# Part-of-Speech Tagging with Recurrent Neural Networks

Juan Antonio Pérez-Ortiz and Mikel L. Forcada

*Departament de Llenguatges i Sistemes Informàtics*
*Universitat d'Alacant*
*E-03071 Alacant, Spain*
{japerez,mlf}@dlsi.ua.es

**Abstract**

*This paper explores the use of discrete-time recurrent neural networks for part-of-speech disambiguation of textual corpora. Our approach does not need a hand-tagged text for training the tagger, being probably the first neural approach doing so. Preliminary results show that the performance of this approach is, at least, similar to that of a standard hidden Markov model trained using the Baum-Welch algorithm.*

## 1 Introduction

This paper explores the use of discrete-time recurrent neural networks (DTRNN) for part-of-speech (PoS) tagging of ambiguous words from the sequential information stored in the network's state. PoS tagging [10] is a very important intermediate step in many natural language processing applications. A PoS tagger is a program that assigns each word in a text a PoS *tag* or *category* from a previously defined set, the *tagset* for that language.

PoS tags may be very coarse (such as "verb") or very fine (such as "transitive lexical verb, present tense, 3rd person singular"), depending on the particular task or application. A large fraction of the words in a text may be easily assigned a single PoS tag by looking them up in a lexicon of forms or using a *guesser* that infers the part of speech from the form of the word (e.g., English words ending in *-ously* are most likely adverbs), but many words are *ambiguous*: they may be assigned more than one PoS tag (for example, the English word *round* may be a noun, an adjective, a preposition or an adverb, or a verb).

The choice of the correct part of speech may be crucial, for example, when translating to another language. Most PoS taggers rely on the assumption that a word can be assigned a single PoS tag in a given context, or at least one of the possible parts of speech is most likely. Choices are usually made (likelihoods are usually estimated) depending on the part of speech of words surrounding it, that is, assuming that *syntax* is the best cue to disambiguation.

There are different approaches to automatic PoS tagging: *rule-based* approaches [1] use linguistic knowledge to formulate simple rules that assign a part of speech to an ambiguous word using context information; *statistical* approaches (of which hidden Markov models trained using the Baum-Welch expectation-maximization algorithm [2, 13] [10, ch. 10] are the standard model) use the statistics collected from ambiguously or unambiguously tagged texts (see below) to estimate the likelihood of each possible interpretation of a sentence or text portion so that the most likely disambiguation is chosen. Of course, *hybrid* approaches are possible which combine the power of rule-based and statistical PoS taggers.

We will refer to a text as *unambiguously tagged* or just *tagged* when each occurrence of each word (ambiguous or not) has been assigned the correct PoS tags. An *ambiguously tagged* text corpus is the one in which all words are assigned (using a lexicon or a morphological analyser and optionally a guesser) the set of possible PoS tags independently of context; in this case, ambiguous and unknown words would receive more than one PoS tag (unknown words are usually assigned the set of *open* categories, that is, categories to which it is very possible to add new words of the language: nouns, verbs, adjectives, adverbs and proper nouns). Words receiving the same set of PoS tags are said to belong to the same *ambiguity class* [2]; for example, the words *tailor* and *book* belong to the ambiguity class {noun, verb}.

A *neural* or *connectionist* approach is also possible; a brief survey of neural PoS tagging work follows:

- Schmid [14] trains a single-layer perceptron to produce the PoS tag of a word as a unary or *one-hot* vector. Input is a window of the $p = 2$ or $p = 3$ words before the current word, the current word, and the $f = 1$ or $f = 2$ words after it; on the one hand, the following words and the current word are represented by a vector in which the $j$-th component is the frequency with which that word gets the $j$-th PoS tag in a large unambiguously tagged corpus; the previous words, on the other hand, are represented by a linear combination of this vector and the output produced by the net for the corresponding word. He reports a 2% improvement over a regular hidden Markov model (HMM) [2] model.
- Marques and Pereira [12] follow a similar approach but use small training sets for Portuguese. In addition, they explore the use of Elman's net [3], but with worse results.
- Ma and Isahara [8] build upon Schmid's [14] model, using a combination of (multilayer) perceptrons with different windows, and use the tagger for the Thai language.
- Ma et al. [9] use a two-layer perceptron but network outputs are corrected using Brill's transformation rule [1] approach and the context window is dynamically sized. Results are marginally better than those in [8].

It has to be noted that all of the surveyed work relies on training the neural net with an unambiguously tagged corpus and using word representations which are also based on the statistics of unambiguously tagged texts. The work presented in this paper is also an example of the *neural* approach to PoS tagging; however, one of the main differences is that we train a DTRNN on an *ambiguously tagged* corpus. As an intermediate step, the network is trained to *predict* the next ambiguity class in the text; then, a single-layer perceptron is trained to extract a PoS from the state representation developed by the network for the text seen so far. A deeper explanation follows.

## 2 Method

Discrete-time recurrent neural networks (DTRNN) have been used since their inception for predictive tasks in the natural language processing arena; the work of Elman [3] stands out. Elman trained his *simple recurrent net* (SRN, a class of DTRNN) to predict the next word in a synthetic corpus of simple, randomly-generated two- or three-word grammatical sentences drawn from a small vocabulary of untagged nonambiguous words. Elman found out that, after learning, the state representation the network developed after reading a word grouped words in categories that could be identified with classical tags such as noun, transitive verb, intransitive verb, etc. For the predictive task to succeed, the network had developed a way to obtain a syntactic representation of the portion of text seen up to that point.

The result of Elman's work has very interesting cognitive implications: humans learn syntax and parts of speech from their exposure to untagged language, only by observing co-ocurrences. The cognitive appeal of Elman's work is, however, obscured by the weak neurobiological plausibility of a discrete-time architecture such as a SRN [5].

### 2.1 Neural Architecture

Elman's [3] *simple recurrent net* (SRN), computes, from its previous *state* or *context* $\mathbf{s}[t-1]$ and from each input vector $\mathbf{a}[t]$ in the sequence, a new state $\mathbf{s}[t]$, using a single-layer perceptron; then, it computes an output $\mathbf{o}[t]$ from that state using another single-layer perceptron. SRN may be easily trained using the standard methods [7] for DTRNN such as RTRL (real-time recurrent learning) —our choice here— or BPTT (backpropagation through time).

### 2.2 Training Phases

In our approach, SRN is trained in two phases. First, the training text is ambiguously tagged using a lexicon or morphological analyser, assigning each word its *ambiguity class*, that is, a set of possible PoS tags (a singleton in the case of a nonambiguous word).

The use of ambiguity classes instead of words or tags reduces drastically the size of the task: Elman [3] worked on a small vocabulary, but real vocabularies have thousands of entries, whereas the number of ambiguity classes is usually in the range of a few hundreds.

After that, a two-phase training process begins:

**First Phase:** The SRN is trained to predict the ambiguity class of the next word $\mathbf{o}[t] \simeq \mathbf{a}[t+1]$ from the ambiguity class of the current word $\mathbf{a}[t]$, and, indirectly, from the classes of preceding words in the sentence, $\mathbf{a}[1], \cdots, \mathbf{a}[t-1]$, ideally stored in the network's state $\mathbf{s}[t-1]$. It is hoped that, in this way, the SRN will learn to develop in its state $\mathbf{s}[t]$ a syntactic representation of each particular ambiguity class in its previous context which allows it to make its best prediction about the ambiguity class of the following word.

The coding of ambiguity classes is one-hot, both for input and for output so that using a quadratic error function allows outputs to be interpreted as probabilitites. We use RTRL and learn the initial state $\mathbf{s}[0]$ as in [4] (the SRN is restarted at that learned state at the beginning of each sentence so that intersentence interferences are avoided).

**Second Phase:** After successful training, each word in the text is tagged with the hidden state vector computed for it by the network; then, for each word a perceptron is trained to predict its *part of speech* from the state vector assigned to the word which is $f$ positions to the right of it. The number $f$ represents the amount of right (forward) context needed to disambiguate a word; the amount of left (past) context is determined by the learning algorithm. Note that for this to work, $f$ end-of-sentence markers (a nonambiguous word) have to be added at the end of each sentence and predicted during the first training phase.

An output unit is assigned to each PoS in this second phase. When an ambiguous word is present, three different target schemes have been used:

a) output units representing tags in the ambiguity class are set to 1 and all other units to zero;

b) as $a$, but using $1/\theta$ instead of 1, where $\theta > 1$ is the size of the ambiguity class;

c) targets of 0 are provided only for tags not in the class; the rest do not contribute to the error function.

The third approach is the only one allowing convergence toward tag probabilities (with a quadratic error function), although we have found the first approach to give the best results.

The combination of the state part of the SRN and the perceptron is used to determine the PoS tag of words in new sentences. Figures 1 and 2 illustrate the training scheme for the first and the second phase, respectively.

### 2.3 Alternative Models

To evaluate the results of our approach we compare it with those obtained with three models:

- A standard hidden Markov model trained using the Baum-Welch algorithm [2, 10, 13].
- A model which randomly chooses a PoS tag from each ambiguity class with probability $1/\theta$, where $\theta$ is the number of tags in the class.
- A winner-takes-it-all model which always selects the most probable PoS tag observed in a large corpus for each ambiguity class.

Tagging results on a test text are compared to a hand-tagged version of the same text. The first model is the standard solution to the problem in hand, whereas the other two models may be used as baselines. In particular, the last baseline in less fair because it involves using more information besides the training text.

## 3 Results and Discussion

Experiments were performed to compute the error rates when tagging text taken from the Penn Treebank (release 3) corpus [11]. A 14276-entry lexicon was built from the first 20 sections of the 24 data sets corresponding to the *Wall Street Journal*, discarding all words appearing less than 4 times (which yielded a 95% coverage), and discarding for each word all tags below 5%. No guesser was used.

The training corpus has 46,461 words; the independent test corpus has 47,397 words, of which 6574 are ambiguous according to the lexicon and 2290 unknown. The 45 original tags in the Penn Treebank corpus were reduced to 19 coarser tags by removing some syntactical distinctions; 82 ambiguity classes were then observed. To train the SRN, pattern RTRL with a learning rate of 0.05 and no momentum term, and initial weights in the range $[-0.2, 0.2]$ were used; all neural results are averages of three differently initialized runs. Results are expressed as the percentage of incorrect tags assigned to *ambiguous words* (including unknown words), not as the overall percentage of correct tags (a common, but confusing error measure).

**HMM results** are shown in table 1 as a function of the number of Baum-Welch iterations.

**Random tagging** yields an incorrect tag rate between 61.8% and 62.9%, as observed in 14 experiments with different random number seed.

**Winner-takes-it-all** results in an incorrect tag rate of 20.5%, a reasonably good result which however relies on the availability of a large hand-tagged corpus.

$$\begin{array}{ccccccccccccccc}
\tilde{\mathbf{a}}[1] & & \tilde{\mathbf{a}}[2] & & \tilde{\mathbf{a}}[3] & & \dots & \tilde{\mathbf{a}}[L] & & \tilde{\#} & & \tilde{\#} & & \dots & \tilde{\#} \\
\uparrow & & \uparrow & & \uparrow & & & \uparrow & & \uparrow & & \uparrow & & & \uparrow \\
\mathbf{s}[0] & \rightarrow & \mathbf{s}[1] & \rightarrow & \mathbf{s}[2] & \rightarrow & \dots & \mathbf{s}[L-1] & \rightarrow & \mathbf{s}[L] & \rightarrow & \mathbf{s}[L+1] & \rightarrow & \dots & \mathbf{s}[L+f] \\
 & & \uparrow & & \uparrow & & & \uparrow & & \uparrow & & \uparrow & & & \uparrow \\
 & & \mathbf{a}[1] & & \mathbf{a}[2] & & \dots & \mathbf{a}[L-1] & & \mathbf{a}[L] & & \# & & \dots & \#
\end{array}$$

**Figure 1:** Training, first phase: learning to predict the next ambiguity class. The symbol "˜" means *predicted value*; the "#"'s are end-of-sentence markers. Note the trailing sequence of $f$ sentence boundaries, which are needed to tag the $f$ last words of a sentence in second phase.

$$\begin{array}{cccccccccccccc}
 & & & & & \mathbf{d}[1] & & \dots & \mathbf{d}[L-f-1] & & \mathbf{d}[L-f] & & \dots & \mathbf{d}[L] \\
 & & & & & \uparrow & & & \uparrow & & \uparrow & & & \uparrow \\
\mathbf{s}[0] & \rightarrow & \mathbf{s}[1] & \rightarrow & \dots & \mathbf{s}[f+1] & \rightarrow & \dots & \mathbf{s}[L-1] & \rightarrow & \mathbf{s}[L] & \rightarrow & \dots & \mathbf{s}[L+f] \\
 & & \uparrow & & & \uparrow & & & \uparrow & & \uparrow & & & \uparrow \\
 & & \mathbf{a}[1] & & \dots & \mathbf{a}[f+1] & & \dots & \mathbf{a}[L-1] & & \mathbf{a}[L] & & \dots & \#
\end{array}$$

**Figure 2:** Training, second phase: learning to obtain the correct tag

**Neural taggers** trained contain roughly the same number of parameters as corresponding HMM; therefore, 12 state units are used, although preliminary results with 24 units were marginally better. Results for $f = 0$ and $f = 1$ are shown in Table 2 as a function of the number of iterations in both phases. As can be seen, the addition of one word of right context confuses rather than help the SRN, which makes better predictions by relying only on past context. Also, it is clear that deep phase 2 training is more critical than deep phase 1 training.

As can be seen, a very simple neural tagger with roughly the same number of adjustable parameters attains basically the same results (about 45% incorrect tags) as a standard Baum-Welch-trained HMM model, but the neural tagger makes a decision on an ambiguous word without taking future words into account, whereas a HMM has to pospone the decision until a nonambiguous word appears, although this capability does not seem to give HMM an edge over the neural net. The overall correct tag rate of both models is around 92%.

## 4 Concluding Remarks

The work presented here shows that the information stored in the state of a discrete-time recurrent neural network trained to predict the ambiguity class of the next word can be useful in the problem of PoS tagging. Results show that the performance of this approach is, at least, similar to that of standard Baum-Welch trained HMM, but at the expense of ignoring the right context of the word.

We are working on the extraction, via clustering, of a finite-state automaton from the network's state [6] to formulate the finite-state tagging rules learnt by the network. The effect of discretization on the error rate attained by this automaton is also worth the study.

## References


[1] E. Brill, "A Simple Rule-Based Part-of-Speech Tagger", in *Proceedings Third Conference on Applied Natural Language Processing*, Trento, Italy, 1992.

[2] Doug Cutting, Julian Kupiec, Jan Pedersen and Penelope Sibun, "A Practical Part-of-Speech Tagger", *Proceedings of Third Conference on Applied Natural Language Processing*, Association for Computational Linguistics, 1992, pp. 133–140.

[3] J. L. Elman, "Finding Structure in Time", *Cognitive Science* 14, 1990, pp. 179–211.

[4] Mikel L. Forcada and Rafael C. Carrasco, "Learning the Initial State of a Second Order Recurrent Neural Network during Regular-Language Inference", *Neural Computation* 7, 1995, pp. 923–930.

[5] Mikel L. Forcada and Rafael C. Carrasco, "Finite-State Computation in Analog Neural Networks: Steps Towards Biologically Plausible Models?", in S. Wermter et al., *Emergent Neural Computational Architectures based on Neuroscience*, col. Lecture Notes in Computer Science 2036, Heidelberg, Springer-Verlag, 2001, in press.

[6] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun and Y. C. Lee, "Learning and Extracting Finite-State Automata with Second-Order Recurrent

**Table 1:** Percent error rate with a hidden Markov model as a function of Baum-Welch iterations

| Iterations | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Error rate | 62.8 | 51.7 | 48.8 | 47.2 | 46.0 | 45.5 | 45.3 | 45.3 | 45.5 |

**Table 2:** Error rates with 12 state units and $f = 1, 0$, according to the number of iterations in each phase

| | | Phase 1 iterations → | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0 | 100 | 200 | 300 | 400 | 500 |
| | 0 | 98.7, 95.4 | 61.8, 67.8 | 62.1, 66.7 | 62.0, 66.6 | 62.1, 66.4 | 62.3, 66.4 |
| Phase 2 | 100 | 96.7, 92.0 | 59.9, 46.4 | 60.1, 47.3 | 60.2, 47.4 | 60.2, 47.4 | 60.2, 47.5 |
| iterations | 200 | 96.8, 90.4 | 55.2, 45.6 | 55.3, 45.4 | 55.2, 45.4 | 55.1, 45.2 | 54.9, 45.0 |
| ↓ | 300 | 96.1, 90.6 | 55.8, 44.9 | 56.0, 44.5 | 55.7, 44.4 | 55.7, 44.2 | 55.4, 44.2 |
| | 400 | 96.5, 90.6 | 55.8, 44.4 | 55.8, 44.9 | 55.8, 45.1 | 55.7, 45.0 | 55.8, 45.0 |
| | 500 | 96.9, 89.6 | 55.5, 45.1 | 55.5, 45.5 | 55.5, 45.5 | 55.5, 45.6 | 55.7, 45.5 |

Neural Networks", *Neural Computation* 4(3), pp. 393–405.

[7] S. Haykin, *Neural Networks: a Comprehensive Foundation*, 2nd. ed., NJ: Prentice-Hall.

[8] Qing Ma and Hitoshi Isahara, "Part-of-Speech Tagging of Thai Corpus with the Logically Combined Neural Networks", *Proceedings of the Natural Language Processing Pacific Rim Symposium*, Linguistics and Knowledge Science Laboratory, 1997, pp. 537–540.

[9] Qing Ma, Masaki Murata, Masao Utiyama, Kiyotaka Uchimoto and Hitoshi Isahara, "Part of Speech Tagging With Mixed Approaches of Neural Networks and Transformation Rules", NLPRS'99 Workshop on Natural Language Processing and Neural Networks, Beijing, China, 1999.

[10] Cristopher D. Manning and Hinrich Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, 1999.

[11] Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz, "Building a Large Annotated Corpus of English: the Penn Treebank", *Computational Linguistics* 19, 1993, pp. 313–330.

[12] Nuno C. Marques and Gabriel Pereira Lopes, "Using Neural Nets for Portuguese Part-of-Speech Tagging", *Proceedings of the Fifth International Conference on The Cognitive Science of Natural Language Processing*, Dublin City University, Ireland, 1996.

[13] Lawrence R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", *Proceedings of the IEEE* 77(2), 1989, pp. 257–286.

[14] H. Schmid, "Part-of-Speech Tagging with Neural Networks", *Proceedings of the International Conference on Computational Linguistics*, 1994, pp. 172–176.