Shallow Learning for sequence tagging

Robert Östling

Department of Modern Languages, University of Helsinki
Department of Linguistics, Stockholm University
`robert.ostling@helsinki.fi, robert@ling.su.se`

Abstract

While most of us appreciate recent developments in machine learning—after all, they allow us to throw some data at a deep neural network and report state-of-the-art results without painstaking feature engineering and probabilistic modeling—we go against the tide and back into the shallow end of machine learning. We present EFSELAB, a system for compiling feature templates into optimized computation graphs for feature hashes, which we then throw at a linear classifier to perform part-of-speech (PoS) tagging and named entity recognition at speeds of millions of words per second on a GPU-less desktop computer. Across the 39 languages evaluated, there is no systematic difference in PoS tagging accuracy between our system and a recently presented LSTM-based neural network system, while the latter requires nearly three orders of magnitude more computing time.

1. Introduction

Recent years have seen natural language processing (NLP), like so many other fields, completely overrun by so-called Deep Learning approaches using multi-layered neural networks. These obtain state-of-the-art results in everything from part of speech tagging (Huang et al., 2015) to machine translation (Luong and Manning, 2016). While this has revolutionized fields such as computer vision, the difference in PoS tagging accuracy between a perceptron-based linear tagger (Shen et al., 2007) and a state-of-the-art neural network system (Huang et al., 2015) is just 0.22 percentage points on the Penn Treebank test set.

Rather than increasing computational resources by orders of magnitude for a (possible) tiny gain in accuracy, our aim is to perform the task as cheaply as possible without sacrificing accuracy.

2. Feature representations

Sequence labeling is the task of finding a sequence of labels y_i given a corresponding sequence of inputs x_i. This has been used in natural language processing for a wide range of tasks, including PoS tagging, named entity recognition and shallow parsing.

Feature-rich models for sequence labeling have been popular for the last couple of decades. They are based on defining a large set of feature functions, $\phi(x, y, i)$, which describe some feature of the sequence x, label sequence y and sequence position i. For instance, we might have

$$\phi(x, y, i) = \left\{ \begin{array}{ll} 1 & \text{if} x_i = \text{cat} \wedge y_{i-1} = \text{DET} \wedge y_i = \text{NOUN} \\ 0 & \text{otherwise} \end{array} \right\}$$

The task of a linear classifier is then to find a weight vector $\bar{w}$ such that its dot product with the feature vector $\bar{\phi}^T \cdot \bar{w} = \sum_k w_k \phi_k(x, y, i)$ is high when x_i has label y_i and low otherwise.

3. Feature hashing

Since $\bar{\phi}$ is typically large but very sparse, it is computationally more efficient to store only its non-zero elements. Furthermore, given some hash function h which maps features to integers, $\bar{\phi}^T \cdot \bar{w}$ can be approximated by $\sum_{k|\phi_k(x,y,i)\neq 0} u_{h(k)}$ if all ϕ_k are binary-valued. The computational advantage is that this amounts to adding a small number of elements from the weight vector u.

For a collision-free function h with a sufficiently large weight vector u, this is identical to $\bar{\phi}^T \cdot \bar{w}$. In practice we want u to be as small as possible to save computational resources. Decreasing the size of u makes the approximation less close, but typically works well despite fairly high collision rates (Ganchev and Dredze, 2008).

We train with a larger than necessary weight vector u followed by repeatedly halving its size until accuracy on held-out data starts to decrease (we do this by simply cutting u in two halves and adding them, so that the new weight vector $u' = u_{1\ldots N/2} + u_{N/2+1\ldots N}$). Empirically, we found this to work as well as optimizing the weight vector length N by re-training at each step, while much less costly as training is only performed once.

4. Redundant feature templates

A naive way to define $h(k)$ would be to construct a string of characters representing the corresponding feature function ϕ_k, for instance "`suffix=ed,tag=VERB`", and then use any function for string hashing to map this into an integer. In most cases the feature functions ϕ are created from templates that generate a number of very similar functions. For instance, with $x = \textit{hinted}$ and $y = \text{NOUN}$ we might have non-zero feature functions with conditions such as these:

$$\begin{array}{l} \text{suffix1} = \textit{d} \quad \wedge \text{tag} = \text{NOUN} \\ \text{suffix2} = \textit{ed} \ \wedge \text{tag} = \text{NOUN} \\ \text{suffix3} = \textit{ted} \wedge \text{tag} = \text{NOUN} \\ \ldots \end{array}$$

A typical hash function over sequences (or trees) of integers works by recursively applying a mixing function $m(a, b)$ that maps integers a and b to another integer in a pseudo-random manner.[1] For the second

[1] The choice of m is arbitrary, but we adapt it from MurmurHash: `https://github.com/aappleby/smhasher`

of the examples above, we might therefore compute $m(\text{suffix2}, m(e, m(d, m(\text{tag}, \text{NOUN}))))$, assuming that suffix2, e, d, tag and NOUN are all symbols that can be represented by integers.

Given the redundancy among these feature functions, it is possible to reduce computation (applications of the mixing function) significantly by sharing subtrees between feature hashes. For instance, the hashes of the three features above can be computed as

$$\begin{aligned} t_1 &= d \\ t_2 &= m(e, t_1) \\ t_3 &= m(t, t_2) \\ t_4 &= m(\text{suffix1}, t_1) \\ t_5 &= m(\text{suffix2}, t_2) \\ t_6 &= m(\text{suffix3}, t_3) \\ t_7 &= m(\text{tag}, \text{NOUN}) \\ h(\phi_1(x,y)) &= m(t_4, t_7) \\ h(\phi_2(x,y)) &= m(t_5, t_7) \\ h(\phi_3(x,y)) &= m(t_6, t_7) \end{aligned}$$

As is standard, only the subset of variables which depends on the current tag label y (in this case t_7) is recomputed when scoring hypotheses with different tags. In addition, we can use other types of redundancy in the feature template to reduce computation, illustrated here by suffixes of varying length (t_4, t_5 and t_6).

5. A practical system

Our main contribution in this work is a practical system, EFSELAB [2], which takes as input a feature template description and produces a sequence learning program based on the structured perceptron (Collins, 2002) using optimized feature hashes as described above. These sequence learning programs can then be trained with actual data to perform sequence labeling tasks such as part of speech tagging or named entity recognition.

A pre-trained Swedish model using extra resources is available, intended as a replacement for Stagger (Östling, 2013). This has also been integrated into an easy-to-use Swedish annotation pipeline, where EFSELAB is used for PoS tagging and named entity recognition, and MaltParser (Nivre et al., 2007) for dependency parsing. Since this pipeline uses additional resources, including a morphological lexicon (Borin and Forsberg, 2009) and the SUC corpus (Källgren, 2006), the error rate (2.0%) is considerably lower than that of the model trained on *only* UD data (3.3%, see Table 1 on the following page).

6. Experiments

We evaluate EFSELAB on 39 languages from the Universal Dependencies 1.3 treebank (Nivre et al., 2016), using the 17-tag universal PoS tagset. These results are compared to

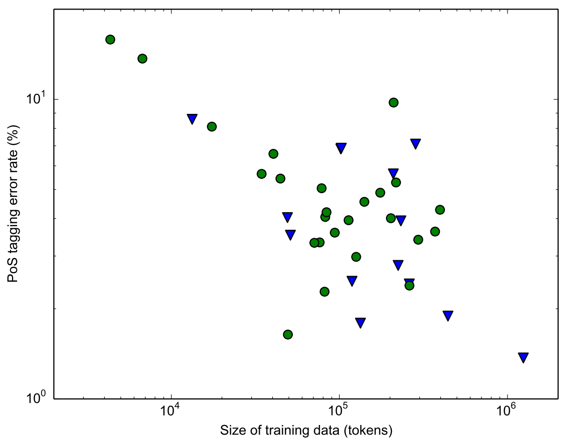

Figure 1: PoS tagging error rate of EFSELAB (green circles) and Plank et al. (2016) (blue triangles) on the UD Treebank 1.3 test sets, showing error rate and training data size. For readability, only the system with the lowest error rate is shown for each language, please refer to Table 1 on the following page for further details.

Plank et al. (2016)[3], who used a model with bidirectional LSTMs on both the character and word level.

Surprisingly, given their very powerful model, EFSELAB performs better on 25 of 39 languages, with a (geometric) mean relative error reduction of 4% over all 39 languages. In other words, the two system would seem to be roughly on par with each other. To some extent this seems to be due to EFSELAB's ability to better handle data sparsity, as there is a moderate correlation (Spearman's $\rho = 0.46$) between training set size and error ratio in favor of Plank et al. This is further illustrated by Figure 1 and Table 1 on the following page.

The difference in computational efficiency, on the other hand, is striking. While EFSELAB achieves a tagging speed of about 7.2 million tokens per second, Plank et al.'s tagger manages 10 700 on the same system.[4]

7. Future work

By reducing the beam size and using greedy search (beam size 1) instead of the default beam size of 4, EFSELAB's performance increases to 13.5 million tokens per second, at the cost of somewhat decreased accuracy (mean error rate increase of 17% on the UD treebank compared to beam size 4). To improve accuracy with small beams, including greedy search, search-based optimization methods such as early updating or LaSO (Daumé and Marcu, 2005) can

[2] `https://github.com/robertostling/efselab` (the experiments reported here can be reproduced with commit `b3f99d1`)

[3] The original article contains only results on version 1.2 of the treebank, with considerably fewer languages, so we instead use their results for version 1.3, which are published at `https://github.com/bplank/bilstm-aux/` (column `i20-h1`)

[4] This was measured using the Swedish UD data with the standard training and test partitioning, on a system with two Intel Xeon E5645 CPUs running at 2.4 GHz, with 12 physical cores in total. 24 parallel processes were used in both cases. For EFSELAB we use the default beam size 4, and for Plank et al.'s tagger model a single LSTM layer.

be used, with no additional test-time cost. This is left for future work.

Acknowledgments

Thanks to Aaron Smith for contributions to the Swedish pipeline integration, and Emil Stenström for contributions to EFSELAB development.

Part of this work has been supported by an infrastructure grant from the Swedish Research Council (SWE-CLARIN, project 821-2013-2003).

Table 1: PoS tagging error rate for EFSELAB and Plank et al. (2016). Best (lowest) value for each language in bold. The rows are sorted by training coprus size (second column, in thousands of tokens).

Language	Size	Error rate (%)	
		EFSELAB	Plank
Kazakh	4	**15.8**	22.3
Tamil	7	**13.7**	15.5
Latvian	13	9.0	**8.6**
Irish	17	**8.1**	9.5
Hungarian	34	**5.6**	6.2
Latin	40	**6.6**	9.8
Turkish	45	**5.4**	6.2
Gothic	49	4.4	**4.0**
Greek	49	**1.6**	2.2
Old Church Slavonic	51	3.6	**3.5**
Swedish	71	**3.3**	3.5
Polish	76	**3.3**	3.7
Basque	78	**5.0**	6.1
Galician	82	**2.3**	3.1
Croatian	82	**4.0**	5.0
Russian	84	**4.2**	4.3
Danish	94	**3.6**	3.9
Indonesian	102	7.1	**6.8**
Chinese	103	8.7	**6.9**
Romanian	113	**4.0**	4.5
Slovenian	119	3.5	**2.5**
Persian	126	**3.0**	3.2
Bulgarian	133	2.1	**1.8**
Hebrew	141	**4.5**	4.8
Finnish	175	**4.9**	5.7
Estonian	202	**4.0**	4.2
Ancient Greek	209	5.9	**5.6**
Dutch	210	**9.8**	10.0
English	217	**5.3**	5.4
Portuguese	224	3.4	**2.8**
Arabic	232	4.0	**3.9**
Norwegian	260	2.6	**2.4**
Italian	261	**2.4**	2.6
German	284	7.3	**7.1**
Hindi	294	**3.4**	3.6
French	371	**3.6**	3.7
Spanish	397	**4.3**	4.8
Catalan	442	2.4	**1.9**
Czech	1242	1.5	**1.4**

References

Lars Borin and Markus Forsberg. 2009. All in the family: A comparison of SALDO and WordNet. In *NODALIDA 2009 Workshop on WordNets and other Lexical Semantic Resources – between Lexical Semantics, Lexicography, Terminology and Formal Ontologies*, pages 7–12, Odense, Denmark.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hal Daumé, III and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 169–176, New York, NY, USA. ACM.

Kuzman Ganchev and Mark Dredze. 2008. Small statistical models by random feature mixing. In *Proceedings of the ACL-2008 Workshop on Mobile Language Processing*. Association for Computational Linguistics.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Gunnel Källgren, 2006. *Manual of the Stockholm Umeå Corpus version 2.0*. Department of Linguistics, Stockholm University, December. Sofia Gustafson-Capková and Britt Hartmann (eds.).

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. In *Association for Computational Linguistics (ACL)*, Berlin, Germany, August.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13:95–135, 6.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Yevgeni Berzak, Riyaz Ahmad Bhat, Cristina Bosco, Gosse Bouma, Sam Bowman, Gülşen Cebirolu Eryiit, Giuseppe G. A. Celano, Çar Çöltekin, Miriam Connor, Marie-Catherine de Marneffe, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Tomaž

Erjavec, Richárd Farkas, Jennifer Foster, Daniel Galbraith, Sebastian Garza, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gokirmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Normunds Grūzītis, Bruno Guillaume, Jan Hajič, Dag Haug, Barbora Hladká, Radu Ion, Elena Irimia, Anders Johannsen, Hüner Kaşkara, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Jessica Kenney, Simon Krek, Veronika Laippala, Lucia Lam, Alessandro Lenci, Nikola Ljubešić, Olga Lyashevskaya, Teresa Lynn, Aibek Makazhanov, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Keiko Sophie Mori, Shunsuke Mori, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Vitaly Nikolaev, Hanna Nurmi, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Lauma Pretkalnia, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Loganathan Ramasamy, Laura Rituma, Rudolf Rosa, Shadi Saleh, Baiba Saulīte, Sebastian Schuster, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Kiril Simov, Aaron Smith, Carolyn Spadine, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Takaaki Tanaka, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Larraitz Uria, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jing Xian Wang, Jonathan North Washington, Zdeněk Žabokrtský, Daniel Zeman, and Hanzhi Zhu. 2016. Universal dependencies 1.3. LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague.

Robert Östling. 2013. Stagger: An open-source part of speech tagger for Swedish. *North European Journal of Language Technology*, 3:1–18.

Barbara Plank, Anders Søgaard, and Yoav Goldberg. 2016. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 412–418, Berlin, Germany, August. Association for Computational Linguistics.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 760–767, Prague, Czech Republic, June. Association for Computational Linguistics.