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

14 marca 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, 14 marca 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 beggining-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 $E^w = \text{MLP}(R^w)$.

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

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 tagger

To prevent overfitting and encourage network to compute morphological features we can add additional training objetive. 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 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 wariations of heaed 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.

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 criterions combined together as linear combination. Those are:

- The negative log-likelihood loss L_h on finding proper head for each word. The training singnal 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-tagger 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 Hyperparameters

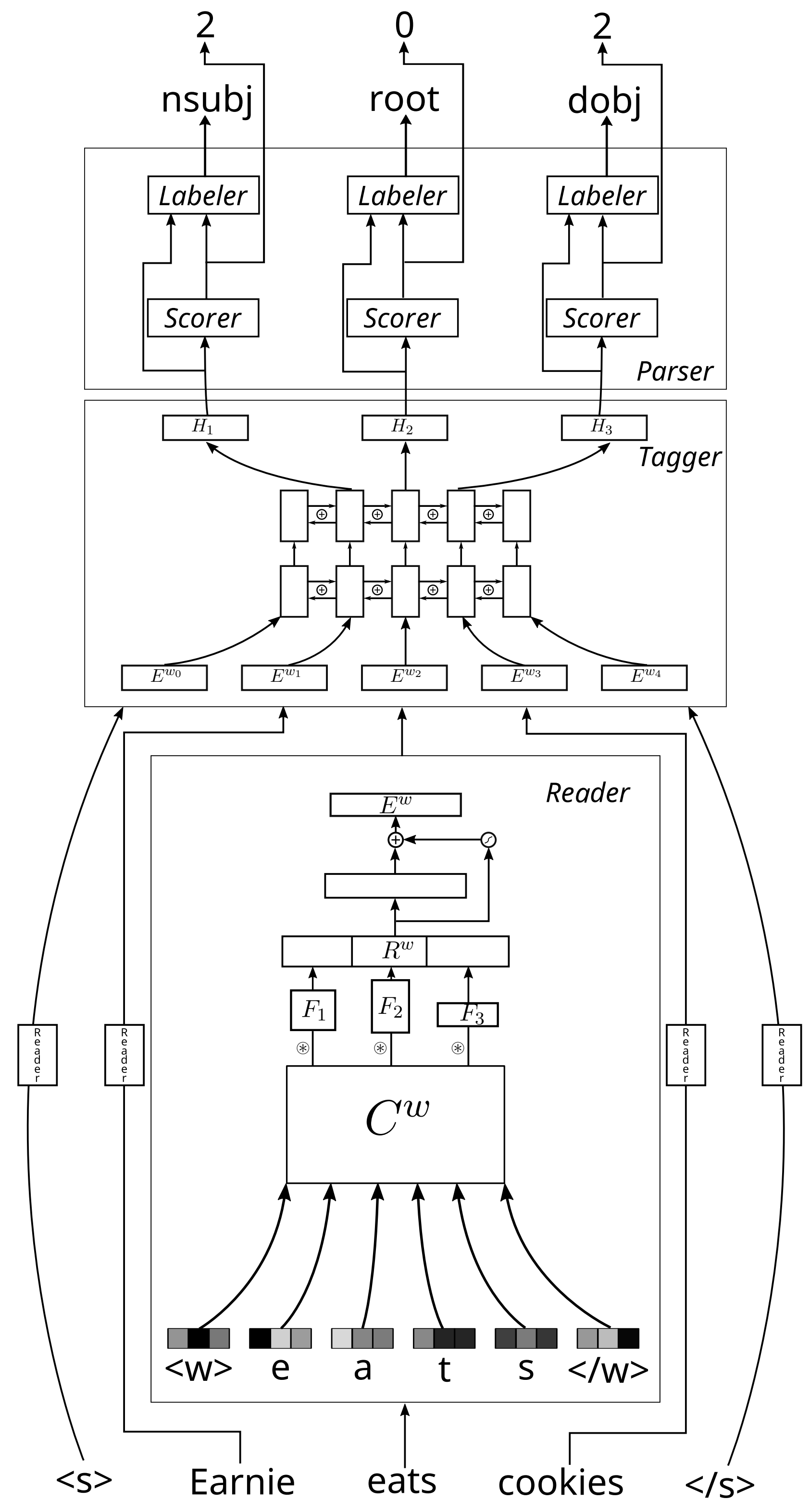

Figure 1.1: The model architecture.

Bibliography

[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.

[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.

[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.

[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.