Assignment 1: Part-Of-Speech Tagging on the Penn Treebank Tagset

Mattia Maranzana, Antonios Pantelis, and **Nalin Sharma**
Master's Degree in Artificial Intelligence, University of Bologna
{mattia.maranzana, antonios.pantelis, nalin.sharma}@studio.unibo.it

Abstract

In this project we are called to address the classification task of Part of Speech (POS) Tagging on the pre-tagged Penn Treebank Tagset. Our goal is to elevate accuracy, leveraging the text pre-processing and NLP model development techniques that we explored in the course and during Tutorials 1 and 2.

1 Introduction

As we mentioned previously, we are addressing the supervised learning task of POS tagging on the Penn Treebank dataset. Let us give a brief summary of our approach.

We encoded the Tagset into a `pandas` dataframes, namely `df` split by sentences. Then we split `df` into training, validation, and test sets, adding an additional "split" column to identify each piece of data with its corresponding dataset.

Then, we performed basic data inspection, visualizations, and text pre-processing, as detailed in section "**??**."

Furthermore, we built the vocabulary of the corpus, along with `idx_to_word` and `word_to_idx` maps, and conducted tokenization.

Next up, we built the embedding matrix and calculated Out of Vocabulary (OOV) words percentage.

Subsequently, we performed some data preparation that included One-Hot Encoding, and size verification.

Later on, we created a variable, `punctuation_indices`, to store punctuation indices for later exclusion during metric evaluation, developed a macro multi-class F_1 score metric and proceeded to create the three models.

Finally, after training our models, we conducted basic error analysis by identifying misclassified tags and their corresponding words to explore potential correlations.

2 System description

For the creation of our models, we used the `Keras`[1] library of `Tensorflow`. As instructed, we developed three models, whose architectures can be seen in Figure **??**.

We worked with GloVe embeddings and after calculating the OOV words percentage we proceeded as follows: For the validation and test sets, we addressed OOV words by replacing them with token "UNK". The subsequent creation of an embedding matrix involved using GloVe embeddings for known words, zero vectors for "UNK", and random vectors for remaining OOV words. Moreover, padding was applied at the beginning of each sequence, and sentences exceeding a length of 600 were truncated.

3 Data

As far as data inspection goes, we calculated the distribution of individual words in sentences and the distribution of POS tags and plotted the 20 *most*[2] and *least frequent*[3] words.

Furthermore, we created a dictionary for the POS labels and performed some minimal text pre-processing on the corpus. In particular, we chose to simply make every word in the corpus lower case, since we could not delete or modify any tokens. This helped greatly with the OOV words percentage. In particular, we obtained 4.98% after lower-casing, while before we had obtained approximately 20%. This made the OOV percentage negligible and thus we decided that no further handling of the OOV words was required.

Moreover, we also plotted two box plots show-

[1]This was precisey the reason why we had to define the metric F_1 ourselves, since the already defined one of `scikit-learn` is incompatible with `Tensorflow`.

[2]Most of the *most frequent* words included punctuation and stop words.

[3]Some of the *least frequent* words included numbers and names.

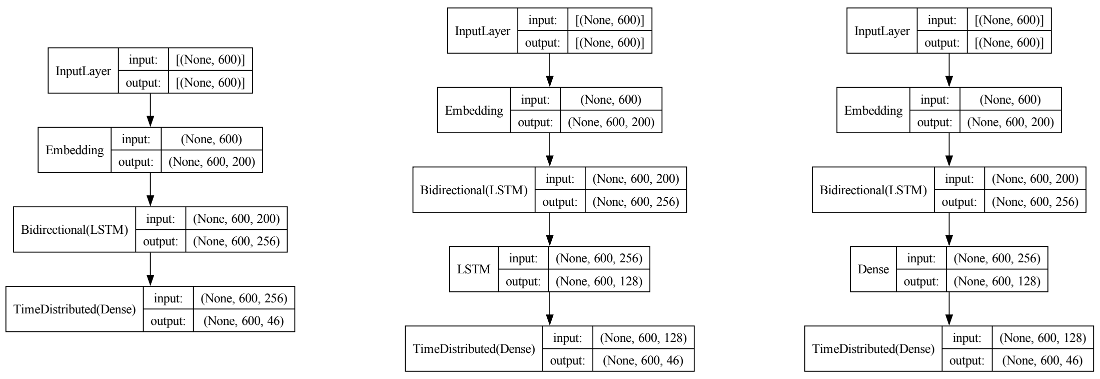
InputLayer | input: | [(None, 600)] | output: | [(None, 600)]
Embedding | input: | (None, 600) | output: | (None, 600, 200)
Bidirectional(LSTM) | input: | (None, 600, 200) | output: | (None, 600, 256)
TimeDistributed(Dense) | input: | (None, 600, 256) | output: | (None, 600, 46)
InputLayer | input: | [(None, 600)] | output: | [(None, 600)]
Embedding | input: | (None, 600) | output: | (None, 600, 200)
Bidirectional(LSTM) | input: | (None, 600, 200) | output: | (None, 600, 256)
LSTM | input: | (None, 600, 256) | output: | (None, 600, 128)
TimeDistributed(Dense) | input: | (None, 600, 128) | output: | (None, 600, 46)
InputLayer | input: | [(None, 600)] | output: | [(None, 600)]
Embedding | input: | (None, 600) | output: | (None, 600, 200)
Bidirectional(LSTM) | input: | (None, 600, 200) | output: | (None, 600, 256)
Dense | input: | (None, 600, 256) | output: | (None, 600, 128)
TimeDistributed(Dense) | input: | (None, 600, 128) | output: | (None, 600, 46)

Figure 1: Architectures of `baseline_model` (left), `model_1` (center), and `model_2` (right).

ing the distribution of sentences' lengths. This helped us make an informed decision about the padding. Since the beginning of the project we had two ideas in mind: either truncate sequences at a fixed padding length with the risk of losing some information, or trying to divide each file in sub-sentences of fixed length and pad them accordingly as not to lose any information at all. However, from the two box plots in question, it was clear that the more convenient one is that of truncating each sentence at length equal to 600. With this choice, we lose only a negligible amount of information out of only one sentence whose length is as one can see from both box plots around twice the padding length.

4 Experimental setup and results

Hyperparameters for the neural POS tagger were tuned using Gridsearch across the following values:

Hyperparameter	**Values**
Embedding Dimensions	50, 100, 200, 300
Batch Sizes	32, 64, 128
LSTM Sizes	32, 64, 128
Learning Rates	0.001, 0.01, 0.1
Trainable embeddings	True, False

After the Gridsearch, we obtained the best-performing hyperparameters as follows: 200 for the embedding dimension, 128 for batch and LSTM size, 0.01 for learning rate and True for trainability of the embedding layer (Results in Appendix A below).

Models were evaluated based on their macro F1 score, reflecting performance across all classes.

5 Discussion

The baseline BiLSTM model coupled with a dense layer, trained on GloVe embeddings produces appreciable performance with respect to models 1 and 2 (Results in Appendix A below). Model 1 shows a noticeable drop in macro f1 score, hinting that the additional complexity does not translate to better representation for this task, potentially interfering with the effective features already provided with GloVe. Model 2 improves on this slightly, suggesting that the layer provides nuanced classification by leveraging its non-linear transformations. However, both models are evidently overfitting with their added complexity.
Regarding error analysis, our most frequently misclassified words were slangs/specialised vocabulary that were in our OOV set. To improve this, we could use Byte Pair Encoding, to allow the model to understand embeddings for unknown words based on known subword units. We can also fine-tune the GloVe embeddings on a domain-specific corpus.

6 Conclusion

Contrary to our expectations, the baseline model remained sufficiently complex for this task, achieving competitive macro f1 scores around 0.7 across all 3 seeds. This speaks to the richness of the GloVe embeddings, and the effectiveness of transfer learning and fine-tuning routine in the defined model.

Nevertheless, static GloVe embeddings do not capture the full contextual boundaries of the data. Using models such as BERT or GPT, we can leverage more context-aware embeddings. The model also misrepresents some compound words, thus better handling of the linguistic variability in the corpora is required for future works.

7 Appendix A

Seed	Model	Loss	Macro F1 Score
47	Baseline	0.69	0.74
	Model 1	0.0198	0.69
	Model 2	0.01	0.72
652	Baseline	0.014	0.68
	Model 1	0.01	0.61
	Model 2	0.01	0.68
72	Baseline	0.01	0.68
	Model 1	0.01	0.63
	Model 2	0.01	0.67

Table 1: Model Performance with Different Seeds