Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS

Petr Němec

Application of Artificial Neural Networks in Computational Linguistics

Institute of Formal and Applied Linguistic
Advisor: *RNDr. Kiril Ribarov*
Study Program: *Computer Science*

I would like to thank Kiril Ribarov for his broad support and lot of ideas that made it possible to achieve the results presented, Jan Hajič for his technical support together with wide range of explanatory remarks, Iveta Mrázová for consultations regarding neural networks, Barbora Hladká for providing me with literature on statistical tagging, and Jana Duchoňová for being such a helpful consultant and proofreader.

I declare that I wrote the thesis by myself and listed all sources I used. I agree with lending the thesis.

Prague, April 10, 2004

Petr Němec

Abstrakt

Název práce: Využití neuronových sítí v počítačové lingvistice
Autor: Petr Němec
Katedra (ústav): Ústav formální a počítačové lingvistiky
Vedoucí diplomové práce: RNDr. Kiril Ribarov
e-mail vedoucího: ribarov@ufal.ms.mff.cuni.cz
Abstrakt:

Neuronové sítě představují perspektivní přístup k řešení problémů, jejichž přímé algoritmické řešení není známé či dostatečně efektivní. Automatické morfologické značkování je jednou z takových úloh na poli počítačové lingvistiky. K jejímu řešení jsme použili neuronovou síť zpětného šíření (backpropagation) v několika typech experimentů. Při určování správné značky na základě spolehlivého kontextu jsme se přesvědčili o základní schopnosti sítě se problému naučit, ačkoli dosažená úspěšnost (89,22%) nedosahovala přesnosti dosahované statistikou (93,47%). Podařilo se nám též určit vhodné parametry sítě a vstupního kontextu pro další experimenty. Pokus určit správnou značku na základě kontextu značek určených předem statistikou přinesl mírné snížení úspěšnosti (88,71%). Konečně experiment, jehož úkolem bylo volit mezi výstupy dvou statistických metod, vykázal vyšší úspěšnost (93,56%) než libovolné z těchto metod (92,74%, 92,58%). Na daném trénovacím korpusu (Pražský závislostní korpus) jde v současné době o absolutně nejlepší dosažený výsledek. Z dosažených výsledků vyplývá doporučení, aby prezentovaná metoda byla vyzkoušena na rozsáhlejší množině dat (Český národní korpus).

Keywords: značkování, morfologie, neuronové sítě

Abstract

Title: Application of Artificial Neural Networks in Computational Linguistics
Author: Petr Němec
Department: Institute of Formal and Applied Linguistics
Supervisor: RNDr. Kiril Ribarov
Supervisor's e-mail address: ribarov@ufal.ms.mff.cuni.cz
Abstract:

Neural networks represent a promising approach to problems, which exact algorithmic solution is unknown or not efficient enough. Morphological tagging is one of such tasks in the area of computational linguistics. We have tried to use a backpropagation neural network in several types of experiments. When determining the correct tag on the basis of reliable context, we have learned that the neural tag is basically capable to handle the problem, although the achieved tagging precision (89,22%) did not reach that of statistical methods (93,47%). We also managed to determine appropriate network and context parameters that we have used in the next experiments. The attempt to determine the correct tag on the basis of beforehand statistically determined tags brought a slight decrease of tagging precision (88,71%). Finally, the experiment, which goal was to vote from the outputs of two statistical taggers, showed higher tagging precision (93,56%) than any of these methods (92,74%, 92,58%). It is therefore the overall best result on the given training data set (Prague Dependency Treebank). Hence, it is recommended to test the method by training it on a larger training set (Czech Corpus).

Keywords: tagging, morphology, neural networks

1 Introduction

Neural networks represent a promising approach to problems, which exact algorithmic solution is unknown. When presented a sufficiently representative training set of problem instances, they are able to adapt in such way that they produce correct results even for the instances that were not trained, i.e. they are able to generalize over the trained data. This makes neural networks a natural candidate for variety of computational tasks.

There are many linguistic problems of this nature, one of which is the morphological tagging – automatic determination of the correct morphological properties of words of a text. This is one of the important problems in the area of computational linguistics as it underlies other crucial tasks such as syntactic parsing, or machine translation. Furthermore, the learning process and the obtained results show us certain facts about the processed language (determination complexities of various morphological categories, ambiguity rate for various lexical entries, context properties etc.).

The task is especially interesting when we deal with highly inflective languages, which morphological system is very rich. An excellent example are Slavonic languages, namely Czech, which is one of the morphologically most complex languages.

Many statistical methods have already dealt with morphological tagging problem and a high tagging precision has been reached. For Czech language, the statistical approach reached more than 95% tagging precision when trained on Czech Corpus [3] (see e.g. [1] or [2] to learn more about statistical approach to the tagging problem). Although this number may seem to be sufficient, at closer look we discover that it may not be so. For instance, if we were to create a grammar checker application based on the result of a morphological tagging, we would have to handle the fact that there is a mistake in each out of twenty words in average. It is therefore still desirable to increase the tagging precision of automatic taggers.

In our thesis, we test the neural networks approach performance on the morphological tagging problem for Czech. Various experiments have been carried out, their results have been discussed and compared to those of the statistical methods. For this purpose, a software tool has been created, too.

Chapter 2 introduces Czech morphology, explains basic terms used in the thesis, and specifies our goal. Chapter 3 describes the neural network architecture and the learning algorithm used, and presents experiments already carried out in this area. Chapters 4 to 6 describe the three types of experiments performed – reliable-context-based determination, statistical-context-based determination, and the voting experiment. Chapter 7 summarizes the achieved results. Information regarding the software provided (User‘s guide, Implementation remarks) together with the description of electronic sources provided on the enclosed CD can be found in appendices.

2 Tagging problem specification

2.1 Czech morphology introduction

Czech language belongs to the group of Slavonic languages, which morphological system is very rich. Czech language recognizes ten part-of-speech types, five of which are inflective: nouns, adjectives, pronouns, numerals, and verbs. For each of them there is usually a large set of inflectional and conjugational patterns. There are other common morphological categories present: gender, number, case, person, tense, grade, voice. There are also other rather special categories (see Section 2.2 for the detailed description). These categories and the values they may obtain form a complex system, which configuration for a given input token[1] may be expressed by a morphological tag – a string of 15 character, each of them representing a value of a morphological category (see Section 2.2). There are more than 2000 different tags commonly present in Czech.

A sole word is rather often morphologically ambiguous. For instance, the word "vlastní" may be either an adjective "own" or a verb "(he/she) owns". A morphological analysis tool that determines for (almost) each Czech word the set of its possible morphological tags is already available [4]. A morphological tagger (a device assigning automatically the correct morphological tag to a word) therefore does not necessarily have to generate the entire tag, it may select it from the generated set.

Prague Dependency Treebank (PDT) training set containing almost 1,5 million words, which correct tags are known, is available as well as testing sets designed for tagging methods comparison [4]. The following example shows how the sentence "Prezident rezignoval na svou funkci" ("The president has stepped down") is coded in a PDT morphological file:

```
<csts>

<f cap>Prezident<l>prezident<t>NNMS1-----A----<MMl>prezident
     <MMt>NNMS1-----A----

<f>rezignoval<l>rezignovat_:T<t>VpYS---XR-AA---<MMl>rezignovat_:T
     <MMt>VpYS---XR-AA---

<f>na<l>na<t>RR--4----------<MMl>na
      <MMt>RR--4----------<MMt>RR--6----------

<f>svou<l>svůj-1_^(přivlast.)<t>P8FS4---------1<MMl>svůj-1_^(přivlast.)
     <MMt>P8FS4---------1<MMt>P8FS7---------1

<f>funkci<l>funkce<t>NNFS4-----A----<MMl>funkce
     <MMt>NNFS3-----A----<MMt>NNFS4-----A----<MMt>NNFS6-----A----

<D><d>.<l>.<t>Z:-------------<MMl>.<MMt>Z:-------------

</csts>
```

The <f> (<d>) tag contains the word form (symbol) as it appears in the text. The <l> tag contains the manually disambiguated lemma of this word form. The <t> tag is the

[1] The notion of „input token" includes a word form in its common sense, but also any standalone symbol in a sentence (parentheses, commas, hyphens, punctuation marks etc.). For simplicity, we will use the term „word" instead of "input token".

manually determined correct morphological tag of the input token (word form or symbol). The <MMl> tag is the lemma determined by the morphological analysis. There may be more than one <MMl> tag present, if the input token is ambiguous it terms of its lemma. For each <MMl> tag there is a set of <MMt> tags. These tags are also generated by the morphological analysis and represent possible morphological tags for the input token with respect to the corresponding lemma. Therefore, the set of all <MMt> tags (regardless of <MMl> tags) represent all possible morphological tags for the given input token.

Furthermore, <MDt> and <MDl> tags may be present. They contain the outputs of statistical taggers – tag and lemma determination. Outputs of two statistical methods (Feature-based tagger and Markov model tagger [4]) are available in PDT1.0 morphologically annotated files as <MDl src="a">, <MDt src="a">, and <MDl src="b">, <MDt src="b"> tags.

2.2 Czech morphology positional tag

Positional tag is a string of 15 characters, where each position (except for the blank placeholder positions 13 and 14) stands for a morphological category and each character indicates a value of this category. For example, "NNFP1-----A----" stands for a plural (position 4) feminine (position 3) noun (position 1) in nominative (position 5).

Table 1[2] shows the categories overview. As shown in Section 4.3.2, the attention has been focused mainly on the gender, number, and case categories. Tables 2 to 4 list possible values of these categories. For the detailed description of values of all categories see Appendix C.

No.	Name	Description
1	POS	Part of Speech
2	SUBPOS	Detailed Part of Speech
3	GENDER	Gender
4	NUMBER	Number
5	CASE	Case
6	POSSGENDER	Possessor's Gender
7	POSSNUMBER	Possessor's Number
8	PERSON	Person
9	TENSE	Tense
10	GRADE	Degree of comparison
11	NEGATION	Negation
12	VOICE	Voice
13	RESERVE1	Unused
14	RESERVE2	Unused
15	VAR	Variant, Style, Register, Special Usage

Table 1: Morphological categories overview

[2] The tables have been taken from the PDT1.0 electronic sources [4].

Value	Description
-	Not applicable
F	Feminine
H	Feminine or Neuter
I	Masculine inanimate
M	Masculine animate
N	Neuter
Q	Feminine (with singular only) or Neuter (with plural only); used only with participles and nominal forms of adjectives
T	Masculine inanimate or Feminine (plural only); used only with participles and nominal forms of adjectives
X	Any of the basic four genders
Y	Masculine (either animate or inanimate)
Z	Not feminine (i.e., Masculine animate/inanimate or Neuter); only for (some) pronoun forms and certain numerals

Table 2: GENDER category values

Value	Description
-	Not applicable
D	Dual
P	Plural
S	Singular
W	Singular for feminine gender, plural with neuter; can only appear in participle or nominal adjective form with gender value Q
X	Any

Table 3: NUMBER category values

Value	Description
-	Not applicable
1	Nominative
2	Genitive
3	Dative
4	Accusative
5	Vocative
6	Locative
7	Instrumental
X	Any

Table 4: CASE category values

2.3 Problem specification

Being acquainted with the terms presented in this chapter, we may now formulate the tagging problem precisely: given an unknown Czech text, given the result of morphological analysis determining the set of possible tags for each word (and possibly the tagging results of statistical methods available), the correct tag for each word of a text is to be determined, i.e. a morphological tagger is to be developed. Assuming that there are morphological taggers available, the aim of our approach towards a morphological tagger will not necessarily be an entire construction of a new one, but also complementations of morphological categories as shown later in Chapters 4 and 5, where the correct value for a partial category (e.g. gender, number, and case) rather than the entire tag is determined.

All experiments we performed are designed to select exactly one tag (or exactly one category value) from the set of possible tags (values). We will evaluate the achieved results by means of tagging precision. For our purposes, the tagging precision is the number of words, which tags (category values) were correctly assigned, divided by the total number of processed words.

2.4 Results of statistical taggers

The output of statistical taggers plays a significant role in our experiments. Not only we compare the achieved results to the statistical ones, but statistical output also serves as the "context provider" as described in Chapter 5. Moreover, the entire Voting experiment (see Chapter 6) consists in the selection of the correct tag from a set of statistical outputs.

We consider three statistical taggers in this thesis: Feature-based tagger, Markov model tagger, and an advanced statistical tagger CZ031219 [1]. Table 5 lists their respective tagging precisions on the evaluation testing file.

The advanced statistical tagger CZ031219 was able to reach overall tagging precision 93,47%, which represents the currently best statistical result when trained on PDT data.

Tagger	Total	GENDER	NUMBER	CASE
CZ031219	93,47	97,82	98,00	95,37
Feature-based	92,74	97,55	97,62	94,67
HMM	92,58	97,62	97,86	94,41

Table 5: Statistical taggers precisions for the entire tag, case, number, and case categories.

We focus our attention on the listed categories, because their tagging precision is the lowest when compared to the other categories. If we reach higher precision than a statistical tagger in tagging any of these categories, it will increase the overall performance of the tagger.

3 Neural networks approach

3.1 Previous research

Compared to other approaches, there are not many current experiments where artificial neural networks (ANN) are applied to Natural Language Processing (NLP) tasks. However, ANN experiments regarding morphological tagging have been performed for English.

Schmid [5] trained recurrent multilayer perceptron network as part-of-speech morphological tagger. Using wide left and right context together with the suffix information, he was able to obtain results similar to those reached by statistical methods (Hidden Markov model systems). Nakamura et al. [6] use massive feed-forward network to predict the part-of-speech of a word on the basis of its left context. Again, the tagging precision was almost the same as that of a trigram-based predictor.

To our best knowledge, no such experiments have been carried out for Czech.

3.2 Network selection

3.2.1 BP-SOM

At the beginning, we considered using BP-SOM architecture [7,8] for our experiments, because BP-SOM learning algorithm showed very good performance over some classical computational problems. For example, it outperformed the classical backpropagation learning algorithm in the binary vector parity classification task and monks tasks [9].

Basically, BP-SOM ANN consists of multilayer feed-forward network (MFN). Furthermore, a self-organizing Kohonen map (SOM) is associated with each hidden layer (see Figure 1). During training of the weights in the MFN, the corresponding SOM is trained on the hidden-unit activation patterns. The self-organization of the SOM is used as an addition to the standard BP learning rule. For details, see e.g. [7]

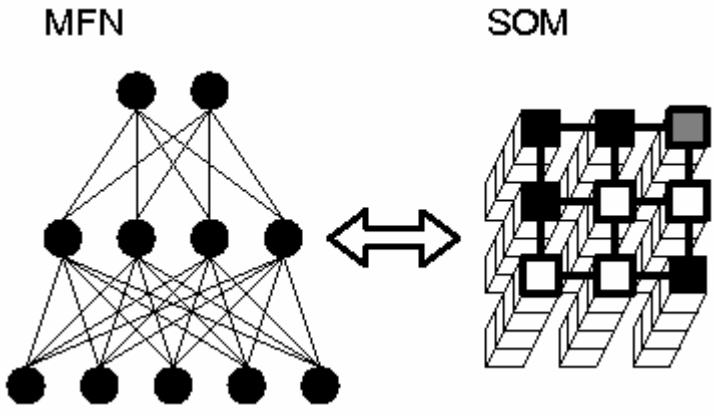


Figure 1: BP-SOM network architecture

This network can be used only as a classifier (each output vector represents a class), and the number of classes must not be too high as each class maps to one neuron in the output layer). For the purpose of experimenting with Czech morphology, there is a need of statistical merging (see Section 4.1.2), which cannot be performed on the BP-SOM architecture. This is

due to the fact that the resulting output vectors do not represent a class anymore. Instead, they form a potentially infinite set of general real number vectors. Without statistical merging, the training set would be not only inconsistent, but also very large. We have therefore decided to abandon BP-SOM approach.

3.2.2 Backpropagation network

A classical option that overcomes the limitation discussed in the previous section is a feed-forward network trained using the backpropagation (BP) learning algorithm. We have tried to test its capabilities on the morphological tagging task.

In this section we give a brief description of multilayer neural network trained by the BP algorithm. Large amount of literature on the backpropagation concept and its details is available; an example reference is [10] .

The network may consist of arbitrary number of layers, each containing an arbitrary number of neurons. Figure 2 shows a 3 layer network, i.e. network with 1 hidden layer. The layers are linearly ordered – the first one is the input layer, the last one is the output layer, between them hidden layers are present. Each neuron of a non-input layer is connected to all the neurons in the preceding layer. Each such connection is assigned a weight.

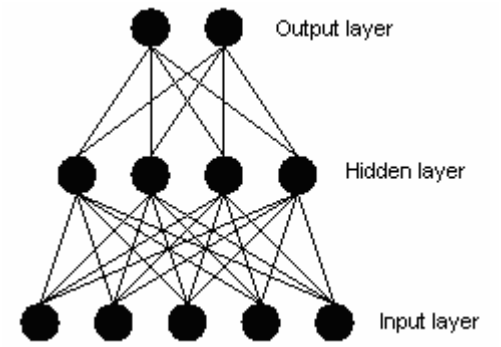


Figure 2: Feed-forward multiperceptron network architecture

The training set consists of vector pairs *(I,D)*, where *I* stands for the input vector and *D* for the desired network output vector. The network is trained in two phases:

1. *Feed-forward phase*. The input vector is applied to the input layer – the vector components represent the respective outputs of the input layer neurons. The outputs of respective neurons in each following layer are computed according to the given activation function T as follows:

$$y_j = T(\sum_i w_{ij} x_i + \Theta_j)$$

where y_j is the output for the j-th neuron in the processed l-th layer, Θ_j is its bias value, w_{ij} is the connection weight from the i-th neuron in the preceding layer l-1 to the j-th neuron of the processed l-th layer, and x_i is the output of that neuron. T is usually a sigmoid function

$$T(x) = \frac{1}{1+e^{-\gamma x}}$$

which is also the case in our experiments. γ is the gain parameter (in our experiments $\gamma = 1$).

2. *Backpropagation phase*. First, the error vector *E* for the output layer (*o*) is computed:

$$E_i^o = y_i^o(1-y_i^o)(D_i - y_i^o)$$

where D_i is the *i*-th component of the desired output vector *D*. This error is backpropagated into the preceding layers (*l*-1) as follows:

$$E_i^{l-1} = y_i^{l-1}(1-y_i^{l-1})\sum_k E_k^l w_{ki}$$

In each layer the weight and bias difference is computed as follows:

$$\Delta w_{ij}^l(t) = \eta E_i^l(t) y_j^{l-1}(t) + \alpha(w_{ij}(t) - w_{ij}(t-1))$$

$$\Delta\Theta_i^l(t) = \eta E_i^l(t) + \alpha(\Theta_i(t) - \Theta_i(t-1))$$

where η is a learning rate parameter and α stands for the momentum term. *t*-1 denotes the variable state in the previous step. The weights and biases are then modified:

$$w_{ij}^l(t+1) = w_{ij}^l(t) + \Delta w_{ij}^l(t)$$

$$\Theta_i^l(t+1) = \Theta_i^l(t) + \Delta\Theta_i^l(t)$$

At the beginning, the weights and biases are set to small random values. In our experiments these values are taken from -0.2 to 0.2. The network is then trained on all vectors repeatedly, until the network's global error on the training data is sufficiently low or some other condition is fulfilled. For instance, in our experiments we may stop training when the performance on the testing files decreases for a long time (see Section 4.4.3). The network parameters are the learning rate (η) and momentum (α). They remain constant during the training.

Once the network is trained, unknown input may be applied. The output is then obtained by performing a single feed-forward step.

3.3 Experiment methodology

As a result a continual process of thinking up various ideas and performing minor tests, we propose the following experiment schema:

First, we tested the BP ANN capabilities on the so-called reliable-context-based determination experiment described in Chapter 4. Although this method does not allow direct tagging of unknown texts, this experiment demonstrates the possibilities of the BP ANN. Moreover, we gathered experience concerning the optimal data coding and the network learning algorithm parameters that we used later on. Chapter 4 contains important details used further in Chapter 5 and Chapter 6.

Second, very similar statistical-context-based determination experiment described in Chapter 5 was performed. This time we used statistical context that can be obtained beforehand by tagging a text by a statistical tagger. The achieved results are therefore directly applicable. The experiment also showed us, how much is the tagging precision influenced by the context quality and whether the use of statistical context is acceptable (in terms of decrease of tagging precision).

Third, we tried to use the BP ANN as a voting device for available statistical taggers. This voting experiment is described in Chapter 6.

4 Reliable-context-based determination

Determination based on reliable context is the key experiment we focused our attention on. The basic idea is based on the *n*-gram model: we determine the correct tag for the given word on the basis of *n*-1 preceding tags (and possibly the suffix information), where *n* is a fixed constant for the experiment. By "reliable context" we mean that we always train and test on correct preceding tags, i.e. we use knowledge of the actual correct tags for the given preceding words (left context). Let *t* be the correct tag for a word *w* in a sentence. So in each sentence

$$s = (w_1, w_2, ..., w_k)$$

of the training or testing set we consider all tuples

$$\{(t_1, t_2, ..., t_{n-1}), (t_2, t_3, ..., t_n), ..., (t_{k-n+1}, t_{k-n+2}, ..., t_{k-1})\}$$

as preceding contexts. Our aim then is to determine the correct tag for w_n, w_{n+1}, ..., w_k, respectively. We have to extend the sentence by adding virtual empty tags at the beginning in order to have a context for every word in the sentence.

This way, however, it is impossible to tag a continual text directly, because we cannot be sure that we know the correct preceding tags[3].

4.1 Formal representation

Let $n \geq 1$, $m \geq 0$ be fixed integers. Let's assume that a word *w*, which tag is being determined, is *k* characters long and is immediately preceded (in the sentence, in which it occurs) by *n*-1 words, which tags $(t_1, t_2, ..., t_{n-1})$ are known. Let

$$(l^1, l^2, ..., l^m)$$

be the determined word's suffix, where l^1 represents the last character of the word, l^2 its second last character, and so forth. If $k < m$, put $l^i = \lambda$ (empty word), for $i > k$. The context of *w* is the pair

$$(T, S) = ((t_1, t_2, ..., t_{n-1}), (l^1, l^2, ..., l^m))$$

The determination of the correct tag can be either an entire generation of that tag or a selection from the set of possible tags (if the result of the morphological analysis is available). The BP ANN is trained on a set of vector pairs *(I,O)*, where *I* is the context for the given word *w* and *O* is the output vector that describes the "quality" of various candidates for the correct

[3] The first error in tagging a word in a sentence would lead to a very poor tagging precision of the remaining words as the context information collapses.

tag of *w*. For the training data[4], the quality of an output vector component represents its frequency of occurrence as the correct value in the context *I* (see Section 4.1.2).

Czech morphological tag consists of several morphological categories (see Section 2.2). There are basically two possible approaches to the determination task. We may try to determine the entire tag at once, so that *O* describes the quality of entire tags as candidates, or we may try to determine the correct value for each category separately, evaluate the results, and propose the best complete tag. For both approaches, an appropriate coding of *I* and *O* has to be chosen. For the second approach we also have to define a measure that allows to evaluate an entire tag quality when the sorted quality lists of values of partial categories are known.

4.1.1 Context coding

Coding the given context requires to code both components *(T,S)* into a single vector – the neural network input. A natural way to do this is to code *T* and *S* and place their codes into the vector as in Figure 3.

Figure 3: Input vector overall structure

There are basically two methods to code $T = (t_1, t_2,..., t_{n-1})$ and $S = (l^1, l^2,..., l^m)$:

1. (Compound coding) We may count all tag (or suffix) sequences of a given length *n* (or *m*) appearing in the corpus, then assign an index – ordinal number to each sequence. The code of this index $C(t_1,t_2,..,t_n)$ (or $C(l^1,l^2,..,l^m)$) will then represent *T* (or *S*). See Figure 4.

Code of T	Code of S
$C(t_1, t_2,..., t_{n-1})$	$C(c_1, c_2,..., c_m)$

Figure 4: Vector structure (compound coding)

2. (Single coding) Each single tag from *T* (or single character from *S*) is coded. The linear string of these codes will form the code for *T* (or *S*) as shown in Figure 5. Every character appearing in the corpus is assigned an ordinal number index. The code of the index $C(l^i)$ represents the given character l^i. As far as tags are concerned, the situation is a bit more complicated. Again, we can assign an ordinal number index to each tag appearing in the corpus (index single coding), or we can deal with the tag structure itself (category single coding). As we have seen in Section 2.2, the positional Czech tag consists of 13 morphological categories. We can code the ordinal number index of the

[4] As far as the real network output for an unknown context is concerned, it is difficult to describe it more precisely as it is a result of „black box".

given value of each category a_i ($1 \le i \le 13$) separately. The linear string of these codes will form the code of the tag (see Figure 6). Eventually, we may also select only a subset of categories to code (e.g. part-of-speech, which tagging precision is very high, see Section 4.3.2).

(Code of T)				(Code of S)			
$C(t_1)$	$C(t_2)$	…	$C(t_n)$	$C(l_1)$	$C(l_2)$	…	$C(l_m)$

Figure 5: Vector structure (single coding)

(Code of a tag)			
$C(a_1)$	$C(a_2)$	…	$C(a_{13})$

Figure 6: Linear coding of the 13 morphological categories.

As it has been shown, in order to code *(T,S)* we have to code a list of ordinal number indices. The code of each index will reside in a part of the vector ("a box") of the fixed length that is equal to the number of vector components required to store the highest index of the given set. This way, the coding function *C* is bijective and the values do not get mixed.

We have performed the tests with two most usual codings of an ordinal: binary and "one from n" coding. Binary coding is the usual binary representation with trailing zeros appended (so that the ordinal occupies exactly the length of the box, which is the number of binary digits required to store the cardinality of the set). In "one from n" coding, the box length is always equal to the cardinality of the set (q). Let i, $0 \le i < q$, is the coded index. Then all box components except fort the i-th are set to 0. The i-th component is set to 1.

4.1.2 Output coding

The simplest way to code a single instance of the desired output for the specified category in a given context is to use "one from n" coding – the dimension n of the output vector is equal to the number of possible values for the category and all vector components are set to 0, except for the index of the correct one, which is set to 1. The major problem of this approach lies in the fact that there is usually more than one correct value for the given context in the training set, i.e. there are more instances of the same context but with different correct value. This language property causes inconsistency in the training set, which affects the BP ANN performance negatively.

It is possible to merge all training instances $\{(I, O_i), i \in K_I\}$ with the same input I, but possibly different outputs $\{O_i, i \in K_I\}$, where K_I is the output vector index set for input I, into a single training vector $(I, S(\{O_i, i \in K_I\}))$, where

$$S : (\{O_i, i \in K_I\}) \to O$$

is a function that returns a statistical distribution vector O of the respective $\{O_i, i \in K_I\}$. We have used a linear model, where the index of the value with maximal number of occurrences (winning value) is set to 1 and other vector components are set to

$$\Theta + \frac{V_j(1-\Theta)}{V_{\max}}$$

where V_j is the number of occurrences of the value with index j, V_{max} is the number of occurrences of the winning value, Θ is a reliability threshold constant. This approach makes the training set consistent and also reduces greatly the number of training vectors.

The output vector design allows to sort the possible output values by the corresponding output vector component value, which lies in [0,1] interval and the higher it is, the more probable should the neural network output value be. The BP neural network works as a "black box", it is therefore difficult to describe the output vector more precisely[5].

It is also possible to code the entire tag to the output vector. We may do so by coding it either as an index into the set of tags or as a set of indices into particular categories (see Section 4.1.1). In the first case (assuming "one from n" coding) we may also perform the statistical merging as described in the previous section.

4.1.3 Example

This section shows an example of neural network input and output for the reliable-context-based determination experiment. As the size of vectors makes it impossible to show the entire coded vectors, only "uncoded" input and output vector forms are presented:

Let the left context length be 2 tags, let the suffix length be 3 characters. Let *s* be the sentence "Prezident rezignoval na svou funkci." ("The president has stepped down."). The correct tags for the respective words of the sentence are listed in Table 6.

Input token	Correct tag
Prezident:	NNMS1-----A----
rezignoval:	VpYS---XR-AA---
na:	RR--4----------
svou:	P8FS4---------1
funkci:	NNFS4-----A----
.:	Z:-------------

Table 6: Correct tags of the input tokens

If the virtual zero tags ("----------") are included in the beginning of the sentence, the training input and output pairs for the gender, number, and case category determination will be those listed in Table 7.

[5] We will use the term "output vector component quality" in the described sense. For the training data, it is the result of statistical merging. For the network output, it is a "black box" value.

Input			Output (GENDER)	Output (NUMBER)	Output (CASE)
---------------	---------------	"ent"	M	S	1
---------------	NNMS1-----A----	"val"	Y	S	-
NNMS1-----A----	VpYS---XR-AA---	"λna"	-	-	4
VpYS---XR-AA---	RR--4----------	"vou"	F	S	4
RR--4----------	P8FS4---------1	"kci"	F	S	4
P8FS4---------1	NNFS4-----A----	"λλ."	-	-	-

Table 7: The training set pairs for the gender, number, and case categories

4.2 Data statistics

The morphological corpus is divided into training set and two testing sets[6]. We have further randomly selected two smaller subsets A,B ($A \subset B \subset C$) from the entire training set (C) in order to be able to test the performance of various codings in reasonable time. The words and sentences counts of the respective sets are listed in Table 8.

	Train set A	Train set B	Train set C	Test set 1	Test set 2
Words	3226	225172	1470644	129574	124957
% of training corpus words	0,22%	15,31%	100,00%	8,81%	8,50%
Sentences	179	13655	87487	7506	7231

Table 8: Sizes of training and testing sets

There are 2061 different tags appearing in both training and testing part of the corpus either as correct tags or as the result of the morphological analysis.

There are 125 different characters appearing in suffices of length 4 in all words of the entire corpus. (We do not consider longer suffices as we believe they do not provide any further essential morphological information. This hypothesis is supported by the fact that a very little difference has been observed between the results obtained for suffices of length 3 and 4 respectively (see Section 4.4).

4.3 Experimental procedure

Because of the computational time limitations, we were unable to test all the codings outlined in Section 4.1. Our basic goal was to find the coding method that gives the best results when used on small training sets (A, B). We then used just this coding for training on the entire set (and also for the statistical-context-based determination and the voting experiment).

[6] The two testing sets are the development and the evaluation testing sets. The highest tagging precision is first achieved on the development testing set, that tagger configuration is then tested on the evaluation testing set. The obtained results are then published.

4.3.1 Rejected codings

The following codings were not considered:

1. Entire tag as the network output. In order to be able to train on the entire training corpus in reasonable time, it turned out to be necessary to reduce the cardinality of the training set C by means of statistical merging (see Section 4.1.2). This is only possible with "one from n" output coding. But we cannot afford as many as 2061 neurons appearing in the output neural network layer.
2. Compound input tags coding. The index tag coding showed to be less effective than category coding (see Section 4.4.2). Compound tags coding is just the indexing of the entire tag sequence rather than each single tag. There is no reason to believe that this formal indexing change would increase the indexing approach precision significantly.
3. Compound suffix coding for the same reason.

So we only tested the performance of the (category and index) single tag coding together with single suffix coding.

4.3.2 Testing categories selection

As far as category single tag coding is concerned, we had to decide, which categories are worth dealing with. These are those, which tagging precision is low in general. Table 9 shows the tagging precision results for the respective categories when the first possible tag is selected. It can be easily seen that only the gender, number, case, and var categories are worth training as the product of tagging precision of the others is 99,73%.

Category	Precision
POS	99,98
SUBPOS	99,92
GENDER	91,71
NUMBER	85,47
CASE	74,33
POSSGENDER	99,99
POSSNUMBER	99,99
PERSON	99,91
TENSE	100,00
GRADE	99,98
NEGATION	99,99
VOICE	100,00
VAR	98,10

Table 9: First candidate selection baselines

The var category can be trained very easily[7] to reach precision greater than 99%, so we will not perform any intermediate tests for it.

[7] This category is somewhat special as it serves as a "garbage collector" – it contains additional information that could not have been placed elsewhere. Therefore its tagging precision will vary significantly, when its value set changes.

4.3.3 Baseline values

We also determined baseline precisions for the gender, number, and case categories. They were measured as the precisions of selecting the most frequent correct tag (on the entire training set) from the set of possible tags (see Table 10).

Category	Precision
GENDER	92,30
NUMBER	94,35
CASE	82,60

Table 10: Category baselines

4.3.4 Testing procedure

All the category tagging experiments, which results are presented in Section 4.4, were performed with statistically merged "one from n" coded output category.

This coding method allows to sort the possible values quality for the given category (obtained from the set of possible tags) according to the neural network output. We select that possible category value, which neural output is the highest, as the winner. Eventually, we may perform template decision first: If the given context - input vector is present in the training set, we consider the trained output instead of the network output. This usually leads to better tagging precision.

As far as entire correct tag determination problem is concerned, we have to define a procedure that evaluates the candidate quality on the basis of the respective category value orderings obtained from the category outputs of the respective neural nets. We propose a measure that minimizes the product of the order indices of the respective tag categories.

For example, let's assume that we obtained neural network outputs for the gender, number, and case categories. The quality of the respective values of these categories are (in the descending order) as follows: N, I, F (gender), S, X, P (number), 1, 4 (case). The score for tag[8] "??NP4???????????" is then 1.3.2 = 6 and for the tag "??FS1???????????" it is 3.1.1 = 3. If these two tags are the only alternatives, the second one will be selected.

[8] ? denotes an arbitrary character

4.4 Results

The electronic form of the experiments presented in this section is stored on the enclosed CD (see Appendix A). The instructions on how to run the reliable-context-based determination experiment with the developed software tool can be found in Section A.1.5 of Appendix A, too.

4.4.1 Binary to "one from n" comparison

First, we focused our attention on the category coding and tried to determine whether binary coding gives significantly worse results than "one from n", because if it does not, we will code the vectors binary as it reduces tag length from 153 to 43 and suffix length from 125 to 7 components. Such reduction would speed up the learning process significantly.

We trained on training set A, because "one from n" coded vectors training would take too long on larger training sets. Neural network with one hidden layer containing 100 neurons was used. The η parameter was set to 0,2 and no momentum was used.

Table 11 shows the test results for gender, number and case categories on A training set for the both codings[9]. We can see that "one from n" coding does not bring any substantial precision increase over binary coding neither when applied to tag nor to suffix[10]. We have therefore decided to code the vectors binary further experiments.

Context	Cntx. coding	Suffix	Sfx. coding	GENDER	NUMBER	CASE
0	-	4	binary	92,19	93,81	81,23
0	-	4	one from n	92,41	93,57	81,22
1	binary	0	-	91,08	92,36	86,69
1	one from n	0	-	90,14	92,46	86,93
1	binary	2	binary	92,06	94,40	90,25
1	one from n	2	binary	92,30	94,69	90,22
1	one from n	2	one from n	92,71	94,87	90,72

Table 11: Tagging precision for various codings (training set A).

4.4.2 Index to category tag coding comparison

Our next question was, which tag coding (index or category) leads to better results. Category coding provides much more information, so our hypothesis favoured this coding method. Additionally, we wanted to find out *m* - the optimal suffix length parameter.

In order to determine suitable experiment parameters, variety of training experiments were performed using the training set B, as it would be impossible to perform such number of tests on the entire training corpus

Hidden layer sizes oscillated between 100-500 neurons depending on the context and suffix length. Table 12 lists the hidden layer sizes for each context and suffix length tested. We have learned that if the hidden layer size is overly below the listed recommended size for the given experiment parameters, the BP ANN learning performance decreases.

[9] "Context" and "suffix" denotes left context and suffix length, respectively.
[10] Note how significantly does suffix information affect the tagging precision.

Learning rate parameter was set from the range [0,1;0,2] and momentum from the range [0,6;0,8] was used. Higher learning rate values led to network oscillation in early training stages. The use of such a high momentum term enhanced the training procedure greatly and the net convergence remained stable. The exact value of the two parameters was set random in each experiment.

Each coding experiment ran at least 500 cycles with 1 iteration. Higher number of iterations affected network performance negatively.

After every 50 cycles the network performance was tested both on the evaluation testing set and the training set B.

Suffix	Left context	Hidden layer
0	1	60-80
1	1	100
2	1	100-150
3	1	150-300
4	1	250-300
0	2	100-200
1	2	200-250
2	2	300
3	2	300
4	2	300

Table 12: Hidden layer sizes for various experiments

Tables 13-16 summarize the best obtained results both on the evaluation testing set and the training set B[11].

	Evaluation testing set			Training set B		
Suffix	GENDER	NUMBER	CASE	GENDER	NUMBER	CASE
0	89,83	91,13	82,2	91,35	92,48	84,24
1	90,83	92,85	84,86	91,97	93,82	86,23
2	92,64	95,16	90,89	93,57	96,10	92,14
3	92,92	95,80	91,98	93,90	96,82	93,05
4	93,22	96,20	92,14	94,61	97,88	93,97

Table 13: Index binary coding, bigrams

	Evaluation testing set			Training set B		
Suffix	GENDER	NUMBER	CASE	GENDER	NUMBER	CASE
0	90,00	92,42	87,28	91,32	93,84	88,87
1	91,57	94,59	89,49	92,77	95,56	90,84
2	92,60	95,5	91,77	94,21	96,93	93,74
3	93,53	96,46	92,79	94,96	98,05	94,61
4	93,72	96,63	92,64	95,34	98,30	94,67

Table 14: Category binary coding, bigrams

[11] The listed results are not directly comparable with the final results, because the first two (one) words for trigram (bigram) model in a sentence were not determined (no zero tags were present). Moreover, the correct tag was always present in the set of possible tags (the result of the morphological analysis was altered). The task was to find the optimal coding, not to give pure tagging results.

	Evaluation testing set			Training set B		
Suffix	**GENDER**	**NUMBER**	**CASE**	**GENDER**	**NUMBER**	**CASE**
0	91,06	93	87,65	92,38	94,43	89,4
1	92,21	94,61	90,3	93,53	95,83	92,15
2	92,88	95,5	91,87	94,18	97,29	94,14
3	93,71	95,97	92,55	95,48	98,34	95,24
4	93,57	96,15	92,65	95,73	98,96	95,6

Table 15: Index binary coding, trigrams

	Evaluation testing set			Training set B		
Suffix	**GENDER**	**NUMBER**	**CASE**	**GENDER**	**NUMBER**	**CASE**
0	91,25	93,04	89,57	92,8	94,56	91,48
1	92,54	95,39	91,32	94,12	96,81	93,37
2	93,37	95,91	91,79	95,03	97,71	95,94
3	93,9	96,57	93,26	95,79	98,39	95,52
4	93,95	96,82	93,57	96,54	98,93	97,01

Table 16: Category binary coding, trigrams

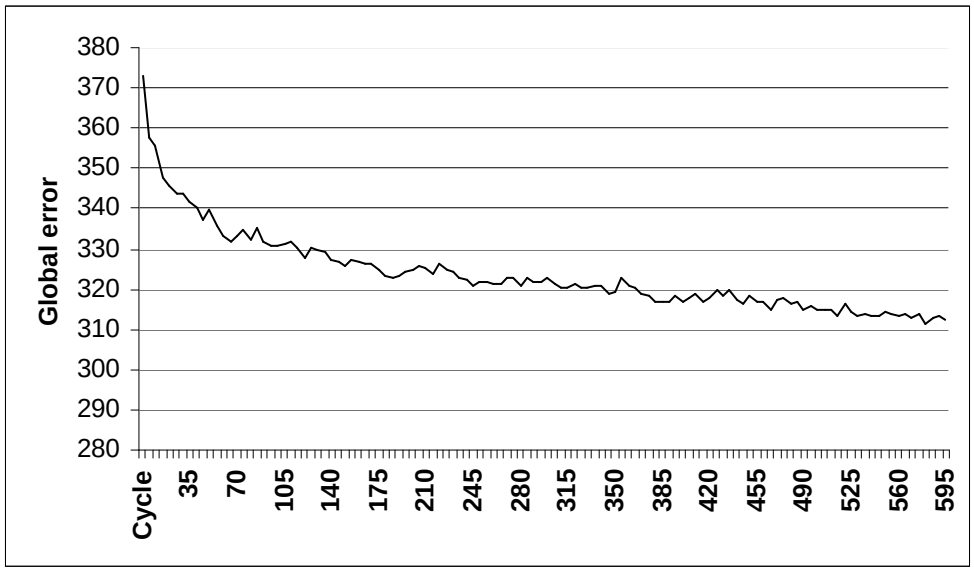
In accordance with our hypothesis, we can see that category coding gives better results than index coding. Moreover, it was harder to achieve low global error (resulting in better tagging precision over the training set) with index coding. We also see that increasing suffix length leads to higher tagging precision, although the difference between suffix of length 3 and 4 is very low.

Having reached the best results 93,95%, 96,82%, and 93,57% (highlighted in Table 16) for the gender, number, and case categories respectively, we have decided to perform the final testing on the entire training corpus C with category single coding and suffix length 4.

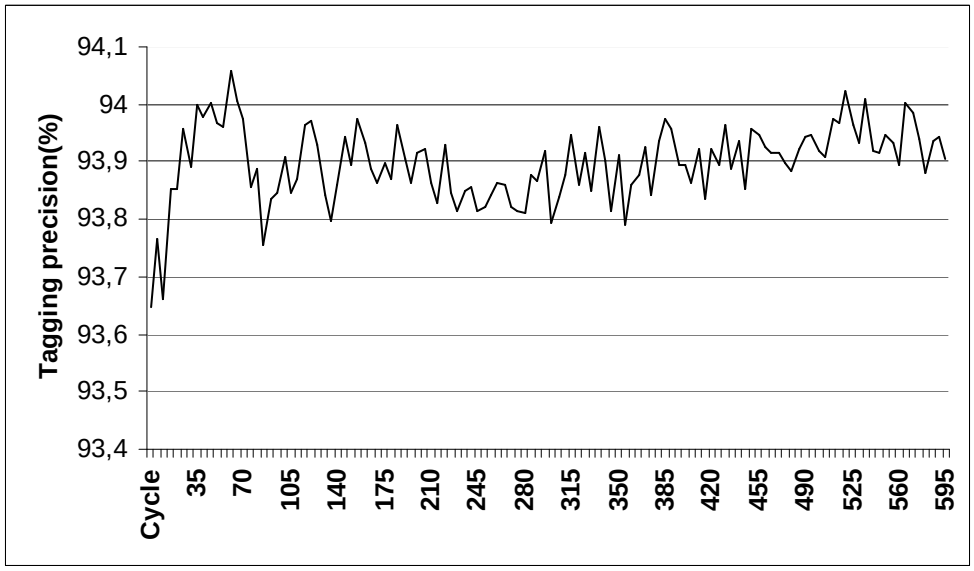
4.4.3 Final tests

The final tests were carried out on the entire training corpus with category single coding and suffix length 4. We tested several parameter configurations of the neural network (always with one hidden layer) and found out that the parameter values discussed in 4.4.2 lead to good learning performance even on the large training set. The hidden layer size was set to 400-500 neurons.

Although the neural network global error was decreasing during training (and the testing performance on the training set steadily increasing), the performance on the testing files usually reached its maximum very soon (before 200-th cycle). Even if it reached the best results on the testing set later, the difference from the early maximum was very small. This behaviour was observed for almost all experiments performed. To illustrate this graphically, Graph 1 and Graph 2 show the network convergence and testing data performance curves in a case category determination experiment.



Graph 1: Network convergence in a case determination experiment.



Graph 2: Tagging precision in a case determination experiment.

Table 17 summarizes the achieved results for the respective categories with and without template decision enhancement (see Section 4.3.3). This time we also trained the var category, which was not necessary in the preliminary tests presented in the previous sections.

	Evaluation testing set				Training set C			
Template	**GENDER**	**NUMBER**	**CASE**	**VAR**	**GENDER**	**NUMBER**	**CASE**	**VAR**
YES	94,51	97,12	94,46	99,46	98,29	99,37	98,3	99,92
NO	94,18	96,91	94,16	99,42	95,35	98,13	95,63	99,76

Table 17: Precision of reliable context category testing

We can see that there is a tagging precision leap between the template enhancement and non-enhanced network use on the training set, which signalizes that the neural network still produces high global error over the training set. However, as mentioned above, the neural network usually reaches its maximum performance over the testing set soon, so it does not necessarily mean that a lower global error over the training set would lead to tagging precision improvement on the testing data.

Finally, the optimal configurations of these partial category determination neural networks were taken and we performed the entire tag determination experiment with them (see Section 4.3.3). We obtained the final tagging precision value of 89,22%.

4.5 Discussion

The tagging precision of the best statistical method on the evaluation set is 93,47%, which is far better result than our 89,22%. However, the tagging precisions of the respective categories are much closer to the statistical approach results. The respective differences for the number and case categories are 0,88% and 0,91%. The significant difference (3,31%) between the two approaches lies in the gender category determination. This difference corresponds to the entire tag evaluation difference 4,25%. It is also possible that the measure we have defined for the entire tag determination may not be the best one, although we found out that similar measures (e.g. sum instead of product) behave in much the same way.

There is a crucial point to understand when comparing the results of the statistical and the neural network approach. Statistical approach is based on the use of Viterby algorithm that finds the optimal path among the respective alternative tags for the words of a sentence. Although using primarily the left context, this way the algorithm can determine the correct tag for a word with respect to the words appearing also in the right context. It also uses the entire word information and is as well further enhanced. On the other side, our approach relies on the left context and suffix information only, so it uses less information than the statistical one.

This may be one of the reasons, why the gender category determination is less successful – in order to determine this category we often need to consider the right context. For instance, it may contain the heading substantive for the adjective, which gender category is being determined[12]. Figure 7 shows a tree representation of the sentence “Evropský měnový systém je ohrožen” (“The European Monetary System is endangered”). The adjectives “evropský” (“European”), “měnový” (“Monetary”), and the noun “systém” (“System”) may be either nominative or accusative case in Czech. We have to know the rest of the sentence to be able to determine the correct case. Left context of arbitrary length does not suffice.

[12] However, this also holds for the number and case categories.

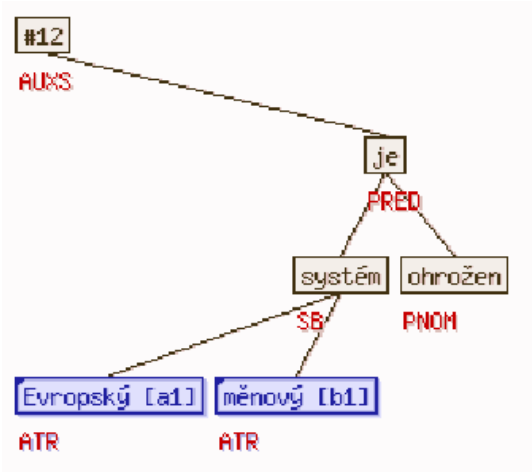


Figure 7: "The European Monetary System is endangered" tree representation (Orakulum query system [11])

As far as the network training is concerned, we believe that we have reached almost optimal state in all experiments. Although in some of them the global error remains high, we found out that neural network reaches its optimal performance over the testing set quite fast and further learning does not lead to significant precision increase.

In summary, we are confident that we were able to exploit the possibilities of the presented method. It shows its limits in comparison to the statistical approach. To be able to get over these limits, we would need to make use of the context in the way statistics does and arrange the experiment to suit neural network behaviour.

5 Statistical-context-based determination

As we have seen, reliable-context-based determination (see Chapter 4) results are not directly applicable, because we are not able to tag a continual text by that method.

Statistical-context-based determination removes this limitation by substituting the correct tags in the left context with the statistical outputs. These outputs are available beforehand as we may run a statistical tagger over the testing set. In every other aspect the experiment runs just like the previous one.

We might be able to directly "repair" the statistics results if we reached better tagging precision with this method. As this is not the case, we may at least compare the results with the reliable context determination approach to see how much would the tagging precision increase of the former experiment affect the precision of the latter and to use the results once we are able to improve the reliable context experiment.

5.1 Formal representation

The formal representation for the statistical-context-based determination experiment is the same as for the reliable-context-based determination experiment (see Section 4.1), except that the input vectors contain statistically determined tags (see Section 2.1) instead of the correct ones.

We will call "context provider" the statistical tagger, which output is used. If more than one statistical method is available, we may choose, which of them will serve as the context provider.

5.2 Results

The statistical tags on the training corpus are result of back tagging as this corpus also served as the training set for the statistics. This brings us a disadvantage, because the training statistical tags are not fully representative – they are too good. We have selected the Feature-based tagger as the context provider, because its tagging precision (92,74%) is slightly better than that of the Markov model tagger (92,58%).

As with the reliable context experiment, the final tests were carried out on the entire training corpus with category single coding and suffix length 4 using the same neural network learning parameters.

Table 18 summarizes the achieved results for the respective categories with and without template decision enhancement (see Section 4.3.3).

	Tagging precision			
Template	**GENDER**	**NUMBER**	**CASE**	**VAR**
YES	94,11	96,67	92,71	99,53
NO	93,81	96,52	92,13	99,52

Table 18: Precision of statistical context category testing

The entire tag tagging precision reached 88,71%.

5.3 Discussion

As expected, the obtained results are worse than those obtained for the reliable context. For the gender, number, and case categories the tagging precision differences are 0,4%, 0,45%, and 1,75%. However, the overall tagging difference is only 0,51%.

Considering the fact that the training set is not fully representative as it is the training set of the context provider itself, the differences are very low, which shows that we may substitute the reliable context with the statistical one and use it in the voting experiment.

6 Voting experiment

In the previous chapters we have described the experiments aimed at the determination of the correct tag given only the result of the morphological analysis. The tagging precision of these methods was significantly lower than that of the statistical approach.

This time we are trying to make use of the statistical methods. When given outputs of several statistical taggers for a given word in a sentence, we attempt to determine the correct one (if it is present among them, of course). The context is the statistical one, so the results are directly applicable (see Chapter 5).

6.1 Formal representation

6.1.1 Input coding

Input of the voting experiment consists of the left statistical context (see Section 5.1) and the candidate statistical tags for the given word. These tags are coded in the same way as the context tags. Additionally, we have tried to add right statistical context – statistical tags appearing right of the word, which tag is being determined. All the tags may be coded in variety of ways discussed in Section 4.1.1. Figure 8 shows the input vector structure of the voting experiment.

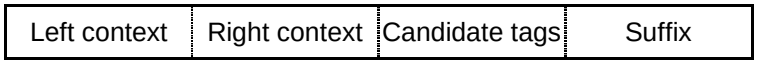


Figure 8: Voting input vector structure

6.1.2 Output coding

Neural network output represents the quality of the candidate statistical tags for the given input context. The size of the output vector is equal to the number of candidate tags and each component represents whether the corresponding candidate tag is the correct tag. If so, it is set to 1, otherwise it is set to 0.

This raw training set is then statistically merged as described in Section 4.1.2, so the output vector values represent the frequencies of the given statistical tag to be the correct tag in the given context.

6.1.3 Example

This section shows an example of neural network input and output for the voting experiment:

Let both left and right context length be 1 tag, let the suffix length be 3 characters. Again, let *s* be the sentence “Prezident rezignoval na svou funkci.” (“The president has stepped down.”). Let the output tags of two statistical methods (*a*, *b*) are available. The output of the statistical method *a* will also serve as the statistical context provider. Let the statistical outputs for the respective words of the sentence are those shown in Table 19.

Input token	Correct tag	Stat. method A	Stat. method B
Prezident:	NNMS1-----A----	NNMS1-----A----	NNMS1-----A----
rezignoval:	VpYS---XR-AA---	VpYS---XR-AA---	VpYS---XR-AA---
na:	RR--4----------	RR--4----------	RR--6----------
svou:	P8FS4---------1	P8FS7---------1	P8FS4---------1
funkci:	NNFS4-----A----	NNFS3-----A----	NNFS6-----A----
.:	Z:----------	Z:----------	Z:----------

Table 19: Voting experiment source information

If the virtual zero tags ("----------") are included, the network is trained on the set of vectors shown in Table 20.

Input					Output
----------	VpYS---XR-AA---	NNMS1-----A----	NNMS1-----A----	"ent"	(1,1)
NNMS1-----A----	RR--4----------	VpYS---XR-AA---	VpYS---XR-AA---	"val"	(1,1)
VpYS---XR-AA---	P8FS7---------1	RR--4----------	RR--6----------	"λna"	(1,0)
RR--4----------	NNFS3-----A----	P8FS7---------1	P8FS4---------1	"vou"	(0,1)
P8FS7---------1	Z:----------	NNFS3-----A----	NNFS6-----A----	"kci"	(0,0)
NNFS3-----A----	----------	Z:----------	Z:----------	"λλ."	(1,1)

Table 20: "Uncoded" (input,output) training pair in the voting experiment

6.2 Baseline values

First, we have measured the baseline results obtained by selecting a random statistical output.

The test has been run 20 times and the obtained values on the evaluation testing set were then averaged. The resulting value 92,69% is less than the sole output of the better statistical method (Feature-based tagger).

We therefore see that the tagging precision of statistical methods cannot be increased in such a naive way.

6.3 Results

Two statistical methods (Feature-based tagger and Markov model tagger) were used as the two statistical outputs. They reached 92,74% and 92,58% tagging precisions on the testing set, respectively.

As in the statistical-context-based determination experiment, we have selected the Feature-based tagger as the context provider.

The neural network parameters were set to the same values as in the previous types of experiments, i.e. learning rate from the range [0,1;0,2], momentum from the range [0,6;0,8]. The exact value of these parameters was set random in each experiment. Each coding experiment ran at least 500 cycles with 1 iteration.

Table 21 lists the hidden layer sizes for each context and suffix length configuration tested. Again, we learned that sizes overly below these numbers affect the BP ANN learning performance negatively.

Left context	Right context	Suffix length	Hidden layer
1	0	0	300
1	0	2	300
1	0	4	300
2	0	4	400
1	1	4	400

Table 21: Hidden layer sizes for the voting experiment

The testing phase was performed after each learning cycle as to be sure not to miss the overall best result.

We have tested several context sizes and suffix lengths. The best obtained results with and without template decision enhancement (see Section 4.3.3) are summarized in tables 22 and 23.

Left context	Right context	Suffix length	Precision(%)
1	0	0	93,33
1	0	2	93,44
1	0	4	93,36
1	1	4	93,47
2	1	4	93,44

Table 22: Voting experiment results

Left context	Right context	Suffix length	Precision(%)
1	0	0	93,56
1	0	2	93,52
1	0	4	93,48
1	1	4	93,51
2	0	4	93,47

Table 23: Voting experiment results, template decision

6.4 Discussion

BP ANN showed to be very successful when used as a voting device. It was able to reach higher tagging precision (93,56%) than any of the input statistical methods (92,74%, 92,58%) and the baseline value (92,69%). It was therefore able to improve the input statistical methods performance by 0,82%.

It also outperformed the currently best tagger CZ031219 that reaches tagging precision of 93,47% (see Section 2.4). Moreover, our results could have been even better if this statistical method had been included into the voting set[13].

[13] However, this experiment was not performed due to technical reasons by the time this thesis is written.

7 Conclusion

We have tried to use the BP ANN in several types of experiments. When determining the correct tag given a reliable context, we have learned that the neural tag is basically capable to handle the problem, although the achieved tagging precision (89,22%) did not reach that of the best statistical method (93,47%). We also managed to determine appropriate network and context parameters that we have used in the next experiments.

The attempt to determine the correct tag on the basis of beforehand statistically determined tags brought a slight decrease of tagging precision (88,71%).

Finally, the experiment, which goal was to vote from the outputs of two statistical taggers, showed higher tagging precision (93,56%) than any of these methods (92,74%, 92,58%). It is therefore the overall best result on the given training data set (PDT).

In summary, neural network approach proved to be very suitable for the morphological tagging task. A simple BP algorithm was able to exceed the tagging precision reached by the best statistical method available. However, it relied itself on the statistical output. This result shows that an union of the statistical methods and the neural network approach could be very promising.

Our approach has yet to be tested on the larger training set (Czech Corpus data), where the tagging precision of statistics reaches more than 95%.

There are many other experiments that are worth performing. For instance, recurrent neural networks performance on the morphological tagging task should be tested. Neural networks could be also trained to determine the optimal context parameters for the tagging task.

References

[1] Hajic J. (2002): Disambiguation of Rich Inflection (Computational Morphology of Czech), MFF UK, 334.

[2] Hladka B. (2000): Czech Language Tagging, PhD thesis, MFF UK, 135.

[3] Czech Corpus reference at http://ckl.ms.mff.cuni.cz/~sgd/UJC_corpus.html

[4] PDT 1.0 resources: http://ufal.ms.mff.cuni.cz/pdt

[5] Schmid H. (1994): Part-of-Speech Tagging with Neural Networks. Proceeding of COLING-94,172-176.

[6] Nakamura M., Maruyama K., Kawabata T., Shikano K. (1990): Neural network approach to word category prediction for English texts. Proceeding of COLING-90, 213-218.

[7] Weijters A. (1995): The BP-SOM architecture and learning rule. Neural Processing Letters, 13-16.

[8] Ton Weijters homepage at http://tmitwww.tm.tue.nl/staff/aweijters/bpsom.htm

[9] Thrun S.B., et. al (1991). The MONK's Problems: a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University.

[10] Mitchell, T. (1997): Machine learning. McGraw-Hill, 95.

[11] Orakulum homepage at http://quest.ms.mff.cuni.cz/orakulum

Appendix A, Software tools

In order to be able to perform the described experiments, several software tools have been designed and implemented:

- Perl scripts managing source data conversions
- **ExpLab**, the experiment manager application

Section A.1 contains instructions regarding data conversions and use of the ExpLab application. Brief remarks on ExpLab implementation can be found in Section A.2.

A.1 User's Guide

This section contains information on the installation of the software tools, conversion of data sources, and running the experiments. The electronic sources are located on the enclosed CD.

The following text assumes that you are already familiar with the terms presented in chapters 1 to 6 of this thesis.

A.1.1 System requirements

ExpLab may be installed on any **Windows** 98 system and higher. In order to be able to run the Perl scripts, a Perl interpreter (e.g. **ActivePerl**[14]) must be installed on your system.

A.1.2 CD contents

The directory structure of the enclosed CD is as follows:

- `\Programs\ExpLab` (contains ExpLab application executable)
- `\Programs\ExpLab\Lab` (ExpLab working directory, see Section B.1)
- `\Programs\ExpLab\Source` (ExpLab source code)
- `\Programs\ExpLab\Source\Doc` (ExpLab HTML and Latex documentation)
- `\Programs\Utils` (Perl scripts)
- `\Programs\Utils\Grader` (Grader tool source files)
- `\Data\Reliable` (compressed reliable-context-based determination experiments)
- `\Data\Stat` (compressed statistical-context-based determination experiments)
- `\Data\Vote` (compressed voting experiments)

Due to the licensing policy, there are no corpus data (original or converted) on the CD. Please obtain them from the Institute of Formal and Applied Linguistics[15].

[14] You may obtain this installation at http://www.activestate.com

[15] Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University, Prague at http://ufal.ms.mff.cuni.cz/pdt

A.1.3 Installation

1. Install ExpLab and the scripts by copying `X:\Programs` directory (where X is your CDROM drive) to an arbitrary directory on your harddrive.

2. If you wish to review the experiments stored on the CD, unpack the selected zipped files from the `\data` directory into a directory on your harddrive. This finishes the installation of resources located on the enclosed CD.

3. Obtain the PDT corpus morphological data[16] at http://ufal.ms.mff.cuni.cz/pdt. These data consist of many small files. (We recommend performing the following steps under **CygWin** – a Linux emulator under Windows, which you can get at http://www.cygwin.com. From now on, it is assumed that you have CygWin installed on your system.)

4. Unpack the obtained corpus files. Launch CygWin, switch to the directory with corpus files and run

   ```
   gunzip *
   ```

5. Merge the corpus files into single training files according to your preferences. Copy the selected files into a separate directory and run

   ```
   cat * > my_file
   ```

   where `my_file` is the name of the newly generated training file.

6. Merge the specified files into the development and the evaluation testing file in the same way as in step 5.

7. Create a special "global" file by merging all the available (training and testing) files.

8. Run the morphological analysis tool (as described in PDT 1.0 sources) on the files you have created in steps 5-6.

9. Convert the data as described in Section A.1.4.

10. You are now ready to run morphological tagging experiments with ExpLab.

[16] Of course, you may also use any other data in the same SGML format.

A.1.4 Data conversion

In order to run experiments in ExpLab, it is necessary to convert the PDT source files, which installation and merging is described in Section A.1.3, into the format readable by the ExpLab application.

Follow these steps to perform the conversion:

1. Copy the training and testing files that you created during the installation (see Section A.1.3) into `\Programs\Utils` in the installation directory. Switch to this directory.

2. Create character map table by running `CreateCharTable.pl` with `–l` parameter set to the maximum suffix length you wish to consider. The resulting file should contain all characters that appear in suffices of all words in the training and testing files. Send the global file (see Section A.1.3) to stdin and retrieve the table file from stdout. For example,

   ```
   perl CreateCharTable.pl –l 4 < global.txt > char_map.txt
   ```

   creates the `char_map.txt` – a character map containing all characters appearing in the suffices of length 4 in the source file global`.txt`.

3. Convert the source files by running either `ConvertFile.pl` (if you generate reliable-context-based determination files) or `ConvertFileStat.pl` (if you generate statistical-context-based determination or voting experiment files). Provide `–f` parameter set to the table file created in step 1, `-l` parameter set to the maximum suffix length you wish to consider, and (when running `ConvertFileStat.pl`) `–m` parameter determining the statistical methods for the context (value of the `<MDt src=“value“>` attribute). For example,

   ```
   perl CreateFileStat.pl –l 4 –f char_map.txt –m a,b
   < s1learn.txt > ..\ExpLab\Lab\Exp_stat\s1learn.bst
   ```

   creates the converted `s1learn.bst` file from the source file and places it into the appropriate directory. Please note that the naming conventions described in Section B.5 must be fulfilled.

4. Convert the global file in the same way as described in step 3 (perform the conversion by `ConvertFileStat.pl`):

   ```
   perl CreateFileStat.pl –l 4 –f char_map.txt –m a,b
               < global.txt > global.cnv
   ```

5. Create tag map table by running `CreateTagTable.pl` with `–a` parameter set to a converted file, which contains all tags present in the training and testing files, i.e. the converted global file. Set `–t` parameter to the file name, on which you want to base the tag occurrence frequency computation (this may be the entire training corpus). For example,

   ```
   perl CreateTagTable.pl -a global.cnv -t s3learn.txt >
               ..\ExpLab\Lab\TagTable.txt
   ```

creates `TagTable.txt` file and places it into the proper directory.

If the format of your files is different from that of PDT, you have to perform the conversion manually. See Section B.6 for the description of converted files format.

A.1.5 ExpLab environment

The testing manager works in a very easy way. It provides you with a single dialog window to set all the experiment parameters (Figure A.1). Once this is done, you click the **Start Experiment** button. The experiment specific output files (see Appendix B) are then generated and you may view them as the experiment runs.

This section contains detailed description of the user interface control items. For instructions on the experiment procedure see Section A.1.6.

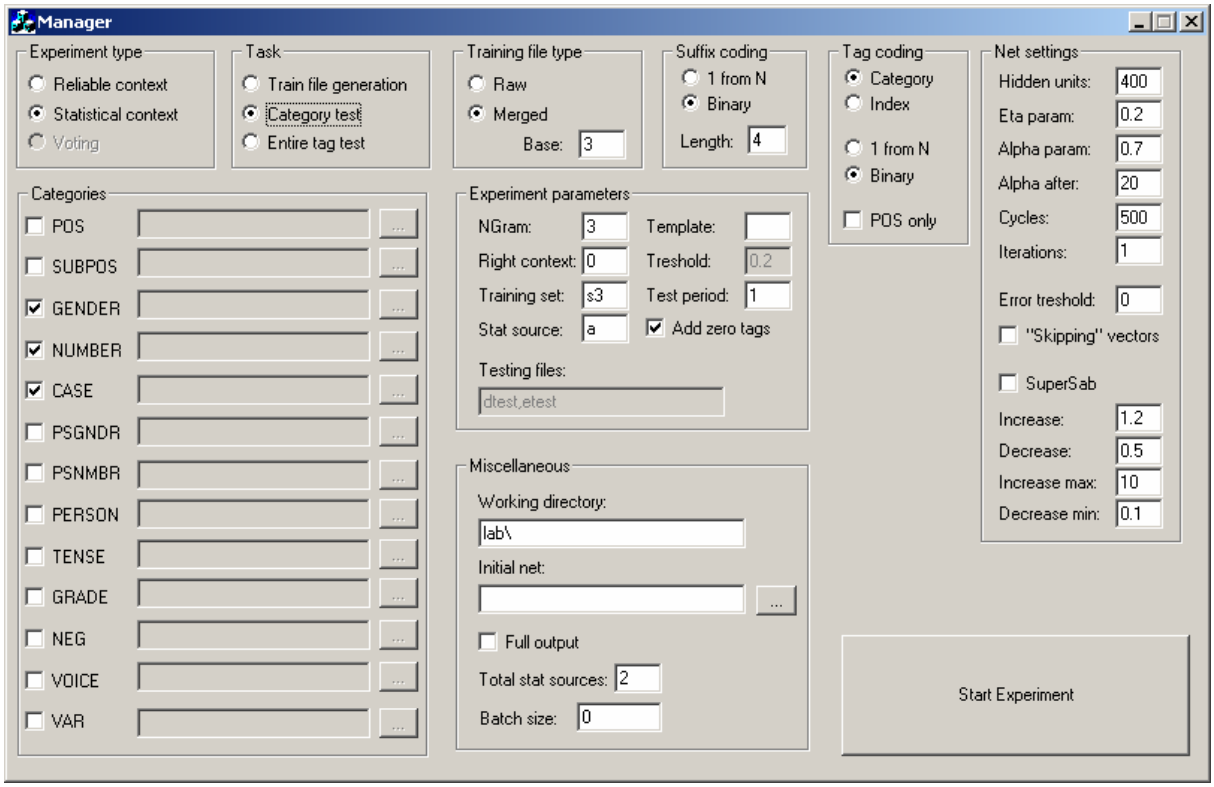


Figure A.1: ExpLab dialog window

- **Experiment type** sets one of the three experiments presented:
  - **Reliable-context-based** determination
  - **Statistical-context-based** determination
  - **Voting** for statistics

- **Task** determines the phase of the experiment set :
  - **Training file generation** for the given experiment type

- **Category test** performs the partial category training and testing for reliable or statistical-context-based determination
- **Entire tag test** starts the voting experiment or performs the complete tag determination based on the category tests for the reliable(statistical)-context based determination experiments

- **Training file type** determines whether the resulting training set (if it is to be generated) or the input training set (if a testing task is selected) is statistically merged:
  - **Raw** – no merging
  - **Merged** – merging with reliability threshold, which first decimal digit is **base**

- **Categories** section sets the morphological categories involved in the experiment.
  - Checking a category will generate the training set or include the category into the reliable(statistical)-context-based experiment for that category.
  - Typing or selecting a partial category experiment network file into the proper edit box will add the category into the complete tag determination experiment.

- **Net settings** sets the learning parameters of the neural network used:
  - **Hidden units** – hidden layer size
  - **Eta param** – weight adaptation coefficient
  - **Alpha param** – momentum
  - **Alpha after** – number of cycles till momentum is used
  - **Cycles** – training cycles (how many times is the training set learned)
  - **Iterations** – training iterations (how many times is each vector learned)
  - **Error threshold** – trains only the vectors, which error is higher than this value
  - **"Skipping" vectors** – if checked, the network is trained only on the vectors, which ordering of respective components differs from that of the target vector
  - **SuperSab** – if checked, SuperSab learning algorithm is used
  - **Increase** – SuperSab increase coefficient
  - **Decrease** – SuperSab decrease coefficient
  - **IncreaseMax** – SuperSab maximal eta value
  - **DecreaseMin** – SuperSab minimal eta value

- **Experiment parameters** sets the general experiment parameters:
  - **NGram** – left context taken into consideration (number of preceding tags plus one)
  - **Right context** – additional right context (number of following tags)
  - **Training set** – base training file prefix (e.g. for the base file "s2learn" prefix is "s2")

- **Stat source** – character defining the context statistical tags source (“a” – the first statistical tag source, “b” – the second one, etc.)
- **Template** – template file prefix. Network output is taken into consideration only if the input vector is not found within this file. If so, the corresponding output will be processed.
- **Threshold** – testing reliability threshold. If the output vector component is lower than this value, it will be considered to be zero.
- **Test period** – testing on the testing files is performed after the given number of training cycles
- **Add zero tags** – virtual zero tags are placed in front of and after the sentence, so that all its tags may be evaluated in the context of specified length
- **Testing files** – comma separated names of base testing files to perform testing on

- **Suffix coding** determines the way suffices are coded:
  - **1 from N**
  - **Binary**
  - **Length** determines the suffix length considered

- **Tag coding** determines the way tags are coded:
  - **Category** – category tag coding
  - **Index** – index tag coding
  - **1 from N**
  - **Binary**
  - **POS only** – only the part-of-speech category is coded for each tag

- **Miscellaneous** consists of all other items:
  - **Working directory** specifies path to the experiments directory
  - **Initial network** specifies initial network file for the experiment
  - **Total stat sources** specifies the number of stat sources in the input vector file
  - **Full output** – when checked, the answer tags list file is generated. Furthermore, network output is assigned to each vector of the testing and training files, and is printed in a special file.

A.1.6 Experiment procedure

This section guides you through the experiment procedure. For the description of the respective experiment parameters see Section A.1.5. For the description of the generated data see Appendix B.

Reliable-context-based determination experiment

1. Start ExpLab. Select *Reliable context* in the *Experiment type* section and set task type to *Training data generation*. Select the morphological categories you want to generate training files for. Set the experiment parameters. Run the experiment. Raw training files for the selected categories will be generated.

2. Once the generation finishes, perform the statistical merging by running the newly created batch file. (If statistical merging is selected.)

3. Start ExpLab. Select *Reliable context* in the *Experiment type* section and set task type to *Category test*. Select the categories you want to run the testing experiment for. (The experiments will run sequentially for each category). Set the experiment parameters. Run the experiment. A new experiment directory will be created and output files will be written there as the experiment runs.

4. Run the entire tag determination experiment. Start ExpLab. Select *Reliable context* in the *Experiment type* section and set task type to *Entire tag test*. Type or browse for the network configuration files from the previous category determination experiment into the respective edit boxes. Set the experiment parameters. Run the experiment. The result is written to the *ct_reliable* directory.

Statistical-context-based determination experiment

The procedure is the same as for the reliable-context-based determination (just select *Statistical context* as the Experiment type).

Voting experiment

1. Start ExpLab. Select *Voting* in the *Experiment type* section and set task type to *Training data generation*. Set the parameters. Run the experiment. A single raw training file will be generated.

2. Once the generation finishes, perform the statistical merging by running the newly created batch file. (If statistical merging is selected.)

3. Start ExpLab. Select *Voting* in the Experiment type section and set task type to *Entire tag test.* Set the experiment parameters. Run the experiment. A new experiment directory will be created and output files will be written there as the experiment runs.

A.2 Implementation remarks

This chapter briefly describes the implementation of the ExpLab application. The source code itself is well documented and the detailed documentation[17] is also available in HTML and Latex format on the enclosed CD (see Section A.1.2).

A.2.1 Development environment

As the experiment evaluation scheme requires tight connection to the neural network instance configuration, we have decided to integrate our own backpropagation network implementation into the application. In order to reach the optimal neural network learning performance, the application has been implemented in C++. To be able to create a user-friendly graphical interface easily, we have decided to use *Microsoft Visual C++ 6.0* as the development tool.

A.2.2 Object skeleton

The basic functionality of the most important classes is described in this section:

- The dialog class **CExpLabDlg** underlies the main (and only) application form, where the user sets the experiment parameters. It handles behaviour of the form controls and runs the selected experiments – creates the instances of the experiment classes described below and runs their generation or testing methods.

- **CExperimentSWTA/CExperimentSWTAStat** generates training set and runs testing for the particular category in the reliable(statistical) context experiment via its **Generate** and **Test** methods respectively.

- **CExperimentCWTA/CExperimentCWTAStat** runs the entire tag determination experiment based on the results (input networks) on the respective categories in the Reliable(Statistical) context experiment via its **Commence** method.

- **CExperimentVoting** generates training set and runs testing for the Voting experiment via its **Generate** and **Test** methods respectively.

- **CDataGenerator/CDataGeneratorStat** reads the source data into internal data structures (a list of **CSentence** objects). These classes are used by the experiment classes to load the source data.

- **CBPNet** class is the implementation of the backpropagation algorithm.

[17] The documentation was created by the Doxygen system (http://www.stack.nl/~dimitri/doxygen)

Appendix B, Electronic sources description

This appendix describes structure and naming of the experiment files and directories.

B.1 Experiment working directory structure

The experiment working directory (`lab` by default) contains the following items:

- `exp_reliable` subdirectory, where the source data files for the reliable-context-based determination experiment are located
- `exp_stat` subdirectory, where the source data files for the statistical-context-based determination experiment are located
- `exp_vote` subdirectory, where the source data files for the voting experiment are located
- `ct_reliable` subdirectory, where the result of entire tag determination for the reliable-context-based determination experiment is written
- `ct_stat` subdirectory, where the result of entire tag determination for the statistical-context-based determination experiment is written
- Testing experiment subdirectories for the respective experiments (see Section B.2)
- Generated training files (together with the batch trigger)
- `Grader.exe` statistical merging tool
- `TagTable.txt` containing tag indexing and frequency information

B.2 Testing directory name

A separate directory is created for each testing experiment. The name of the directory consists of codes for the set parameters that are separated by underscore. The codes in the respective order are explained below:

- *Experiment type* is
  - **r** for the reliable-context-based determination
  - **s** for the statistical-context-based determination
  - **v** for the voting experiment
- *Training set* is the prefix of the training data file (e.g. “s1” for “s1learn”)
- *Category* is the name of the category for the experiments dealing with particular category, **na** otherwise
- *Tag coding* is
  - **c** for category tag coding
  - **i** for index tag coding

- *Tag value* coding is
  - **b** for binary coding
  - **o** for one from n coding
- *Suffix coding and length* is
  - **b***l* for binary coding, *l* is the suffix length
  - **o***l* for one from n coding, *l* is the suffix length
  - **0**, if no suffix is present
- *Training set type and reliability threshold* is
  - **m***t* for statistically merged data set with reliability threshold value *t*/10
  - **r**, if raw data set is trained
- *n in n-gram* (i.e. left context length + 1)
- *Additional right context length*
- *Zero tags status* is
  - **z0**, zero tags are not included at the begging and end of each sentence
  - **z1**, if they are included
- *Whole tag status* is
  - **c**, if only part-of-speech part of a tag is coded
  - **f**, if entire tag is coded
- *Statistical source* is the code of the context provider statistical method (statistical-context-based determination and voting experiments only, **na** otherwise)
- *Total statistical methods* is the number of statistical taggers, which outputs are voted from (voting experiment only)
- *SuberSab status* is
  - **s0**, if standard backpropagation learning algorithm is used
  - **s1**, if SuperSab learning algorithm is used
- *Number of units in the hidden layer*
- *Number of training cycles*
- *Number of training iterations*
- *Momentum term * 100*
- *Learning rate * 100*
- *SuperSab increase term * 100*
- *SuperSab decrease term * 100*
- *SuperSab maximum increase term * 100*

- *SuperSab minimum decrease term * 100*
- *Skipping vectors only* status is
  - **s0** or **k0** if all vectors are trained
  - **s1** or **k1** if the network is trained only on the vectors, which ordering of respective components differs from that of the target vector
- *Training tolerance threshold * 100* (only vectors, which error – distance from the target vector - is higher than this number will be trained on)
- *Testing reliability threshold * 100* (network output below this value is considered to be 0)

B.3 Testing output files

The testing report and performance output files are simple plain text files containing self-explanatory entries. Hence, we do not describe their internal structure in this appendix.

B.3.1 Category testing and voting experiment

The following set of output files is generated during the category testing in the reliable (statistical)-context-based determination and voting experiments:

- *cycle_number*`.net`, the network configuration file, which may be later loaded as the initialize the net
- *cycle_number*`.rf`*i*, testing report on the i-th testing file (testing files are numbered in order specified in the user interface edit box)
- *cycle_number*`.pln`, neural network performance output over the training set

Additionally, `progress.txt` file records the testing results summary for each testing period and keeps track of the overall best results.

B.3.2 Entire tag determination

The following files are generated in `ct_reliable` (`ct_stat`) directory during the entire tag determination in the reliable (statistical)-context-based determination experiment:

- `test_`*name*`.rf`, testing report on the testing file *name*

B.4 Training file names

The names of generated training files for a particular experiment are coded similarly as the directory names. However, the parameters set is somewhat reduced. It contains the following items (which meaning is the same as described in Section B.2):

- *Experiment type*
- *Training set*
- *Tag coding*
- *Tag value*
- *Suffix coding and length*
- *Training set type and reliability threshold*
- *n in n-gram*
- *Additional right context length*
- *Zero tags status*
- *Whole tag status*
- *Statistical source*
- *Total statistical methods*

B.5 Base file names

Base files are located in the proper experiment directories (exp_reliable, exp_stat, exp_vote).

The names of base training files match the following pattern: `s`*`i`*`learn.`*`suffix`*, where i is an arbitrary digit[18] and suffix is

- `brl` (in `exp_reliable`)
- `bst` (in `exp_stat`)
- `bvt` (in exp_vote)

The names of base testing files can be arbitrary, the suffix is determined in the same way as that of the training files.

B.6 Internal files format

B.6.1 Reliable-context-based determination source files

The structure of the reliable-context-based determination experiment files is as follows (`\n` denotes the newline character):

[18] We recommend to number the training sets sequentially.

```
Number_of_suffix_digits Width_of_suffix_digit\n
\n
\n #Sentence 1
CorrectTag1\n #Correct tag of the first word of the sentence
PossibleTag11 PossibleTag12 ...\n #Possible tags for this word
CorrectTag2\n #Correct tag of the secord word of the sentence
PossibleTag21 PossibleTag22 ...\n #Possible tags for this word
.
.
\n #Sentence 2
CorrectTag1\n #Correct tag of the first word of the sentence
PossibleTag11 PossibleTag12 ...\n #Possible tags for this word
CorrectTag2\n #Correct tag of the secord word of the sentence
PossibleTag21 PossibleTag22 ...\n #Possible tags for this word
.
.
.
```

B.6.2 Statistical-context-based determination and voting source files

The structure of the statistical-context-based determination experiment and voting experiment files is as follows (\n denotes the newline character):

```
Number_of_suffix_digits Width_of_suffix_digit\n
\n
\n #Sentence 1
CorrectTag1\n #Correct tag of the first word of the sentence
#Statistical tags of the respective taggers, possible tags
StatTag11 StatTag12 ... PossibleTag11 PossibleTag12 ...\n
CorrectTag2\n
StatTag21 StatTag21 ... PossibleTag21 PossibleTag22 ...\n
.
.
\n #Sentence 2
CorrectTag1\n #Correct tag of the first word of the sentence
#Statistical tags of the respective taggers, possible tags
StatTag11 StatTag12 ... PossibleTag11 PossibleTag12 ...\n
CorrectTag2\n
StatTag21 StatTag21 ... PossibleTag21 PossibleTag22 ...\n
.
.
.
```

B.6.3 Tag table

The structure of the tag table is as follows (\n denotes the newline character):

```
Tag1 Number_of_occurrences Order\n
Tag2 Number_of_occurrences Order\n
.
.
.
```

Appendix C, Positional tag definition

The following tables give detailed description of the positional tag morphological categories and the values these may acquire. They were taken from the PDT1.0 resources [4].

No.	Name	Description
1	POS	Part of Speech
2	SUBPOS	Detailed Part of Speech
3	GENDER	Gender
4	NUMBER	Number
5	CASE	Case
6	POSSGENDER	Possessor's Gender
7	POSSNUMBER	Possessor's Number
8	PERSON	Person
9	TENSE	Tense
10	GRADE	Degree of comparison
11	NEGATION	Negation
12	VOICE	Voice
13	RESERVE1	Unused
14	RESERVE2	Unused
15	VAR	Variant, Style, Register, Special Usage

Table C.1: Morphological tag categories

Value	Description
A	Adjective
C	Numeral
D	Adverb
I	Interjection
J	Conjunction
N	Noun
P	Pronoun
V	Verb
R	Preposition
T	Particle
X	Unknown, Not Determined, Unclassifiable
Z	Punctuation (also used for the *Sentence Boundary* token)

Table C.2: POS category values description

Value	Description
!	Abbreviation used as an adverb (now obsolete)
#	Sentence boundary (for the *virtual* word ###)
*	Word krát (lit.: *times*) (POS: C, numeral)
,	Conjunction subordinate (incl. aby, kdyby in all forms)
.	Abbreviation used as an adjective (now obsolete)
0	Preposition with attached -ň (pronoun něj, lit. *him*); proň, naň, ... (POS: P, pronoun)
1	Relative possessive pronoun jehož, jejíž, ... (lit. *whose* in subordinate relative clause)
2	Hyphen (always as a separate token)
3	Abbreviation used as a numeral (now obsolete)
4	Relative/interrogative pronoun with adjectival declension of both types (*soft* and *hard*) (jaký, který, čí, ..., lit. *what*, *which*, *whose*, ...)
5	The pronoun *he* in forms requested after any preposition (with prefix n-: něj, něho, ..., lit. *him* in various cases)
6	Reflexive pronoun se in long forms (sebe, sobě, sebou, lit. *myself* / *yourself* / *herself* / *himself* in various cases; se is personless)
7	Reflexive pronouns se (CASE = 4), si (CASE = 3), plus the same two forms with contracted -s: ses, sis (distinguished by PERSON = 2; also number is singular only)
8	Possessive reflexive pronoun svůj (lit. *my*/*your*/*her*/*his* when the possessor is the subject of the sentence)
9	Relative pronoun jenž, již, ... after a preposition (n-: něhož, niž, ..., lit. *who*)
:	Punctuation (except for the virtual sentence boundary word ###, which uses the SUBPOS #)
;	Abbreviation used as a noun (now obsolete)
=	Number written using digits (POS: C, numeral)
?	Numeral kolik (lit. *how many*/*how much*)
@	Unrecognized word form (POS: X, unknown)
A	Adjective, general
B	Verb, present or future form
C	Adjective, nominal (short, participial) form rád, schopen, ...
D	Pronoun, demonstrative (ten, onen, ..., lit. *this*, *that*, *that ... over there*, ...)
E	Relative pronoun což (corresponding to English *which* in subordinate clauses referring to a part of the preceding text)
F	Preposition, part of; never appears isolated, always in a phrase (nehledě (na), vzhledem (k), ..., lit. *regardless*, *because of*)
G	Adjective derived from present transgressive form of a verb
H	Personal pronoun, clitical (short) form (mě, mi, ti, mu, ...); these forms are used in the second position in a clause (lit. *me*, *you*, *her*, *him*), even though some of them (mě) might be regularly used anywhere as well
I	Interjections (POS: I)
J	Relative pronoun jenž, již, ... not after a preposition (lit. *who*, *whom*)
K	Relative/interrogative pronoun kdo (lit. *who*), incl. forms with affixes -ž and -s (affixes are distinguished by the category VAR (for -ž) and PERSON (for -s))
L	Pronoun, indefinite všechnen, sám (lit. *all*, *alone*)
M	Adjective derived from verbal past transgressive form
N	Noun (general)
O	Pronoun svůj, nesvůj, tentam alone (lit. *own self*, *not-in-mood*, *gone*)
P	Personal pronoun já, ty, on (lit. *I*, *you*, *he*) (incl. forms with the enclitic -s, e.g. tys, lit. *you're*); gender position is used for third person to distinguish on/ona/ono (lit. *he/she/it*), and number for all three persons
Q	Pronoun relative/interrogative co, copak, cožpak (lit. *what*, *isn't-it-true-that*)
R	Preposition (general, without vocalization)
S	Pronoun possessive můj, tvůj, jeho (lit. *my*, *your*, *his*); gender position used for third person to distinguish jeho, její, jeho (lit. *his, her, its*), and number for all three pronouns
T	Particle (POS: T, particle)
U	Adjective possessive (with the masculine ending -ův as well as feminine -in)
V	Preposition (with vocalization -e or -u): (ve, pode, ku, ..., lit. *in*, *under*, *to*)
W	Pronoun negative (nic, nikdo, nijaký, žádný, ..., lit. *nothing*, *nobody*, *not-worth-mentioning*,

	no/none)
X	(temporary) Word form recognized, but tag is missing in dictionary due to delays in (asynchronous) dictionary creation
Y	Pronoun relative/interrogative co as an enclitic (after a preposition) (oč, nač, zač, lit. *about what*, *on/onto what*, *after/for what*)
Z	Pronoun indefinite (nějaký, některý, číkoli, cosi, ..., lit. *some*, *some*, *anybody's*, *something*)
^	Conjunction (connecting main clauses, not subordinate)
a	Numeral, indefinite (mnoho, málo, tolik, několik, kdovíkolik, ..., lit. *much/many*, *little/few*, *that much/many*, *some (number of)*, *who-knows-how-much/many*)
b	Adverb (without a possibility to form negation and degrees of comparison, e.g. pozadu, naplocho, ..., lit. *behind*, *flatly*); i.e. both the NEGATION as well as the GRADE attributes in the same tag are marked by - (Not applicable)
c	Conditional (of the verb být (lit. *to be*) only) (by, bych, bys, bychom, byste, lit. *would*)
d	Numeral, generic with adjectival declension (dvojí, desaterý, ..., lit. *two-kinds/...*, *ten-...*)
e	Verb, transgressive present (endings -e/-ě, -íc, -íce)
f	Verb, infinitive
g	Adverb (forming negation (NEGATION set to A/N) and degrees of comparison GRADE set to 1/2/3 (comparative/superlative), e.g. velký, za\-jí\-ma\-vý, ..., lit. *big*, *interesting*
h	Numeral, generic; only jedny and nejedny (lit. *one-kind/sort-of*, *not-only-one-kind/sort-of*)
i	Verb, imperative form
j	Numeral, generic greater than or equal to 4 used as a syntactic noun (čtvero, desatero, ..., lit. *four-kinds/sorts-of*, *ten-...*)
k	Numeral, generic greater than or equal to 4 used as a syntactic adjective, short form (čtvery, ..., lit. *four-kinds/sorts-of*)
l	Numeral, cardinal jeden, dva, tři, čtyři, půl, ... (lit. *one*, *two*, *three*, *four*); also sto and tisíc (lit. *hundred*, *thousand*) if noun declension is not used
m	Verb, past transgressive; also archaic present transgressive of perfective verbs (ex.: udělav, lit. *(he-)having-done*; arch. also udělaje (VAR = 4), lit. *(he-)having-done*)
n	Numeral, cardinal greater than or equal to 5
o	Numeral, multiplicative indefinite (-krát, lit. (*times*): mnohokrát, tolikrát, ..., lit. *many times*, *that many times*)
p	Verb, past participle, active (including forms with the enclitic -s, lit. *'re* (*are*))
q	Verb, past participle, active, with the enclitic -ť, lit. (perhaps) *-could-you-imagine-that?* or *but-because-* (both archaic)
r	Numeral, ordinal (adjective declension without degrees of comparison)
s	Verb, past participle, passive (including forms with the enclitic -s, lit. *'re* (*are*))
t	Verb, present or future tense, with the enclitic -ť, lit. (perhaps) *-could-you-imagine-that?* or *but-because-* (both archaic)
u	Numeral, interrogative kolikrát, lit. *how many times?*
v	Numeral, multiplicative, definite (-krát, lit. *times*: pětkrát, ..., lit. *five times*)
w	Numeral, indefinite, adjectival declension (nejeden, tolikátý, ..., lit. *not-only-one*, *so-many-times-repeated*)
x	Abbreviation, part of speech unknown/indeterminable (now obsolete)
y	Numeral, fraction ending at -ina (POS: C, numeral); used as a noun (pětina, lit. *one-fifth*)
z	Numeral, interrogative kolikátý, lit. *what* (*at-what-position-place-in-a-sequence*)
}	Numeral, written using Roman numerals (XIV)
~	Abbreviation used as a verb (now obsolete)

Table C.3: SUBPOS category description

Value	Description
-	Not applicable
F	Feminine
H	Feminine or Neuter
I	Masculine inanimate
M	Masculine animate
N	Neuter
Q	Feminine (with singular only) or Neuter (with plural only); used only with participles and nominal forms of adjectives
T	Masculine inanimate or Feminine (plural only); used only with participles and nominal forms of adjectives
X	Any of the basic four genders
Y	Masculine (either animate or inanimate)
Z	Not fenimine (i.e., Masculine animate/inanimate or Neuter); only for (some) pronoun forms and certain numerals

Table C.4: GENDER category description

Value	Description
-	Not applicable
D	Dual
P	Plural
S	Singular
W	Singular for feminine gender, plural with neuter; can only appear in participle or nominal adjective form with gender value Q
X	Any

Table C.5: NUMBER category description

Value	Description
-	Not applicable
1	Nominative
2	Genitive
3	Dative
4	Accusative
5	Vocative
6	Locative
7	Instrumental
X	Any

Table C.6: CASE category description

Value	Description
-	Not applicable
F	Feminine possessor
M	Masculine animate possessor (adjectives only)
X	Any gender
Z	Not feminine (both masculine or neuter)

Table C.7: POSSGENDER category description

Value	Description
-	Not applicable
P	Plural (possessor)
S	Singular (possessor)

Table C.8: POSSNUMBER category description

Value	Description
-	Not applicable
1	1st person
2	2nd person
3	3rd person
X	Any person

Table C.9: PERSON category description

Value	Description
-	Not applicable
F	Future
H	Past or Present
P	Present
R	Past
X	Any (Past, Present, or Future)

Table C.10: TENSE category description

Value	Description
-	Not applicable
1	Positive
2	Comparative
3	Superlative

Table C.11: GRADE category description

Value	Description
-	Not applicable
A	Affirmative (not negated)
N	Negated

Table C.12: NEGATION category description

Value	Description
-	Not applicable
A	Active
P	Passive

Table C.13: VOICE category description

Value	Description
-	Not applicable

Table C.14: RESERVE1 category description

Value	Description
-	Not applicable

Table C.15: RESERVE2 category description

Value	Description
-	Not applicable (basic variant, standard contemporary style; also used for standard forms allowed for use in writing by the Czech Standard Orthography Rules despite being marked there as colloquial)
1	Variant, second most used (*less frequent*), still standard
2	Variant, rarely used, bookish, or archaic
3	Very archaic, also archaic + colloquial
4	Very archaic or bookish, but standard at the time
5	Colloquial, but (almost) tolerated even in public
6	Colloquial (standard in spoken Czech)
7	Colloquial (standard in spoken Czech), less frequent variant
8	Abbreviations
9	Special uses, e.g. personal pronouns after prepositions etc.

Table C.16: VAR category description