Assignment 1 - POS tagging using neural architecture

Mattia Bertè* **Simone Giordani*** **Omar G. Younis***
{mattia.berte, simone.giordani2, omargallal.younis}@studio.unibo.it

Abstract

In this report we present our solution to the Part-of-Speech tagging problem. We faced and solved the problem using an RNN for processing sequences, using LSTM and GRU cells. The dataset used for training is the public available *Dependency Treebank*, while concerning the embedding we used the pretrained *GloVe* embeddings. After a preprocessing step for handling OOV terms we tested different architectures with different hyperparameters. We kept the two bests architectures and we analyzed their results. All the code and the trained models can be found on GitHub[1].

1 Problem description

POS tagging is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context. Usually it is faced using RNN since given a sequence of words in input it returns the corresponding sequence of tag in output.

2 Dataset analysis

The dataset is composed of 94084 words divided in 45 classes. We have analyzed the distribution of the classes after the splitting in train, validation and test sets to verify that they are similar. We also analyzed the distribution of the sentence length distribution to guide the decision of splitting/truncating lengthy sentences. Finally, we have carefully analyzed the OOV words to understand which processing we might apply to encode them properly. At the end we decide to perform the preprocessing described in the following section.

2.1 Preprocessing

Regarding OOV words, we discovered that the majority of them were due to the following issues:

1. words composed with the slash or dash, such as 'new-home';
2. decimal numbers;
3. typos;

The first one has been solved computing the embeddings as the mean of the embeddings of the two words. Regarding the second, GloVe has just embeddings for a finite set of numbers (mostly integers) so we decided to approximate them to the nearest number in the vocabulary. Typos have been handled using Levenshtein distance to correct them. After that, we also noticed that some OOV words are composed by two subwords like *subskills*. Again, we encode them using the mean between the encoding of the two subwords. To recognize the two subwords, we try to split only in correspondence of a syllable (using the package *NLTK*) and then check if the two words are present in the vocabulary. Unexpectedly, we then discovered that just splitting without the syllable constraint works better. After this preprocessing phase, the number of OOV words decreases from 676 to 7.

During the analysis we also discovered that the max length sentence in the train set was due to a list of sentences separated by semi-columns. We decided to split them to avoid wasting computation. In fact, we need to pad small sequences in the same batch with lengthy ones, resulting in a waste of computational resources and memory.

[1] https://github.com/younik/pos-tagging

3 Model and training description

We started, as baseline, with a bidirectional LSTM layer followed by a fully connected ones for the classification. The fully connected layer processes the output at each time step independently. After that, we tried to make the following changes:

- a GRU layer instead of the LSTM one (less complex than the previous one and easier to train);
- two LSTM layers instead of one;
- two fully connected layers on top, instead of one;

For each architecture, we also tried to add a *Conditional Random Field* module.
We tuned several hyperparameters using *grid search*. In particular, for each architecture, we tuned the hidden dimension, the amount of dropout, the weight decay, and the learning rate. Due to our limited computational capacity, we decide to divide the tuning part into two steps: first, we find the optimal hyperparameters for the hidden dimension and the dropout; then we fixed them and we tuned the learning rate and the weight decay. We use Adam to train the models.
We find that, across architectures, the best combination of hidden dimension and dropout is, respectively, 128 and 0.6. The architectures that perform better are the bidirectional LSTM with two-layer and the architecture with 2 fully-connected layers. Using CRF module results in an accuracy that is, on average, 0.5% better. Regarding learning rate, the default one of Adam performs best while for weight decay, we find 10^{-3} to be a good choice. The final results can be found in table 1.

4 Results and error analysis

After training the two best architectures, we evaluated them. To have a better insight on the real behaviour of the model we used the F1-score as the main evaluation metric. The scores presented in Table 1 don't take in account the punctuation classes (except for the accuracy), however they have been used during the training phases because they bring useful information. After the evaluation steps we analyzed the results. Even if the accuracy of the model is pretty high, the F1-score decreases a lot due to some classes, such as RBS whose numbers of samples is very low and so very difficult to be learned by the models.
To understand better which are the main errors, we plotted the confusion matrix, from it we can see that the majority of errors involves the class NN and JJ, there some words that depending on the context change the class such as *first* and *general*. Other errors can be found between the classes NN and NNP, the noun and plural noun classes and it's clear why sometimes this error happens. Other error can be found between JJ and VBN but they are just a few.

Model	**Macro average**			**Weighted average**			
	Precision	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**	**Accuracy**
2-LSTM-50	85.49%	82.98%	83.31%	90.54%	90.55%	90.44%	91.70%
2-FC-50	86.82%	84.57%	84.82%	90.88%	90.88%	90.77%	91.91%
2-LSTM-100	88.34%	88.86%	88.20%	93.56%	93.57%	93.51%	94.27%
2-FC-100	87.84%	88.68%	87.93%	93.64%	93.70%	93.61%	94.35%
2-LSTM-300	87.30%	87.49%	87.18%	93.09%	93.14%	93.07%	93.97%
2-FC-300	87.94%	87.32%	87.08%	93.08%	93.10%	93.03%	93.97%

Table 1: Results of our best models with different embedding sizes. 2-LSTM indicates the model with 2 layers of bidirectional LSTM and a fully-connected layer on top. 2-FC indicates the model with one bidirectional LSTM and two layers of a fully-connected network. All the models use CRF. The last number indicates the embedding size of *GloVe*.

5 Conclusion

We achieve good performances even if we kept our models simple. As best of our knowledge, state-of-the-art results accuracy is 97.97%[2], but they use a more complex architecture and they exploit multi-tasking that helps in achieving better generalization.

[2]Dat Quoc Nguyen and Karin Verspoor. An improved neural network model for joint. Proceedings of the, 2018.