Morphosyntactic Label Disambiguation

Bernardo Casaleiro
bernardo.casaleiro@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2018

Abstract

Recently, several models based on recurrent neural networks have been proposed for parts-of-speech tagging and achieving state-of-the-art performance. This paper presents a variety of recurrent neural network models that use a combination of bi-directional Long Short-Term Memory and Conditional Random Fields to disambiguate morphosyntactic labels. Focusing mainly on the Portuguese language the final goal is to try to surpass the current statistical morphosyntactic labels disambiguator for STRING, a hybrid statistical and rule-based Natural Language Processing chain for Portuguese, and integrate the best model into STRING.

Keywords: Morphosyntactic Label Disambiguation, Natural Language Processing, Deep Learning, Restricted CRF

1. Introduction

Part-of-Speech (POS) tagging is a fundamental task in Natural Language Processing (NLP). It provides insights into the role of each word in a sentence, among other contextual information. The more information a POS tagger can infer about the sentence structure and each word morphology, the better.

Current state-of-the-art POS taggers \cite{11, 14} use deep learning methods to predict one single label for each word in a sentence. This label usually represents the word role in the sentence. For instance, \cite{14} evaluate their model on the Universal Dependencies (UD) dataset and also report accuracies on the Wall Street Journal (WSJ) portion of the Penn Treebank, which is also used by \cite{11}. While the UD tagset contains 17 different labels that categorize each word as for its role in the sentence, the WSJ tagset tries to complement this with information about the word morphology resulting in a single set of 45 labels.

This paper presents a different approach to this problem. It proposes three models based on Long-Short Term Memory Networks (LSTM) and Conditional Random Fields (CRF), that in conjunction with word-embeddings and hand-crafted rules aim to disambiguate 11 different morphosyntactic labels for each word. These labels are divided into the Category of the word, that in turn can have a Subcategory, and its inflection features, such as Mood, Tense, Person, Number, Gender, Degree, Case, Syntactic and Semantic.

The proposed models were trained and evaluated for the Portuguese language, with the final goal of trying to surpass MARv’s results. MARv \cite{MARv4} is the statistical disambiguator module of STRING, a hybrid statistical and rule-based NLP chain for Portuguese.

The remaining structure of this paper is as follows. Section 2 briefly describes previous work on machine learning methods to address sequence labeling. Section 3 provides a detailed description of the models’ architecture and implementation. Section 4 details implementation methodologies and experiment hyperparameters. Section 5 presents and analyses the results. Finally, Section 6 concludes and presents some ideas for future work.

2. Related Work

POS tagging has been a classic NLP task. It has drawn research attention for a few decades. Previously, most POS tagging models were linear statistical models like Hidden Markov Models (HMM) and CRFs. However, recently, deep learning methods based on Recurrent Neural Networks (RNN), have been proposed and have excelled in this task, pushing the state-of-the-art forward.

The simpler models include LSTM and Bi-LSTM networks \cite{7}. As well as Bi-LSTM networks with an auxiliar loss \cite{14}. In this, the model jointly predicts the label for each word and the log frequency of the next word. This auxiliar loss has been proved to help increase model performance on rare and out-of-vocabulary (OOV) words. The use of sub-word token embeddings also proved to lead to better re-

\footnote{http://string.12f.inesc-id.pt//index.php/MARv4}
sults, namely the combination of word and character embeddings [2] [14].

Some architectures combine these models with a final CRF layer [7] [11]. This enhancement proves to improve the model performance and makes it more robust and less dependent on the word embeddings.

Finally, some architectures train their character embeddings during the training of the model instead of using pre-trained embeddings [10] [11]. Creating task-sensitive character representations, that extract morphological information, like the prefix and suffix of the word. These embeddings can be computed using CNN’s [11] or LSTMs [13].

3. Neural Network Architecture

This section introduces the neural network layers used in the presented models, the word representations used as input for the model training and evaluation and, the CRF restrictions that enforce compatible predictions. Lastly, it describes in detail the presented model architectures.

3.1. CNN for Character-Level Representations

Convolutional Neural Networks (CNN) [2] [11] are a particular type of neural network composed of a sequence of layers that make the explicit assumption that the inputs are images. Unlike a regular neural network, the layers of a CNN have neurons arranged in 3 dimensions. The neurons in a layer connect only to small local regions, called the filters, of the layer before it. Each filter computes the dot product between their weights and a small region of the layer before it. A Max Pooling layer connects to them in the input. A Max Pooling layer computes the final character representation using the output from each filter.

3.2. Bi-directional LSTM

3.2.1 LSTM Unit

RNNs [5] are a type of neural network architecture with loops that allow information from one step to pass to the next. Although RNNs can capture recent past information, they fail to capture long-distance dependencies due to vanishing gradient problems.

LSTM networks [6] are a particular kind of RNN designed to avoid the vanishing gradient problem. While LSTMs keep the core principles of the RNNs, they have three gates instead of only one (i.e., forget gate, update gate and output gate) interacting together to preserve long-term dependencies. The gates regulate the portions of information to add, keep or forget before moving to the next unit. Formally, the formulas to update the LSTM unit at time $t$ are:

$$ f_t = \sigma(W_f \cdot [h_{t-1}, y_t] + b_f) \quad (1) $$
$$ i_t = \sigma(W_i \cdot [h_{t-1}, y_t] + b_i) \quad (2) $$

$$ o_t = \sigma(W_o \cdot [h_{t-1}, y_t] + b_o) \quad (3) $$

$$ \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, y_t] + b_c) \quad (4) $$

$$ C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \quad (5) $$

$$ h_t = o_t \times \tanh(C_t) \quad (6) $$

where $\sigma$ is the element-wise sigmoid function, $y_t$ is the input vector and $h_{t-1}$ is the output from the last cell. $W_f, W_i, W_c$ and $W_o$ are the weight matrices of the different gates and $b_f, b_o, b_c$ and $b_o$ denote the bias vectors.

3.2.2 Bi-LSTM

When processing sequences of data, such as POS tagging, it is usually beneficial to consider both past and future information. A simple solution is to use bi-directional LSTMs. The basic idea is to present each sequence to two independent LSTMs that are trained simultaneously in a forward and backward direction, to capture both past and future context, respectively. The two LSTM outputs are concatenated to form the final output.

3.3. Conditional Random Fields

A CRF [9] [10] is a probabilistic model for structured, sequential prediction. It encodes known relationships between observations by predicting a vector $x$ of random variables given an observed vector $y$.

CRFs model the conditional probability $P(x|y)$ using a feature map $\Phi(y_1, ... y_n; x_1, ... x_n)$ responsible
for mapping the entire observed sequence \( y \) to an entire sequence \( x \). The score of each feature \( \Phi(y, x) \) can be computed by assigning a weight vector \( w \) to it:

\[
\text{score}(y, x) = w \cdot \Phi(y, x)
\] (7)

The conditional probability \( P(x|y) \) can then be estimated by exponentiating and normalizing the score function for feature \( \Phi(y, x) \):

\[
P(x|y; W) = \frac{e^{w \cdot \Phi(y, x)}}{\sum_{x'} e^{w \cdot \Phi(y, x')}}
\] (8)

In the previous equation, \( \gamma \) defines all possible label sequences and \( w \) is the weight vector the corresponding feature. The model is trained using maximum likelihood estimation. For a given pair \((y_i, x_i)\), the model updates the weight \( w_i \) in order to maximize:

\[
L(w) = \sum_i \log P(x_i|y_i; W)
\] (9)

The decoding corresponds, thus, to searching the label sequence with the highest conditional probability:

\[
x^* = \arg\max_x P(x|y; W)
\] (10)

The naive way would be to calculate \( P(x|y; W) \) for every possible labeling sequences. However, this can be solved efficiently by using the Viterbi algorithm [18].

3.4. Word Representation

The input given to the neural network has a strong influence on the network capacity to extrapolate relations and extract information. For this reason and to evaluate the impact of using sub-token embeddings, the neural network was tested with only word embeddings and with a combination of word, character and case embeddings. In the later, the final word representation is obtained by concatenating of the three embeddings.

3.4.1 Word Embeddings

In the experiments, all architectures use the FastText\textsuperscript{2} pre-trained embeddings for Portuguese. FastText provides high-quality word embeddings that demonstrate good performance when generalizing OOV words. The embeddings were trained on Wikipedia\textsuperscript{3} corpus using a variant of the skip-gram model that represents words as a bag of n-grams and accounts for word order [1, 12] instead of projecting every word to the same position. Each character n-gram has an embedding representation. The word representation is then the sum of the representations of its n-gram constituents.

3.4.2 Character Embeddings

For the character embeddings two different approaches were explored, the use of a CNN and the use of a Bi-LSTM network. Both of these methods have been shown to be an effective way to extract morphological information. In both cases, the character embeddings were randomly initialized and updated during the model training.

3.4.3 Case Embeddings

Some labels have a characteristic word morphology, e.g., Numerals. Case embeddings provide a distinctive representation for these labels by assigning each word with a one-hot embedding, which consists of a 7 dimensional vector where each position represents a class, such that the desired class index is 1 and the remaining values 0. This embedding represents the word distinct morphology by assigning it to one of the classes: capital first letter, all capital letters, all lower letters, numeric value, mainly numeric, contains a digit, and other.

3.5. Restricted CRF

One of the problems when predicting more than one label independently for each word is consistency.

\[\text{http://fasttext.cc}\]

\[\text{http://www.wikipedia.org/}\]

3
Most of the times the neural network tends to predict label combinations that are not compliant with the grammar rules, e.g., predicting the Category Noun and Subcategory Definite or predicting the label Mood when Noun has no Mood inflection feature. Table 1 describes the possible relationships between the labels.

Two different strategies were implemented to enforce structure into the predictions, i.e., the Simple and Kernel methods. Although each approach has different advantages, both have a cornerstone in common. First, the models predict the Category label and then, assuming this prediction is correct, they predict the remaining labels, which are congruent with the Category prediction.

The first approach and the simpler one is the naive approach. Instead of using the Viterbi algorithm this approach computes the marginal probabilities. It then applies a filter to the output of the algorithm by multiplying the entries that are possible by one and the remaining by zero, invalidating them. Although it enforces the grammar rules, this method has no impact on the neural network weights training as it only affects the output. Thus, not improving the model understanding of the grammar rules for future occurrences.

The second method, the Kernel restriction, utilizes the Viterbi algorithm. It employs a filter to the $\psi$ table, effectively blocking any transition to impossible states, by multiplying every non-possible element by the ”infinite” constant provided by the Python library numpy, and the remaining elements by one. The reason for this is because the implemented CRF layer minimizes the score function instead of maximizing it.

Finally, the Rules restriction employs the same method as the Kernel restriction with the exception that on top of the grammar rules it uses information from the STRING modules before Marv, i.e., LexMan and RuDriCo2.

The LexMan module performs a text segmentation and assigns all possible POS tags and any other relevant morphosyntactic features to each word, immediately restricting the number of tags to take into account for each word. In turn, RuDriCo2 applies hand-crafted rules to the output from LexMan, applying two types of rules: segmentation rules that can expand words or contract them into one, and disambiguation rules that assign the most probable POS tag by considering the surrounding context.

The final filter is obtained by multiplying the grammar rules filter (Table 1) with the output from RuDriCo2 and is employed like in the Kernel restriction.

\[\text{http://string.12f.inesc-id.pt/w/index.php/LexMan}\]
\[\text{http://string.12f.inesc-id.pt/w/index.php/RuDriCo2}\]
3.6. Bi-LSTM-CRF

Finally, there are four presented models named after the restriction employed in their CRF layers: the Baseline, the Simple restricted, the Kernel restricted and the Rules restricted.

The models’ input is the representation of each word. The representation can have one of two configurations. It can be solely the word embedding, provided by FastText (i.e., $\vec{w}$) or it can complement the word embedding with character and case embeddings. The character embeddings creation can use either a CNN architecture (i.e., $\vec{w} + \vec{c} + c$) (Figure 4) or a Bi-LSTM architecture (i.e., $\vec{w} + \vec{c}$) (Figure 3). In both cases, the character embeddings are randomly initialized from uniform samples from $[-0.5, 0.5]$ and updated during the model training. In particular, when using the CNN architecture before the character embeddings are input to the CNN, a dropout layer is applied. The final word representation is obtained by concatenating the three embeddings. Lastly, a dropout layer is employed before each word representation is fed into the Bi-LSTM network.

All presented models utilize a Bi-LSTM network at its core. This layer encodes the sentence past and future context, to feed to the CRF layers that jointly decode the best labels sequence. The Bi-LSTM has 50 units with recurrent dropout. Once again a dropout layer is applied before the output from the Bi-LSTM is fed to the CRF layers.

Although all models share the same architecture until this point, the final component is the distinguishing one between the four. The Baseline model (Figure ??) last component consists of 11 CRFs, each responsible for predicting one label. The Simple, Kernel and Rules restricted models (Figure ??) receive their names from the restriction approach implemented in their CRFs. They first predict the Category label using a CRF and then feed this prediction alongside with the Bi-LSTM output to the remaining 10 CRFs, in order to restrict the predictions based on the predicted category. Moreover, the Rules restricted model have one additional input layer for each label to receive the restrictions from RuDriCo2.

Lastly, it was tested the use of an auxiliary loss. The models jointly predict the 11 labels and the logarithmic frequency of the current word as estimated from the training data. The intuition is that the auxiliary loss helps to differentiate the representations of rare words and common words.

4. Experiments

All models were implemented using the Python deep learning library Keras with the same hyperparameters in all models, i.e., Adam AMSGrad variant optimizer[8, 15] with default learning rate (0.001), batch size 100, 100 epochs, 300 dimensions word embeddings, provided by FastText, 50 dimensions character embeddings. Throughout training and fine-tuning of the models, a fixed seed was used to ensure the results comparability.

To mitigate overfitting, the models have several Dropout layers. During the character embeddings generation, before the embeddings are fed into the CNN a 0.1 dropout rate is applied. A dropout rate of 0.25 and 0.5 are, respectively, applied to the input and output vectors of the core Bi-LSTM.

4.1. Dataset

The proposed models were trained and evaluated on a segment of the LE-PAROLE manually annotated with STRING tagset. It has a total of 13205 sentences, summing up to a total of 293737 tokens. 

### Table 1: Labels considered based on the Category label

<table>
<thead>
<tr>
<th>CAT</th>
<th>SCT</th>
<th>MOD</th>
<th>TEN</th>
<th>PER</th>
<th>NUM</th>
<th>GEN</th>
<th>DEG</th>
<th>CAS</th>
<th>SYN</th>
<th>SEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>.</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Verb</td>
<td>.</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>.</td>
<td>10</td>
<td>.</td>
<td>11</td>
</tr>
<tr>
<td>Adjective</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>.</td>
<td>10</td>
<td>.</td>
</tr>
<tr>
<td>Pronoun</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>.</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Article</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>6</td>
<td>7</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>11</td>
</tr>
<tr>
<td>Adverb</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>8</td>
<td>.</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Preposition</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>6</td>
<td>7</td>
<td>.</td>
<td>.</td>
<td>10</td>
</tr>
<tr>
<td>Conjunction</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>11</td>
</tr>
<tr>
<td>Numerical</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>6</td>
<td>7</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>11</td>
</tr>
<tr>
<td>Punctuation</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>Symbol</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Tags are considered based on the Category label are marked with its index. The remaining labels are marked with a ‘.’.
5. Results

I started by experimenting to understand the effectiveness of different component of the neural network architectures. I experimented with different word embeddings (i.e., Word2Vec, FastText), different models’ core RNNs (i.e., RNNs, LSTMs, Gated Recurrent Units (GRU)), and the use of dense layers instead of the CRF layers. From this experiments, FastText vastly outperformed Word2Vec pre-trained embeddings, notably for OOV words. LSTM and GRU networks also exceeded the pure RNN performance, and although the aftermath was close between the two, the LSTM provided better results. Lastly, according to expectations, the CRF layers vastly surpassed dense layers performance, leading to the current neural network architectures.

The focus then shifted towards improved word representations, by using character and case embeddings, and different CRF restriction methods. These changes proved to be more impactful in the model performance, and thus, are the locus of the presented results.

5.1. Main results

Overall all models show good performance, showing that a single neural network can successfully encode more than one label for POS tagging, extending the information obtained from the POS tagger. Table 2 presents the accuracies obtained for the four presented models with different input word representations using the auxiliary loss LOGFREQ. Tables 3 to 13 displays the label specific F1-scores obtained with different word representations.

Analyzing Table 2 it is immediately evident the benefits of using sub-token word representations. Even though FastText embeddings provide high-quality pre-trained embeddings, it falls short when training a single model to predict several labels, especially for a morphosyntactic rich language like Portuguese. It is also clear that the use of character embeddings drastically reinforces the model capability to predict the labels associated with cer-
from Tables 3 to 13, it becomes clear that this model performs close to the more advanced models when looking at the label specific metrics. The two strategies used for the creation of the character embeddings show similar performance, but ultimately, the Bi-LSTM architecture, i.e., \( \vec{w} + \vec{c} + c \), representation performs better.

Although at first glance, it may seem that the Baseline model performs slightly worse, the F1-Scores reported in Tables 3 to 13 reflect the fact that some labels occur more frequently than their counterparts. As the model is biased due to the fact that some labels occur more frequently than their counterparts.

Even though the models suffer penalties when predicting the most common label, this leads to higher accuracy and recall but lower precision. This reflects on the F1-Scores reported in Tables 3 to 13. The restriction strategies, on the other hand, provide better results that are coherent with the grammar rules and show a more diverse range of predictions.

<table>
<thead>
<tr>
<th>Model</th>
<th>CAT</th>
<th>SCT</th>
<th>MOD</th>
<th>TEN</th>
<th>PER</th>
<th>NUM</th>
<th>GEN</th>
<th>DEG</th>
<th>CAS</th>
<th>SYN</th>
<th>SEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>( \vec{w} )</td>
<td>16.23</td>
<td>42.43</td>
<td>80.31</td>
<td>78.16</td>
<td>71.91</td>
<td>33.45</td>
<td>32.67</td>
<td>48.18</td>
<td>92.98</td>
<td>62.21</td>
</tr>
<tr>
<td>( \vec{w} + \vec{c} + c )</td>
<td>82.77</td>
<td>72.48</td>
<td>93.72</td>
<td>91.94</td>
<td>88.82</td>
<td>76.46</td>
<td>74.69</td>
<td>93.32</td>
<td>94.97</td>
<td>95.94</td>
<td>99.20</td>
</tr>
<tr>
<td>( \vec{w} + \vec{c}_i + c )</td>
<td>73.74</td>
<td>68.86</td>
<td>89.58</td>
<td>85.85</td>
<td>83.48</td>
<td>70.32</td>
<td>69.47</td>
<td>88.59</td>
<td>94.10</td>
<td>92.85</td>
<td>99.10</td>
</tr>
<tr>
<td>Simple Restricted</td>
<td>( \vec{w} )</td>
<td>12.71</td>
<td>40.34</td>
<td>70.62</td>
<td>70.41</td>
<td>65.92</td>
<td>31.27</td>
<td>38.89</td>
<td>46.73</td>
<td>92.80</td>
<td>59.95</td>
</tr>
<tr>
<td>( \vec{w} + \vec{c} + c )</td>
<td>77.78</td>
<td>76.83</td>
<td>90.78</td>
<td>89.81</td>
<td>85.91</td>
<td>80.46</td>
<td>74.91</td>
<td>89.91</td>
<td>94.31</td>
<td>84.01</td>
<td>99.26</td>
</tr>
<tr>
<td>( \vec{w} + \vec{c}_i + c )</td>
<td>61.62</td>
<td>68.86</td>
<td>86.71</td>
<td>85.96</td>
<td>72.44</td>
<td>77.48</td>
<td>69.57</td>
<td>86.08</td>
<td>84.30</td>
<td>71.62</td>
<td>99.10</td>
</tr>
<tr>
<td>Kernel Restricted</td>
<td>( \vec{w} )</td>
<td>12.61</td>
<td>37.42</td>
<td>71.08</td>
<td>71.19</td>
<td>65.97</td>
<td>29.66</td>
<td>35.07</td>
<td>51.01</td>
<td>92.70</td>
<td>61.73</td>
</tr>
<tr>
<td>( \vec{w} + \vec{c} + c )</td>
<td>80.09</td>
<td>78.25</td>
<td>92.47</td>
<td>91.49</td>
<td>88.84</td>
<td>77.09</td>
<td>75.81</td>
<td>72.46</td>
<td>96.13</td>
<td>55.44</td>
<td>99.27</td>
</tr>
<tr>
<td>( \vec{w} + \vec{c}_i + c )</td>
<td>71.35</td>
<td>71.84</td>
<td>88.76</td>
<td>88.12</td>
<td>83.28</td>
<td>76.11</td>
<td>72.03</td>
<td>88.36</td>
<td>93.02</td>
<td>81.93</td>
<td>99.13</td>
</tr>
<tr>
<td>Rules Restricted</td>
<td>( \vec{w} )</td>
<td>88.56</td>
<td>90.23</td>
<td>91.30</td>
<td>91.20</td>
<td>87.86</td>
<td>91.44</td>
<td>92.69</td>
<td>78.69</td>
<td>92.37</td>
<td>71.93</td>
</tr>
<tr>
<td>( \vec{w} + \vec{c} + c )</td>
<td>97.32</td>
<td>93.58</td>
<td>95.88</td>
<td>94.34</td>
<td>90.21</td>
<td>96.38</td>
<td>96.50</td>
<td>93.40</td>
<td>93.90</td>
<td>91.35</td>
<td>99.80</td>
</tr>
<tr>
<td>( \vec{w} + \vec{c}_i + c )</td>
<td>96.67</td>
<td>94.19</td>
<td>95.20</td>
<td>95.23</td>
<td>91.06</td>
<td>96.12</td>
<td>96.35</td>
<td>97.15</td>
<td>92.73</td>
<td>80.27</td>
<td>99.82</td>
</tr>
<tr>
<td>Marv4</td>
<td>97.53</td>
<td>97.16</td>
<td>95.78</td>
<td>99.15</td>
<td>95.16</td>
<td>96.45</td>
<td>99.89</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Table 2: Models' accuracy with LOGFREQ auxiliary loss on all words, ambiguous words and out-of-vocabulary words.

Tain inflection features that have characteristic morphologies such as the Gender, Number and Degree labels. The two strategies used for the creation of the character embeddings show similar performance, but ultimately, the Bi-LSTM architecture, i.e., \( \vec{w} + \vec{c} + c \), representation performs better.
predicting the wrong Category label, both models make more intelligible predictions. Between the two, the Kernel restricted model outperforms the Simple restricted model in most labels except for the Number, Degree and Syntactic labels, presumably due to the dataset unbalance. Since the Kernel restricted model learns patterns, it tends to make predictions based on these, while the Simple Restricted model tends to predict the most common tag from the possible ones after applying the restrictions. The Kernel Restricted model with the $\vec{w} + \vec{c} + c$ representation is thus the more consistent model with an average accuracy of 82.49%, without using any pre-processing data other than the grammar rules table (Table 1).

Lastly, as expected the Rules Restricted model largely exceeds the other models’ performance. It greatly benefits from RuDriCo2 hand-crafted rules, that further restrict the labels to take into account on top of the grammar rules. This model achieves the best results when combined with the $\vec{w} + \vec{c} + c$ word representation, with an average accuracy of 94.79%.

On a label specific note, from analyzing Tables 3 to 13, there are some interesting patterns that become evident. Due to the dataset unbalance between each tag, tags with fewer occurrences are heavily punished. It is also evident that the Verb Tense label highly benefits from the use of the CNN architecture to generate the character-level embeddings. Furthermore, tags that partner with words that are generally created by composition also profit from this architecture. While tags that associate with words that are generally created by derivation benefit the most from the use of the Bi-LSTM architecture for the character-level embeddings, like the Category label tag Adverb. Lastly, the use of case embeddings also proves to improve the model results in tags with specific morphologies, such as the Category label tag Numeral and the Subcategory label tag Cardinal.

Even though the Rules Restricted performs well, considering it only uses a single Bi-LSTM neural network to predict several labels, the model falls slightly short with an average accuracy of 94.89% (on the comparable labels) and do not surpass the current STRING morphosyntactic disambiguator, MARv4, with an average of 97.30%.

5.2. Results on Ambiguous and Out-of-Vocabulary words
The biggest challenge when performing POS tagging is the chance of the same word have more than one possible tag. It is thus, crucial that the model be able to choose the right label. Table 2 presents the models’ accuracy on ambiguous words using the LOGFREQ auxiliary loss with different word representations as input. To test the effectiveness of
the LOGFREQ auxiliary loss experiments were also conducted with and without this enhancement, Table 14 presents the obtained results.

The Baseline model obtains disastrous results always predicting the most common tag in the dataset without any restriction, leading to terrible results. The Simple restricted model outperforms the Kernel Restricted model. Since the Simple Restricted model predicts the most common tag from the possible ones after applying the restrictions, the model ends up correctly predicting the tag more times than the Kernel Restricted model. Finally, the Rules Restricted model is the best performing model, making use of the RuDriCo2 hand-crafted rules.

Once more it is also evident that the use of character-level embeddings improves the models’ performance drastically. Between the two representations, the Bi-LSTM architecture shows the best results consistently, outperforming the CNN architecture by on average 4% for the Rules Restricted model, the best performing model.

Analyzing Table 14 it is evident that the use of the auxiliary log frequency loss LOGFREQ proves to improve the models’ performance on ambiguous words. We can see an increase of approximately 2% consistently across all models’ accuracies.

On the other hand, the models’ capability to generalize for OOV words is also essential. The use of FastText pre-trained word embeddings and character embeddings gave the models a valuable toolkit to infer the right tag for each label for OOV words. Table 15 shows the models’ accuracies on OOV with the LOGFREQ auxiliary loss using different word representations. In order to confirm the effectiveness of the LOGFREQ auxiliary loss on OOV words, Table 15 shows the models’ accuracies
Table 14: Models’ accuracy on ambiguous words using the Bi-LSTM architecture for character embeddings generation.

| LOGFREQ | CAT | SCT | MOD | TEN | PER | NUM | GEN | DEG | CAS | SYN | SEM |
|---------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Baseline | Yes | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| No | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Simple Restricted | Yes | 68.35 | 69.07 | 59.37 | 43.98 | 79.93 | 64.13 | 38.75 | 64.38 | 79.12 | 79.81 | 93.46 |
| No | 65.16 | 63.25 | 55.19 | 40.15 | 75.34 | 61.07 | 32.11 | 63.75 | 75.39 | 69.35 | 87.34 |
| Kernel Restricted | Yes | 65.54 | 64.18 | 58.32 | 52.88 | 86.53 | 57.01 | 48.86 | 51.98 | 86.57 | 34.36 | 94.39 |
| No | 63.31 | 61.32 | 53.35 | 49.32 | 83.45 | 53.31 | 44.56 | 46.33 | 82.88 | 32.47 | 90.83 |
| Rules Restricted | Yes | 93.40 | 83.89 | 53.89 | 54.13 | 68.66 | 83.08 | 85.09 | 55.26 | 71.97 | 84.08 | 92.98 |
| No | 90.65 | 81.93 | 51.16 | 50.20 | 66.66 | 79.30 | 80.23 | 53.73 | 68.98 | 79.79 | 92.98 |

on OOV words with the Bi-LSTM architecture for word representation generation, with and without the use of the auxiliary loss LOGFREQ.

Just like for ambiguous words, the Rules Restricted model is the best performing model. The Baseline model obtains disastrous results always predicting the most common tag in the dataset when a word is not in the training set. The Simple restricted model performs close to the Kernel Restricted model, with a negative difference of approximately 1% in almost every label, except for the Number, Degree and Syntactic labels where it excels. Once again, probably due to the Simple Restricted model predicts the most common tag from the possible ones after applying the restrictions.

6. Conclusions
This dissertation presented several deep learning models that in conjunction with word representations and hand-crafted rules address the POS tagging task with the ambition to disambiguate 11 different morphosyntactic labels using a single neural network. It also explored the restriction of the labels to take into account in the final CRF layer, presenting an approach that successfully blocks transitions to restricted labels during the Viterbi algorithm. The implementation of these restrictions into the CRF layers allowed to complement the deep learning models with hand-crafted rules, significantly improving the model performance. The Rules Restricted model shows good results across all labels by combining the potential of RNN and the reliability of hand-crafted rules.

Furthermore, this work explored the use of different levels of word representations as well as different approaches to generate these representations. Results show that the use of sub-word representations is essential when dealing with a morphosyntactic rich language such as the Portuguese language. During the development of this work, it was tested different architectures for the generation of character-level embeddings. The Bi-LSTM architecture proved to be the most efficient when it is crucial to capture the affix information, working best for words created by derivation, while the CNN architecture proves to be more efficient for words formed by composition. Lastly, the use of case embeddings to reinforce certain morphological features proved to be crucial for labels with characteristic morphologies.

Finally, this dissertation explored the use of an auxiliary loss that predicts the current word logarithmic frequency in the training set to improve the model’s performance on rare and OOV words. Results show that this auxiliary loss effectively improves the model performance not only on OOV words but also on ambiguous words.

6.1. Future Work
For future work, it would be interesting to experiment with different methodologies to combine the word, character and case embeddings. For instance by performing a weighted projection of this embeddings into a final word representation. Another alternative would be to use a neural network layer, such as a dense layer or even a CNN. On the same note, it would be interesting to study the use of an Attention layer to understand better the impact of each embedding on the model’s performance.

The proposed models use Bi-LSTMs, but more complex types of RNNs could be used, such as Multi-Function Recurrent Units (MuFuRUs). MuFuRUs use an intricate gating mechanism that allows for additional differentiable functions as composition operations, leading to models that can better capture nuances involved in encoding sequences.

References
<table>
<thead>
<tr>
<th>LOGFREQ</th>
<th>CAT</th>
<th>SCT</th>
<th>MOD</th>
<th>TEN</th>
<th>PER</th>
<th>NUM</th>
<th>GEN</th>
<th>DEG</th>
<th>CAS</th>
<th>SYN</th>
<th>SEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Yes</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Simple Restricted</td>
<td>Yes</td>
<td>76.63</td>
<td>75.35</td>
<td>90.30</td>
<td>88.21</td>
<td>83.07</td>
<td>78.08</td>
<td>76.22</td>
<td>84.61</td>
<td>99.79</td>
<td>96.60</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>74.16</td>
<td>73.85</td>
<td>88.17</td>
<td>82.10</td>
<td>80.19</td>
<td>77.05</td>
<td>74.11</td>
<td>82.63</td>
<td>99.49</td>
<td>96.35</td>
</tr>
<tr>
<td>Kernel Restricted</td>
<td>Yes</td>
<td>76.29</td>
<td>75.81</td>
<td>91.34</td>
<td>86.48</td>
<td>85.07</td>
<td>72.10</td>
<td>71.23</td>
<td>65.88</td>
<td>99.79</td>
<td>96.19</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>75.91</td>
<td>74.22</td>
<td>89.75</td>
<td>84.32</td>
<td>81.73</td>
<td>69.69</td>
<td>69.27</td>
<td>50.03</td>
<td>99.38</td>
<td>60.37</td>
</tr>
<tr>
<td>Rules Restricted</td>
<td>Yes</td>
<td>93.34</td>
<td>83.39</td>
<td>96.34</td>
<td>93.21</td>
<td>94.32</td>
<td>84.35</td>
<td>82.31</td>
<td>78.89</td>
<td>99.81</td>
<td>99.12</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>91.25</td>
<td>80.20</td>
<td>93.29</td>
<td>89.88</td>
<td>89.94</td>
<td>76.88</td>
<td>78.15</td>
<td>72.95</td>
<td>96.43</td>
<td>93.10</td>
</tr>
</tbody>
</table>

Table 15: Models’ accuracy on out-of-vocabulary words using the Bi-LSTM architecture for character embeddings generation.


