Fonseca *et al. Journal of the Brazilian Computer Society* (2015) 21:2
DOI 10.1186/s13173-014-0020-x

Journal of
the Brazilian Computer Society
a SpringerOpen Journal

RESEARCH Open Access

Evaluating word embeddings and a revised corpus for part-of-speech tagging in Portuguese

Erick R Fonseca*, João Luís G Rosa and Sandra Maria Aluísio

Abstract

Background: Part-of-speech tagging is an important preprocessing step in many natural language processing applications. Despite much work already carried out in this field, there is still room for improvement, especially in Portuguese. We experiment here with an architecture based on neural networks and word embeddings, and that has achieved promising results in English.

Methods: We tested our classifier in different corpora: a new revision of the Mac-Morpho corpus, in which we merged some tags and performed corrections and two previous versions of it. We evaluate the impact of using different types of word embeddings and explicit features as input.

Results: We compare our tagger's performance with other systems and achieve state-of-the-art results in the new corpus. We show how different methods for generating word embeddings and additional features differ in accuracy.

Conclusions: The work reported here contributes with a new revision of the Mac-Morpho corpus and a state-of-the-art new tagger available for use out-of-the-box.

Keywords: Natural language processing; Part-of-speech tagging; Neural networks; Word embeddings

Background

Part-of-speech (POS) tagging is an important natural language processing (NLP) task, used as a preprocessing step in many applications. Its objective is to tag each token in a sentence with the corresponding part-of-speech tag.

While POS tagging has a simple definition and is widely employed, the way it is implemented varies a lot in practice. First, there is no single best algorithm to perform the task, and second, there is the question of which tagset to use.

Tagsets may differ in their granularity, e.g., having different tags for plural and singular nouns or grouping them under a single tag. In English, for example, the most widespread ones are the Penn Treebank tagset [1], with 45 tags including punctuation; the CLAWS5 and CLAWS7 ones, used in the British National Corpus, with 62 and 137 tags, respectively. The tagset a system uses is usually determined by the annotated corpus it is trained on, which brings the issue of the corpus itself.

In order to be useful in practice, a corpus with POS annotation should contain sentences that reflect the kind of text one expects to apply a POS tagger on. Also, given the nature of POS tagging methods in current NLP research, the tags should be deductible without having to analyze syntactic structure. In previous research [2], we revised the Mac-Morpho corpus [3] in order to provide a more reliable resource. Here, we present further revisions in the corpus, combining tags that were difficult for a POS tagger to discriminate correctly. Our goal was to evaluate if these revisions would result on further improvements on POS tagger accuracy.

As for methods, while rule-based systems exist [4], machine learning-based techniques have been dominant for years, as in most NLP research. Many algorithms have been proposed and successfully applied in this task, usually capable of achieving good results in different

*Correspondence: erickrf@icmc.usp.br
Institute of Mathematics and Computer Science (ICMC), University of São Paulo, Av. Trabalhador São-Carlense, 400, 13566-590, São Carlos, Brazil

languages. Still, POS tagging continues to be explored in search of better algorithms and improvements over existing ones, since better tagging performance usually helps other NLP applications. Also, POS tagged data are a relatively big test bed for machine learning methods dealing with sequence classification.

In Portuguese, recent experiments reported in the literature include transformation-based learning [5], hidden Markov models (HMM) and variable length Markov chain (VLMC) [6], HMM with a character language model [7], and neural networks with word embeddings [2,8].

Word embeddings are representations of words as real-valued vectors in a multidimensional space. They can be generated by many different techniques, ranging from classical approaches based on word frequencies and co-occurrences [9] to neural language models [10]. Embeddings are usually obtained from huge unannotated corpora and can encode syntactic and semantic information about words.

In contrast, the traditional way to represent words in NLP systems is to treat each token as a completely independent feature. A classifier thus views each word as a sparse vector with the size of the known vocabulary having the value 1 in a given dimension and zeros elsewhere. Word embeddings, in comparison, have two evident advantages: they map words to a relatively small space (a few hundred dimensions or even less) and capture similarities between words (the type of similarity varies according to the method used to generate the embeddings).

In the last few years, their advantages have attracted the interest of the NLP community in both using them as input to automatic classifiers [11-13] and in developing new ways of inducing embeddings [11,14,15].

As in our aforementioned paper [2], we explored the neural network-based tagger. In this paper, we tested it under different conditions, experimenting with varied word embedding models and additional explicit features such as the presence of capital letters and word endings. We carried out comparisons with other taggers and found that ours achieves state-of-the-art performance with 97.57% overall accuracy on the original Mac-Morpho corpus, 97.48% on our first revised version, and 97.33% on the second revision presented here.

The remainder of this paper is organized as follows. The section 'The corpus and its tagset' presents the Mac-Morpho corpus, its tagset, and the changes we applied to them. In the section 'Methods', we described the model we used to train the tagger. In the section 'Experimental setup', we describe in detail the data used in the experiments and parameters evaluated. Results and comparison with other systems is found in 'Results and discussion', and we present our final words in the section 'Conclusions'.

The corpus and its tagset

The most widespread corpora annotated with POS tags in Portuguese are Tycho Brahe [16], Mac-Morpho [3], and Bosque [17]. The first two are larger, with around one million tokens. Bosque has around 200,000 tokens in its most recent version (8.0) and is composed by both Brazilian and European Portuguese texts. Unfortunately, the three corpora have incompatible tagsets and are tokenized differently, which prevents them from being combined into one larger resource without losing the richness of each particular annotation.

Tycho Brahe is a corpus of historical texts, whose authors were born from the fourteenth to the nineteenth century. It exhibits a very different writing style and vocabulary than those of modern Portuguese, and thus, training a POS tagger on it is only of practical interest to people dealing with texts from that time span.

Here, we focus on Mac-Morpho because it is the largest manually POS-tagged corpus of modern Portuguese, collected from Brazilian newspaper articles. Its tagset is displayed in Table 1 (there are also 19 punctuation tags, not displayed in the table). We took the revised version presented in [2], in which many errors from the original

Table 1 Original Mac-Morpho tagset with 22 tags, without including punctuation tags

Tag	Meaning
ADJ	Adjective
ADV	Adverb
ADV-KS	Subordinating connective adverb
ADV-KS-REL	Subordinating relative adverb
ART	Article
CUR	Currency
IN	Interjection
KC	Coordinating conjunction
KS	Subordinating conjunction
N	Noun
NPROP	Proper noun
NUM	Number
PCP	Participle
PDEN	Denotative word
PREP	Preposition
PROADJ	Adjective pronoun
PRO-KS	Subordinating connective pronoun
PRO-KS-REL	Subordinating connective relative pronoun
PROPESS	Personal pronoun
PROSUB	Nominal pronoun
V	Verb
VAUX	Auxiliary verb

version were corrected, such as sentences with missing words, repeated sentences, tag mistypes, and a few others.

Missing words were checked following a few heuristic rules, based on impossible (or near-impossible) combinations like an article being followed by a verb or two contiguous commas. By reading these sentences, it was clear that a word was supposed to be there, but for some reason, it was not.

Such sentences were discarded, since there was no way to recover the missing word. Repeated sentences were also removed. Tag mistypes were easily found checking for any tag not in the corpus tagset and could be manually corrected.

Besides error correction, the revised version properly delimits sentence boundaries and does not separate word contractions (like *do* for *de* + *o*, meaning 'of the'), which reflects a real-world scenario. The original corpus presented such tokens separately but indicated where there were contractions. Thus, it was easy to redo them when they occurred. Contracted tokens had as their POS tag the concatenation of both tags with a plus sign between them; so, the preposition and article contraction *do* is tagged as `PREP+ART`. Finally, all punctuation tokens were mapped to a single tag `PU`.

In this work, we performed more corrections on the corpus. We found a few sentences where empty strings were assigned a tag, as if they were tokens. This was mainly due to the problems with punctuation tokens from the original corpus, and we discarded such sentences.

Also, we introduced another change in the tagset, combining some related tags. This new changes were based on the Penn Treebank tagset [1], the most widespread POS tagset in English. These are the tags we merged:

1. `V` and `VAUX`, namely, for non-auxiliary and auxiliary verbs. Both were merged into `V`. The motivation for combining them is that this distinction is actually related to the syntactic analysis level, where sentence structure is examined. A POS tagger should not be expected to do this level of analysis. In fact, it has been pointed out that at least 30 verbs in Brazilian Portuguese can work as auxiliaries [18], usually only needing to precede an infinite verb form (infinitive, gerund, or participle), and most of them can also be main verbs on their own. In the Penn Treebank, there is a tag for modals, which refers to a closed set of 11 verbs, but other ones working as auxiliaries are tagged the same way as main verbs. Examples:

   Mas se o Brasil ***começar*** *a vender café (...)* (But if Brazil starts to sell coffee) *Começar* (start) was originally tagged as `VAUX` and retagged as `V`.

   Os leilões de café ***começaram*** *no mês passado.* (Coffe auctions started last month) Here, *começaram* (a different inflection of the same verb above) is the main verb and therefore has the tag `V`. However, a possible sentence in Portuguese is '*Os leilões de café* ***começaram*** *no mês passado a dar lucro*' (Coffe auction started to make profit last month). In this case, *começaram* works as an auxiliary and is not close to the main verb (*dar*), which would make it very hard for the tagger to get the right answer. For comparison, the verb *start* in the English translation would have the same tag in both cases.
2. `PRO-KS` and `PRO-KS-REL` merged into `PRO-KS` (and also their variants including preposition contraction). Both tags refer to relative pronouns introducing subordinate clauses, but `PRO-KS-REL` must have a referent in the sentence. Both usages are exemplified below:

   Pode exigir apenas ***o que*** *havia deixado de cobrar.* (One can demand only what he had not charged before). Here, *o que* (what) does not have an explicit referent, and thus, both tokens are tagged as `PRO-KS`.

   O cliente terá acesso a 70% da remuneração ***que*** *exceder a da poupança.* (The client will have access to 70% of the remuneration that exceeds the saving funds). In this case, *que* (that) refers to something mentioned before and was tagged as `PRO-KS-REL`.

   The motivation here is that detecting correferences is a difficult NLP problem itself, so we can't expect a POS tagger to perform it as a subtask. The Penn Treebank makes no such distinction.
3. `ADV-KS` and `ADV-KS-REL` merged into `ADV-KS`. Analogous to the last item but related to adverbs instead of pronouns. The Penn Treebank also does not make this distinction. Examples:

   A descoberta pode ajudar médicos a decidirem ***quando*** *aplicar um tratamento (...)* (The discovery may help doctors decide when to apply a treatment). Here, *quando* (when) is tagged as `ADV-KS` because there is no referred time.

   Em comparação a outubro, ***quando*** *a alta havia sido de 5,7% (...)* (In comparison with October, when the increase had been of 5.7%). Here, *quando* refers to an explicit moment and thus is tagged as `ADV-KS-REL`.

Lastly, we changed the way clitic pronouns are split from verbs. The original version presented verbs and pronouns separately, with an indication in the verb tag that it was followed by a clitic. For example:

Sente se *frustrado por não ter jogado o mundial de os EUA?* (Do you feel frustrated for not playing the USA

World Cup?). In actual running text, the first two tokens would be written *sente-se* but here were split into two. The verb itself, *sente*, receives the tag `V|+`, while the clitic has the usual pronoun tag `PROPESS`.

The revised version, instead, presented verbs with a trailing hyphen, and no distinction in its tag. This was done in order to conform to the Bosque corpus. The example sentence above became *Sente- se frustrado por (...)*, with the tags `V` and `PROPESS` for the first two tokens.

However, leaving the hyphen with the verb increases data sparsity, and so we decided to leave it with the pronoun. Thus, in the latest Mac-Morpho version, we have *Sente -se frutrado (...)*, with the same tags as in the first revision.

Figure 1 illustrates the tag frequency distribution after our changes in the corpus. As usual, nouns are by far the most common part of speech, followed by punctuation signs, verbs, proper names, and prepositions. Some preposition contractions are very rare, as are interjections.

It is interesting to note that two points make the context of each token very important in order to determine its correct tag. In short adverbial or prepositional multiword expressions, tokens are tagged as adverbs or prepositions. For example, in the phrase *por antecipação* (by anticipation), both tokens are tagged as adverbs instead of preposition and noun, since they are equivalent to *antecipadamente* (anticipatedly).

Additionally, the proper name category covers not only person and place names (such as *João* or *Brasil*) but also entity names composed of existing words. For example, whereas *ministério* and *agricultura* are common nouns when used on their own, the phrase *Ministério da Agricultura* (Ministry of Agriculture), referring to the actual entity, is a proper name, and the three tokens are tagged as such. Without text understanding, the only hint of their actual POS is the presence of capital letters.

Methods

We implemented the model introduced in [19] for training a POS tagger. It is based on a multilayer perceptron neural network that receives word representations in the form of numeric vectors and learns to predict their part-of-speech from a set of predefined tags U. These word representations, also known as *word embeddings*, have a fixed number d of dimensions and are generated by unsupervised methods, explained later in this section.

In order to tag the tokens in a sentence, the tagger takes a window of tokens at a time and maps them to their vectors, which are concatenated and fed to the neural network. In the case of tokens in the beginning or the end of a sentence, the window is filled with pseudo-tokens serving as padding. These pseudo-tokens also have their own vectors.

The concatenated vectors go through a standard hidden layer, which performs a weighted sum followed by a nonlinear function. We follow [19] and use an approximate and faster version of the hyperbolic tangent, shown in Equation 1. The output layer performs another weighted sum on the resulting values.

$$\text{hardtanh}(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x \geq 1 \\ x, & \text{otherwise.} \end{cases} \tag{1}$$

The network outputs a score $f_u(t)$ for assigning each tag $u \in U$ to the token t in the middle of the input window. Figure 2 exemplifies a window of three tokens mapped into vectors, and Figure 3 shows the outline of the network that processes it.

The output of the network is combined with a tag transition score, also learned during training, which encodes knowledge such as 'an article is very likely to be followed by a noun'. A matrix A has in each cell $A_{u,v}$ the score for

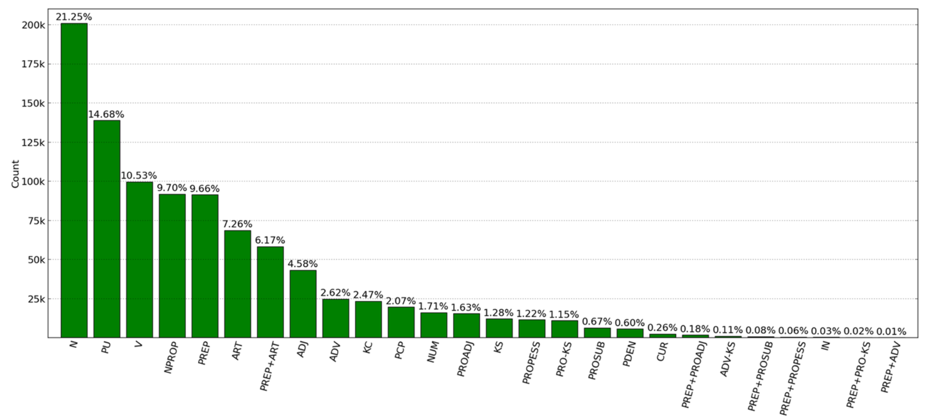

Figure 1 Tag frequency distribution in Mac-Morpho after changes performed in this work.

| Dados | **serão** | apurados | $\longrightarrow$ $\boxed{0.82 \mid 0.45 \mid \ldots}$ $\boxed{-0.65 \mid 0.18 \mid \ldots}$ $\boxed{0.13 \mid -0.77 \mid \ldots}$

Figure 2 Example of a window of three tokens being converted into feature vectors.

a token tagged with u being followed by a token tagged with v. Additionally, the row $A_{0,u}$ contains the scores for starting a sentence with each tag u.

Thus, after network scores have been produced for all tokens in the sentence x of length T, the final score for a tag sequence y is the following:

$$s(x,y) = \sum_{t=1}^{T} \left(f_{y_t}(x_t) + A_{y_{t-1},y_t} \right). \qquad (2)$$

The final system answer is the tag sequence y^* with the highest score. It can be determined in polynomial time using the Viterbi algorithm [21].

Training

Training the system consists in adjusting three sets of parameters: the neural network connection weights, the values of the input vectors, and the tag transition scores. Our training procedure follows [22]: all adjustments are made in order to maximize the likelihood over the training data. Formally, we want to maximize the following sum:

$$\sum_{(x,y)\in\mathcal{T}} p(y|x,\theta). \qquad (3)$$

Where $\mathcal{T}$ symbolizes our set of (sentence, POS tags) pairs, and θ represents the system parameters. The probability $p(y|x,\theta)$ for a given tag sequence y can be computed from the model output using a softmax operation:

$$p(y|x,\theta) = \frac{e^{s(x,y)}}{\sum_j e^{s(x,j)}}. \qquad (4)$$

Where j iterates over all possible tag sequences. The log-likelihood, more suited to work with low probabilities, is determined as:

$$\log p(y|x,\theta) = s(x,y) - \mathrm{logadd}_j s(x,j). \qquad (5)$$

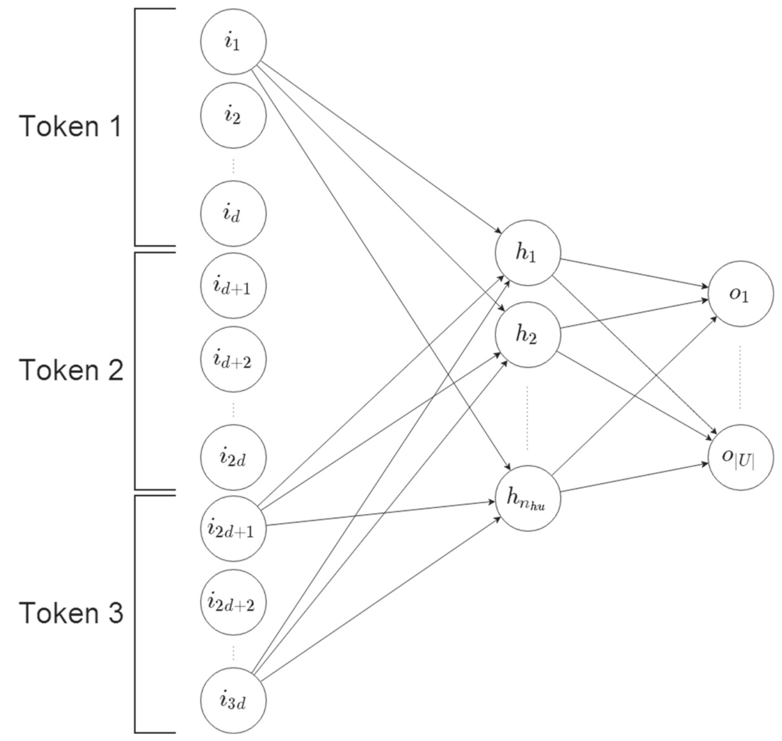

Figure 3 Neural network outline. In the input layer, there is one neuron for each dimension in the combined vectors of the token window (each vector having length d). The hidden and output layers are as in conventional MLP networks [20]. They have n_{hu} and $|U|$ neurons, respectively. For simplicity, biases and some connections were omitted.

With logadd$_i$ being shorthand for

$$\text{logadd}_i z_i = \log\left(\sum_i e^{z_i}\right) \tag{6}$$

The logadd term includes every possible tag sequence, and its size grows exponentially with the sentence size. However, it can be computed in linear time using dynamic programming techniques. The computation starts with the first token x_1, whose score for each tag u is simply the sum of the tag transition score from the sentence beginning to u and the neural network output for u.

$$s(x_1, u) = f_u(x_1) + A_{0,u}. \tag{7}$$

The logadd in the first token is then trivially resolved to the score itself:

$$\begin{aligned}\text{logadd}_k s(x_1, u) &= \log \sum_{t=1}^{1} e^{s(x_t, j_t)} \\ &= \log e^{s(x_1,u)} \\ &= s(x_1,).\end{aligned} \tag{8}$$

Then, considering partial tag sequences j_1^t from the first to the tth token and ending with tag u, we have:

$$\begin{aligned}\delta_t(u) &= \text{logadd}_{(j_1^t | j_t = u)} s\left(x_1^t, j_1^t\right) \\ &= \text{logadd}_v\ \text{logadd}_{(j_1^t | j_{t-1}=v, j_t=u)} s\left(x_1^{t-1}, j_1^{t-1}\right) \\ &\quad + A_{v,u} + f_u(x_t) \\ &= \text{logadd}_v\ \delta_{t-1}(v) + A_{v,u} + f_u(x_t) \\ &= f_u(x_t) + \text{logadd}_v\left(\delta_{t-1}(v) + A_{v,u}\right).\end{aligned} \tag{9}$$

With v iterating over all tags at the token in position $t-1$. At the last token, we have:

$$\text{logadd}_{j_1^T} s\left(x_1^T, j_1^T\right) = \text{logadd}_v\ \delta_{j_T}(v). \tag{10}$$

Recalling Equation 5, we want to maximize the score for the correct tag sequence at the expense of the scores for all others. We call the opposite of that equation the cost function, which we want to minimize. It is a function of all the system parameters, which include the neural network and the transition score matrix:

$$C(\theta) = \text{logadd}_j\ s(x, j) - s(x, y). \tag{11}$$

We adjust all the parameters doing a gradient descent over the cost function. In other words, it means finding the negative gradient of the error function with respect to each of the system parameters in all instances of the training set. Then, for each parameter, its respective gradient is multiplied by the learning rate and added to it.

During training, after the system has produced an answer for a given sentence, we initialize two gradient matrices. One has the gradients with respect to the neural network score for each token/tag combination, and the other has the gradients with respect to the score of each tag transition. Both matrices are initialized with the value 0 in all cells:

$$\frac{\partial C}{\partial f_u(t)} = 0\ , 1 \le t \le T, \forall u \in U \text{ and } \frac{\partial C}{\partial A_{u,v}} = 0\ \forall u, v \in U. \tag{12}$$

First, we accumulate the negative gradients over the logadd part of (11), which we want to minimize. Applying the chain rule, we have:

$$\frac{\partial C_{\text{logadd}}}{\partial \delta_T(u)} = -\frac{e^{\delta_T(u)}}{\sum_v e^{\delta_T(v)}}\ \forall u. \tag{13}$$

Then, we compute the gradient at each token, from the last to first:

$$\frac{\partial C_{\text{logadd}}}{\partial \delta_{t-1}(u)} = \sum_v \frac{\partial C_{\text{logadd}}}{\partial \delta_t(v)} \frac{e^{\delta_{t-1}(u) + A_{u,v}}}{\sum_w e^{\delta_{t-1}(u) + A_{w,v}}}. \tag{14}$$

At each token t, we update the matrices with gradients for the neural network and the transition scores:

$$\begin{aligned}\frac{\partial C}{\partial f_u(t)} &-= \frac{\partial C_{\text{logadd}}}{\partial \delta_t(u)} \frac{\partial \delta_t(u)}{\partial f_u(t)} = \frac{\partial C_{\text{logadd}}}{\partial \delta_t(u)} \\ \frac{\partial C}{\partial A_{u,v}} &-= \frac{\partial C_{\text{logadd}}}{\partial \delta_t(v)} \frac{\partial \delta_t(v)}{\partial A_{u,v}} = \frac{\partial C_{\text{logadd}}}{\partial \delta_t(v)} \frac{e^{\delta_{t-1}(u) + A_{u,v}}}{\sum_w e^{\delta_{t-1}(k) + A_{w,v}}}.\end{aligned} \tag{15}$$

Finally, we deal with the other term from (11), which we want to maximize. We add 1 to the gradient matrices at each point corresponding to a correct token/tag assignment or a tag/tag transition in the training tag sequence y.

$$\frac{\partial C}{\partial f_{y_t}(t)} += 1 \quad \text{and} \quad \frac{\partial C}{\partial A_{y_{t-1}, y_t}} += 1 \quad \forall t. \tag{16}$$

We use the well-established backpropagation algorithm to adjust the network weights. When performing it, we backpropagate error gradients until the input layer, which allows us to adjust the word representations as if they were neural network weights.

Word representations

The first level of this architecture is based on word embeddings, i.e., a vector space where each word has a corresponding real-valued vector[a]. The only desideratum in the representations is that similar words have similar vectors, according to euclidian distance or cosine similarity. By *similar*, we mean words that tend to be used in the same contexts and usually belong to the same part-of-speech category.

Word embeddings allow the automatic classifier to detect words that should be treated similarly. Also, words not seen in the training data for a tagging task are not completely unknown, as long as they have a feature vector. Thus, out-of-vocabulary (OOV) impact is expected to be smaller.

Collobert et al. [22] used a neural language model to initialize their word representations. This model is based on a neural network that assigns scores to windows of 11 tokens and learns to output values for positive examples (token windows extracted from a corpus) higher than for negatives ones (random perturbations of positive examples). The corrections in the network parameters were backpropagated to the word representations, which were adjusted similarly to network weights. After training the model, words with similar meaning and usage had vectors with a small euclidian distance.

These kinds of word representations have been successfully used in a myriad of NLP tasks. Collobert et al. [22] also used their architecture to deal with named entity recognition, syntactic chunking, and semantic role labeling. In [11], this model was extended to perform constituency parsing. A new version of the semantic role labeling system was successfully applied to Portuguese in [23]. Luong et al. [24] employed a recursive model to learn vector representations for phrases and used them to perform syntactic parsing. In [12], a similar model is used in the task of sentiment analysis.

The number of dimensions in the vectors may vary. In general, the more dimensions they have, the better representations can be achieved. However, if vectors are too big, processing them will be slower. The embeddings used in systems like the ones cited above have 50 to 100 dimensions.

Feature vectors can also represent discrete attributes such as the presence of capitalization. To this end, each possible value of the attribute must have a corresponding vector; in the case of capitalization, values could be: all lowercase letters, initial uppercase letter, other combinations, and a N/A value for punctuation and numbers. Thus, in order to create the full representation of a token, its word vector is concatenated with the vectors for all attributes. Figure 4 exemplifies this process.

Word representations, however, may be obtained by other means. Huang et al. [14] present a variation of the neural language model, where the network score is based not only on a small window of text but also on a vector obtained by averaging all words in a document. By doing this, the similarity between their resulting vectors also reflects words that appear in the same documents - and which tend to describe similar concepts. However, since we are dealing with POS tagging, we are more interested in a finer grained similarity, not *concept* similarity.

Turian et al. [25] report that different kinds of representations can be successfully used by a classifier in NLP tasks. They tested the impact of word representations when used as additional features for a classifier in the tasks of named entity recognition and syntactic chunking. While some representations were better than others in some situations, all of them improved performance.

Due to the fact that training a neural language model is very time consuming, Fonseca and Rosa [2] employed methods based on distributional semantics [9] for their POS tagger, which are much faster. They used the hyperspace analogue to language (HAL) [26], a method based

Feature vectors for word types in vocabulary

⋮				
náutico	$\longrightarrow$	-0,81	0,09	-0,21
não	$\longrightarrow$	0,97	-0,34	0,16
nós	$\longrightarrow$	0,49	0,82	0,63
⋮				

Feature vectors for capitalization

All lowercase	$\longrightarrow$	0,04	0,72
Uppercase initial	$\longrightarrow$	-0,59	0,18
Other combinations	$\longrightarrow$	-0,12	-0,65
N/A	$\longrightarrow$	0,94	0,51

Tokens in running text

nós	$\longrightarrow$	0,49	0,82	0,63	0,04	0,72
não	$\longrightarrow$	0,97	-0,34	0,16	0,04	0,72
Não	$\longrightarrow$	0,97	-0,34	0,16	-0,59	0,18
NÃO	$\longrightarrow$	0,97	-0,34	0,16	-0,12	-0,65

Figure 4 Examples of five-dimensional feature vectors representing tokens. They are obtained by combining a three-dimensional word-type vector with a two-dimensional capitalization vector.

Table 2 Corpora size

Corpus		Train	Development	Test	Total	Tagset
Mac-Morpho v1	Sentences	42,022	2,211	9,141	53,374	41
	Tokens	957,439	50,232	213,794	1,221,465	
Mac-Morpho v2	Sentences	42,742	2,249	4,999	49,990	30
	Tokens	807,818	43,145	94,995	945,958	
Mac-Morpho v3	Sentences	37,948	1,997	9,987	49,932	26
	Tokens	728,497	38,881	178,373	945,751	
Tycho Brahe	Sentences	29,163	1,535	10,234	40,932	265
	Tokens	734,922	40,679	259,991	1,035,592	

on counting occurrences of words next to each other, thus obtaining a large count matrix, and then reducing the matrix dimensionality by selecting the dimensions with the highest variance (50 in their experiments). In [27], word representations generated with similar methods are also employed for POS tagging in English; however, the system architecture in that work is very different from the one explored here.

Another interesting strategy for generating vectors is the skip-gram modeling from [15], in which a log-linear classifier tries to predict words surrounding a given word. The prediction is based on the neural language model introduced in [10], which assigns a score to each word in the vocabulary. In order to overcome the huge quantity of possible answers, the output is organized in a binary tree. Therefore, instead of expensive operations involving the vocabulary size V, the system output follows a path of size $log_2 V$. Since the log-linear classifier has no hidden layer, the algorithm is substantially faster than the original neural language model. The authors show that vectors induced with skip-gram capture syntactic and semantic similarity between words.

In this work, we experimented with word embeddings generated by different methods, aiming at comparing their usefulness in POS tagging. We used HAL implementation from semantic vectors [28,29], like Fonseca and Rosa. In addition, we tried a neural language model (the same algorithm used by Collobert et al. in [22]) with the implementation from word2embeddings [30] and the skip-gram implementation from gensim [31,32].

Besides trying new embeddings, we significantly increased our data: we combined a January 2014 Wikipedia dump with the PLN-BR corpus [33] and a new corpus we compiled collecting articles from the G1 news website (http://www.g1.com.br). In total, this yielded around 240 million tokens, roughly 100 million more than the data used in [2].

In the preprocessing step, we converted any digit in the corpus to nine, in order to reduce data sparsity. Our tokenizer, based on regular expressions, also splits clitic pronouns from verbs in the same way they appear in our new version of Mac-Morpho (the hyphen is left with the pronoun). We generated vectors for all word types appearing at least 20 times in the combined corpus (ignoring differences in uppercase or lowercase), which resulted in a vocabulary of 160,270 items. All words not in this list were

Table 3 Attributes represented by vectors

Attributes	Vector length
Word-type *pessoa* itself	50
All lowercase	5
Ending in *a*	2
Ending in *oa*	2
Ending in *soa*	2
Ending in *ssoa*	2
Ending in *essoa*	2
Beginning with *p*	2
Beginning with *pe*	2
Beginning with *pes*	2
Beginning with *pess*	2
Beginning with *pesso*	2
Total	75

Table 4 Tagger parameters used in the experiments

Parameter	Value
Word window size	5
Hidden layer size	100
Word embedding dimensions	50
Capitalization embedding dimensions	5
Prefix embedding dimensions	2
Suffix embedding dimensions	2
Learning rate (at epoch *i*)	$\frac{0.01}{i}$
Training epochs	20

Table 5 Tagging accuracy on Mac-Morpho v1

Input features	Only words		Capitalization		Prefix + suffix		All three	
	All (%)	OOV (%)	All (%)	OOV (%)	All (%)	OOV (%)	All (%)	OOV (%)
Random	95.10	70.75	96.65	84.58	96.12	83.64	97.33	92.37
HAL	95.84	78.14	97.04	86.87	96.36	86.35	97.41	92.34
NLM	96.34	85.68	97.44	91.71	96.56	88.62	97.57	93.38
SG	96.10	82.83	97.24	89.99	96.47	87.70	97.44	93.32

mapped to a single vector that was generated randomly. Our representations had 50 dimensions.

The running time needed to obtain the representations is also a factor to be considered. The neural language model vectors were obtained after around 10 days running on a 2.7 GHz Intel Xeon processor in a Linux server. There is no exact time after which the vectors are ready, so we verified the quality of the representations empirically by querying the model for the most similar words to a given one. When we saw no further improvement, we finished the process. Skip-gram and HAL vectors, on the other hand, were generated in a few hours in the same computer architecture.

Experimental setup

Corpora

We trained our tagger on three versions of the Mac-Morpho corpus: the original one, using the same train/test split from [34] (referred to as Mac-Morpho v1); the revision presented in [2] (Mac-Morpho v2); and the new version described here (Mac-Morpho v3). Additionally, we experimented on the Tycho Brahe corpus [16], also using the same split from [34].

We split Mac-Morpho v3 into 80% of the sentences for training and the remaining 20% for testing (In the work presented in [2], the split was 90% and 10%). We also set aside 5% of all training portions for parameter validation, following [34]. Table 2 shows the size in numbers of sentences and tokens for all corpora we used.

The Tycho Brahe tagset is much larger than the others because it includes morphological information in its tags, such as gender, number, verb tense, etc. It also has a separate tag for common verbs like *ser*, *estar* (both can translate into to be), and *ter* (to have).

Word embeddings

We tried different word representations as input to our neural networks in order to evaluate their impact, if any. As already mentioned, we generated representations with HAL, a neural language model (NLM) and skip-grams (SG). As a baseline, we also tried randomly initialized vectors, which were generated for all word types appearing at least twice in the training data. All other tokens were mapped to a single vector. In all models, vectors representing the padding to the left and to the right of the sentence were generated randomly.

One of the additional attributes we evaluated was capitalization, which is very important to correctly detect proper names. The possible values for capitalization were five: all lowercase, initial uppercase, other case combinations, non-alphabetic tokens, and padding. Vectors with five dimensions were randomly generated for these and adjusted during training.

We also experimented training models without any information about capitalization. The purpose of this was not only measuring the performance gain of observing capital letters but also obtaining a tagger insensitive to case variation. This is particularly interesting when dealing with user-generated texts from the Web, which often display inconsistent capitalization usage, a potential problem for POS taggers [35].

Moreover, we measured the impact of adding two other attributes: prefixes and suffixes, also used in the ESL tagger from [36] (for simplicity, we call the first and the last few characters of a word, respectively, a *prefix* and a *suffix*, even if it is not linguistically accurate). For each length from one to five, we collected all prefixes and suffixes appearing in at least five word types in the training data and initialized a random vector with two dimensions for

Table 6 Tagging accuracy on Mac-Morpho v2

Input features	Only words		Capitalization		Prefix + suffix		All three	
	All (%)	OOV (%)	All (%)	OOV (%)	All (%)	OOV (%)	All (%)	OOV (%)
Random	94.99	74.46	96.50	83.72	96.04	86.37	97.25	93.40
HAL	95.61	77.81	96.94	85.95	96.25	87.35	97.29	92.91
NLM	96.31	85.64	97.32	91.16	96.46	90.04	97.48	94.34
SG	95.89	82.56	97.14	88.64	96.41	89.17	97.35	93.61

Table 7 Tagging accuracy on Mac-Morpho v3

Input features	Only words		Capitalization		Prefix + suffix		All three	
	All (%)	OOV (%)	All (%)	OOV (%)	All (%)	OOV (%)	All (%)	OOV (%)
Random	94.27	69.18	96.06	81.55	95.49	84.02	96.93	91.62
HAL	95.12	74.97	96.61	84.61	95.79	85.49	97.10	91.60
NLM	95.95	85.04	97.21	91.21	96.14	89.01	97.33	93.66
SG	95.55	81.58	96.92	88.20	96.01	88.13	97.19	93.01

it. Rare prefixes and suffixes of each size share a single vector.

This means that, in effect, prefixes and suffixes are implemented as five separate features each. For example, the vector representation of the word *pessoa* (person), using all attributes described above, would be the concatenation of vectors as shown in Table 3. If any of the suffixes or prefixes does not have a feature vector of its own, then it is mapped to the 'rare' suffix or prefix vector.

In all tagger setups, we employed an input word window of size 5 and 100 neurons in the hidden layer. When capitalization, prefixes, and suffixes features are used, our resulting input layer has 375 neurons. The implementation we used to train our networks has been improved since its previous publication in [2], allowing for much faster convergence. We trained our neural networks for 20 epochs using learning rate decay, such that the rate at the ith epoch was $\frac{0.01}{i}$. Table 4 summarizes the parameters.

Results and discussion

Here, we present results obtained in our experiments for each corpus. Results for all our configurations are shown in Tables 5, 6, 7 and 8. For each type of embeddings used as input, we evaluate accuracy for all tokens and for OOV ones.

The NLM representations achieved the best results overall, with the performance difference being more evident in OOV tokens. This indicates that this vector space model captures very well the morphosyntactic similarities between words before any supervised training. SG vectors deliver a slightly lower accuracy, but the surprising fact is that HAL vectors can be worse than random initialization for OOV words in some situations.

The only exception to this tendency is the Tycho Brahe corpus, in which random vectors and NLM are tied with the best performance considering all tokens. Therefore, we decided to evaluate if the accuracy differences in this corpus were statistically significant using the method described below.

The base rationale was to shuffle the outputs of the two models being evaluated and estimate how likely it would be for two random partitions (both containing all sentences in the test data) to have an accuracy difference greater than or equal to the one we observed in the actual results. More specifically, we consider the null hypothesis that both tagger results came from the same distribution and estimate the probability p (or significance level) that it would generate the actual accuracy difference.

For each sentence in the test corpus, we pick its corresponding tag sequences y_1^* and y_2^*, produced by both taggers. We assign y_1^* to either a set S_1 or S_2 with the same probability and y_2^* to the other. We repeat this process R times and count the number r of times in which the accuracy difference between S_1 and S_2 was greater than or equal to the difference in the actual results. We then interpret $\frac{r+1}{R+1}$ as p. In our experiments, we used $R = 1,000$.

Using this method, we found that the difference between NLM and random vectors is not statistically significant in any of the four configurations (we found $p > 0.3$ in all of them). However, in the first two configurations (no extra attributes and using only capitalization), the results using random vectors are significantly better than HAL and SG with $p < 0.01$ and also better than SG in the fourth configuration with the same value.

Table 8 Tagging accuracy on Tycho Brahe

Input features	Only words		Capitalization		Prefix + suffix		All three	
	All (%)	OOV (%)	All (%)	OOV (%)	All (%)	OOV (%)	All (%)	OOV (%)
Random	93.59	37.98	94.65	42.73	95.89	78.41	96.93	82.61
HAL	93.25	44.95	94.39	49.28	95.83	78.97	96.89	83.02
NLM	93.58	52.21	94.67	55.67	95.86	80.91	96.91	84.14
SG	93.46	50.57	94.51	53.82	95.87	80.10	96.83	83.10

Table 9 Comparison values for Mac-Morpho v1

Model	All (%)	OOV (%)
OpenNLP MaxEnt	96.11	89.04
OpenNLP Perceptron	95.31	88.79
ESL	97.12	—
CharWNN [34]	**97.47**	92.49
WNN, caps + suffixes [34]	97.42	**92.64**

Best values shown in bold.

We do not have any conclusive explanation for the better performance of random vectors in this corpus. A possibility is that the word embedding models are biased towards the modern usage of the Portuguese language and it affects negatively their performance on historical texts, but this is very hard to measure. Still, all of them had better performance on OOV tokens than random vectors.

Considering all corpora, results indicate that the more explicit information that is supplied to the tagger, the lower is the performance gap between different models. This makes sense, since having more input data, the importance of word vectors themselves is relatively smaller.

Apparently, the addition of all three attributes reaches a threshold where there is no advantage in using HAL vectors over random ones for OOV words. This does not mean that HAL vectors are useless for this task: overall accuracy using them is always higher than with random vectors in Mac-Morpho and so is OOV accuracy without all three attributes.

Capitalization, prefix, and suffix are beneficial to all models. This is on par with most works in POS tagging for European languages. We note that their benefit is much more evident in OOV tokens, as they help the tagger generalizes beyond the vector space (or, in the case of random vectors, to have any generalization at all).

For comparison, we present results obtained by other tagging systems. For Mac-Morpho v1 and v2, and the Tycho Brahe corpus, we present results available in the literature. We also add comparisons on Mac-Morpho v3 with the ESL-based tagger described in [36][b]. In all corpora, we also trained models from the Apache OpenNLP toolkit [37], which implement maximum entropy- and perceptron-based classifiers.

Table 10 Comparison values for Mac-Morpho v2

Model	All (%)	OOV (%)
OpenNLP MaxEnt	95.92	90.17
OpenNLP Perceptron	95.44	90.03
CharWNN [34]	**97.31**	**93.43**
WNN, Caps + suffixes [34]	97.24	92.64

Best values shown in bold.

Table 11 Comparison values for Mac-Morpho v3

Model	All (%)	OOV (%)
OpenNLP MaxEnt	95.59	88.19
OpenNLP Perceptron	94.24	86.16
ESL	**96.96**	**91.50**

Best values shown in bold.

The comparisons include results reported in [34] with a neural network system similar to ours. Apart from minor parameter differences, they only used word vectors obtained with the skip-gram model over a corpus smaller than ours. Their WNN model is essentially the same as the one used here, and the CharWNN uses a deep architecture to analyze all characters in a word instead of fixed length suffixes.

Tables 9, 10, 11 and 12 show comparison values obtained with each model. The maximum entropy algorithm consistently achieves the best performance among the ones in OpenNLP, although it still falls with a considerable gap behind all our models that use the three extra attributes. The ESL-based tagger achieves a high accuracy but still lower than ours.

The CharWNN overall performance is very close to our best model: the difference is 0.1% absolute in Mac-Morpho v1 and 0.17% in v2. In OOV performance, the gap is slightly larger: 0.74% compared to their best performing WNN model in v1 and 0.91% in v2. Even comparing with our skip-gram model, there is still an advantage. This is probably due to the fact that we use a bigger corpus for word vector induction.

In the Tycho Brahe corpus, the CharWNN has the best performance. The finer granularity of the tagset plays an important role, and the convolutional model seems to adapt better to it. The WNN network, on the other hand, performs worse than our best models. This is probably because we employ larger suffixes and also prefixes. In general, all models (including ours) perform visibly worse on OOV tokens from the Tycho Brahe in comparison with other corpora.

We see a little performance drop in all taggers from v1 to v2 and from v2 to v3, indicating that later versions are harder to work with (even more so because the ESL tagger version used with Mac-Morpho v2 is an improvement

Table 12 Comparison values for Tycho Brahe

Model	All (%)	OOV (%)
OpenNLP MaxEnt	95.89	75.49
OpenNLP Perceptron	94.45	76.80
CharWNN [34]	**97.17**	**86.58**
WNN, caps + suffixes [34]	96.79	80.61

Best values shown in bold.

Table 13 Per-tag F^1 score on versions 2 and 3 of Mac-Morpho

Tag	v2 F^1 (%)	v3 F^1 (%)	Tag	v2 F^1 (%)	v3 F^1 (%)
ADJ	94.93	94.40	PREP+ADV	92.31	91.80
ADV	91.97	91.39	PREP+ART	99.25	99.15
ADV-KS	60.71 (77.09)	78.42	PREP+PROADJ	99.10	98.72
ADV-KS-REL	84.44	—	**PREP+PRO-KS**	0 (82.61)	92.44
ART	98.76	98.76	**PREP+PRO-KS-REL**	100.00	—
CUR	99.58	99.83	PREP+PROPESS	99.07	93.83
IN	71.43	55.11	PREP+PROSUB	80.29	82.81
KC	97.83	97.92	PROADJ	96.62	96.05
KS	89.86	88.19	**PRO-KS**	74.27 (88.45)	89.76
N	97.54	97.33	**PRO-KS-REL**	91.16	—
NPROP	97.00	96.68	PROPESS	98.37	98.36
NUM	96.76	95.85	PROSUB	88.69	86.07
PCP	96.80	96.29	PU	99.98	99.98
PDEN	88.71	89.24	**V**	98.31 (97.77)	99.11
PREP	98.08	97.83	**VAUX**	94.83	—

Values between parentheses are averaged over merged pairs, weighted by frequency.
Merged pairs are shown in bold.

of the one used in v1). The increased difficulty in Mac-Morpho v2 might be attributed to the new tags created for preposition contractions, which increased the size of the tagset.

Concerning v2 and v3, it is more difficult to answer. Table 13 shows F_1 scores per tag obtained by our best models (NLM vectors with capitalization, suffix, and prefix features) in Mac-Morpho v2 and v3. While there is a performance increase for all tag pairs we merged, confirming that they were difficult distinctions for the tagger, for most others performance is actually worse. A possible explanation is that learning to distinguish those pairs also helped to distinguish other neighboring tags.

Table 14 shows the tokens which our best model tagged wrongly most often, together with the tags they have in the Mac-Morpho v3 test set and the number of times they appear. Previous work [38] has also pointed out that *que* and *a* are the two main sources of errors for POS taggers in Portuguese, since they are frequent words and can be used in different ways: *que* may be a pronoun, relative or not, and a subordinating conjunction; *a* may be an article, preposition, or pronoun. Additionally, in the Mac-Morpho annotation, they may be considered part of a proper name or multiword expressions tagged as adverb or conjunction.

Having all these results in mind, we see that the tagger explored here achieves competitive performance and sets the state-of-the-art for POS tagging with the new version of Mac-Morpho. The benefit of distributed word representations obtained with a neural language model (or even with a skip-gram model) is more evident when it comes to tagging words that did not appear in the supervised training data, but that are part of the model's vocabulary.

Conclusions

In this work, we presented a new version of the Mac-Morpho corpus, with error corrections, a new train/test split and some tag merges. We used this new resource to evaluate the performance of a neural network-based

Table 14 Most common wrongly tagged tokens and the tags they have in Mac-Morpho v3 test data

Token	Tags	Times
que	ADV, PDEN, PROSUB, NPROP, ADV-KS, PRO-KS, KS, PROADJ, ADJ	383
a	ADV, KC, PROSUB, NPROP, PROPESS, KS, PROADJ, IN, ART, PREP	161
o	ADV, PROSUB, NPROP, PROPESS, PRO-KS, N, KS, ART	137
como	ADV-KS, ADV, NPROP, KC, KS, IN, PREP	127
de	ADV, PDEN, NPROP, N, KS, PREP	103
um	ADV, PROSUB, ART, N, PROADJ, NUM, NPROP	68
até	ADV, PDEN, KS, PREP	60
uma	ADV, PROSUB, NPROP, N, NUM, IN, ART	59
ao	ADV, PDEN, PREP+PRO-KS, NPROP, PREP+PROSUB, PREP+ART, PREP	59
mais	ADV, KC, PROSUB, NPROP, KS, PROADJ, PREP	52

POS tagger and the impact of different kinds of word embeddings and attributes used as input.

We compared the effect of using word vectors generated with the hyperspace analogue to language, a neural language model, and the skip-gram model. The neural language model is clearly the best suited for the task, being especially useful for providing knowledge about out-of-vocabulary words. SG vectors come a little behind, while HAL vectors do not perform as well as the others.

The impact of observing capitalization, prefixes, and suffixes was also measured. While they were always beneficial in our experiments, a tagger trained with the neural language model has most substantial performance improvements in OOV words.

Our new version of Mac-Morpho is available at http://nilc.icmc.usp.br/macmorpho/, along with previous versions of the corpus. Our tagger is at http://nilc.icmc.sc.usp.br/nlpnet/. We provide two versions of trained model files to be used with our tagger, both obtained with the neural language model and prefix/suffix features, but one with capitalization features and the other without. The latter is expected to be more useful when tagging texts with little consistency in capital letter usage. Our tagger can also be trained in new corpora or in other languages.

Endnotes

[a] Actually, not only words, but rather all tokens, including punctuation, numbers, etc. can have a feature vector. We use the term *word* here because it is commonly found in the literature.

[b] We thank Eraldo Fernandes for training and testing his EBL tagger with our data.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

ERF performed the two revisions of the data, implemented the models and performed experiments. JLGR and SMA supervised experiments. SMA contributed to the second revision of the data. All authors read and approved the final manuscript.

Acknowledgements

This paper is an extension of a previous work presented in STIL 2013. We thank Giuseppe Attardi for helping in improving our neural network implementation; André Cunha and José Eduardo Silva for pointing out errors in the corpus.

Received: 1 May 2014 Accepted: 5 December 2014
Published online: 06 February 2015

References

1. The Penn Treebank Project (2014). http://www.cis.upenn.edu/~treebank/ Accessed April 2014
2. Fonseca ER, Rosa JLG (2013) Mac-Morpho revisited: towards robust part-of-speech. In: Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology – STIL, Fortaleza, Brazil. pp 98–107
3. Aluísio S, Pelizzoni J, Marchi A. R, de Oliveira L, Manenti R, Marquiafável V (2003) An account of the challenge of tagging a reference corpus for brazilian portuguese. In: Proceedings of the 6th International Conference on Computational Processing of the Portuguese Language. PROPOR'03, Springer, Berlin, Heidelberg. pp 110–117
4. Bick E (2000) The parsing system PALAVRAS: automatic grammatical analysis of Portuguese in a constraint grammar framework. PhD thesis. Department of Linguistics – Aarhus University
5. dos Santos CN, Milidiú RL, Rentería RP (2008) Proceedings of International Conference on Computational Processing of Portuguese (PROPOR 2008), vol. 5190. In: Teixeira A, Lima VL, Oliveira LC, Quaresma P (eds). Springer, Berlin Heidelberg. pp 143–152
6. Kepler FN, Finger M (2006) Advances in Artificial Intelligence - IBERAMIA-SBIA 2006. In: Sichman JS, Coelho H, Rezende SO (eds). Springer, Berlin Heidelberg. pp 482–491
7. Maia MRdH, Xexéo GB (2011) Part-of-speech tagging of Portuguese using hidden Markov models with character language model emissions. In: Proceedings of the 8th Brazilian Symposium in Information and Human Language Technology, Cuiabà, Brazil. pp 159–163
8. dos Santos CN, Zadrozny B (2014) Learning character-level representations for part-of-speech tagging. In: Proceedings of the 31st International Conference on Machine Learning, Beijing, China. pp 1818–1826
9. Turney PD, Pantel P (2010) From frequency to meaning: vector space models of semantics. J Artif Intell Res 37:141–188
10. Bengio Y, Ducharme R, Vincent P, Jauvin C (2003) A neural probabilistic language model. J Mach Learn Res 3:1137–1155
11. Collobert R (2011) Deep learning for efficient discriminative parsing. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, vol. 15, Ft. Lauderdale, USA. pp 224–232
12. Socher R, Perelygin A, Wu J, Chuang J, Manning C, Ng A, Potts C (2013) Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2013), Seattle, USA
13. Socher R, Huval B, Manning CD, Ng AY (2012) Semantic compositionality through recursive matrix-vector spaces. In: Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Stroudsburg, PA, USA. pp 1201–1211
14. Huang EH, Socher R, Manning CD, Ng AY (2012) Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Stroudsburg, PA, USA. pp 873–882
15. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. CoRR
16. Namiuti C (2004) O Corpus Anotado do Português Histórico: um Avanço para as Pesquisas em Lingüística Histórica do Português. Revista Virtual de Estudos da Linguagem 2:1–9
17. Afonso S, Bick E, Haber R, Santos D (2002) Floresta sintá(c)tica: a treebank for Portuguese. In: Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002). ELRA, Paris. pp 1698–1703
18. Duran MS, Aluísio SM (2010) Verbos auxiliares no português do Brasil. Technical Report NILC-10-05, University of São Paulo
19. Collobert R, Weston J (2008) A unified architecture for natural language processing: deep neural networks with multitask learning. In: International Conference on Machine Learning, ICML. Helsinki, Finnland. pp 160–167
20. Haykin S (1998) Neural networks: a comprehensive foundation. 2nd edn. Prentice Hall PTR, Upper Saddle River, NJ, USA
21. Viterbi AJ (1967) Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans Inf Theory 13:260–269
22. Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa PP (2011) Natural language processing (almost) from scratch. J Mach Learn Res 12:2493–2537
23. Fonseca ER, Rosa JLG (2013) A two-step convolutional neural network approach for semantic role labeling. In: IJCNN 2013 – International Joint Conference on Neural Networks. IEEE, Dallas, USA. pp 2955–2961
24. Luong M-t, Socher R, Manning CD (2013) Better word representations with recursive neural networks for morphology. In: Proceedings of the Seventeenth Conference on Computational Natural Language Learning (CoNLL), Sofia, Bulgaria. pp 104–113
25. Turian J, Ratinov L, Bengio Y (2010) Word representations: a simple and general method for semi-supervised learning. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Stroudsburg, USA. pp 384–394

26. Lund K, Burgess C (1996) Producing high-dimensional semantic spaces from lexical co-occurrence. Behav Res Methods Instruments Comput 28(2):203–208
27. Huang F, Yates A (2009) Distributional representations for handling sparsity in supervised sequence-labeling. In: Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP. Association for Computational Linguistics, Stroudsburg, USA. pp 495–503
28. Semantic Vectors. https://code.google.com/p/semanticvectors. Accessed April 2014
29. Widdows D, Ferraro K (2008) Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08), Marrakech, Morocco. pp 1183–1190
30. word2embeddings. https://bitbucket.org/aboSamoor/word2embeddings. Accessed April 2014
31. Gensim. http://radimrehurek.com/gensim/. Accessed April 2014
32. Řehůřek R, Sojka P (2010) Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. ELRA, University of Malta, Valletta, Malta. pp 45–50
33. Bruckschen M, Muniz F, Souza JGC, Fuchs JT, Infante K, Muniz M, Gonçalves PN, Vieira R, Aluísio SM (2008) Anotação Lingüística em XML do Corpus PLN-BR. Technical Report NILC-TR-09-08, University of São Paulo
34. dos Santos CN, Zadrozny B (2014) Training state-of-the-art Portuguese POS taggers without handcrafted features. In: Proceedings of the International Conference on Computational Processing of Portuguese (PROPOR 2014). Springer International Publishing. pp 82–93
35. Duran M, Avanço LV, Aluísio SM, Pardo TAS, Nunes MGV (2014) Some issues on the normalization of a corpus of products reviews in Portuguese. In: Proceedings of the 9th Web as Corpus Workshop (WaC-9) @ EACL 2014. Association for Computational Linguistics, Stroudsburg, USA. pp 22–28
36. Fernandes ELR (2012) Entropy guided feature generation for structure learning. PhD thesis. Pontifícia Universidade Católica do Rio de Janeiro
37. Apache OpenNLP. http://opennlp.apache.org/. Accessed April 2014
38. Kepler FN, Finger M (2010) Variable-length Markov models and ambiguous words in Portuguese. In: Proceedings of the NAACL HLT 2010 Young Investigators Workshop on Computational Approaches to Languages of the Americas. Association for Computational Linguistics, Stroudsburg, USA. pp 15–23