Large tagset labeling using Feed Forward Neural Networks. Case study on Romanian Language

Tiberiu Boroș
Research Institute for Artificial Intelligence "Mihai Drăgănescu", Romanian Academy
tibi@racai.ro

Radu Ion
Research Institute for Artificial Intelligence "Mihai Drăgănescu", Romanian Academy
radu@racai.ro

Dan Tufiș
Research Institute for Artificial Intelligence "Mihai Drăgănescu", Romanian Academy
tufis@racai.ro

Abstract

Standard methods for part-of-speech tagging suffer from data sparseness when used on highly inflectional languages (which require large lexical tagset inventories). For this reason, a number of alternative methods have been proposed over the years. One of the most successful methods used for this task, called Tiered Tagging (Tufiș, 1999), exploits a reduced set of tags derived by removing several recoverable features from the lexicon morpho-syntactic descriptions. A second phase is aimed at recovering the full set of morpho-syntactic features. In this paper we present an alternative method to Tiered Tagging, based on local optimizations with Neural Networks and we show how, by properly encoding the input sequence in a general Neural Network architecture, we achieve results similar to the Tiered Tagging methodology, significantly faster and without requiring extensive linguistic knowledge as implied by the previously mentioned method.

1 Introduction

Part-of-speech tagging is a key process for various tasks such as `information extraction, text-to-speech synthesis, word sense disambiguation and machine translation. It is also known as lexical ambiguity resolution and it represents the process of assigning a uniquely interpretable label to every word inside a sentence. The labels are called POS tags and the entire inventory of POS tags is called a tagset.

There are several approaches to part-of-speech tagging, such as Hidden Markov Models (HMM) (Brants, 2000), Maximum Entropy Classifiers (Berger et al., 1996; Ratnaparkhi, 1996), Bayesian Networks (Samuelsson, 1993), Neural Networks (Marques and Lopes, 1996) and Conditional Random Fields (CRF) (Lafferty et al., 2001). All these methods are primarily intended for English, which uses a relatively small tagset inventory, compared to highly inflectional languages. For the later mentioned languages, the lexicon tagsets (called morpho-syntactic descriptions (Calzolari and Monachini, 1995) or MSDs) may be 10-20 times or even larger than the best known tagsets for English. For instance Czech MSD tagset requires more than 3000 labels (Collins et al., 1999), Slovene more than 2000 labels (Erjavec and Krek, 2008), and Romanian more than 1100 labels (Tufiş, 1999). The standard tagging methods, using such large tagsets, face serious data sparseness problems due to lack of statistical evidence, manifested by the non-robustness of the language models. When tagging new texts that are not in the same domain as the training data, the accuracy decreases significantly. Even tagging in-domain texts may not be satisfactorily accurate.

One of the most successful methods used for this task, called Tiered Tagging (Tufiș, 1999), exploits a reduced set of tags derived by removing several *recoverable* features from the lexicon morpho-syntactic descriptions. According to the MULTEXT EAST lexical specifications (Erjavec and Monachini, 1997), the Romanian tagset consists of a number of 614 MSD tags (by exploiting the case and gender regular syncretism) for wordforms and 10 punctuation tags (Tufiş et al., 1997), which is still significantly larger than the tagset of English. The MULTEX EAST version 4 (Erjavec, 2010) contains specifications for a total of 16 languages: Bulgarian, Croatian, Czech, Estonian, English, Hungarian, Romanian, Serbian, Slovene, the Resian dialect of Slovene, Macedonian, Persian, Polish, Russian, Slovak, and Ukrainian

The strategy of the Tiered Tagging methodology is to use a reduced tagset (called CTAG-set), where a CTAG is a generalization of a MSD, from which recoverable context-irrelevant features are removed. For instance, the attribute for gender (masculine ‘m’ or feminine ‘f’) from MSDs ‘Ncfsrn’ and ‘Ncmsrn’ is deleted to obtain the CTAG ‘NSRN’, because the gender information can be deterministically recovered based on the CTAG and the wordform itself. The recovering of the left out attributes (Figure 1) is based on the lexicons, linguistic rules and, in the case of unknown words, on ML techniques (Ceauşu, 2006). When tagging with CTAGs, one can use any statistical POS tagging method such as HMMs, Maximum Entropy Classifiers, Bayesian Networks, CRFs, etc., followed by the CTAG to MSD recovery.

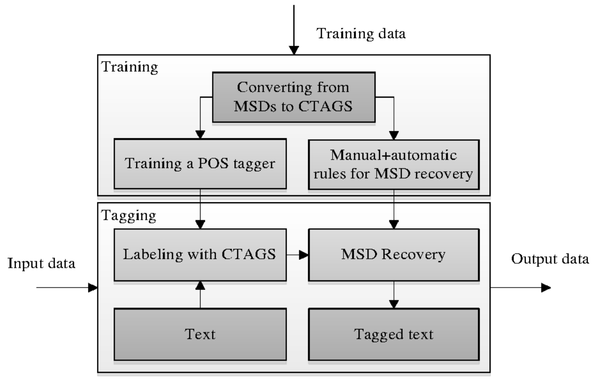

Figure 1 - Tiered Tagging methodology

The language dependent process of manually inferring linguistic rules for MSD recovery requires good knowledge of the target language and also extensive amounts of time invested in testing and re-design. It is difficult even for a native speaker to create such rules without in-depth linguistic knowledge.

In this article, we propose an alternative solution based on local optimizations with feed-forward neural networks. Our method eliminates the need for the two stage processing, is much faster at run time and is comparatively accurate with the Tiered Tagging implemented in the TTL tagger (Ion, 2007), available in the METASHARE Platform[1].

[1] http://ws.racai.ro:9191/

2 Large tagset part-of-speech tagging with feed-forward neural networks

Although removing the recoverable attributes, as proposed in the Tiered Tagging approach, helps the goal of squeezing the lexical tagset to a reasonable size, valuable information for contextual disambiguation is also lost. For example, the gender agreement rule, valid in many languages, cannot be exploited unless the gender attribute is present in the tags.

Our proposal to deal with large tagsets without removing contextually useful information is based on Feed Forward Neural Networks (FFNN). FFNN are known for their simplicity and robustness in finding and recombining patterns inside data.

Several neural network architectures have been proposed for the task of part-of-speech tagging. Schmid (1994) proposed a FFNN architecture for part-of-speech tagging obtaining a 96.22% accuracy. In his paper, he argues that neural networks are preferable to other methods, when the training set is small. He compares his results with a HMM tagger (94.24%) and a trigram tagger (96.02%), both trained and tested on the same corpora as his FFNN tagger (the Penn-Treebank corpus). A similar approach is presented by Marques and Lopes (1996). In their paper, the authors come to similar conclusions as those presented in Schmid (1994). We support these findings with an additional argument, namely the better fit for managing large tagsets.

In both approaches mentioned before, the network is trained so that from the input vector, to output a real valued vector. Each value in the output vector is generated by a distinct neuron, and corresponds to a unique tag in the tagset (e.g. 100 tags means the network contains 100 neurons on the output layer). The input vector for predicting the tag of the current word encodes the tags for the previously tagged words and the probable tags for the current and following two words, estimated using Maximum Likelihood Estimation (MLE):

$$P(t|w) = \frac{C(w,t)}{C(w)} \qquad (1)$$

$P(t\|w)$	-	The probability of the word w having tag t
$C(w,t)$	-	The total number of times, the word w appears with tag t in the training corpus
$C(w)$	-	The total number of times, the word w appears in the training corpus

In the case of out-of-vocabulary (OOV) words, both approaches use suffix analysis to determine the most probable tags that can be assigned to the current word.

To clarify how these two methods work, if we want to train the network to label the current word, using a context window of 1 (previous tag, current possible tags, and possible tags for the next word) and if we have, say 100 tags in the tagset, the input is a real valued vector of 300 sub-unit elements and the output is a vector which contains 100 elements, also sub-unit real numbers. As mentioned earlier, each value in the output vector corresponds to a distinct tag from tagset and the tag assigned to the current word is chosen to correspond to the maximum value inside the output vector.

The previously proposed methods still suffer from the same issue of data sparseness when applied to MSD tagging. However, in our approach, we overcome the problem through a different encoding of the input data (see section 2.1).

The power of neural networks results mainly from their ability to attain activation functions over different patterns via their learning algorithm. By properly encoding the input sequence, the network chooses which input features contribute in determining the output features for MSDs (e.g. patterns composed of part of speech, gender, case, type etc. contribute independently in selecting the optimal output sequence). This way, we removed the need for explicit MSD to CTAG conversion and MSD recovery from CTAGs.

2.1 The MSD binary encoding scheme

A MSD language independently encodes a part of speech (POS) with the associated lexical attribute values as a string of positional ordered character codes (Erjavec, 2004). The first character is an upper case character denoting the part of speech (e.g. 'N' for nouns, 'V' for verbs, 'A' for adjectives, etc.) and the following characters (lower letters or '-') specify the instantiations of the characteristic lexical attributes of the POS. For example, the MSD 'Ncfsrn', specifies a noun (the first character is 'N') the type of which is common ('c', the second character), feminine gender ('f'), singular number ('s'), in nominative/accusative case ('r') and indefinite form ('n'). If a specific attribute is not relevant for a language, or for a given combination of feature-values, the character '-' is used in the corresponding position. For a language which does not morphologically mark the gender and definiteness features, the earlier exemplified MSD will be encoded as 'Nc-sr-'.

In order to derive a binary vector for each of the 614 MSDs of Romanian we proceeded to:

1. List and sort all possible POSes of Romanian (16 POSes) and form a binary vector with 16 positions in which position *k* is equal 1 only if the respective MSD has the corresponding POS (i.e. the *k*-th POS in the sorted list of POSes);
2. List and sort all possible values of all lexical attributes (disregarding the wildcard '-') for all POSes (94 values) and form another binary vector with 94 positions such that the *k*-th position of this vector is 1 if the respective MSD has an attribute with the corresponding value;
3. Concatenate the vectors from steps 1 and 2 and obtain the binary codification of a MSD as a 110-position binary vector.

2.2 The training and tagging procedure

The tagger automatically assigns four dummy tokens (two at the beginning and two at the end) to the target utterance and the neural network is trained to automatically assign a MSD given the context (two previously assigned tags and the possible tags for the current and following two words) of the current word (see below for details).

In our framework a training example consists of the features extracted for a single word inside an utterance as input and it's MSD within that utterance as output. The features are extracted from a window of 5 words centered on the current word. A single word is characterized by a vector that encodes either its assigned MSD or its possible MSDs. To encode the possible MSDs we use equation 2, where each possible attribute a, has a single corresponding position inside the encoded vector.

$$P(a|w) = \frac{C(w,a)}{C(w)} \qquad (2)$$

Note that we changed the probability estimates to account for attributes not tags.

To be precise, for every word w_k, we obtain its input features by concatenating a number of 5 vectors. The first two vectors encode the MSDs assigned to the previous two words (w_{k-1} and w_{k-}

$_2$).The next three vectors are used to encode the possible MSDs for the current word (w_k) and the following two words (w_{k+1} and w_{k+2}).

During training, we also compute a list of suffixes with associated MSDs, which is used at run-time to build the possible MSDs vector for unknown words. When such words are found within the test data, we approximate their possible MSDs vector using a variation of the method proposed by Brants (2000).

When the tagger is applied to a new utterance, the system iteratively calculates the output MSD for each individual word. Once a label has been assigned to a word, the word's associated vector is edited so it will have the value of 1 for each attribute present in its newly assigned MSD.

As a consequence of encoding each individual attribute separately for MSDs, the tagger can assign new tags (that were never associated with the current word in the training corpus). Although this is a nice behavior for dealing with unknown words it is often the case that it assigns attribute values that are not valid for the wordform. To overcome these types of errors we use an additional list of words with their allowed MSDs. For an OOV word, the list is computed as a union from all MSDs that appeared with the suffixes that apply to that word.

When the tagger has to assign a MSD to a given word, it selects one from the possible wordform's MSDs in its wordform/MSDs associated list using a simple distance function:

$$\min_{e \in P} \sum_{k=0}^{n} |o_k - e_k| \qquad (3)$$

P	-	The list of all possible MSDs for the given word
n	-	The length of the MSD encoding (110 bits)
o	-	The output of the Neural Network for the current word
e	-	Binary encoding for a MSD in P

3 Network hyperparameters

In our experiments, we used a fully connected, feed forward neural network with 3 layers (1 input layer, 1 hidden layer and 1 output layer) and a sigmoid activation function (equation 3). While other network architectures such as recurrent neural networks may prove to be more suitable for this task, they are extremely hard to train, thus, we traded the advantages of such architectures for the robustness and simplicity of the feed-forward networks.

$$f(t) = \frac{1}{1+e^{-t}} \qquad (3)$$

$f(t)$	-	Neuron output
t	-	The weighted sum of all the neuron outputs from the previous layer

Based on the size of the vectors used for MSD encoding, the output layer has 110 neurons and the input layer is composed of 550 (5 x 110) neurons.

In order to fully characterize our system, we took into account the following parameters: accuracy, runtime speed, training speed, hidden layer configuration and the number of optimal training iterations. These parameters have complex dependencies and relations among each other. For example, the accuracy, the optimal number of training iterations, the training and the runtime speed are all highly dependent on the hidden layer configuration. Small hidden layer give high training and runtime speeds, but often under-fit the data. If the hidden layer is too large, it can easily over-fit the data and also has a negative impact on the training and runtime speed. The number of optimal training iterations changes with the size of the hidden layer (larger layers usually require more training iterations).

To obtain the trade-offs between the above mentioned parameters we devised a series of experiments, in all of which we used the "1984" MSD annotated corpus, which is composed of 118,025 words. We randomly kept out approximately 1/10 (11,960 words) of the training corpus for building a cross-validation set. The baseline accuracy on the cross-validation set (i.e. returning the most probable tag) is 93.29%. We also used an additional inflectional wordform/MSD lexicon composed of approximately 1 million hand-validated entries.

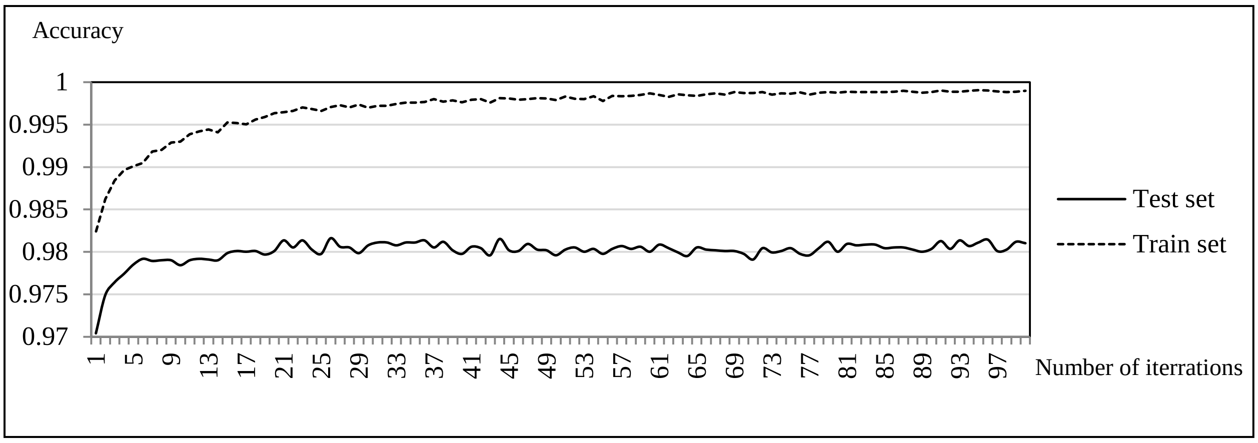

Figure 2 - 130 hidden layer network test and train set tagging accuracy as a function of the number of iterations

The first experiment was designed to determine the trade-off between the run-time speed and the size of the hidden layer. We made a series of experiments disregarding the tagging accuracy.

Hidden size	Time (ms)	Words/sec
50	1530	7816
70	1888	6334
90	2345	5100
110	2781	4300
130	3518	3399
150	5052	2367
170	5466	2188
190	6734	1776
210	7096	1685
230	8332	1435
250	9576	1248
270	10350	1155
290	11080	1079
310	12364	967

Table 1 - Execution time vs. number of neurons on the hidden layer

Because, for a given number of neurons in the hidden layer, the tagging speed is independent on the tagging accuracy, we partially trained (using one iteration and only 1000 training sentences) several network configurations. The first network only had 50 neurons in the hidden layer and for the next networks, we incremented the hidden layer size by 20 neurons until we reached 310 neurons. The total number of tested networks is 14. After this, we measured the time it took to tag the 1984 test corpus (11,960 words) for each individual network, as an average of 3 tagging runs in order to reduce the impact of the operating system load on the tagger (Table 1 shows the figures).

Determining the optimal size of the hidden layer is a very delicate subject and there are no perfect solutions, most of them being based on trial and error: small-sized hidden layers lead to under-fitting, while large hidden layers usually cause over-fitting. Also, because of the trade-off between runtime speed and the size of hidden layers, and if runtime speed is an important factor in a particular NLP application, then hidden layers with smaller number of neurons are preferable, as they surely do not over-fit the data and offer a noticeable speed boost.

hidden layer	Train set accuracy	Cross validation accuracy
50	99.18	97.95
70	99.20	98.02
90	99.27	98.03
110	99.29	98.05
130	99.35	98.12
150	99.35	98.09
170	99.41	98.07
190	99.40	98.10
210	99.40	98.21

Table 2 - Train and test accuracy rates for different hidden layer configurations

As shown in Table 1, the runtime speed of our system shows a constant decay when we increase the hidden layer size. The same decay can be seen in the training speed, only this time by an order of magnitude larger. Because training a single network takes a lot of time, this experiment was designed to estimate the size of the hidden layer which offers good performance in tagging. To do this, we individually trained a number of networks in 30 iterations, using various hidden layer configurations (50, 70, 90, 110, 130, 150, 170, 190, and 210 neurons) and 5 initial random initializations of the weights. For each configuration, we stored the accuracy of reproducing the learning data (the tagging of the training corpus) and the accuracy on the unseen data (test sets). The results are shown in Table 2. Although a hidden layer of 210 neurons did not seem to over-fit the data, we stopped the experiment, as the training time got significantly longer.

The next experiment was designed to see how the number of training iterations influences the tagging performance of networks with different hidden layer configurations. Intuitively, the training process must be stopped when the network begins to over-fit the data (i.e. the train set accuracy increases, but the test set accuracy drops). Our experiments indicate that this is not always the case, as in some situations the continuation of the training process leads to better results on the cross-validation data (as shown in Figure 2). So, the problem comes to determining which is the most stable configuration of the neural network (i.e. which hidden unit size will be most likely to return good results on the test set) and establish the number of iterations it takes for the system to be trained. To do this, we ran the training procedure for 100 iterations and for each training iteration, we computed the accuracy rate of every individual network on the cross-validation set (see Table 3 for the averaged values). As shown, the network configuration using 130 neurons on the hidden layer is most likely to produce better results on the cross-validation set regardless of the number of iterations.

Although, some other configurations provided better figures for the maximum accuracy, their average accuracy is lower than that of the 130 hidden unit network. Other good candidates are the 90 and 110 hidden unit networks, but not the larger valued ones, which display a lower average accuracy and also significantly slower tagging speeds.

The most suitable network configuration for a given task depends on the language, MSD encoding size, speed and accuracy requirements. In our own daily applications we use the 130 hidden unit network. After observing the behavior of the various networks on the cross-validation set we determined that a good choice is to stop the training procedure after 40 iterations.

Hidden units	**Avg. acc.**	**Max. acc.**	**St. dev.**
50	97.94	98.31	0.127002
70	98.03	98.31	0.12197
50	97.94	98.37	0.139762
70	98.03	98.43	0.124996
90	98.07	98.39	0.134487
110	98.08	98.45	0.127109
130	98.14	98.44	0.136072
150	98.01	98.36	0.143324
170	97.94	98.36	0.122834

Table 3 - Average and maximum accuracy for various hidden layer configuration calculated over 100 training iterations on the test set

To obtain the accuracy of the system, in our last experiment we used the 130 hidden unit network and we performed the training/testing procedure on the 1984 corpus, using 10-fold validation and 30 random initializations. The final accuracy was computed as an average between all the accuracy figures measured at the end of the training process (after 40 iterations). The first 1/10 of the 1984 corpus on which we tuned the hyperparameters was not included in the test data, but was used for training. The mean accuracy of the system (98.41%) was measured as an average of 270 values.

4 Comparison to other methods

In his work, Ceaușu (2006) presents a different approach to MSD tagging using the Maximum Entropy framework. He presents his results on the same corpus we used for training and testing (the 1984 corpus) and he compares his method (98.45% accuracy) with the Tiered Tagging methodology (97.50%) (Tufiș and Dragomirescu, 2004).

Our Neural Network approach obtained similar (slightly lower) results (98.41%), although it is arguable that our split/train procedure is not identical to the one used in his work (no details were given as how the 1/10 of the training corpus was selected). Also, our POS tagger detected cases where the annotation in the Gold Standard was erroneous. One such example is in “lame de ras” (English “razor blades”) where “lame” (English “blades”) is a noun, “de” (“for”) is a preposition and “ras” (“shaving”) is a supine verb (with a past participle form) which was incorrectly annotated as a noun.

5 Network pattern analysis

Using feed-forward neural networks gives the ability to outline what input features contribute to the selection of various MSD attribute values in the output layer which might help in reducing the tagset and thus, redesigning the network topology with beneficial effects both on the speed and accuracy.

To determine what input features contribute to the selection of certain MSD attribute values, one can analyze the weights inside the neural network and extract the input → output links that are formed during training. We used the network with 130 units on the hidden layer, which was previously trained for 100 iterations. Based on the input encoding, we divided the features into 5 groups (one group for each MSD inside the local context – two previous MSDs, current and following two possible MSDs). For a target attribute value (noun, gender feminine, gender masculine, etc.) and for each input group, we selected the top 3 input values which support the decision of assigning the target value to the attribute (features that increase the output value) and the top 3 features which discourage this decision (features that decrease the output value). For clarity, we will use the following notations for the groups:

- G_{-2}: group one – the assigned MSD for the word at position *i-2*
- G_{-1}: group two – the assigned MSD for the word at position *i-1*
- G_0: group three – the possible MSDs for the word at position *i*
- G_1: group four– the possible MSDs for the word at position *i+1*
- G_2: group five – the possible MSDs for the word at position *i+2*

where *i* corresponds to the position of the word which is currently being tagged. Also, we classify the attribute values into two categories (*C*): *(P) want to see* (support the decision) and *(N) don't want to see* (discourage the decision).

Table 4 shows partial (G_{-1} G_0 G_1) examples of two target attribute values (cat=Noun and gender =Feminine) and their corresponding input features used for discrimination.

Target value	Group	C	Attribute values
Noun	G_{-1}	P	main (of a verb), **article**, masculine (gender of a noun/adjective
		N	particle, conjunctive particle, auxiliary (of a verb), demonstrative (of a pronoun)
	G_0	P	**noun, common/proper (of a noun)**
		N	adverb, pronoun, numeral, interrogative/relative (of a pronoun)
	G_1	P	**genitive/dative (of a noun/adjective)**, particle, punctuation
		N	conjunctive particle, strong (of a pronoun), non-definite (of a noun/adjective), exclamation mark
Fem.	G_{-1}	P	main (of a verb), preposition, **feminine (of a noun/adjective)**
		N	auxiliary (of a verb), particle, demonstrative (of a pronoun)
	G_0	P	**feminine (of a noun/adjective)**, nominative/accusative (of a noun/adjective), past (of a verb)
		N	masculine (of a noun/adjective), auxiliary (of a verb), interrogative/relative (of a pronoun), adverb
	G_1	P	dative/genitive (of a noun/adjective), indicative (of a verb), **feminine (of a noun/adjective)**
		N	conjunctive particle, future particle, nominative/accusative (of a noun/adjective)

Table 4 – P/N features for various attribute values.

For instance, when deciding on whether to give a noun (N) label to current position (G_0), we can see that the neural network has learned some interesting dependencies: at position G_{-1} we find an article (which frequently determines a noun) and at the current position it is very important for the word being tagged to actually be a common or proper noun (either by lexicon lookup or by suffix guessing) and not be an adverb, pronoun or numeral (POSes that cannot be found in the typical ambiguity class of a noun). At the next position of the target (G_1) we also find a noun in genitive or dative, corresponding to a frequent construction in Romanian, e.g. "mașina băiatului" being a sequence of two nouns, the second at genitive/dative.

If the neural network outputs the feminine gender to its current MSD, one may see that it has actually learned the agreement rules (at least locally): the feminine gender is present both before (G_{-1}) the target word as well as after it (G_1).

6 Conclusions and future work

We presented a new approach for large tagset part-of-speech tagging using neural networks. An advantage of using this methodology is that it does not require extensive knowledge about the grammar of the target language. When building a new MSD tagger for a new language one is only required to provide the training data and create an appropriate MSD encoding system and as shown, the MSD encoding algorithm is fairly simple and our proposed version works for any other MSD compatible encoding, regardless of the language.

Observing which features do not participate in any decision helps design custom topologies for the Neural Network, and provides enhancements in both speed and accuracy. The configurable nature of our system allows users to provide their own MSD encodings, which permits them to mask certain features that are not useful for a given NLP application.

If one wants to process a large amount of text and is interested only in assigning grammatical categories to words, he can use a MSD encoding in which he strips off all unnecessary features. Thus, the number of necessary neurons would decrease, which assures faster training and tagging. This is of course possible in any other tagging approaches, but our framework supports this by masking attributes inside the MSD encoding configuration file, without having to change anything else in the training corpus. During testing the system only verifies if the MSD encodings are identical and the displayed accuracy directly reflects the performance of the system on the simplified tagging schema.

We also proposed a methodology for selecting a network configurations (i.e. number of hidden units), which best suites the application requirements. In our daily applications we use a network with 130 hidden units, as it provides an optimal speed/accuracy trade-off (approx. 3400 words per second with very good average accuracy).

The tagger is implemented as part of a larger application that is primarily intended for text-to-speech (TTS) synthesis. The system is free for non-commercial use and we provide both web and desktop user-interfaces. It is part of the METASHARE platform and available online[2]. Our primary goal was to keep the system language independent, thus all our design choices are based on the necessity to avoid using language specific knowledge, when possible. The application supports various NLP related tasks such as lexical stress prediction, syllabification, letter-to-sound conversion, lemmatization, diacritic restoration, prosody prediction from text and the speech synthesizer uses unit-selection.

From the tagging perspective, our future plans include testing the system on other highly inflectional languages such as Czech and Slovene and investigating different methods for automatically determining a more suitable custom network topology, such as genetic algorithms.

Acknowledgments

The work reported here was funded by the project METANET4U by the European Commission under the Grant Agreement No 270893

[2] http://ws.racai.ro:9191

References

Berger, A. L., Pietra, V. J. D. and Pietra, S. A. D. 1996. *A maximum entropy approach to natural language processing*. Computational linguistics, 22(1), 39-71.

Brants, T. 2000. *TnT: a statistical part-of-speech tagger*. In Proceedings of the sixth conference on applied natural language processing (pp. 224-231). Association for Computational Linguistics.

Calzolari, N. and Monachini M. (eds.). 1995. *Common Specifications and Notation for Lexicon Encoding and Preliminary Proposal for the Tagsets*. MULTEXT Report, March.

Ceaușu, A. 2006. *Maximum entropy tiered tagging*. In Proceedings of the 11th ESSLLI Student Session (pp. 173-179).

Collins, M., Ramshaw, L., Hajič, J. and Tillmann, C. 1999. *A statistical parser for Czech*. In Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics (pp. 505-512). Association for Computational Linguistics.

Erjavec, T. and Monachini, M. (Eds.). 1997. *Specifications and Notation for Lexicon Encoding*. Deliverable D1.1 F. Multext-East Project COP-106.

Erjavec, T. 2004. *MULTEXT-East version 3: Multilingual morphosyntactic specifications, lexicons and corpora*. In Fourth International Conference on Language Resources and Evaluation, LREC (Vol. 4, pp. 1535-1538).

Erjavec, T. and Krek, S. 2008. *The JOS morphosyntactically tagged corpus of Slovene*. In Proceedings of the Sixth International Conference on Language Resources and Evaluation, LREC'08.

Erjavec, T. 2010. *MULTEXT-East Version 4: Multilingual Morphosyntactic Specifications, Lexicons and Corpora*. In Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10), Valletta, Malta. European Language Resources Association (ELRA) ISBN 2-9517408-6-7.

Lafferty, J., McCallum, A. and Pereira, F. C. 2001. *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*.

Marques, N. C. and Lopes, G. P. 1996. *A neural network approach to part-of-speech tagging*. In Proceedings of the 2nd Meeting for Computational Processing of Spoken and Written Portuguese (pp. 21-22).

Ratnaparkhi, A. 1996. *A maximum entropy model for part-of-speech tagging.* In Proceedings of the conference on empirical methods in natural language processing (Vol. 1, pp. 133-142).

Samuelsson, C. 1993. *Morphological tagging based entirely on Bayesian inference*. In 9th Nordic Conference on Computational Linguistics.

Schmid, H. 1994. *Part-of-speech tagging with neural networks*. In Proceedings of the 15th conference on Computational linguistics-Volume 1 (pp. 172-176). Association for Computational Linguistics.

Tufiș, D., Barbu A.M., Pătrașcu V., Rotariu G. and Popescu C. 1997. *Corpora and Corpus-Based Morpho-Lexical Processing*. In Recent Advances in Romanian Language Technology, (pp. 35-56). Romanian Academy Publishing House, ISBN 973-27-0626-0.

Tufiș, D. 1999. *Tiered tagging and combined language models classifiers*. In Text, Speech and Dialogue (pp. 843-843). Springer Berlin/Heidelberg.

Tufiș, D., and Dragomirescu, L. 2004. *Tiered tagging revisited*. In Proceedings of the 4th LREC Conference (pp. 39-42).