Assignment 1
Recurrent Neural Architectures for POS tagging

Vincenzo Collura, Gianmarco Pappacoda, and **Anthea Silvia Sasdelli**
Master's Degree in Artificial Intelligence, University of Bologna
{ vincenzo.collura2, gianmarco.pappacoda, anthea.sasdelli }@studio.unibo.it

Abstract

Part-Of-Speech (POS) tagging is an important task at the base of more sophisticated NLP tasks. The goal of this brief report is to explore various (Recurrent) Neural architectures to perform POS tagging. The most important distinction with respect to the task at hand is the use of context which is better exploited by neural solutions as opposed to rule-based techniques. One of the most striking observations was the worsened performance of model variants with respect to the baseline model.

1 Introduction

The objective of this assignment is to explore and evaluate (Recurrent) Neural architectures with dense vector representations to perform Part-Of-Speech (POS) Tagging. Both the dataset, architectures and vector representations were provided. The dataset is the Penn Treebank(Marcus et al., 1994) which contains a labeled series of words and their POS tags as well as dependencies between words arranged in phrases, documents. The data was downloaded and transformed into a suitable format for data inspection. The third column, representing dependencies between words, has been dropped as per instructions. Each word has been associated with its document and its phrase by means of two different identifiers. This allowed the authors to perform an analysis based on the split methods. For the full analysis please refer to the operative material.

2 System description

The implemented system can be roughly divided into three main components:

- Pre-processing pipeline
- Word embedding
- Neural architecture

Each subsequent component is dependent on all the previous ones.

The pre-processing pipeline has been purposely kept simple, words have only been converted to lower-case as GloVe(Pennington et al., 2014) only has lower-case mappings. Out-Of-Vocabulary (OOV) elements have been dealt with in a rigorous process, the first vocabulary (V1) built started is based on GloVe, then OOV in the training set are computed, a new vocabulary (V2) used for validation purposes only is then created, hence OOV in the validation set are computed, the resulting vocabulary (V3) is then used for testing purposes. A final vocabulary (V4) is computed from the test set.

The word embedding has been implemented as part of the neural network in the form of a static (non-trainable) layer. The embedding layer depends on the building process of the embedding matrix which in turn depends on the vocabulary. During this step words whose embedding is known are translated, words whose embedding is not known (OOVs) are translated into vectors whose dimensions have a uniform distribution. This is due to the fact a hard requirement was imposed, embeddings couldn't be learned or be dynamically dependent on context.

Finally, the following architectures have been tested, all of them start with the aforementioned embedding layer:

Baseline: a bidirectional LSTM layer followed by a Time-distributed Fully-Connected (Dense) layer with softmax activation.

GRU: the same architecture as baseline, with GRU units instead of LSTM.

TwoLSTM: the same architecture as baseline, with an additional bidirectional LSTM layer on top of the first one.

TwoDense: the same architecture as baseline, with an additional (non-distributed) dense layer at the end, before the softmax activation. In this case the Time-distributed Dense layer has three times the units of the last added layer as to avoid two mirrored layers.

3 Experimental setup and results

A seed has been set for all the involved frameworks as to ensure reproducibility. The dataset was split phrase-wise rather than document-wise as a quick iteration through the baseline model confirmed the authors' intuition of models performing better with phrases contexts rather than document ones. The dataset has been then split into three sets: training, validation, test.

Each neural network has been trained using early stopping, model checkpointing and adaptive learning rate. The chosen optimizer is Nadam (Adam with Nesterov Momentum) as it should provide a faster convergence(Dozat, 2016). The main hyperparameter is the number of units in the GRU/LSTM layer. After several attempts authors deemed 100 to be a suitable amount for all the subsequent experiments. For a comprehensive list of hyperparameters, please refer to the operative material.

The selected metric to evaluate the models is the F1-score (rounded 4 digits) excluding punctuation classes. The results of the experiments are shown below.

	Validation F1	Test F1
baseline	0.6998	0.7810
GRU	0.6797	0.7657
twoLSTM	0.6356	0.6790
twodense	0.6243	0.7284

Notice the test F1 score was calculated for all models, but only the two best models (according to validation F1) should be accounted for.

4 Discussion

The two best models are *baseline* and *GRU*. Surprisingly the number of parameters did not translate into a direct performance improvement, on the contrary the most complex models actually performed worse than the most simple ones. The difference in performance between baseline and GRU is negligible while GRU uses $\approx 23.5\%$ less parameters.

Most of the models errors involve nouns (single/plural, common/proper), adjectives and verbs (especially in past tense/past participle). These errors are relevant and can be addressed through a many different approaches, either based on feature engineering or more neural-oriented.

5 Conclusion

It has been possible to implement and evaluate all the models proposed in While bigger models should give better results, the sheer amount of parameters is not indicative of model performance, the baseline model being the most performant while being the simplest among the proposed models. Architecture is actually more important than the amount of stacked layers.

The preprocessing pipeline, albeit short, is actually very important yet the use of GloVe embedding compensate for such absence. As a possible improvement, enriching the pipeline by handling common OOV cases such as dashed-words or numbers in the dataset may improve models' performance.

References

Timothy Dozat. 2016. Incorporating Nesterov Momentum into Adam. *ICLR*.

Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1994. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.