End-to-End neural dependency parsing

(Parsing zależnościowy za pomocą sieci neuronowej)

Michał Zapotoczny

Praca magisterska

Promotor: dr Jan Chorowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

5 kwietnia 2017

Michał Zapotoczny

. . . .

. . . .

(adres zameldowania)

. . . .

. . . .

(adres korespondencyjny)

PESEL:	. . . .
e-mail:	. . . .

Wydział Matematyki i Informatyki
stacjonarne studia II stopnia

kierunek:	informatyka
nr albumu:	248100

Oświadczenie
o autorskim wykonaniu pracy dyplomowej

Niniejszym oświadczam, że złożoną do oceny pracę zatytułowaną *End-to-End neural dependency parsing* wykonałem samodzielnie pod kierunkiem promotora, dr. Jana Chorowskiego. Oświadczam, że powyższe dane są zgodne ze stanem faktycznym i znane mi są przepisy ustawy z dn. 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity: Dz. U. z 2006 r. nr 90, poz. 637, z późniejszymi zmianami) oraz że treść pracy dyplomowej przedstawionej do obrony, zawarta na przekazanym nośniku elektronicznym, jest identyczna z jej wersją drukowaną.

Wrocław, 5 kwietnia 2017

(czytelny podpis)

Abstract

. . .

. . .

Contents

Chapter 1

Neural dependency parser

1.1 Overview of the network architecture

The network architecture consists of three main parts: reader, tagger and parser (see Figure 1.1). The reader subnetwork is evaluated on each individual word in a sentence, and using convolutions on their orthographic representation produces words embeddings. Next, the tagger subnetwork implemented as bidirectional RNN equips each word with a context of whole sentence. Finally parser part computes dependency tree parent for each word using attention mechanism [Vinyals et al., 2015] after which network computes appropriate dependency label. In the following paragraphs we will describe all parts in detail.

1.1.1 Reader

As stated before the reader subnetwork is run on each word producing its embedding. This architecture is based on [Kim et al., 2015]. Each word w is represented by sequence of its characters plus a special beginning-of-word and end-of-word tokens. Firstly we find low-dimensional characters embeddings which are concatenated to form a matrix C^w. Then we run 1D convolutional filters on C^w which then is reduced to vector of filter responses computed as:

$$R_i^w = \max(C^w * F^i) \tag{1.1}$$

Where F^i is i-th filter. The purpose of the convolutions is to react to specific part of words (because we have bow and eow tokens it can also react to prefixes and suffixes) which in morphologically rich languages such as Polish can depict its grammatical role.

Finally we transform filter responses R^w with a simple multi-layer perceptron [1] obtaining the final word embedding $E^w = \mathrm{MLP}(R^w)$.

[1]Which are just linear transformations followed be non-linearity

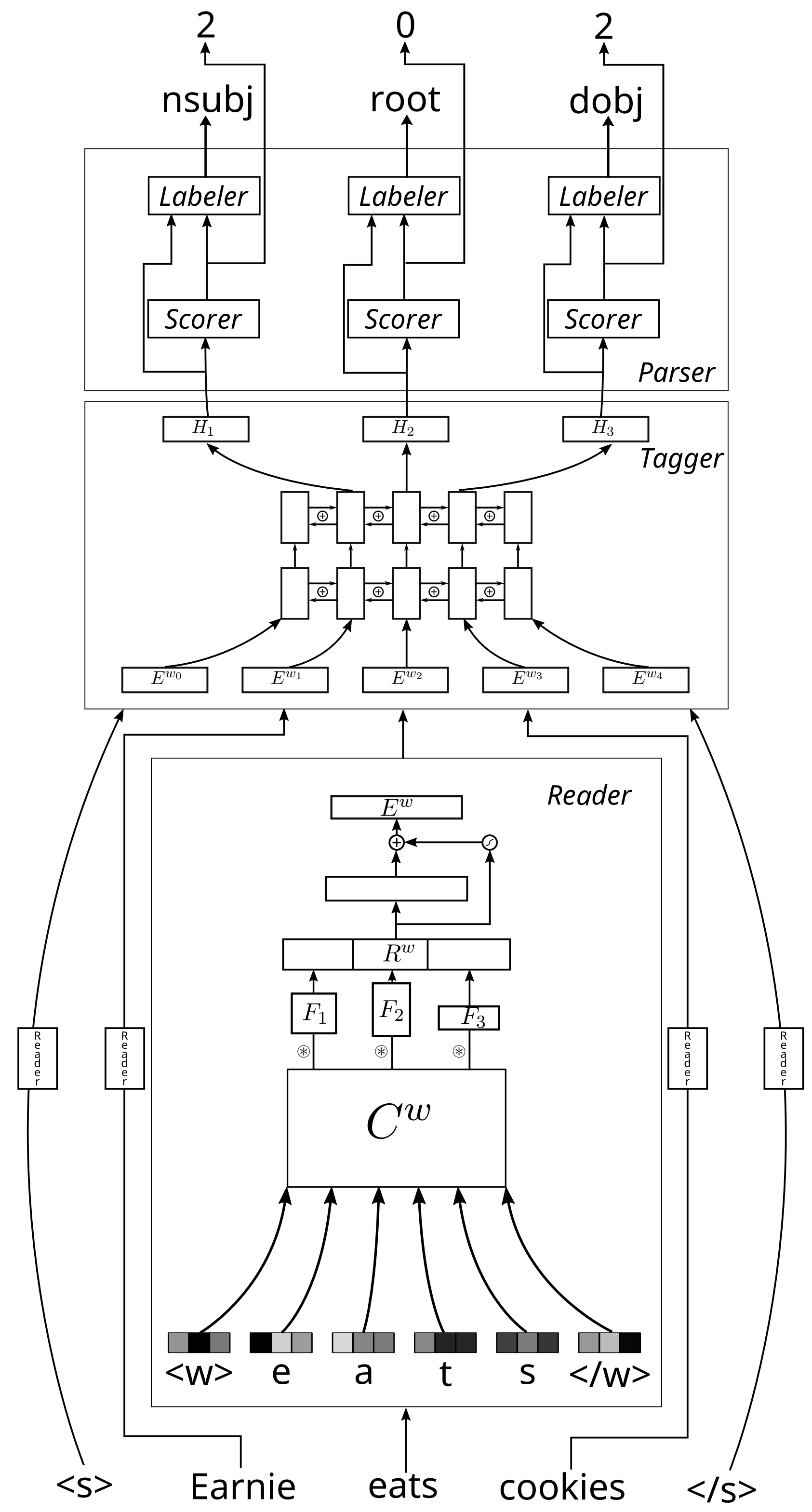

Figure 1.1: The model architecture.

1.1.2 Tagger

Having obtained the word embeddings E^w we can proceed with actually "reading" whole sentence. To do this we use multi-layer bidirectional RNN ([Schuster and Paliwal, 1997]) (we have evaluated LSTM[Hochreiter and Schmidhuber, 1997] and GRU[Cho et al., 2014] types). We combine the backward and forward passes by adding them.

POS tag predictor

To prevent overfitting and encourage network to compute morphological features we can add additional training objective. It works by taking output from one of the tagger BiRNN layers and use it to predict available part-of-speech tags for each word. The result is not feeded back to the rest of the network because we think it would introduce too much noise.

1.1.3 Parser

The last part of the network serves two purposes: to find head for each word and to compute label for that edge.

Finding head word

We use method similar to [Vinyals et al., 2015]. The input to this part are word annotations $H_0, H_1, \ldots, H_n$ (where H_0 serves as a root word) produced by the tagger. For each of the words $1, 2, \ldots, n$ we compute probability distribution which tell us where the head of current word should be $(0, 1, 2, \ldots, n)$. This computation (called *scorer*) is implemented as small feedforward network $s(w, l) = f(H_w, H_l)$, where $w \in 1, 2, \ldots, n$, $l \in 0, 1, 2, \ldots, n$.

Finding edge label

Output of the scorer can already be interpreted as pointer network, but in order to use it as attention for computing dependency label we have to normalize it:

$$p(w, l) = \sum_{i=0}^{n} \frac{f(H_w, H_l)}{f(H_w, H_i)} \tag{1.2}$$

The dependency label is computed by small Maxout network [Goodfellow et al., 2013], which takes the current word annotation H_w and heads annotation A_w. This part is called *labeler*. We investigated two variations of head annotation A_w

Soft attention
Which is a weighted average of normalized attention 1.2 and words annotations H.

$$A_w = \sum_{i=0}^{n} p(w,i) H_i$$

Hard attention
Here we use a only head annotation.

$$A_w = H_h$$

During training we use ground-truth head location, whereas during evaluation we use head word computed in previous step.

Decoding algorithm

The *scorer* give us a $nxn + 1$ matrix of head dependency preference for each word. From this matrix we have find a set of dependencies that satisfy some constraints (exactly one root dependant, no cycles). We investigated two possibilities for such decoding: a greedy algorithm, and Chu-Liu-Edmonds [Edmonds, 1966].

1.2 Training

1.2.1 Training criterion

Every neural network needs a training criterion which will be optimized. Here we have 3 individual training criterion combined together as linear combination. Those are:

- The negative log-likelihood loss L_h on finding proper head for each word. The training signal is propagated from scorer down to the reader.
- The negative log-likelihood loss L_l on finding dependency label. With soft attention it is propagated through the whole network (excluding pos tagging part), while with the hard attention we do not propagate through the scorer.
- The (optional) negative log-likelihood loss L_t on predicting pos tags. This error is backpropagated only through pos-predictor and part of tagger down to the reader.

So the final loss is:

$$L = \alpha_h L_h + \alpha_l L_l + \alpha_t L_t \tag{1.3}$$

1.2.2 Regularization, optimizer and hyperparameters

Czy należy wszystko opisywać? weight decay, adadelta, dropout, spearmint itd.

Regularization is a essential part of neural network training. It improves generalization and prevents overfitting. One of the most popular regularization technique is called Dropout [Srivastava et al., 2014]. Using it, we randomly drop part of the connections from a certain network layer during training. In our case dropout is applied to the *reader* output, between the BiRNN layers of the *tagger* and to the *labeller*.

The models are trained using Adedelta[Zeiler, 2012] learning rule, with weight decay and adaptive gradient clipping [Chorowski et al., 2014]. All experiments are early stopped on validation set UAS.

Hyperparameter selection is crucial for neural networks to obtain good results. For example, comparing our first experiments with architecture depicted above to the best single-language results (using the same basic architecture) gave us about 5% boost in UAS score on Polish language.

To find the best hyperparameters we have used the Spearmint system [Snoek et al., 2012] invoked on polish dataset. The chosen parameters are as follows. The *reader* embeds each character into 15 dimensions, and contains 1050 filters (50·k filters of length k for k = 1, 2,. . . , 6) whose outputs are projected into 512 dimensions transformed by a 3 equally sized layers of feedforward neural network with ReLU activation. The *tagger* contains 2 BiRNN layers of GRU units with 548 hidden states for both forward and backward passes which are later aggregated using addition. Therefore the hidden states of the tagger are also 548-dimensional. The *POS tag predictor* consists of a single affine transformation followed by a SoftMax predictor for each POS category. The *scorer* uses a single layer of 384 tanh for head word scoring while the *labeller* uses 256 Maxout units (each using 2 pieces) to classify the relation label [Goodfellow et al., 2013]. The training cost used the constants $\alpha_h = 0.6, \alpha_l = 0.4, \alpha_t = 1.0$. We apply 20% dropout to the *reader* output, 70% between the BiRNN layers of the *tagger* and 50% to the *labeller*. The weight decay is 0.95.

1.3 Dataset

UD v1.3

1.4 Multilingual training

The big problem of learning to parse is small number of gold standard dependency trees for many languages (including Polish). With small number of examples neural networks do not generalize well and can more easily overfit. There also exist languages with good, standardized treebank like Czech Prague Treebank[Bejček et al., 2013].

Because our model is purely neural network we can incorporate a multitask learning [Caruana, 1997]. It allows the network to learn multiple tasks at the same time, sharing common patterns and distinguishing differences. Additionally, because we are using Universal Dependencies treebanks [Nivre et al., 2015] we have common standardized format across many languages, which allowed for easy implementation of multitask learning and experiments across different languages.

The multilingual model for n languages can be viewed as n copies of our basic model, but sharing part of the parameters. To unify input/output for each of the models we sum possible outputs for each data category (characters, POS categories, dependency labels). If some category doesn't exist within a particular language then we use a special UNK token. To actually make use of multitask learning we must share at least part of the parameters of all models. We experimented with different sharing strategies , from share-everything to only sharing the *parser* part. Additionally to prevent over-representation of some languages during training we sample (on each epoch) only portion of the available data so that each language have equal number of examples (equal to number of samples of the smallest language).

Chapter 2

Results

2.1 Single language

Our basic results for subset of UD v1.3[Nivre et al., 2015] as shown in table **??**. We compare ourselves to SyntaxNet and ParseySaurus, ... predictor, reader impact

Table 2.1: Baseline results of models trained on single languages from UD v1.3. Our models use only the orthographic representation of tokenized words during inference and can work without a separate POS tagger.

language	#sentences	Ours		SyntaxNet		ParseySaurus	
		UAS	LAS	UAS	LAS	UAS	LAS
Czech	87 913	**91.41**	**88.18**	89.47	85.93	89.09	84.99
Polish	8 227	90.26	85.32	88.30	82.71	**91.86**	**87.49**
Russian	5 030	83.29	79.22	81.75	77.71	**84.27**	**80.65**
German	15 892	82.67	76.51	79.73	74.07	**84.12**	**79.05**
English	16 622	87.44	83.94	84.79	80.38	**87.86**	**84.45**
French	16 448	**87.25**	**83.50**	84.68	81.05	86.61	83.1
Ancient Greek	25 251	**78.96**	**72.36**	68.98	62.07	73.85	68.1

2.1.1 Soft vs Hard attention

2.1.2 Decoding algorithm

2.1.3 Word pieces

2.1.4 Pointer softening

2.1.5 Recurrent state size

2.2 Multilanguage

2.3 Error analysis

Bibliography

[Bejček et al., 2013] Bejček, E., Hajičová, E., Hajič, J., Jínová, P., Kettnerová, V., Kolářová, V., Mikulová, M., Mírovský, J., Nedoluzhko, A., Panevová, J., Poláková, L., Ševčíková, M., Štěpánek, J., and Zikánová, Š. (2013). Prague dependency treebank 3.0.

[Caruana, 1997] Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1):41–75.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

[Chorowski et al., 2014] Chorowski, J., Bahdanau, D., Cho, K., and Bengio, Y. (2014). End-to-end Continuous Speech Recognition using Attention-based Recurrent NN: First Results. *arXiv:1412.1602 [cs, stat]*. 00000 arXiv: 1412.1602.

[Edmonds, 1966] Edmonds, J. (1966). Optimim Branchings. *JOURNAL OF RESEARCH of the National Bureau of Standards - B.*, 71B(4):233–240. 00000.

[Goodfellow et al., 2013] Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout Networks. In *ICML*, pages 1319–1327. 00106.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Kim et al., 2015] Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2015). Character-aware neural language models. *arXiv preprint arXiv:1508.06615*. 00011 bibtex: kim_character_2015a.

[Nivre et al., 2015] Nivre, J. et al. (2015). Universal Dependencies 1.2. *http://universaldependencies.github.io/docs/*.

[Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681. 00157.

[Snoek et al., 2012] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959. 00414.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958. 00282.

[Vinyals et al., 2015] Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2674–2682. 00010.

[Zeiler, 2012] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. *arXiv:1212.5701 [cs]*. 00017 arXiv: 1212.5701.