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

27 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, 27 kwietnia 2017

(czytelny podpis)

Abstract

. . .

. . .

Contents

Chapter 1

Introduction

The ability to communicate with the user in a natural language is a major driving force in development of natural language processing algorithms. One of the basic tasks in a NLP pipeline is parsing by which we can describe sentence structure. There exist two basic parsing techniques: constituency and dependency parsing. With constituency parser we break the sentence into phrases, which can be furher broken into smaller sub-phrases. Example of such parsing is shown in figure 1.1.

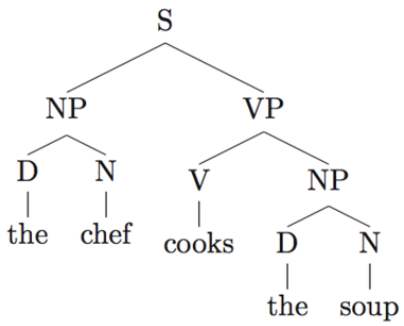

Figure 1.1: A sample constituency parse treeściągnięte z internetu, przegenerować wektorowo

In dependency parsing each word (called *dependent*) is connected via labelled arc to another word of the sentence (called *head*) or to the special $ROOT$ vertice, forming a directed tree. Example of such parsing is depicted on figure 1.2.

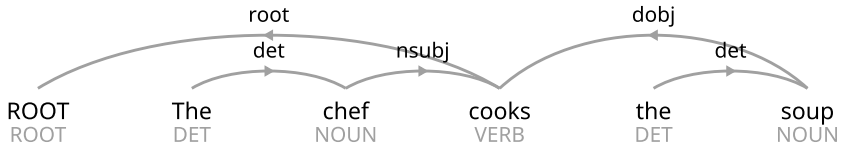

Figure 1.2: A sample dependency parse tree

1.1 Dependency parsing

A major advantage of dependency parsers over constituency ones are their independence from the word ordering. It is important in morphologically rich languages like Polish or Czech where the ordering is very flexible, and thus related words can be far apart from each other. Additionaly the head-dependent relationship is a good approximation to semantic relationship between words [Covington, 2001] which is important for tasks like question answering or information extraction.

The labels of head-dependent arcs tells us about grammatical function that dependent word have in respect to the head. In table 1.1 we show some of the most popular labels for the english language.

Label	Description	Example
CASE	Case marking	*From* Friday 's Daily **Star**
NSUBJ	Nominal subject	*Musharraf* **calls** the bluff
NMOD	Nominal modifier	India **defensive** over Sri *Lanka*
DET	Determiner	That he missed *a* **physical** ?
DOBJ	Direct object	Did you **know** *that* ?
ADVMOD	Adverbial modifier	*So* Bush **stopped** flying .
AMOD	Adjectival modifier	Six weeks of *basic* **training** .
COMPOUND	Compound	Bush 's *National* **Guard** years

Table 1.1: Most popular labels from English treebank. Dependents are *italic*, while heads are **bold**

Projectivity

We can distinguish two types of dependency trees: projective and nonprojective. We say that head-dependent arc is projective is there exists a path from head to every word between head and dependent in the sentence. A dependency tree is projective if all its arcs are projective. Intuitively we can say that nonprojective trees are trees that cannot be drawn without crossing the edges (see figure 1.3). Altough English dataset has less than 5% of nonprojective sentences other languages can have significant amount - like Czech (12%) or Ancient Greek (63%) [Straka et al., 2015]. It is important to acknowledge this fact because some of the presented parsing methods can only produce projective trees.

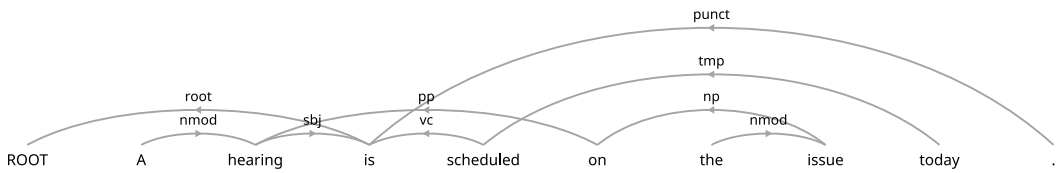

Figure 1.3: A sample nonprojective dependency parse tree. The on → hearing arc is nonprojective and thus the whole tree becomes nonprojective.

1.1.1 Parsing algorithms

In following section we will present two basic approaches to dependency parsing. Although the differ in complexity and output flexibility, both of them depends on supervised machine learning techniques.

Transition based

In transition based parsing we find a transition sequence from initial configuration to terminal one. A configuration c is a triple (s, b, A) containing stack s, buffer b and set of dependency arcs A. For given sentence $w_1, \cdots w_n$ the corresponding initial configuration would be $([\text{ROOT}], [w_1, \cdots w_n], \emptyset)$. The terminal configuration is $(\text{ROOT}, [], A)$ in which A is resulting parse tree. There are three possible transitions:

- `LEFTARC(l)` - Pop from the stack elements s_1 and s_2, add arc $s_1 \rightarrow s_2$ with label l to the set A and push s_1 back to the stack
- `RIGHTARC(l)` - Pop from the stack elements s_1 and s_2, add arc $s_2 \rightarrow s_1$ with label l to the set A and push s_2 back to the stack
- `SHIFT` - Move first element from the buffer to the stack

An example of parsing sequence is shown in table 1.2.

Stack	Buffer	Transition	New arcs
[ROOT]	[He has good control]	`SHIFT`	-
[ROOT He]	[has good control]	`SHIFT`	-
[ROOT He has]	[good control]	`LEFTARC(nsubj)`	nsubj(has,He)
[ROOT has]	[good control]	`SHIFT`	-
[ROOT has good]	[control]	`SHIFT`	-
[ROOT has good control]	[]	`LEFTARC(amod)`	amod(control, good)
[ROOT has control]	[]	`RIGHTARC(dobj)`	dobj(has, control)
[ROOT has]	[]	`RIGHTARC(root)`	root(ROOT, has)
[ROOT]	[]	-	-

Table 1.2: Transitions neede to parse sentence "He has good control"

Because every word of the sentence has be shifted to the stack exactly once and `ARC` operations reduce stack size by one we will have exactly $2n$ transitions from initial to terminal configuration, so the time complexity of this algorithm is $O(n)$ where n is number of words in a sentence. Using this algorithm we can represent any projective tree [Nivre, 2008] (nonprojectives cannot be represented).

The decision which transition to use between consecutive configurations is obtained through machine learning techniques. This can vary from simple linear SVM [Nivre et al., 2005] to neural networks [Chen and Manning, 2014, Andor et al., 2016].

Graph based

Alternative approach is to use graph-based algorithms. The idea is that we define a space of candidate arcs, find a model to score each arc and then use some parsing algorithm to find dependency tree with highest score.

The candidate space and scoring depends on the used learning model, but we can distinguish two main parsing algorithms: Chu-Liu-Edmonds and Eisner.

Chu-Liu-Edmonts Tutaj opis cle

Eisner Tutaj opis eisnera

1.1.2 Evaluation of parsing algorithms

Having obtained dependency tree for a particular sentence we have to be able to compare it with gold-standard parse. There are two main evaluation metrics:

- **Unlabelled Attachnment Score** (UAS), is a percentage of words with correct predicted head
- **Labelled Attachnment Score** (LAS), is a percentage of words with correct predicted head **and** correct predicted label

In the experimental section we will show both scores, whenewer available.

1.1.3 Universal Dependencies

Up till recently most development of dependency parsers was done in single language setting, where we have different parser for every language we want to use, each using language-specific morphosyntactical features. The Universal Dependencies project [Nivre et al., 2015] aims to provide a cross-linguistically consistent treebank annotation. It is based on previous work on universal Stanford Dependencies [De Marneffe et al., 2006], Google universal pos tags [Petrov et al., 2011] and the Interset interlingua [Zeman, 2008]. The UD project unifies part of speech tags and dependency relations labels for 30+ languages under common CONNL-U format. The main advantage is that we can use the same parsing system for every language (data has consistent format) additionaly because the morphosyntactical data has common format we can try to combine data coming from different languages (see section 2.3).

1.2 Neural Networks

A neural network is a mathematical model loosely based on how brain works. The models proven to be very flexible and obtained state of the art results in many tasks (cytaty). A neural network is defined by its topology, activation functions and parameters. The first two are choosen by the author, while the parameters are obtained by learning procedure.

The topology of the network is telling us how the layers of neurons are connected to each other. Each layer have a number of neurons which have some incoming indicies i, associated activation function ϕ and some parameters w. The neuron combines the incoming signals, usually by taking weighted average and then applying the activation function: $\phi(\sum_{k=0}^{|i|} w_k i_k)$. This value is then feeded to the connected neurons from the next layer.

The activation function can have any form, but the most common are hyperbolic tangent tanh and ReLu [Nair and Hinton, 2010].

By the learning procedure we understand a optimization process that given a loss function L, data x and some parameters W to tune, tries to minimize $L(x)$. Because usually the data x size is big we cannot use simple gradient descent algorithm on all data at once because it would not fit into the memory. Instead we are using stochastic gradient descent to train only on small subset of data at once (this subset is called a minibatch).

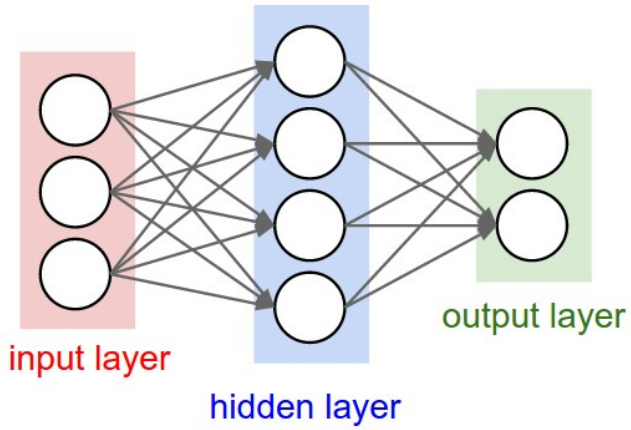

Figure 1.4: An sample neural network architectureinternet...

Chapter 2

Neural dependency parser

2.1 Overview of the network architecture

The network architecture consists of three main parts: reader, tagger and parser (see Figure 2.1). 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.

2.1.1 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:

$$R_i^w = \max(C^w * F^i) \tag{2.1}$$

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

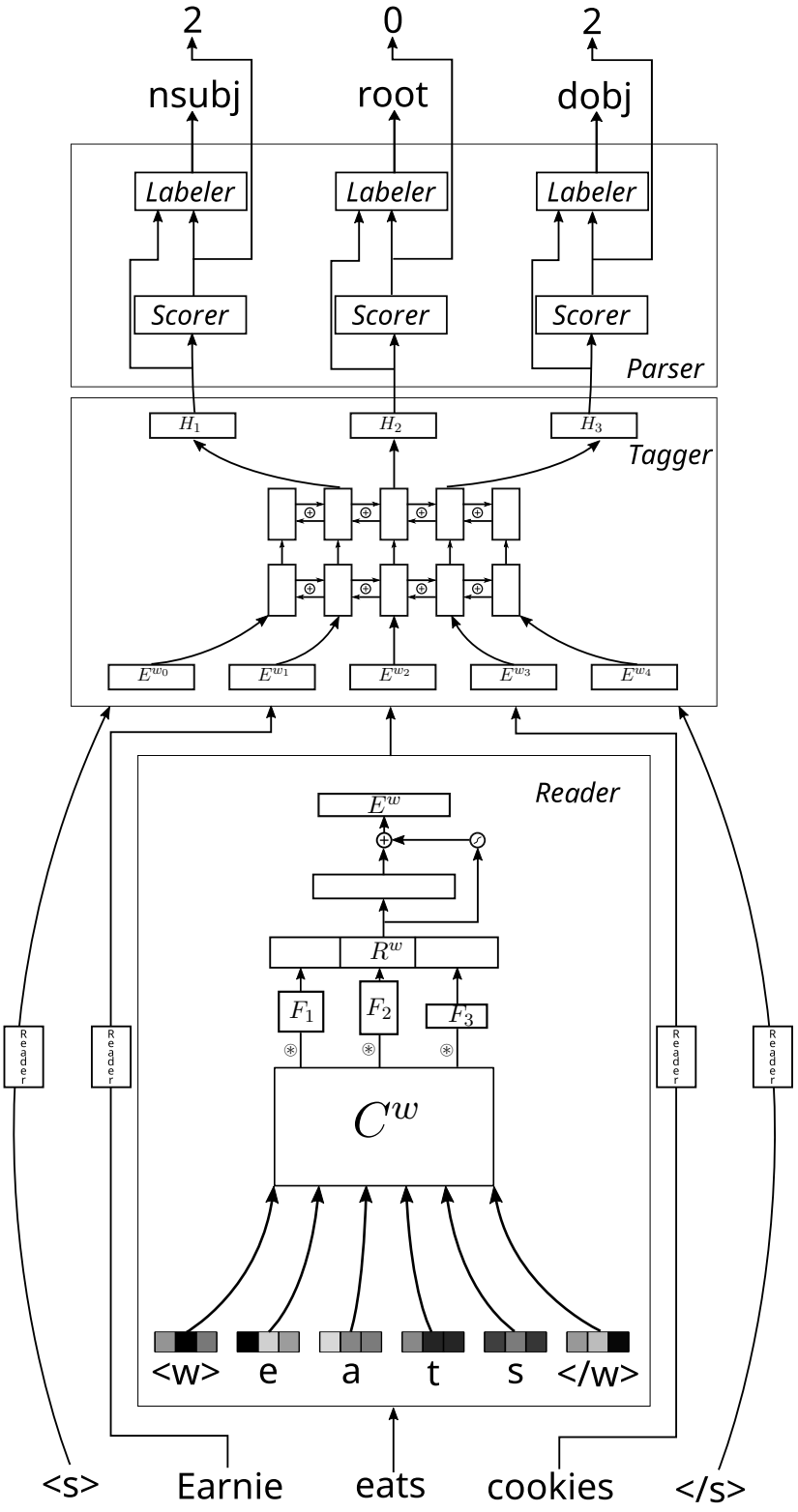

Figure 2.1: The model architecture.

2.1.2 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.

2.1.3 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:

$$p(w, l) = \sum_{i=0}^{n} \frac{f(H_w, H_l)}{f(H_w, H_i)} \tag{2.2}$$

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 2.2 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].

2.2 Training

2.2.1 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:

$$L = \alpha_h L_h + \alpha_l L_l + \alpha_t L_t \tag{2.3}$$

2.2.2 Regularization, optimizer and hyperparameters

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.

2.3 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 3

Results

3.1 Single language

All experiments depicted in this section are based on configuration shown in section 2.2.2. In table 3.1 we show our baseline results for subset of the UD languages (due to limited computational resources we were not able to train models for all of them). The results are compared to SyntaxNet[Andor et al., 2016] and ParseySaurus[Alberti et al., 2017], both from Google. The ParseySaurus is based on SyntaxNet, and uses characters as input, while the original SyntaxNet uses word embeddings. Both are transition-based, and needs no external POS tagger. It is worth noting that SyntaxNet has different hyperparameters configuration for each language, while our parser uses the same configuration across all languages.

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

Table 3.1: 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 works without a separate POS tagger.

In the following sections we will show impact of different network parameters on the result.

3.1.1 Impact of *reader* and *predictor*

Firstly we will inspect the impact of different *reader* and *predictor* settings.

training inputs	Czech		English		Polish	
	UAS	LAS	UAS	LAS	UAS	LAS
Gold POS tags						
base word	**91.7**	**88**	**88.6**	85.1	**93.4**	**89.3**
Predicted POS tags or no POS tags						
words	82.4	72.1	81.9	74.7	74.6	61.6
chars, soft att.	**90.1**	85.7	86.5	82.1	89.1	82.5
chars, tags, soft att.	89.6	82.8	86.2	81.3	90.4	83.9
chars, tags, hard att.	**90.1**	**86.7**	**87.6**	**83.6**	**91.3**	**86**

Table 3.2: Model performance on selected languages and different training inputs. Experiments were run on **UD v 1.2**

In the upper part "Gold POS tags" we show a favorable situation when we have access to gold pos tag (approved by human). The inputs are base words embedded in a vector - basically instead of whole *reader* part we have just simple table lookup. This setting allowed us to obtain best results (especially for Polish) but gold pos tag are not available in real-word setting so this result is noted just as a trivia.

The words and chars, soft att. settings don't use *predictor*. Using just words as inputs (embedded in the same manner as base word) gives worst results. This is easy to predict because most words in the dataset occurs only once and so the network doesn't have a clue about role of most of the words. Instead, using character-level input allows us to obtain much better results, especially because in morphosyntactically rich languages the word spelling catches many word features that can be used in deducing the pos tag information.

The result of adding a *predictor* depends on attention type used. Used with soft attention it decreased the results in Czech and English, but increased in Polish. When hard attention is used it can obtain the best results in all cases. The intuition behind these results are that with hard attention the labbeler only sees annotation of the choosen head, and the error is backpropagated only to this element. On the other hand with soft attention the head annotation may refer to many incorrect words introducing more noise. Additionally the soft attention labeller error is backpropagated through all words, which can introduce even more noise.

3.1.2 Impact of decoding algorithm

In the table 3.3 we show the comparison of greedy and Chu-Liu-Edmonds decoding algorithms.

language	Greedy UAS	Edmonds UAS
Czech	**91.41**	91.40
Polish	**90.26**	90.17
Russian	**83.29**	83.23
German	**82.67**	82.66
English	**87.44**	87.41
French	**87.25**	87.18
Ancient Greek	**78.96**	78.92

Table 3.3: The comparison of greedy and edmonts decoding algorithms

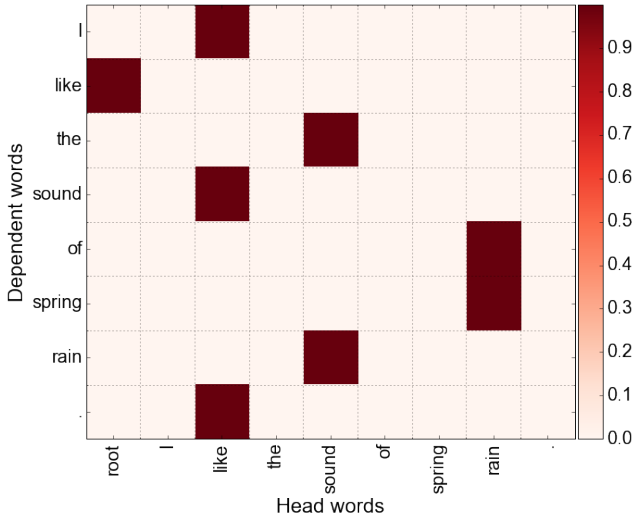

Figure 3.1: A typical scorer output showing probabilities assigned for head-word location.

Counterintuitively the CLE algorithm makes the decoding results slighty worse. We have investigated this and the problem lies in the confidence of the *scorer* predictions which are very sharp (see figure 3.1). Additionaly non-top scores do not reflect alternatives but are rather only a noise, so when CLE algorithm finds a cycle it is very probable that it will break it in wrong place, creating another error. The example of such behaviour is depicted on figure 3.2.

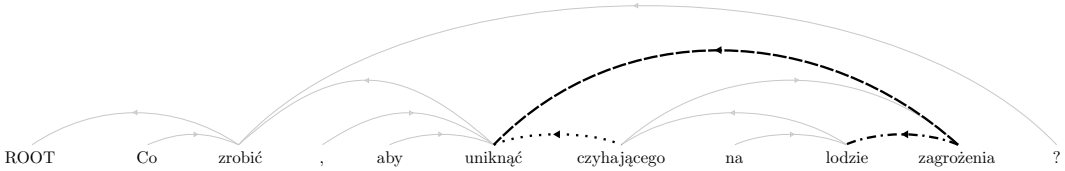

Figure 3.2: Dependency tree for sentence "What to do to advert threat lurking on the ice?". The dashed-dotted (zagrożenia → lodzie), dotted (czyhającego → uniknąć) and dashed (zagrożenia → uniknąć) arcs denote, respectively greedy, CLE, and groundtruth decodings. The greedy decoder produced the incorrect dependency zagrożenia → lodzie instead of zagrożenia → uniknąć which created a cycle zagrożenia → lodzie → czychającego → zagrożenia. However, the CLE algorithm replaced the good connection czyhającego → zagrożenia with czyhającego → uniknąć which removed the cycle but introduced a new error.

We tried to counteract this by using label-smoothing regularization (LSR) [Szegedy et al., 2015]. It is a very simple method of regularization which changes 1-hot vector encoding of groundtruth data to some other distribution which have effect of lowering overconfidence of the network. In our case we soften the head position groundtruth vector by giving the proper head 51% probability and uniformly distribute the rest 49% among other words. Unfortunately this method did not improve the overall CLE algorithm score.

3.1.3 Recurrent state size

We have also investigated the impact of *tagger* recurrent state size. It is desirable to have as small recurrent state size as possible because this part of the network is activated as many times as there are words in the sentence, making it a big performance factor.

We can see that in small dataset language like Polish there is not a big difference between base and 50% base, whereas in Czech the performance drop is significant. Bacause we want to use the same architecture for every language the original 548 units recurrent state size is optimal.

Language	% of recurrent state size	UAS	LAS
Polish	100%	90.26	85.32
	50%	**90.54**	**85.64**
	25%	89.07	82.80
Czech	100%	**91.41**	**88.18**
	50%	89.98	85.92
	25%	85.56	79.16

Table 3.4: Impact of the *tagger* recurrent state size on the performance. We report this value as percent of baseline rnn size.

3.1.4 Word pieces model

Following [Sennrich et al., 2015] we experimented with subword units. Instead of using single characters as inputs we can find frequent common characters groups and depict each word as combination of those units. Of course to be able to present every word, each single letter also has to be treated as small subword unit. For example lets consider two polish words: *lepszy* and *najlepszy* (better, the best). The *naj* prefix in polish used with adjective means that it is in superlative degree, so we can imagine that this letter composition is pretty common and thus will be treated as subword unit. Then those two words will be shown to the network as *naj-l-e-p-s-z-y* and *l-e-p-s-z-y* (- acting as subword delimiter).

Our word pieces model are obtained as follows: first we compute word frequency dictionary and initialiaze pieces dictionary with all characters in the dataset. From frequency dictionary we take the most common character bigram and we treat it as new character and replace all previous occurences with this new unit. The number of such iterations is a parameter given by the user. To convert word to pieces we use a greedy strategy. For word w we find the longest piece p that is prefix of w. The next step takes w suffix of size $\|w\| - \|p\|$.

#pieces	UAS	LAS
	Polish	
25	90.29	84.91
50	**90.40**	**85.46**
75	90.23	84.87
100	89.97	84.44
	Czech	
25	90.29	86.50
50	90.03	86.08
75	90.17	86.40
100	**90.84**	**87.31**

Figure 3.3: Results on word pieces model. #pieces denotes how many new multi-character tokens were used. To convert word to pieces we use a greedy algorithm in which we choose the longest piece that is equal to prefix of word.

In table 3.3 we have depicted the results of word pieces model for different number od additional multi-character tokens. There is only minor improvement for

Polish while for Czech we have actually worse results. Additionaly this input model is not well suited for multilanguage parser, because the pieces frequency is different for each language.

3.2 Multilanguage training

The main difference between multilanguage and single language configurations is that we have to decide which parts of the network should have shared weights. We tested it on 3 language pairs: Polish-Czech, Russian-Czech, Polish-Russian. They all are from the same slavic family, but Russian uses cyryllic alphabet as opossed to latin script for Polish and Czech. The main advantage of Czech language that made us choose it as auxiliary language is the quality and quantity of training data - it is over 6 times bigger than Polish and Russian datasets combined. The results of different sharing strategies are depicted in Table 3.2.

Shared parts	Main lang	Aux lang	UAS	LAS
-	Polish	-	90.31	85.21
Parser	Polish	Czech	90.72	85.57
Tagger, Parser	Polish	Czech	91.19	86.37
Tagger, POS Predictor, Parser	Polish	Czech	91.65	86.88
Reader, Tagger, POS Predictor, Parser	Polish	Czech	**91.91**	**87.77**
Parser	Polish	Russian	90.31	85.07
Tagger, POS Predictor, Parser	Polish	Russian	**91.34**	**86.36**
Reader, Tagger, POS Predictor, Parser	Polish	Russian	89.16	82.94
-	Russian	-	83.43	79.24
Parser	Russian	Czech	83.15	78.69
Tagger, POS Predictor, Parser	Russian	Czech	83.91	79.79
Reader, Tagger, POS Predictor, Parser	Russian	Czech	**84.78**	**80.35**

Table 3.5: Impact of parameter sharing strategies on main language parsing accuracy when multilingual training is used for additional supervision.

Multilingual training (Table 3.2) improves the performance on low-resource languages. We observe that the optimal amount of parameter sharing depends on the similarity between languages and corpus size – while it is beneficial to share all parameters of the PL-CZ and RU-CZ parser, the PL-RU parser works best if the reader subnetworks are separated.

Simillary as in single language configuration we have tested the impact of *tagger* recurrent state size on the results (see Table 3.2).

The results are computed on best Polish-Czech model (with share all strategy). We can see that the recurrent state size have similar behaviour as in single language czech model - the 50% size has significant drop in the performance. It is worth noting

% of recurrent state size	UAS	LAS
200%	91.45	86.36
100%	**91.65**	**86.88**
50%	89.53	83.66
25%	87.28	78.93

Table 3.6: Impact of the *tagger* recurrent state size on the performance of multilanguage pl-cz model. We report this value as percent of baseline rnn size.

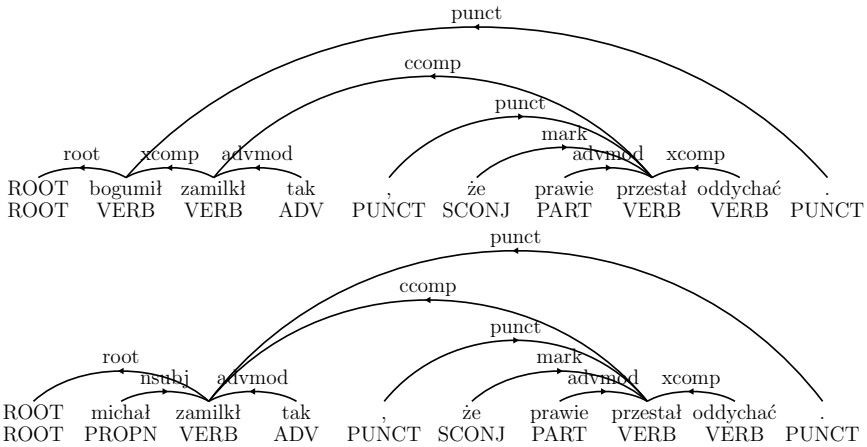

Figure 3.4: In the upper panel the sentence "Bogumił felt silent so much that he almost stopped breathing." is incorrectly parsed. We attribute the error to the *reader* or to the *tagger* because the predicted POS tag for "Bogumił", a male given name, is wrong (VERB instead of NOUN). In the lower panel we replace "Bogumił" with the more common "Michał" the network assigns a proper tag and the whole sentence becomes a valid parse tree.

that increasing the size changes the result only slighty (and are actually worse). This mean that choosen *tagger* size is well-suited for both single and multi language parsers.

3.2.1 Polish-Polish multilang parser

After investigating results of the parser (this problem applies to both single and muli parser) we have found that if the *tagger* assigns wrog POS tag to a particular word it can mislead the scorer and labeller producing wrong parse tree. In many cases after replacing wrong word with its synonym such that the *tagger* will assing the right POS tag - the tree becomes valid parse. For an example see 3.2.1.

Chapter 4

Summary

Bibliography

[Alberti et al., 2017] Alberti, C. et al. (2017). SyntaxNet Models for the CoNLL 2017 Shared Task. *arXiv:1703.04929*.

[Andor et al., 2016] Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., and Collins, M. (2016). Globally Normalized Transition-Based Neural Networks. *arXiv:1603.06042 [cs]*. 00001 arXiv: 1603.06042.

[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.

[Chen and Manning, 2014] Chen, D. and Manning, C. D. (2014). A Fast and Accurate Dependency Parser using Neural Networks. In *EMNLP*, pages 740–750. 00188.

[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.

[Covington, 2001] Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102. Citeseer. 00206.

[De Marneffe et al., 2006] De Marneffe, M.-C., MacCartney, B., Manning, C. D., et al. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454. Genoa.

[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.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[Nivre, 2008] Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Comput. Linguist.*, 34(4):513–553. 00220.

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

[Nivre et al., 2005] Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., KüBler, S., Marinov, S., and Marsi, E. (2005). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, page 1. 00666.

[Petrov et al., 2011] Petrov, S., Das, D., and McDonald, R. (2011). A universal part-of-speech tagset. In *LREC 2012*.

[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.

[Sennrich et al., 2015] Sennrich, R., Haddow, B., and Birch, A. (2015). Neural Machine Translation of Rare Words with Subword Units. *ArXiv e-prints*.

[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.

[Straka et al., 2015] Straka, M., Hajič, J., Straková, J., and Jan Hajič, j. (2015). Parsing universal dependency treebanks using neural networks and search-based oracle. In *14th International Workshop on Treebanks and Linguistic Theories (TLT 2015)*, pages 208–220, Warszawa, Poland. IPIPAN, IPIPAN.

[Szegedy et al., 2015] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567.

[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.

[Zeman, 2008] Zeman, D. (2008). Reusable tagset conversion using tagset drivers. In *LREC*.