bot.zen @ EmpiriST 2015 - A minimally-deep learning PoS-tagger (trained for German CMC and Web data)

Egon W. Stemle
EURAC research
Bozen-Bolzano, Italy
`egon.stemle@eurac.edu`

Abstract

This article describes the system that participated in the Part-of-speech tagging subtask of the *EmpiriST 2015 shared task on automatic linguistic annotation of computer-mediated communication / social media*.

The system combines a small assertion of trending techniques, which implement matured methods, from NLP and ML to achieve competitive results on PoS tagging of German CMC and Web corpus data; in particular, the system uses word embeddings and character-level representations of word beginnings and endings in a LSTM RNN architecture. Labelled data (Tiger v2.2 and EmpiriST) and unlabelled data (German Wikipedia) were used for training.

The system is available under the APLv2 open-source license.

1 Introduction

Part-of-speech (PoS) tagging is an essential processing stage for virtually all NLP applications. Subsequent tasks, like parsing, named-entity recognition, event detection, and machine translation, often utilise PoS tags, and benefit (directly or indirectly) from accurate tag sequences. However, frequent phenomena in computer-mediated communication (CMC) and Web corpora such as emoticons, acronyms, interaction words, iteration of letters, graphostylistics, shortenings, addressing terms, spelling variations, and boilerplate (Androutsopoulos, 2007; Bernardini et al., 2008; Beißwenger, 2013) deteriorate the performance of PoS-taggers (Giesbrecht and Evert, 2009; Baldwin et al., 2013).

To this end, the EmpiriST shared task (ST) invited developers of NLP applications to adapt their tokenisation and PoS tagging tools and resources for the processing of written German CMC and Web data (Beißwenger et al., 2016). The ST was divided into two subtasks, tokenisation and PoS tagging, and for each subtask two data sets were provided (see Subsection 4.1.3). The systems were evaluated by the organisers on raw data for the tokenisation subtask, and on unlabelled but pre-tokenised data for the PoS tagging subtask (both on the same approx. 14,000 tokens).

We participated in the PoS tagging subtask of the ST with our new minimally-deep learning PoS-tagger: We combine `word2vec` (`w2v`) word embeddings (WEs) with a single-layer Long Short Term Memory (LSTM) recurrent neural network (RNN) architecture; strictly speaking, `w2v` is *shallow*. Therefore we call the combination with a single hidden layer *minimally-deep*. The sequence of unlabelled `w2v` representations of words is accompanied by the sequence of n-grams of the word beginnings and endings, and is fed into the RNN which in turn predicts PoS labels.

The paper is organised as follows: We present our system design in Section 2, the implementation in Section 3, and its evaluation in Section 4. Section 5 concludes with an outlook on possible implementation improvements.

2 Design

Overall, our design takes inspiration from as far back as Benello et al. (1989) who used four preceding words and one following word in a feed-forward neural network with backpropagation for PoS tagging, builds upon the strong foundation laid down by Collobert et al. (2011) for a NN architecture and learning algorithm that can be applied to various natural language processing tasks, and ultimately is a variation of Nogueira dos Santos and Zadrozny (2014) who trained a NN for PoS tagging, with character-level and WE representations of words.

2.1 Word Embeddings

Recently, state-of-the-art results on various linguistic tasks were accomplished by architectures using neural-network based WEs. Baroni et al. (2014) conducted a set of experiments comparing the popular `w2v` (Mikolov et al., 2013a; Mikolov et al., 2013b) implementation for creating WEs to other distributional methods with state-of-the-art results across various (semantic) tasks. These results suggest that the word embeddings substantially outperform the other architectures on semantic similarity and analogy detection tasks. Subsequently, Levy et al. (2015) conducted a comprehensive set of experiments and comparisons that suggest that much of the improved results are due to the system design and parameter optimizations, rather than the selected method. They conclude that "there does not seem to be a consistent significant advantage to one approach over the other".

Word embeddings provide high-quality low dimensional vector representations of words from large corpora of unlabelled data, and the representations, typically computed using NNs, encode many linguistic regularities and patterns (Mikolov et al., 2013b).

2.2 Character-Level Sub-Word Information

The morphology of a word is opaque to WEs, and the relatedness of the meaning of a lemma's different word forms, i.e. its different string representations, is *not* systematically encoded. This means that in morphologically rich languages with long-tailed frequency distributions, even some WE representations for word forms of common lemmata may become very poor (Kim et al., 2015).

We agree with Nogueira dos Santos and Zadrozny (2014) and Kim et al. (2015) that sub-word information is very important for PoS tagging, and therefore we augment the WE representations with character-level representations of the word beginnings and endings; thereby, we also stay language agnostic—at least, as much as possible—by avoiding the need for, often language specific, morphological pre-processing.

2.3 Recurrent Neural Network Layer

Language Models are a central part of NLP. They are used to place distributions over word sequences that encode systematic structural properties of the sample of linguistic content they are built from, and can then be used on novel content, e.g. to rank it or predict some feature on it. For a detailed overview on language modelling research see Mikolov (2012).

A straight-forward approach to incorporate WEs into feature-based language models is to use the embeddings' vector representations as features. Having said that, WEs are also used in neural network architectures, where they constitute (part of) the input to the network.

Neural networks (NNs) consist of a large number of simple, highly interconnected processing nodes in an architecture loosely inspired by the structure of the cerebral cortex of the brain (O'Reilly and Munakata, 2000). The nodes receive weighted inputs through these connections and *fire* according to their individual thresholds of their shared activation function. A firing node passes on an activation to all successive connected nodes. During learning the input is propagated through the network and the output is compared to the desired output. Then, the weights of the connections (and the thresholds) are adjusted stepwise so as to more closely resemble a configuration that would produce the desired output. After all input cases have been presented, the process typically starts over again, and the output values will usually be closer to the correct values.

RNNs are NNs where the connections between the elements are directed cycles, i.e. the networks have loops, and this enables them to model sequential dependencies of the input. However, regular RNNs have fundamental difficulties learning long-term dependencies, and special kinds of RNNs need to be used (Hochreiter, 1991); a very popular kind is the so called long short-term memory (LSTM) network proposed by Hochreiter and Schmidhuber (1997).

3 Implementation

We maintain the implementation in a source code repository at `https://github.com/bot-zen/`. The version tagged as `0.9` comprises the version that was used to generate the results submitted to the ST. The version tagged as `1.0` is identical at its core but comes with explicit documentation on how to download and install external software, and how to download and pre-process required corpora.

Our system feeds WEs and character-level sub-word information into a single-layer RNN with a LSTM architecture.

3.1 Word Embeddings

We incorporates `w2v`'s original C implementation for learning WEs[1] in an independent pre-processing step, i.e. we pro-compute the WEs. Then, we use gensim[2], a Python tool for unsupervised semantic modelling from plain text, to load the data, and to extract the vector representations of the embedded words as input to our NN.

3.2 Character-Level Sub-Word Information

Our implementation uses a *one-hot encoding* with a few additional features for representing sub-word information. The one-hot encoding transforms a categorical feature into a vector where the categories are represented by equally many dimensions with binary values. We convert a letter to lower-case and use the sets of ASCII characters, digits, and punctuation marks as categories for the encoding. Then, we add dimensions to represent more binary features like *'uppercase'* (was upper-case prior to conversion), *'digit'* (is digit), *'punctuation'* (is punctuation mark), *whitespace* (is white space, except the new line character; note that this category is usually empty, because we expect our tokens to *not* include white space characters), and *unknown* (other characters, e.g. diacritics). This results in vectors with more than a single *one-hot* dimension.

3.3 Recurrent Neural Network Layer

Our implementation uses Keras, a minimalist, highly modular NNs library, written in Python and capable of running on top of either TensorFlow or Theano (Chollet, 2015). In our case it runs on top of Theano, a Python library that allows to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently (The Theano Development Team et al., 2016).

The input to our network are sequences of the same length as the sentences we process. During training we group sentences of the same length into batches. Each single word in the sequence is represented by its sub-word information and two WEs that come from two sources (see Section 4). Unknown words, i.e. words without a WE, are mapped to a randomly generated vector representation once, and this representation is reused later. In Total, each word is represented by $1,800$ features: two times 500 (WEs), and ten times 80 for two 5-grams (word beginning and ending). (If words are shorter than 5 characters their 5-grams are zero-padded.)

This sequential input is fed into a LSTM layer that, in turn, projects to a fully connected output layer with softmax activation function. We use categorical cross-entropy as loss function and backpropagation in conjunction with the RMSprop optimization for learning. At the time of writing, this was the Keras default—or the explicitly documented option to be used—for our type of architecture.

4 Case Study

We used our implementation to participate in the EmpiriST 2015 shared task. First, we describe the corpora used for training, and then the specific system configuration(s) for the ST.

4.1 Training Data for `w2v` and PoS Tagging

4.1.1 Tiger v2.2 (PoS)

Tiger v2.2[3] is version 2.2 of the TIGERCorpus (Brants et al., 2004) containing German newspaper texts. The corpus was semi-automatically PoS tagged, and is one of the standard corpora used for German PoS tagging. It contains 888,238 tokens in 50,472 sentences. For research and evaluation purposes, the TIGERCorpus can be downloaded for free.

4.1.2 German Wikipedia (`w2v`)

de.wiki'15[4] are user talk pages (messages from users to users, often questions and advice), article talk pages (questions, concerns or comments related to improving a Wikipedia article), and article pages of the German wikipedia from 2015, made available by the *Institut für Deutsche Sprache*[5]. The corpus contains 2 billion tokens (talk:379m,

[1]https://code.google.com/archive/p/word2vec/

[2]https://radimrehurek.com/gensim/

[3]http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger.html

[4]http://www1.ids-mannheim.de/kl/projekte/korpora/verfuegbarkeit.html#Download

[5]http://www.ids-mannheim.de

article talk:447m, article:1,1bn) in 79 million sentences (talk:15m, article talk:17m, article:47m), is well-sized for `w2v`, and also (partly) resembles or target data. It is available under the CC BY-SA 3.0[6] license.

4.1.3 EmpiriST 2015 Data (PoS and `w2v`)

empirist[7] is the CMC and Web data made available by the organizers of the ST. It contains data samples from different CMC genres and samples from text genres on the Web. The training corpus contains 10,053 tokens and was PoS tagged by two annotators (unclear cases were decided by a third person). The trial corpus contains around 3,600 tokens (2,100 CMC[8], 1,500 Web) and was PoS tagged by one annotator (without systematic error checks). See Beißwenger et al. (2016) for more details.

4.2 EmpiriST 2015 shared task

For the ST we used one overall configuration for the system, but we used three different corpus configurations for training. Consequently, we participated in the ST with three runs: we used PoS tags from *empirist* (run 1), from *Tiger v2.2* (run 2), and from both (run 3). For `w2v` we trained a 500-dimensional skip-gram model on *empirist* that ignored all words with less than 3 occurrences within a window size of 10; it was trained with negative sampling (value 5) and erroneously[9] also with hierarchical softmax. We also trained a 500-dimensional continuous bag-of-words model on *de.wiki'15* that ignored all words with less than 25 occurrences within a window size of 10; it was trained with negative sampling (value 3) and erroneously also with hierarchical softmax.

[6]Creative Commons Attribution-ShareAlike 3.0 Unported, i.e. the data can be copied and redistributed, and adapted for any purpose, even commercially. See `http://creativecommons.org/licenses/by-sa/3.0/` for more details.

[7]`https://sites.google.com/site/empirist2015/home/shared-task-data`

[8]For evaluation during the development phase we used *empirist*-trial. Unfortunately, we found out only later that the CMC part of the trial data is also part of the training data, i.e. for the CMC data our evaluation data was identical with the training data.

[9]According to `w2v`'s author, technically negative sampling and hierarchical softmax can be combined but one should avoid combining them (see `https://groups.google.com/forum/#!topic/word2vec-toolkit/WUWad9fL0jU`).

We had forgotten to deactivate an option in a data processing script.

The rational behind training the two models differently was that according to `w2v` author's experience[10] a skip-gram model "works well with small amount[s] of the training data, [and] represents well even rare words or phrases", and a cbow model is "several times faster to train than the skip-gram, [and has] slightly better accuracy for the frequent words". The other `w2v` parameters were left at their default settings[11].

To optimize the system's output we ran a simple grid search for three parameters: the hidden LSTM layer's size, the dropout value for the projections from the LSTM to the output layer during training, and the number of epochs during training. The found values were size:1024, dropout:0.1, epochs:20.

	CMC	Web
(1) *empirist*	81.03	86.97
(2) *Tiger v2.2*	73.56	89.73
(3) *empirist+Tiger v2.2*	85.42	90.63
Winning Team	87.33	93.55

Table 1: Official results of our PoS tagger for the three runs on the EmpiriST 2015 shared task data.

5 Conclusion & Outlook

We presented our submission to the EmpiriST 2015 shared task, where we participated in the PoS tagging sub-task with fair results on the CMC data and adequate results on the Web data. Still, our implementation, albeit following state-of-the art designs and methods, is quite unpolished, and can certainly gain performance with more detailed tuning. For example, adding special sequence start and sequence stop symbols to the input is typically done as a pre-processing step, which might improve the results at the beginning and the end of sentences; or we might gain some performance by adding additional hidden layers to enable the network to learn more intermediate abstractions. A more profound design change could also help, e.g. Recurrent Memory Network are a novel recurrent architecture that have been shown to outperform LSTMs on some language modelling tasks. Finally, for learning the word embeddings we

[10]`https://groups.google.com/d/msg/word2vec-toolkit/NLvYXU99cAM/E5ld8LcDxlAJ`

[11]`-sample 1e-3 -iter 5 -alpha 0.025` for skip-gram and `-alpha 0.05` for continuous bag-of-words

could use different corpora, or selectively extract parts from large web-corpora resembling—as much as possible—the type of data that is to be tagged.

References

Jannis K. Androutsopoulos. 2007. Neue Medien – neue Schriftlichkeit? *Mitteilungen des Deutschen Germanistenverbandes*, 1:72–97.

Timothy Baldwin, Paul Cook, Marco Lui, Andrew MacKinlay, and Li Wang. 2013. How noisy social media text, how diffrnt social media sources? In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 356–364, Nagoya, Japan, October. Asian Federation of Natural Language Processing.

Marco Baroni, Georgiana Dinu, and German Kruszewski. 2014. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247. Association for Computational Linguistics.

Michael Beißwenger, Sabine Bartsch, Stefan Evert, and Kay-Michael Würzner. 2016. EmpiriST 2015: A Shared Task on the Automatic Linguistic Annotation of Computer-Mediated Communication, Social Media and Web Corpora. In *Proceedings of the 10th Web as Corpus Workshop (WAC-X)*, Berlin, Germany.

Michael Beißwenger. 2013. Das Dortmunder Chat-Korpus: ein annotiertes Korpus zur Sprachverwendung und sprachlichen Variation in der deutschsprachigen Chat-Kommunikation. *LINSE - Linguistik Server Essen*, pages 1–13.

Julian Benello, Andrew W. Mackie, and James A. Anderson. 1989. Syntactic category disambiguation with neural networks. *Computer Speech & Language*, 3(3):203–217, July.

Silvia Bernardini, Marco Baroni, and Stefan Evert. 2008. A WaCky Introduction. In *Wacky! Working papers on the Web as Corpus*, pages 9–40. GEDIT, Bologna, Italy.

Sabine Brants, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. 2004. TIGER: Linguistic Interpretation of a German Corpus. *Research on Language and Computation*, 2(4):597–620.

Franois Chollet. 2015. Keras: Deep Learning library for Theano and TensorFlow. `https://github.com/fchollet/keras`.

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*, 12:2493–2537.

Eugenie Giesbrecht and Stefan Evert. 2009. Is Part-of-Speech Tagging a Solved Task? An Evaluation of POS Taggers for the German Web as Corpus. *Web as Corpus Workshop (WAC5)*.

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

Sepp Hochreiter. 1991. *Untersuchungen zu dynamischen neuronalen Netzen*. diploma thesis, TU München.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2015. Character-Aware Neural Language Models. *CoRR*, abs/1508.0.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed Representations of Words and Phrases and their Compositionality. *CoRR*, abs/1310.4546, October.

Tomáš Mikolov. 2012. *Statistical Language Models Based on Neural Networks*. Ph.D. thesis, Brno University of Technology.

Cícero Nogueira dos Santos and Bianca Zadrozny. 2014. Learning Character-level Representations for Part-of-Speech Tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826.

Randall C. O'Reilly and Yuko Munakata. 2000. *Computational Explorations in Cognitive Neuroscience Understanding the Mind by Simulating the Brain*. MIT Press.

The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, and et al. 2016. Theano: A Python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688.