

Neuro-Symbolic Word Tagging

Nuno C. Marques^{1a}, Sebastian Bader^{2b}, Vitor Rocio^{3c}, and Steffen Hölldobler^{2d}

¹ Centria, DI-FCT-UNL Lisbon, Portugal

^a `nmm@di.fct.unl.pt`

² ICCL, TU Dresden, Germany

^b `Sebastian.Bader@inf.tu-dresden.de`, ^d `sh@iccl.tu-dresden.de`

³ CITI, DI-FCT-UNL Lisbon and Universidade Aberta, Portugal

^c `vjr@univ-ab.pt`

Abstract. Part-of-speech tagging (POS) assigns grammatical tags (like noun, verb, etc.) to a word depending on its definition and its context. This is a first step before parsing may be applied. POS tagging and more generically word tagging, plays an important role in computational linguistics and in many information retrieval and text mining tasks. Neither pure rule based, nor machine learning approaches give satisfying results: rule based systems can not adapt well to existing samples; machine learning systems ignore available background knowledge. A combination of both is needed. In particular, we show some cases where the initialization of a neural network based tagger with background knowledge obtains better results. In this preliminary work we study some implications of embedding background knowledge for POS and word tagging problems. Preliminary results show that the combined system outperforms a purely machine learning system when only limited samples are available.

1 Introduction

In text mining and information extraction tasks it is useful to annotate words with their grammatical tags, so that text mining systems can then identify linguistic, semantic or even purely task oriented information. E.g., in the GATE-system [CMBT02], this kind of processing allows the detection of several kinds of named entities and some relations among them. If we want to mine more specific or domain dependent knowledge, we can also adapt the tagset to better suit our specific needs [MG04]. In the latter case, most annotations can even be used directly for text mining tasks.

The main problem with word tagging is that many words are ambiguous. For example, the word "past" can be an adjective, an adverb or a noun. *Part-of-speech (POS)* disambiguation, also called syntactic wordclass tagging, is the process that assigns these grammatical tags to a word depending on its context [vH99]. In the past, rule based taggers were developed as well as systems based on machine learning technologies. But as described in Section 2, both approaches have contrasting advantages and disadvantages. In this work, we try to combine the advantages of both ideas into an integrated system. This includes the usage of available background knowledge in form of symbolic rules and the usage of

the powerful learning capabilities provided by artificial neural networks trained by backpropagation.

Our contribution in this paper is twofold. On the one hand, we will show that methods from neural symbolic integration [BH05] can be applied in this real world domain and lead to improvements. On the other hand, this paper opens a new research line in the area of POS Tagging. We show how to combine two established approaches such that the advantages of both are retained.

In the next section, we will introduce some preliminary notions and provide background knowledge required for the subsequent sections. We will discuss basics of neural networks and part-of-speech tagging first. Afterwards, we will show how to use neural networks for POS tagging and present some techniques from neural-symbolic (NeSy) integration, i.e., techniques that allow the embedding of background knowledge into neural networks. We will describe how that information can be used and refined for part-of-speech tagging. Then a preliminary experimental framework used to test these systems is also described. Namely, we will describe how to adapt the Susanne corpus [Sam95] for our experiments, how to use rules and how to build the data-sets from the corpus. The acquired results will then be analyzed and several conclusions will be drawn regarding the applicability of the NeSy approach for word tagging.

2 Preliminaries & Related work

Artificial neural networks consist of simple computational units (neurons), which receive real numbers as inputs via weighted connections and perform simple operations: the weighted inputs are usually added and a simple function (like threshold, sigmoidal) is applied to the sum. We will consider networks, where the units are organized in layers. Neurons which do not receive input from other neurons are called *input neurons*, and those without outgoing connections to other neurons are called *output neurons*. Such, so-called *feed-forward networks* compute functions from \mathbb{R}^n to \mathbb{R}^m , where n and m are the number of input and output units, respectively. Neural networks are well known for their capabilities to handle high dimensional, noisy and sometimes incomplete data samples, as in most natural language problems. For a general introduction on machine learning including neural networks, we refer to [Mit97].

Neural-symbolic integration aims at the combination of the advantages of neural networks with those of symbolic systems. For a general introduction including many references, we refer to [BH05]. This includes the embedding of propositional logic rules into connectionist systems and the extraction of rules. As shown in [dGBG02], the embedding of background knowledge into neural networks leads to faster convergence in many problem domains. We will describe some of the basic techniques below. In particular, we will show how to embed rules into a neural network following [HK94, dGBG02].

Part-of-speech (POS) tagging is usually part of any task involving natural language processing (NLP). In traditional NLP, POS tagging can be seen as a level between lexical look-up and sentence parsing. The idea is to determine the word category based on its context. Several systems have been developed for this particular task. A good review can be found in [vH99]. Although it is difficult to make comparisons (since no single dataset, or even tagset is used), a precision that is usually in the 96% level is reported.

First, *rule-based POS taggers* were developed [vH99]. But these systems are usually hard to build and to maintain by hand. Also most of the rules, instead of being associated with syntax, are purely heuristic [vH99]. It is quite easy (and most often needed) to devise a small set of rules that make sense, even in a broader syntactic context. But tuning those systems to achieve good performance is a laborious and dedicated task.

An alternative are POS taggers based on *machine learning* or on *statistical methods*, among many others [ML01, Bri95, DZB96, Rat96]. These perform supervised classification based on annotated text corpora. E.g., in [ML01, Sch94] it was shown that a feed forward neural network, similar to the ones used here, can achieve high precision. But to achieve good results, a reasonable amount of correctly tagged text should be available. Furthermore, those taggers are highly specific for the domain they were trained for. This means that for every new domain laborious annotations of text are required.

Usually domain experts can easily specify some rules for any annotation task. But as mentioned above, pure rule based systems need a lot of effort to achieve good accuracy. On the other hand, machine learning systems might improve with the help of rules. For example, Eric Brill's TBL-system [Bri95] is based on both, rules and machine learning techniques. So called transformation based rules are learned from a tagged corpus. Unfortunately the rule sets generated by TBL method are very extensive and highly difficult to change. Also most learned rules are either incompressible or too specific to the type of text being handled.

In this paper, we propose a new *integrated system*, that conjoins the advantages of background knowledge expressed as rules and the learning power of neural networks. This is a first case study on improving a neural network machine learning tagger with background knowledge. Neural network taggers have already been shown to be very efficient in learning from scarce data [ML01]. In our approach rules will be used to initialize a neural network model. These rules can encode important information that is not present in training data with limited number of examples. Although explicit encoding of background knowledge is possible in other machine learning based systems, the core NeSy method [BH05] used here provides a very elegant and efficient theory for this integration. Since rules are directly encoded in the learning model, no direct overhead exists in the learning system. Also the extension for richer rulesets (namely including first-order logic) is possible and will be subject to further research.

3 The Neural-Symbolic Part-of-Speech Tagger

The basic model for our neural-symbolic tagger follows [ML01]:

1. Consider a context window of n words.
2. Represent each word by its probability vector $p_i \in \mathbb{R}^t$.
3. Compute the resulting probability vector $p \in \mathbb{R}^t$.
4. Use the tag with maximum probability as result.

To tag a word w , we consider its n -word context, represented as a probability distribution. Depending on those probabilities, we compute a tag-relevance vector for w and pick the maximal element as result. The function ($f : \mathbb{R}^{n \cdot t} \rightarrow \mathbb{R}^t$, with t being the number of possible tags and n being the window size), is learned and computed by a neural network as depicted in Figure 1.

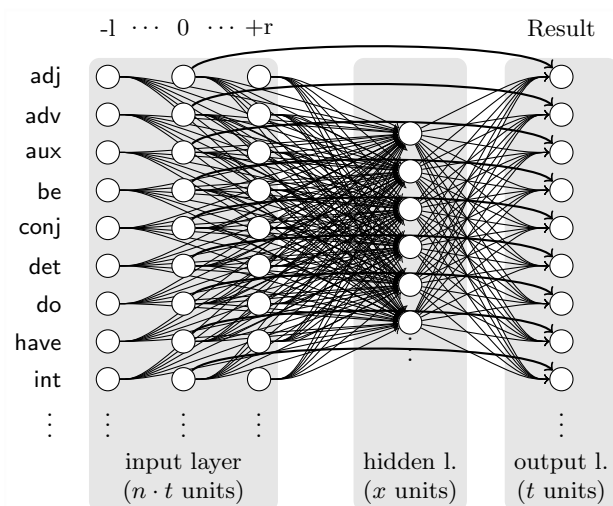


Fig. 1. General Network architecture. The input layer contains $n \cdot t$ units (one for each offset and tag) and the output layer t units (one for each tag). The size of the hidden layer will vary, depending on the experiment.

The Relation between Words and Probability Vectors Before classifying a word by the network, the word is represented by its probability vector. As in previous work [ML01], a dictionary is extracted from a sub sample of an annotated corpus (that will be called training corpus, since it is used to train our model⁴) by counting the number of occurrences of each word and each tag. This results in

⁴ Further details how the annotated corpus should be used are given in section 4.

a probability distribution of all words over the tag set. Following [ML01], we assign a probability vector $(p_w(T_1), \dots, p_w(T_t))$ to a word w as follows:

$$p_w(T_i) := \frac{\text{freq}(w, T_i)}{\text{freq}(w)}.$$

We use $\text{freq}(w, T_i)$ to denote the number of times w is tagged as T_i ($1 \leq i \leq t$) and $\text{freq}(w)$ to denote the total number of occurrences of w in the corpus. Unknown words are treated as in [BS96]: A global probability vector is computed containing the relative frequencies of all tags for unknown words, using those words that occur only once in the train corpus.

Since the dictionary is computed from the selected training corpus, it can be considered as part of the trained model. This way no interferences occur with subsamples of the corpus used to test this learned model (please check sec. 4) and a more realistic scenario is applied, since unknown words may occur when testing the model.

The Generation of Training Data Using one part of the corpus and the dictionary we generated training-data for our network as follows: All words in the training corpus (and their associated context words) are mapped to their probability vectors by the dictionary. The resulting composed probability vector together with a unit vector (i.e., a vector of 0's and a single 1 for the correct tag) are collected as a training-set. The validation data is built analogously from a different subset of the original corpus (except that the dictionary is the one extracted from the training data).

For a simple example, let us consider the word "amount" and its 5-word context in the following sentence: "the medical *plan would amount to* 1.5 billion dollars". From our dictionary, we obtain the following probabilities: "plan" is a noun with probability (0.94) or a verb (0.06), "would" is usually an auxiliary verb (0.99), "amount" can be a noun (0.74) or a verb (0.25), "to" is usually a preposition (0.99) and "1.5" is mostly tagged as noun (0.95). The resulting input vector contains those probabilities on the according positions. From the corpus, we know that "amount" should be tagged as verb in this case (even though it is more likely a noun). Therefore, we set the desired output vector to be 0 everywhere, except for the position of the tag v .

Representation of Background Knowledge Our background knowledge will be expressed in the form of implications as follows:

$$T \leftarrow T_{-l}, \dots, T_0, \dots, T_r$$

with $l, r \geq 0$ and $l + r + 1 = n$. T_{-l}, \dots, T_r are tags required on the given position and T is the desired output tag. For example, the rule $v \leftarrow \text{aux}_{-1}, n_0$ states that a noun preceded by an auxiliary verb is actually a verb. The most frequent if-then-rules, found in our corpus are shown in Table 1 together with small examples.

Rule	Example
1) $v \leftarrow \text{aux}_{-1}, n_0$	would <i>force</i> , would <i>amount</i>
2) $wh \leftarrow \text{det}_0, wh_1$	a few, a lot, <i>the</i> rest
3) $be \leftarrow \text{pront}_{-1}, n_0$	it 's, there 's
4) $wh \leftarrow \text{prep}_{-1}, \text{conj}_0$	of <i>that</i> , during <i>that</i> , like <i>that</i>
5) $v \leftarrow \text{pron}_{-1}, n_0$	they <i>work</i> , we <i>need</i> , i <i>hope</i>
6) $\text{pront} \leftarrow \text{det}_0, \text{aux}_1$	<i>one</i> might, <i>one</i> can

Table 1. The 6 most frequent rules from the training corpus.

These rules are very similar to the *transformation rules* used in Eric Brill Tagger [Bri95], and a simplified version of the hand-crafted rules proposed by Voutilainen in [vH99]. Rule 1 and 5 resolve noun-verb ambiguities: the previous 1-tag context (auxiliary verb in rule 1 and pronoun in rule 5) determine that the most likely tag (noun) should be changed into a verb. In rule 3, the existence of a third-person singular pronoun (pront) determines that "'s" is in fact the verb "to be", not part of a noun (possessive case). In rule 6, a determiner that is followed by an auxiliary verb is in fact a pronoun. As we will see below, these rules can easily be embedded into a neural network following in principle [HK94].

Embedding Rules into the Network Each of the $t \times n$ input units will be associated with a tag and a position, namely, the input unit $u_{i,j}$ will represent the probability that the word at position j is of type i . Analogously, we associate each output unit with one of the tags. To embed a given rule into a network we follow in principle [HK94, dGBG02] with some modifications.

Usually, the most probable entry in the dictionary and the correct tag agree. Therefore, we added shortcut connections from each offset-0-unit in the input layer with the corresponding unit in the output layer. For example, the input unit n_0 is connected with the output unit n . Therefore, the network has to learn the exceptions only. Those exceptions are given by the rules. The rule $v \leftarrow \text{aux}_{-1}, n_0$ should force the network to switch the output from n to v , if the n is preceded by an aux . We add a hidden layer unit for each rule such that this unit will be active if and only if the preconditions of the rule are met. For example, for this rule we generate a hidden layer unit u with threshold 1.5ω (with ω being a user defined constant) and add connections from aux_{-1} and n_0 , both weighted ω . This ensures that u receives an input of 0.5ω if and only if both preconditions of the rule are met. By setting $\omega = 5$ we can ensure that u 's output (after applying the sigmoidal activation function) is still close to 1. Furthermore, we connect u with the output unit for v with weight ω and to the output unit n with weight $-\omega$.

A given output unit o shall receive a high input if at least one hidden unit created for a rule with head o is active, or if the corresponding 0-index unit is active. This can be achieved by setting the threshold of all output units to 0.5ω . The negative link to the "wrong" output unit ensures that these default rules are overwritten. Furthermore, we connect u to all other input and output units and set those "free" weights to small random numbers. Part of the resulting network

is shown in Figure 2. In the following sections, we will show the (positive) effect of embedding rules this way.

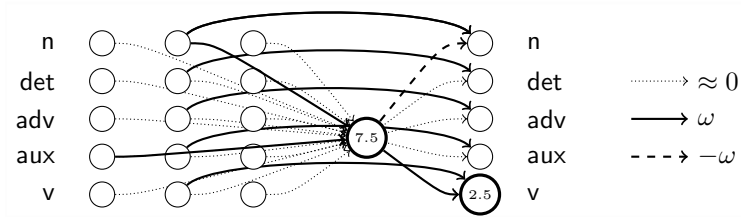


Fig. 2. Embedding the rule $v \leftarrow \text{aux}_{-1}, n_0$. The numbers denote the biases. The weights of the connection are given at the right.

Additional hidden units (with random connection weights) are added to the network. These "free" units and the "free" connections enable the network to learn new rules and to revise the embedded ones. Please note that randomizing all weights yields a normal feed-forward neural network.

4 Experimental Evaluation

To evaluate our approach, we ran several experiments using the Susanne corpus and different networks. After describing the setup of our experiments, we will describe some experiments in detail. All networks were trained using the batchman utility of the SNNS software [Zel92].

4.1 Description of the framework

The original tagset of the Susanne corpus is too big for our purposes: the morphological information is very detailed, and as a consequence many tags occur only once or twice in the corpus, potentially causing problems due to data sparseness. In a preliminary step, we mapped the Susanne tags into a reduced tagset containing only 18 tags. Table 2 shows our tags and their meaning. From a linguistic point of view, these tags have enough information to allow us to build a basic grammar without the need for further lexical knowledge.

The Susanne corpus contains 156.610 tagged words. As shown in Table 3, we split this set into different sub-corpora. $\approx 1/5$ is used as testset to compare the different tagging performances, $\approx 1/5$ is used for validation during the training to avoid overfitting phenomena and the remaining samples were used for training and dictionary building. We will use three different subsets of the training set in our experiments, which are called *full*, *small* and *tiny* training set.

Tag	Meaning	Frequency	Tag	Meaning	Frequency
adj	adjective	6022	adv	adverb	4460
aux	auxiliary verb	1055	be	verb to be	2852
conj	conjunction	4124	det	determiner	10829
do	verb to do	294	have	verb to have	856
int	interrogative pronoun	131	n	noun	22493
pto	punctuation	14605	pn	proper name	457
prep	preposition	11050	pron	pronoun	1332
pront	pronoun (3rd person singular)	2544	vt	verb (3rd person singular)	548
v	verb	8476	wh	wh-word	1895

Table 2. Our tagset for the Susanne corpus. The table also shows the total number of occurrences for each tag.

Size	Usage
31.282	Testset to compare the tagging performance (not used for training)
31.305	Validation during the training to avoid overfitting (not used for training)
94.023	Training set <i>full</i>
9.424	Training set <i>small</i>
2.817	Training set <i>tiny</i>

Table 3. Sub-corpora used in our experiments.

4.2 Comparing the Training Error

As a first experiment, we computed the mean square error (mse) of different networks trained for the three training sets described above. We will use MSE for measuring training accuracy since it measures the actual encoding of training patterns into the model that its being learned (i.e., mean squared difference between desired and actual outputs). This will be done on train and validation data. After training process is finished tagging precision over the test set will be used to actually evaluate trained models.

First, we embedded the most frequent rule into a network and added 10 additional hidden units (referred to as NeSy₁₊₁₀). Another network was initialized with 11 rules and without additional units (NeSy₁₁₊₀) and another one with 11 hidden units and completely randomized weights (Rand₁₁). All three networks were trained on the full, small and tiny dataset and the error on the validation set was computed. The results are shown in Figure 3. On the full data set, the randomized network was the best, but on the small and tiny sets, the NeSy networks outperform the randomized one. This is probably due to the fact that the rules were retained and are actually needed to achieve good results on the validation set, but could not be learned from the examples provided in the smaller datasets.

Figure 4 shows a typical run for the small and tiny training corpus. Even though the initialized network achieves from the very beginning good results

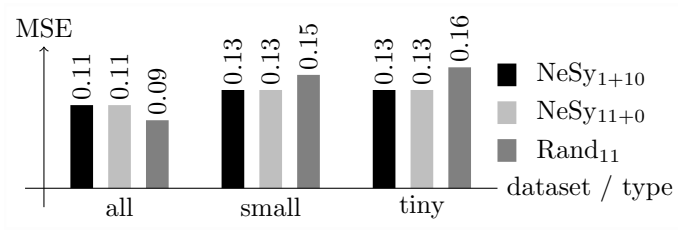


Fig. 3. Mean square error on the validation set for different networks and training sets.

on the training corpus, the randomized network gets better after only a few iterations. But the initialized network remains better on the validation corpus (which is actually the important one). This means that the initialized network generalizes better to unseen samples.

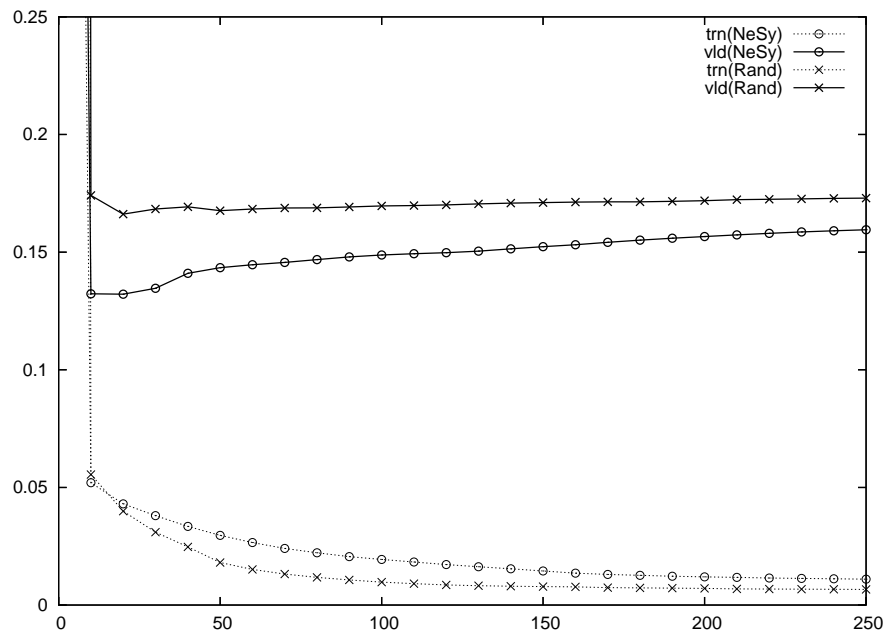


Fig. 4. The evolution of the error for the first 250 iterations of training.

4.3 Preservation of Rules

To evaluate whether the embedded rules are retained during the training process, we evaluated the tagging performance for examples covered by the first rule

from Table 1. We chose the networks trained on the tiny training corpus, with minimum error on the validation set. A network initialized with the 6 rules and 10 free hidden units (so that better learning could be achieved), was used. The NeSy network tags 70% of those cases correctly, whereas the randomized one, obtains only 20%. This result was observable for all rules and all training corpora. Which means that embedded rules are preserved during the training. But this also means, that the network is strongly biased. This can be seen from the results discussed above, as the randomized network achieved a slightly better overall performance.

4.4 Comparing the Tagging Performance

To evaluate the actual tagging performance, we tagged the the testset using the best NeSy and randomized network and using a baseline unigram tagger. The same network of previous subsection was used (either with initialized weights or with random ones). Also the unigram tagger is shown as a baseline. It assigns the most probable tag for each word according to the dictionary obtained from the training corpus. Unknown words are always labeled as noun, which is the most frequent tag. Comparing the precisions for the nouns, we find that the unigram tagger outperforms the other ones. This is probably due to the strategy just described. However, this has the drawback as the unigram tagger is very bad for verbs, which are frequently ambiguous with nouns. On most of the other tags, the NeSy tagger is the best. Only for auxiliary verbs, prepositions and adjectives, the randomized network is better. We think, this results from the fact that we did not embed any rule concerning those tags and that the network is biased to concentrate on the provided rules.

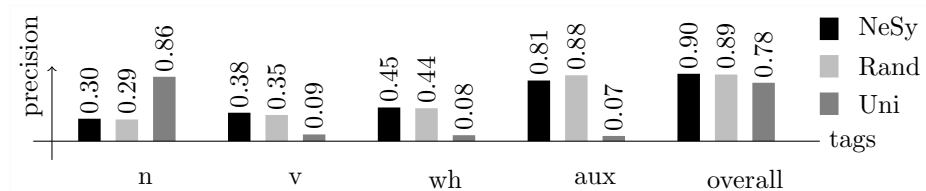


Fig. 5. The tagging precision for different tags.

5 Conclusions & Future Work

Syntactic wordclass tagging, usually assigns grammatical tags to a word, depending on its definition and its context [vH99]. There are several applications of word tagging in text mining, either using the tags directly [CMBT02] or by requiring the tags to become more semantically, or simply task oriented (as in e.g. in

[MG04]). Until now two alternative approaches have been used for POS-tagging: On the one hand, there are rule based systems, which require a lot of work in system development and adaptation. On the other hand there are supervised machine learning approaches, that require a lot of learning samples. Here, we propose a new approach that conjoins both, by retaining the advantages of rule based methods and of machine learning methods. This is done by using a neural network as in [ML01], but instead of randomizing the weights, we initialized the network using rules that express background knowledge [HK94, dGBG02].

We showed that the methods of neural symbolic integration can be successfully applied in the context of POS tagging. Furthermore, we showed that the embedding of generic background knowledge is particularly helpful if only limited training data is available. It suffices to embed generic rules into a network which is then trained using only few examples to achieve good results. To the best of our knowledge, this is the biggest problem ever approached with neural symbolic methods and the first application to natural language processing and text mining.

However, much remains to be done. First, we will compare our approach with other existing approaches on different data- and tagsets. Some of the disambiguities could not be solved due to our limited tagset. Therefore, more experiments with better tagsets⁵ are necessary. Furthermore, we will investigate whether the background rules can be automatically generated from existing grammars, or whether the grammar itself can be embedded into a network. Finally, we will try to apply this methods to specific tagsets appropriate for text mining problems. We expect to improve the current state of the art of the neural network tagger for Portuguese (based on [ML01]) and probably extend its application to English and other languages.

We found that neuro-symbolic networks allow an elegant encoding of rules inside a learning system. The evaluation shows the potential of our approach, because the NeSy networks are able to learn from raw data and also preserve embedded background knowledge. This way, the described approach seems to be an excellent way to encode and refine background knowledge into one of the best and most successful supervised learning methods.

References

- [BH05] Sebastian Bader and Pascal Hitzler. Dimensions of neural-symbolic integration — a structured survey. In S. Artemov, H. Barringer, A. S. d’Avila Garcez, L. C. Lamb, and J. Woods, editors, *We Will Show Them: Essays in Honour of Dov Gabbay*, volume 1, pages 167–194. King’s College Publications, JUL 2005.
- [Bri95] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.

⁵ Provided that enough and meaningful tagging examples are available for each tag.

- [BS96] Harald Baayen and Richard Sproat. Estimating lexical priors for low-frequency morphologically ambiguous forms. *Computational Linguistics*, 22(2):155–166, 1996.
- [CMBT02] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust nlp tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.
- [dGBG02] Artur S. d’Avila Garcez, Krysia B. Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems — Foundations and Applications*. Perspectives in Neural Computing. Springer, Berlin, 2002.
- [DZB96] W. Daelemans, J. Zavrel, and S. Berck. Mbt: A memorybased part of speech tagger-generator. 1996.
- [HK94] Steffen Hölldobler and Yvonne Kalinke. Towards a massively parallel computational model for logic programming. In *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI, 1994.
- [MG04] Nuno C. Marques and Sérgio Goncalves. Part-of-speech tagging for postal address text mining. 2004.
- [Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, March 1997.
- [ML01] Nuno C. Marques and Gabriel Pereira Lopes. Neural networks, part-of-speech tagging and lexicons. In *Proceedings of the International Conference on Intelligent Data Analysis (IDA ’01)*, number 2189 in LNCS, pages 63–72, Cascais, Portugal, September 2001. Springer Verlag.
- [Rat96] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In Eric Brill and Kenneth Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142. Association for Computational Linguistics, Somerset, New Jersey, 1996.
- [Sam95] Geoffrey Sampson. *English for the Computer: The SUSANNE Corpus and Analytic Scheme*. Oxford University Press, Oxford, 1995.
- [Sch94] H. Schmid. Part-of-speech tagging with neural networks. In *Proceeding of COLING-94*, pages 172–176, 1994.
- [vH99] Hans van Halteren. *Syntactic Wordclass Tagging*. Kluwer Academic Publishers, 1999.
- [Zel92] Andreas Zell. SNNS, stuttgart neural network simulator, user manual, version 2.1. Technical report, Stuttgart, 1992.