The Hunvec Framework For NN-CRF-based Sequential Tagging

Katalin Pajkossy
BUTE, Institute of Mathematics
H-1111 Egry József u. 13-17, Budapest
pajkossy@mokk.bme.hu

Attila Zséder
HAS Research Institute for Linguistics
H-1068 Benczúr u 33, Budapest
zseder@mokk.bme.hu

Abstract

In this work we present the open source hunvec framework for sequential tagging, built upon Theano and Pylearn2. The underlying statistical model, which connects linear CRF-s with neural networks, was used by Collobert and co-workers, and several other researchers. For demonstrating the flexibility of our tool, we describe a set of experiments on part-of-speech and named-entity-recognition tasks, using English and Hungarian datasets, where we modify both model and training parameters, and illustrate the usage of custom features. Model parameters we experiment with affect the vectorial word representations used by the model; we apply different word vector initializations, defined by Word2vec and GloVe embeddings and enrich the representation of words by vectors assigned trigram features. We extend training methods by using their regularized (l2 and dropout) version. When testing our framework on a Hungarian named entity corpus, we find that its performance reaches the best published results on this dataset, with no need for language-specific feature engineering. Our code is available at http://github.com/zseder/hunvec

Keywords: neural networks, sequential tagging, named entity recognition

1. Introduction and related work

Sequential labeling models provide algorithms for NLP tasks of basic importance, including part-of-speech tagging (POS) and named entity recognition (NER). In this work we use a model which connects linear CRF-s with neural networks, introduced in (Collobert and Weston, 2008; Peng et al., 2009; Do and Artieres, 2010). Our implementation is based on (Collobert et al., 2011). This work attracted much attention among NLP researchers, inspiring several adaptations (see (Zheng et al., 2013)) and extensions (see (Santos and Zadrozny, 2014)). The model is capable of utilizing distributed word representations as part of the model's initialization; the effect of using different embeddings has also been studied ((Demir and Ozgur, 2014), (Ling et al., 2015)). Although the results of the previous work are impressive, the number of tools publicly available for researchers is limited. For creating word embeddings there are excellent open source packages, Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014). There are also several low-level libraries for working with deep neural networks, like Theano (Bergstra et al., 2010; Bastien et al., 2012), Torch (Collobert et al., 2002), or the recently published Tensorflow (Abadi et al., 2015). However, working with these frameworks requires solid programming skills and to the best of our knowledge, there is no publicly available, high-level library for sequential tagging, that would let a broader range of researchers investigate the domain. Our framework, hunvec, built upon Theano, attempts to fill this gap.

2. Model and architecture

Our model, following the work of (Collobert et al., 2011), uses their sentence-level training scheme. The log likelihood of a sentence $s_1 s_2 \dots s_T$ having labels indexed by $t_1 t_2 \dots t_T$ is defined as

$$\sum_{i=1}^{T} (A_{[t_i, t_{i+1}]} + f_{\theta, t_i}(\vec{s_i})) - \log \mathcal{Z}_\theta(\vec{s}) \qquad (1)$$

where the trainable elements $A_{[i,j]}$ form a transition matrix A, f_{θ,t_i} scores are given by a neural network's output with trainable parameters θ, $\vec{s_i}$ is the input vector computed for s_i, and $\mathcal{Z}_\theta$ is the partition function which accounts for normalizing the probabilities.

We follow a windowing approach: the network's input, $\vec{s_i}$ is the concatenation of a fixed length sequence (window) surrounding the target word s_i in the sentence, where the units in the sequence are the words, possibly augmented with word features (see Figure 1 for illustration). In the first layer these units are mapped to a continuous vector space, which can be of different dimension for the words and features (practically, the feature space is of much lower dimension). The first layer's output is the concatenation of these vectors, then, transformations defined by regular multilayer perceptron layers follow, and the output layer's (unnormalized) activations correspond to the tag scores. We chose tanh as activation function.

The window size, the dimension of the word and feature representations and the number of hidden layers and neurons are hyperparameters of the model. We can enhance our models by initializing the mapping of words using pretrained word vectors (embeddings). We train the parameters of our model (network weights and transition scores) using stochastic gradient descent.

3. Implementation

As a basis of our implementation, we chose Pylearn2 (Goodfellow et al., 2013), which is a machine learning library for conducting scientific experiments built upon Theano (Bergstra et al., 2010; Bastien et al., 2012). Therefore, we get all the benefits of Theano; the automatic calculation of the gradients will be optimized and stabilized, and the same code can be compiled to GPU if preferred to CPU. One training epoch on a tagged corpora of average size (15-50 thousand sentences) runs for 15-90 minutes on a regular 2-core CPU, and it usually took 15-20 epochs until converged. Tagging the same amount of sentences takes

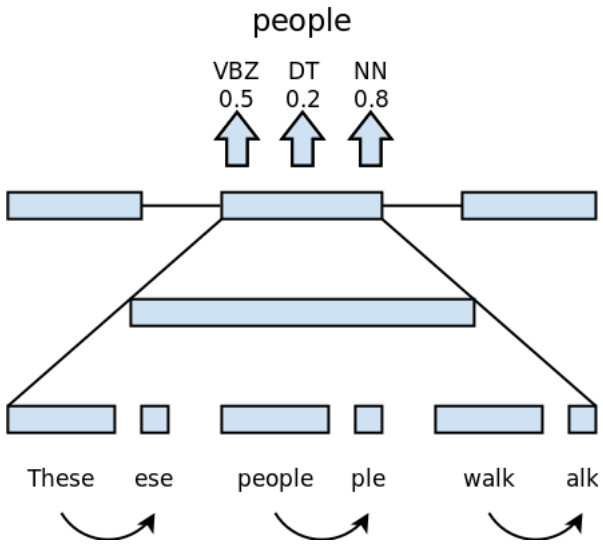

Figure 1: Window-based approach with a 3-length window, feature vectors represent the last trigrams

less than one minute.

We designed `hunvec` in a way that models and training options can be modified with very little effort. Modifying the network architecture (number of hidden layers and neurons), the training parameters (like using momentum, learning rate/decay, regularization), or the input (like changing the window size, replacing the initial word embeddings, or adding new features) are as easy as replacing one line in the code or a command line parameter, so researchers with little programming skill can experiment with it.

In our experiments we use a shallow network architecture with one hidden layer, this can also be replaced by a deep network as in (Do and Artieres, 2010). Since Pylearn2 is used and actively developed by the deep learning community, many of the new methods of deep learning get implemented quickly, and can be used in an easy, plug-and-play fashion, which lets us further develop our tool easily. Some of the methods of training deep architectures suggested in the literature, such as Nesterov momentum (Sutskever et al., 2013) or dropout regularization (Srivastava, 2013) are available as training option; in some of the experiments we present in the next sections we used dropout regularization. We publish our code on github at `http://github.com/zseder/hunvec`, under MIT License.

4. Experiments

Although the library is still rough on the edges, we are continuously experimenting with it. We publish the library in this early phase, because we believe that it can be already useful for researchers and also the development could benefit from their feedback. As a sanity check, we repeated the experiments of (Collobert et al., 2011) on English POS-tagging and NER, using the same experimental setup (explained in Section 4.1., 4.2., and 4.3.). For demonstrating the flexibility of our tool, we also present results of modifying some of the hyperparameters of our model (see Section 4.2. and 4.3.). In order to investigate whether our tool is applicable to different datasets with good results, we also trained models on a Hungarian NER corpus (see 4.1.).

4.1. Datasets and preprocessing

For English POS task we use the Wall Street Journal dataset from Penn Treebank III. (Marcus et al., 1993), and use a canonical split, section 0-18 for training, section 19-21 for validation and section 22-24 for testing.

For English NER we use the CoNNL-2003 benchmark (Reuters) dataset (Tjong Kim Sang and De Meulder, 2003).

For Hungarian NER we use the Szeged NER Corpus (Szarvas et al., 2006a), which is CoNNL-style annotated corpus of 200 thousand tokens. We use the same train-development-test split as the authors of (Szarvas et al., 2006b) and (Varga and Simon, 2007)

We preprocess our corpora by replacing the numerical strings with a common symbol.

4.2. Network and training parameters

Following (Collobert et al., 2011), we use a network architecture having a single hidden layer of 300 units, and use a fixed learning rate, scaled by the respective layer's size. We experiment with applying different regularization methods; $l2$ penalty term and dropout regularization (Srivastava, 2013).

4.3. Pretrained word embeddings and features

As in (Collobert et al., 2011), we use a window size of 5, and use word vectors of 50 and feature vectors of 5 dimensions. In our experiments on English datasets we apply different embeddings as word vector initialization; the embedding published by (Collobert et al., 2011) (which will be referred to as `Senna` in the later sections), and the word vectors published by the `GloVe` project, which were trained on Wikipedia 2014 and Gigaword 5 corpora. For models trained on the Hungarian dataset we use `Word2Vec` vectors, which we trained on the Hungarian Webcorpus (Halácsy et al., 2004) with negative sampling, using the tool's skipgram model. For the feature vectors we use no specific initialization.

Following (Collobert et al., 2011), we use a capitalization feature in all our experiments, which we extend with additional features in some setups. Our additional feature set for the English POS task is the same that was used by Collobert, consisting only of features corresponding to the two-long suffices. As our aim is to demonstrate the flexibility of our tool, we tested our models the Hungarian dataset without any laborious language-specific feature engineering; instead, we chose to use a simple feature set containing three features representing the last three trigrams of the given word. We use this feature set also for the English NER task.

5. Results

Our results on the different datasets are described in Section 5.1., 5.2. and 5.3., and some conclusions we could draw regarding the used word vector initializations and regularization methods can be read in Section 5.4. and 5.5..

For evaluation we use metrics commonly used for these tasks; for evaluating POS tagging we calculate per word

		$l2$	dropout	Collobert
only caps.	no init.	96.80	96.87	96.37
	senna	97.27	97.15	97.20
	glove	97.12	97.09	*
caps. + bigram suffix	no init.	97.05	97.13	*
	senna	**97.33**	97.23	97.29
	glove	97.22	97.15	*

Table 1: Results of English POS tagging (precision)

		$l2$	dropout	Collobert
only caps.	no init.	83.78	83.38	81.47
	senna	88.01	88.31	88.61
	glove	87.70	87.81	*
caps. + trigram (last 3)	no init.	84.05	84.42	*
	senna	88.24	**88.74**	*
	glove	88.33	88.35	*

Table 2: Results of English NER (F1-measure)

precision and for NER the weighted average of the F1 score of each categories.

5.1. English POS tagging task

Table 1 shows our results on this dataset, compared to those reported in (Collobert et al., 2011). As can be seen, results of identical setup are similar, our results of the simplest model being slightly better (see first row). Our best result on this dataset is 97.33% precision, the model in question was trained using `Senna` initialization and $l2$ regularization, and uses bigram suffix features.

5.2. Results on English NER

Table 2 shows our results on English NER. On this dataset we experimented with the setup used in (Collobert et al., 2011) (see first block) and with the addition of simple trigram features (see second block). As the table shows, when using identical setup than reported by Collobert and co-workers, our results are similar (the performance of simplest models being somewhat higher and those using `Senna` initialization slightly lower). The results also show that the usage of the trigram features resulted in a definite boost of performance in every setup. Our best result on this dataset is 88.74 F1-measure, it was reached using `Senna` initialization and dropout regularization, with the usage of trigram features.

5.3. Results on Hungarian NER

For Hungarian NER task the best published results of NER systems trained and tested on Szeged NER Corpus reach 94,77 (Szarvas et al., 2006b) and 95.05 F1-measure(Varga and Simon, 2007). For this task we extended our basic models using `Word2Vec` initialization, and with added trigram features. Our results are shown in Table 3. As can be seen, here the impact of the word vector initialization is less, than on the CoNNL-2003 benchmark dataset, still, it provides a consistent boost of performance, as well as the usage of the trigram features. Our models trained with both `Word2Vec` initialization and trigram features reach the performance of the above two, more sophisticated systems on this dataset (see last row of Table 3).

		$l2$	dropout
only caps.	no init.	94.28	94.15
	word2vec	94.36	94.65
caps.+ trigram	no init.	94.49	94.92
	word2vec	94.88	**95.48**

Table 3: Results of Hungarian NER (F1-measure)

5.4. Effect of dropout regularization

We compared the effect of dropout regularization (with a dropout rate of 0.5 for all weights) versus $l2$ weight decay in all experimental setups. The results are mixed; while on the POS dataset models trained with $l2$ weight decay performed somewhat better in most of the setups, on the other datasets (English and Hungarian NER) dropout regularization resulted in better results, especially in the case where we used additional features.

5.5. Effect of different word vector initializations

On both of our English datasets we compared the effect of using `Senna` and `GloVe` initialization. As the results presented show, models with `Senna` initialization outperformed those initialized with `GloVe` in almost all experimental setups; slightly on the POS and in a larger extent on the NER dataset. However, since these vectors were trained on different datasets, we cannot conclude that the model that generated `Senna` is the better suited to this task, we only compare the utility of the resulting word vectors.

On the Hungarian dataset we experimented with `Word2Vec` initialization; using these vectors which we could train in only a few hours improved our results in all setups.

6. Conclusions and future work

In this work, we presented an open-source library for neural network based sequential tagging, and showed its basic functionalities of it on common NLP tasks. When testing our framework on a standard Hungarian named entity corpus, we found that without much optimization or language-specific feature engineering its performance reaches the best published results on this dataset. In the future we plan to further investigate the effect of different hyperparameters, including training parameters (like tuning the dropout rate), different word vector initializations and feature vectors, and as a furher goal, network depth.

7. Bibliographical References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2015). Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow.org*.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June. Oral Presentation.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA. ACM.

Collobert, R., Bengio, S., and Mariéthoz, J. (2002). Torch: a modular machine learning software library. Technical report, IDIAP.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research (JMLR)*.

Demir, H. and Ozgur, A. (2014). Improving named entity recognition for morphologically rich languages using word embeddings. In *13th International Conference on Machine Learning and Applications, ICMLA 2014, Detroit, MI, USA, December 3-6, 2014*, pages 117–122.

Do, T.-M.-T. and Artieres, T. (2010). Neural conditional random fields. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9. JMLR: W&CP, 5.

Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., and Bengio, Y. (2013). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*.

Halácsy, P., Kornai, A., Németh, L., Rung, A., Szakadát, I., and Trón, V. (2004). Creating open language resources for Hungarian. In *Proc. LREC2004*, pages 203–210.

Ling, W., Dyer, C., Black, A., and Trancoso, I. (2015). Two/too simple adaptations of word2vec for syntax problems.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330.

Mikolov, T., Yih, W.-t., and Geoffrey, Z. (2013). Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT 2013*, pages 746–751.

Peng, J., Bo, L., and Xu, J. (2009). Conditional neural fields. In Y. Bengio, et al., editors, *Advances in Neural Information Processing Systems 22*, pages 1419–1427. Curran Associates, Inc.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*.

Santos, C. D. and Zadrozny, B. (2014). Learning character-level representations for part-of-speech tagging. In Tony Jebara et al., editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826. JMLR Workshop and Conference Proceedings.

Srivastava, N. (2013). Improving Neural Networks with Dropout. Master's thesis, University of Toronto, Toronto, Canada, January.

Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta et al., editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings.

Szarvas, G., Farkas, R., Felföldi, L., Kocsor, A., and Csirik, J. (2006a). A highly accurate named entity corpus for hungarian. In *Proceedings of International Conference on Language Resources and Evaluation*.

Szarvas, G., Farkas, R., and Kocsor, A. (2006b). A multilingual named entity recognition system using boosting and C4.5 decision tree learning algorithms. In *Discovery Science, 9th International Conference, DS 2006, Barcelona, Spain, October 8-10, 2006, Proceedings*, pages 268–278.

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In Walter Daelemans et al., editors, *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada.

Varga, D. and Simon, E. (2007). Hungarian named entity recognition with a maximum entropy approach. *Acta Cybern.*, 18(2):293–301, February.

Zheng, X., Chen, H., and Xu, T. (2013). Deep learning for chinese word segmentation and pos tagging. In *EMNLP*, pages 647–657. ACL.