entities, and a representation of the extracted relation. It may additionally contain auxiliary information that the parser may have generated, such as part-of-speech tags.

2. **Score extractions using trained neural network**: Each candidate extraction generated, together with the section of the document it was found in, is encoded into feature data X. A deep neural network is used to compute a neural network correctness score $\tilde{s}$ for each extraction candidate. We detail the input and architecture of the neural network in Section 2.2 and its training in Section 2.3.

3. **Score extractions based on cheap, noisy supervision**: We compute a consistency score s for the candidate extraction, measuring if the extracted relation is consistent with cheaply available, but potentially noisy supervision from e.g. an existing database. We discuss how s is computed in Section 2.4.

4. **Classify extractions as correct or incorrect**: A linear classifier classifies extraction candidates as correct and incorrect extractions, based on neural network correctness score $\tilde{s}$, database consistency score s, and potentially other features. Candidates classified as incorrect are discarded.

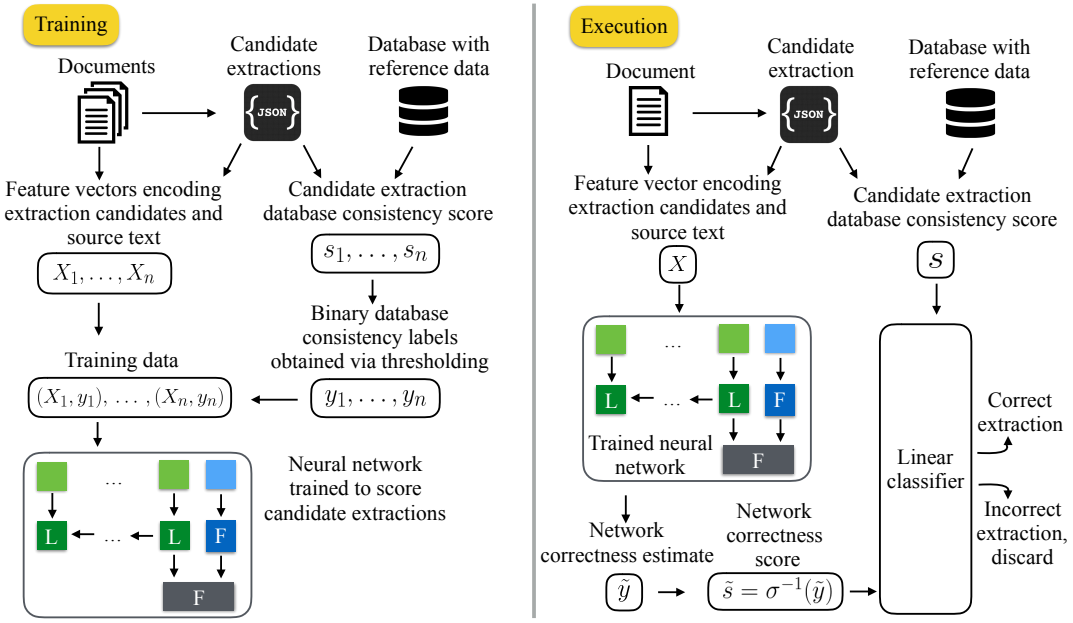

Figure 2: Training (left) and execution (right) set-up. Blocks marked "L" are neural network LSTM cells, while blocks marked "F" are fully-connected layers.

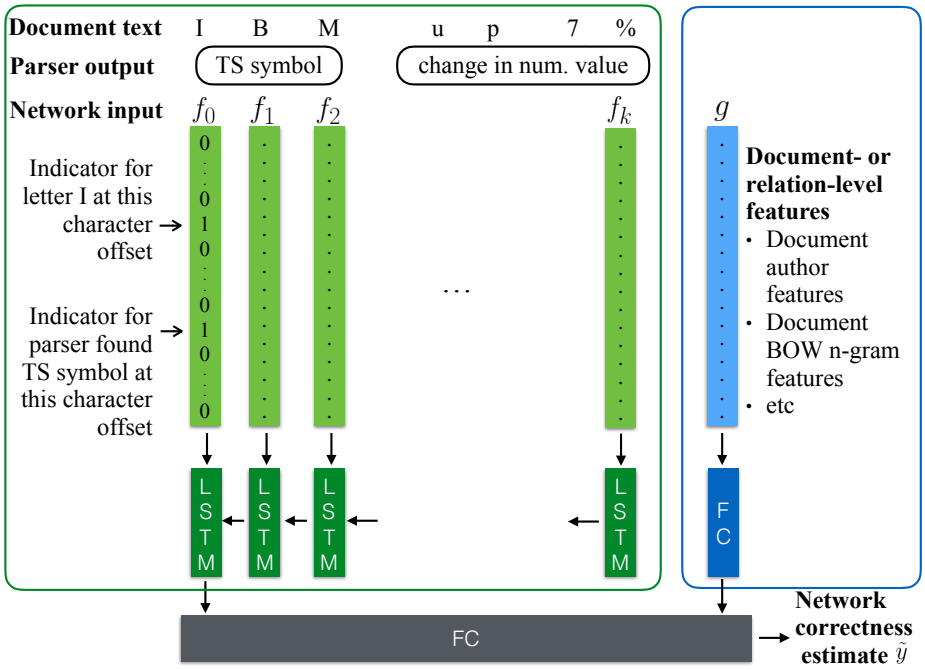

Figure 3: We use a neural network comprised of (i) an LSTM processing encoded parser output and document text character-by-character (labeled LSTM, green), (ii) a fully-connected layer (FC, blue) that takes document-level features as input, and (iii) a fully-connected layer (FC, grey) that takes the output vectors of the layers (i) and (ii) as input to generate a correctness estimate for the extraction candidate. Layer (iii) uses a sigmoid activation function to generate a correctness estimate $\tilde{y} \in (0, 1)$, from which we compute the network correctness score as $\tilde{s} := \sigma^{-1}(\tilde{y})$.

2.2 Neural network input and architecture

The neural network processes each candidate extraction independently. To estimate the correctness of an extracted candidate, the network is provided with two pieces of input (see Figure 3 for the full structure of the neural network). First, the

network is provided with a vector g (right box in Figure 3) containing document- or extraction-level features, such as attributes of the document's author, or word-level n-gram features. The second piece of input consists of a sequence of vectors f_i (left box in Figure 3), encoding the document text and the parser's output at the character level. There is one vector f_i for each character c_i of the document section where the extraction candidate is found.

The vectors f_i are a concatenation of (i) a one-hot encoding of the character, and (ii) information about entities the parser identified at the position of c_i. For (i) we use a restricted character set of size 94, including `[a-zA-Z0-9]` and several whitespace, special characters, and an indicator to represent characters not present in our character set. For (ii), we include in f_i a vector of indicators specifying whether or not any of the entities appearing in the relations supported by the parser were found in the position of character c_i.

The input vectors f_i feed into an LSTM, which accumulates state until the input sequence has been exhausted. The global feature vector g feeds into a fully-connected layer, which is then concatenated with the output of the LSTM, and passed through another fully-connected layer with sigmoid activation σ to produce a network correctness estimate $\tilde{y}$. We subsequently compute the network correctness score $\tilde{s}$ for the candidate extraction via $\tilde{s} = \sigma^{-1}(\tilde{y})$.

We regularize the LSTM weight matrices and state vector using dropout (see (Srivastava et al., 2014) and (Zaremba et al., 2015)). Similar to (Sutskever et al., 2014), we found that reversing the LSTM's input resulted in a substantial increase in accuracy.

2.3 Neural network training

We train the neural network by referencing candidates extracted by the high-recall candidate-generating parser against a potentially noisy reference source (see Figure 2, left panel on "training"). In our application, this reference is Bloomberg's proprietary database of historical time series data, which enables us to check how well the extracted numerical data fits into time series in the database. Concretely, we compute a consistency score $s \in (-\infty, \infty)$ that measures the degree of consistency with the database. Depending on the application, the score may for instance be a squared relative error, an absolute error, or a more complex error function. In many applications, the score s will be noisy (see Section 2.4 for further discussion). We threshold s to obtain binary correctness labels $y \in \{0, 1\}$. We then use the binary correctness labels y for supervised neural network training, with binary cross-entropy loss as the loss function. This allows us to train a network that can compute a pseudo-likelihood $\tilde{y} \in (0, 1)$ of a given extraction candidate to agree with the database. Thus, $\tilde{y}$ estimates how likely the extraction candidate is correct.

The neural network's training data consists of candidates generated by the candidate-generating parser, and noisy binary consistency labels y.

In our application, the database labeled a large majority of candidates as correct. To obtain balanced training data, we generated 6 sets of training data, each containing the same set of around 1 million negative cases and disjoint sets of 1 million positive cases each. Correspondingly, we trained an ensemble of 6 networks, and averaged their network scores $\tilde{s}$. We found this to work much better than oversampling the smaller class.

2.4 Noisy supervision

We assume that the noise in the source of supervision is limited in magnitude, e.g. $< 5\%$. We moreover assume that there are no strong patterns in the distribution of the noise: if the noise correlates with certain attributes of the candidate extraction, the pseudo-likelihoods $\tilde{y}$ might no longer be a good estimate of the candidate extraction's probability of being a correct extraction.

There are two sources of noise in our application's database supervision. First, there is a high rate of false positives. It is not rare for the parser to generate an extraction candidate *ts_tick_abs (TS symbol, numerical value)* in which the numerical value fits into the time series of the time series symbol, but the extraction is nonetheless incorrect. False negatives are also a problem: many financial time series are sparse and are rarely observed. As a result, it is common for differences between reference numerical values and extracted numerical values to be large even for correct extractions. Limits for acceptable differences between extracted data and reference data are incorporated into the computation of the database consistency score s. The formula for computing s is application dependent, and may involve auxiliary

data related to the extracted entities.

3 Results

Compared to using the existing, highly-tuned, client-facing extraction systems, this work reduced the number of false positive extractions by more than 90%, while the drop in recall was smaller than 1%. Our character-level neural network considerably boosted the precision of the existing information extraction systems, and is being deployed to production.

We are not aware of alternative approaches that can improve an existing extraction engine in a way that is directly comparable to ours. Purely neural network-based or purely statistical approaches require large amounts of human annotation, while our method does not. Constraint-based or hybrid statistical approaches are competitors to the existing extraction system rather than our work; indeed, our work could also be used to boost the precision of state-of-the-art constraint or hybrid methods.

When trained with 256 LSTM hidden units, and 2 million samples for 150 epochs, the neural network alone achieved a training accuracy of 95.8% and a test accuracy of 94.9%, relative to the noisy database supervision. Note that since the supervision provided by the databases is imperfect, it is not unexpected that the network's accuracy is substantially below 100%. We examined the errors in the training set, and found that they are primarily due to the network generalizing correctly, with the network being correct almost always when strongly disagreeing with the database's label.

We include in Figure 4 how data size and network size affect network accuracy. Note that the network's accuracy decreases substantially if the network size (number of LSTM units) is reduced. Accuracy also decreases if smaller quantities of training data are available. To achieve acceptably small latencies in client-serving scenarios, we limit the network size to 256 LSTM hidden units, even though larger networks could achieve slightly greater accuracies. We moreover limited the document input text provided to the LSTM neural network to the lines on which the extraction candidate was found.

Besides the neural network architecture described in this paper, we also considered an approach based on character-level n-grams (see Figure 5). In this approach, we replaced the LSTM

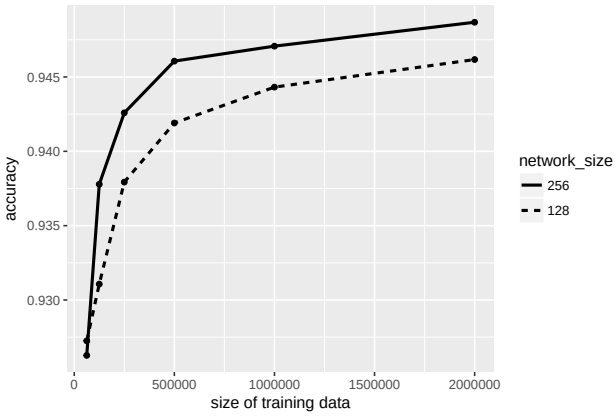

Figure 4: Neural network accuracies for different network and training data sizes.

unit of the neural network with a single fully-connected layer. At the same time, we replaced the character-level input sequence $f_0, \ldots, f_k$ with a single binary vector representing a bag-of-words of n-grams. More precisely, we computed character-level feature vectors $f'_0, \ldots, f'_k$ analogously to the f_i, except that the f'_i contain encodings of character classes (upper case letter, lower case letter, digit, and the same list of special characters used in the f_i) rather than of characters themselves. Each binary vector f'_i was then mapped to a string $\tilde{f'}_i$ containing the same sequence of 0s and 1s as f'_i. Finally, we computed n-grams over the sequence of strings $\tilde{f'}_i$. All n-gram features were tf-idf normalized. We cross-validated on a validation set to tune various hyperparameters, and found that using 2-grams up to 12-grams worked best. The resulting classifier achieve an accuracy of 96.9% on the training set, and 90.4% on the test set. This constitutes a big gap to the 94.9% test accuracy achieved by the neural network, especially since we estimate the accuracy of the database supervision to be around 95%. This suggests that the recurrent neural network was able to internally compute higher-quality features than the bag-of-words n-gram features.

4 Discussion

In this work, we presented an architecture for information extraction from text using a combination of an existing parser and a deep neural network. The deep neural network can boost the precision of an existing high-recall parser. To train the neural network, we use measures of consistency

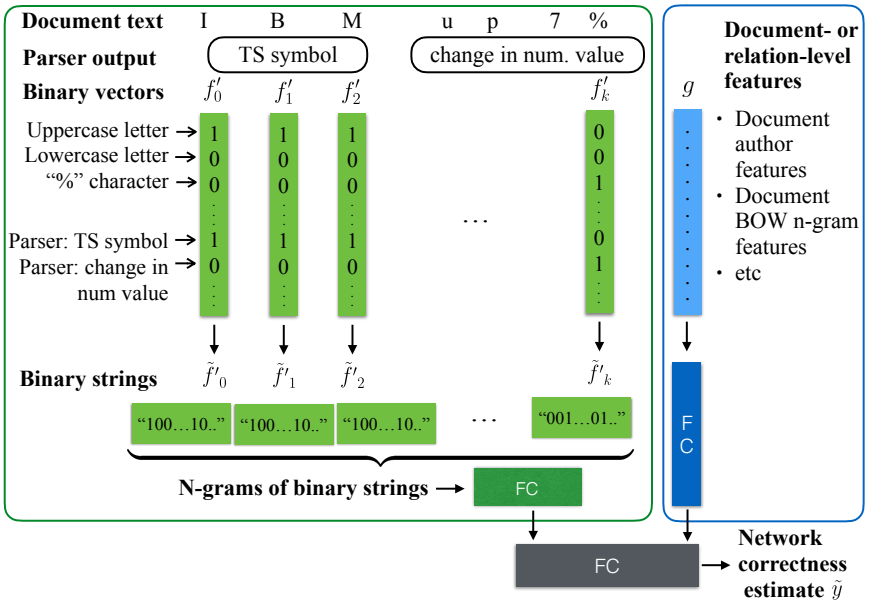

Figure 5: Extraction candidate correctness estimation with an n-gram-based classifier. It differs from Figure 3 only in the green fully connected layer labeled FC, and its input.

between extracted data and existing databases as a form of noisy supervision. The architecture resulted in substantial improvements over mature and highly-tuned information extraction systems for financial language text at Bloomberg. While we used time series databases to derive measures of consistency for candidate extractions, our set-up can easily be applied to a variety of other information extraction tasks for which potentially noisy reference data is cheaply available.

We believe our encoding of document text and parser output makes learning particularly easy for the neural network. We encode the candidate-generating parser's document annotations character-by-character into vectors f_i that also include a one-hot encoding of the character itself. We believe that this encoding makes it easier for the network to learn character-level characteristics of the entities that are part of the semantic frame. As an additional benefit, our encoding lends itself well to processing both by recurrent architectures (processing character-by-character input vectors f_i) and convolutional architectures (performing 1D convolutions over an input matrix whose columns are vectors f_i).

Our architecture can easily incorporate global attributes of the document. In our application, we found it useful to add one-hot encoded n-gram and word shape features of the document, allowing the neural network to consider information that might be located far from where the extraction candidate was found. Other potentially useful features could be based on document length, document embeddings, document creator features, and more.

We experimented with other neural network architectures. A slight variation would be to use a bidirectional LSTM instead of a simple LSTM. In addition to LSTM architectures, we experimented with character-level convolutional neural networks. In this setting, we concatenated the vectors f_i into a single matrix for all character indices, and performed 1D convolutions and pooling operations 3 times, and pass the result through a fully-connected layer. We found the performance of this approach to be very similar to that of the LSTM we used. Finally, a hybrid approach, stacking an LSTM on a single convolutional layer, gave very similar results as the LSTM and convolutional architectures.

One could use a variety of sources of information as distant and cheap supervision. While we used existing databases, other applications may use supervision e.g. from user interactions with the extracted data, say if the extracted data is presented to a user who can accept, modify, or reject the extraction. In such a system, the linear classifier classifying candidate extractions would have only a single feature, i.e. the neural network score.

Acknowledgments

We would like to thank my managers Alex Bozic, Tim Phelan, and Joshwini Pereira for supporting this project, as well as David Rosenberg from the CTO's office for providing access to GPU infrastructure.

References

Miguel Ballesteros, Chris Dyer, and A. Noah Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. In *EMNLP*. pages 349–359.

Leonard E. Baum and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Math. Statistics* .

Ming-Wei Chang, Lev Ratinov, and Dan Roth. 2012. Structured learning with constrained conditional models. *Machine Learning* 88(3):399–431.

Laura Chiticariu, Yunyao Li, and R. Frederick Reiss. 2013. Rule-based information extraction is dead! long live rule-based information extraction systems! In *EMNLP*. pages 827–832.

Jason PC Chiu and Eric Nichols. 2015. Named entity recognition with bidirectional LSTM-CNNs. http://arxiv.org/pdf/1511.08308v4.pdf.

Cicero dos Santos and Maira Gatti. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*. pages 69–78.

Felix Gehrs. 2001. *Long Short-Term Memory in Recurrent Neural Networks*. Ph.D. thesis, École polytechnique Fédérale de Lausanne.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780.

Andrej Karpathy. 2015. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn_effectiveness/.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*. pages 1746–1751.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *AAAI*. pages 2741–2749.

Stefan Kombrink, Tomas Mikolov, Martin Karafiát, and Lukás Burget. 2011. Recurrent neural network based language modeling in meeting recognition. In *INTERSPEECH*.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*. pages 282–289.

Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using lstms on sequences and tree structures. In *ACL*. Assoc. Comp. Ling., pages 1105–1116.

Huu Thien Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016. Joint event extraction via recurrent neural networks. In *NAACL*. pages 300–309.

Huu Thien Nguyen and Ralph Grishman. 2015. Event detection and domain adaptation with convolutional neural networks. In *ACL*. pages 365–371.

M. Schuster and K. K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Trans. Signal Processing* 45(11):2673–2681.

Richard Socher, John Bauer, D. Christopher Manning, and Ng Andrew Y. 2013. Parsing with compositional vector grammars. In *ACL*. pages 455–465.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *JMLR* 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*. pages 3104–3112.

Oscar Täckström, Kuzman Ganchev, and Dipanjan Das. 2015. Efficient inference and structured learning for semantic role labeling. *Transactions of the Association of Computational Linguistics* 3:29–41.

Kristina Toutanova, Aria Haghighi, and Christopher D. Manning. 2008. A global joint model for semantic role labeling. *Comp. Ling.* 34(2).

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *ICML*. pages 104–.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2015. Recurrent neural network regularization. https://arxiv.org/abs/1409.2329.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*. pages 649–657.

Jie Zhou and Wei Xu. 2015. End-to-end learning of semantic role labeling using recurrent neural networks. In *ACL*. pages 1127–1137.