Assignment 1

Pelinsu Acar, Rubin Carkaxhia, Calin Diaconu and **Ruud Johannes Wilhelmus Korsten**
Master's Degree in Artificial Intelligence, University of Bologna
{ pelinsu.acar, rubin.carkaxhia, calin.diaconu, ruudjohannes.korsten }@studio.unibo.it

Abstract

The current assignment had the purpose of creating a solution capable of part-of-speech (POS) tagging. To achieve this, three different configurations of Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) models were tested and compared. The winning one includes one embedding layer, two bidirectional layers, and one fully connected (FC) layer. It achieves an F1-score of 0.91 on a test set of 13,676 elements. The resulting code is available as a Python notebook, with the models implemented using the TensorFlow machine learning library.

1 Introduction

POS tagging is defined in (Ske, 2022) as the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and context.

According to (Lyons, 1977), in grammar, a POS is a category of words that have similar grammar properties. Words assigned to the same POS display similar syntactic behavior, and, sometimes, similar morphological behavior. Commonly listed English parts of speech are noun, verb, adjective, adverb, pronoun, preposition, conjunction, interjection, numeral, article, and determiner.

A solution where an algorithm just looks for the word in a list of words which already had the POS assigned would not work, because oftentimes the POS of a word can only be determined based on the context in which it was used. For example, the word "record" can be both a verb, when used in the sense of storing a piece of information for later use, and a noun, when used to name a musical album released by an artist.

The first accomplishments in the field of POS tagging appeared in the mid-1960s, with the creation of the Brown Corpus (Kucera and Francis, 1967), an early dataset of 1,000,000 words, with an initial program also proposed by the authors. Later on, researchers developed solutions involving hidden Markov models (like (Leech et al., 1994)), dynamic programming methods (such as (Coombs et al., 1987), and (Church, 1989)), and unsupervised taggers. Current highly-performing algorithms include the Viterbi algorithm (Forney, 1973), the Brill tagger (Brill, 1992), constraint grammar (Karlsson, 1990), and the Baum-Welch algorithm (Hoffman and Welch, 1968). Some machine learning solutions have also been developed for solving the problem of POS tagging by using different methods, with most of them achieving accuracies above 95%. By comparison, our solution reaches an accuracy of 91%.

2 System description

The three models used and tested for this task were the baseline model - a bidirectional LSTM layer followed by a Dense layer, model1 - two bidirectional LSTM layers followed by a Dense layer, and model2: a bidirectional LSTM followed by two Dense layers.

The word embedding was done using the GloVe dense embedding (Pennington et al., 2014). A 50-dimensional embedding model was chosen, and it was loaded via the gensim library (Rehurek and Sojka, 2011). It was then used in the trainable embedding layer placed in the beginning of every model. The OOV terms within the validation and test sets were handled by assigning a special token and a random embedding vector, while those in the training set were added to the vocabulary and assigned a random embedding vector. To make sure that the network receives only fixed size input, all the sentences were zero-padded up to a predefined length. The hyperparameters of the network have been chosen in an automatic manner, using the keras tuner library (O'Malley et al., 2019), which, in turn, is based on (Li et al., 2018).

3 Experimental setup and results

The experiment is set up using TensorFlow in a Python environment. The hypermodels are created, and the Keras tuner is used to tune the hyperparameters. Namely, the number of hidden units in each layer, the activation function, the learning rate, and the number of epochs are tuned with the Keras tuner. The selection is based on the highest validation accuracy. After tuning them, the models are created with the optimal hyperparameters.

With all models created, the training process is started. Each model is trained with 3 different seeds to account for randomness, with the F1 score being computed for each model and seed. Finally, the training history is plotted for each model, and the weights are saved. An overview of each model is presented, including the used hyperparameters (Table 1). Additionally, the results of the trained models are presented, including the seeds used, the test loss, the test accuracy, and test F1 score for each seed (Table 2).

4 Discussion

4.1 Analysis of Results

4.1.1 Quantitative Analysis

According to the training history, we can compare four models as in Table 3.

Observations: The training and validation loss curves show a steady decrease over the epochs for all the models with early stopping triggered before 10th epoch, suggesting that further training may not be beneficial. The additional LSTM layer in Model1 contributes to a slight improvement in accuracy. Both model1_timedist and model2 have a lower accuracy compared to model1. Therefore, using TimeDistributed on the final layer and an additional dense layer does not seem to provide a significant improvement over both Model1 and baseline models. Model1 with an additional LSTM layer performs slightly better than the other models in terms of accuracy.

4.1.2 Error

Considering the classification report of model1, we can perform an error analysis by examining the precision, recall, and F1-score for each class. Most of the classes (e.g., CC, EX, IN) have high precision, recall, and F1-score, indicating that the model performs well on these classes. Classes like JJR, NN, VBN have reasonably good metrics, although there is some room for improvement.

To get a deeper understanding, we can consider some examples for class NN. Further analysis shows that the model confuses samples originally belonging to class NN (noun singular) with class NNP (proper noun singular) and vv., meaning that the classification boundary between these two classes wasn't learned well by the classifier. One misclassified example could be "world". In the training set, this word is classified as NNP for 10 times and NN for 12 times showing the inherent difficulty in distinguishing this word according to the context.

Some classes have a very small number of instances (e.g., LS, FW, UH, PDT), leading to either very low or zero metrics. So, collecting more data for these classes or using techniques like oversampling, undersampling may improve the performance of the model.

For certain classes, there is a noticeable difference between precision and recall (e.g., JJ, JBR, RBS: low precision, high recall and WP$, NNPS,VBG: high precision, low recall). This indicates that the model may have a tendency to either overpredict or underpredict these classes. Further analysis and potentially adjusting the model's threshold might help in finding a balance.

4.2 Future Developments

- Considering the distribution of the sentence lengths, a smaller number could be used for the input length.
- Using a dropout layer as a regularization approach may also prevent overfitting for some classes and lead to a better generalization.
- The micro and macro averages indicate a decent overall performance, but there is room for improvement, especially for classes with lower metrics by addressing class imbalances, and further tuning the model could enhance its overall effectiveness as well.

5 Conclusion

Through this assignment, we got the occasion to test and compare different models built for POS classification of words. We defined three different models based on bidirectional LSTM and FC layers, with different numbers of layers and hyperparameters settings. Beside the learning value of this project, we discovered that the best accuracy was yielded by the model containing two bidirectional LSTM layers.

6 Links to external resources

The notebook can be found here: http://bit.ly/3QYTUAB . The weights of the trained models can be found here: https://bit.ly/47gieUi .

References

2022. [link].

Eric Brill. 1992. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, ANLC '92, page 152–155, USA. Association for Computational Linguistics.

Kenneth Ward Church. 1989. A stochastic parts program and noun phrase parser for unrestricted text. In *International Conference on Acoustics, Speech, and Signal Processing,*, pages 695–698. IEEE.

James H Coombs, Allen H Renear, and Steven J DeRose. 1987. Markup systems and the future of scholarly text processing. *Communications of the ACM*, 30(11):933–947.

G.D. Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Frederick Hoffman and Lloyd R Welch. 1968. Totally variant sets in finite groups and vector spaces. *Canadian Journal of Mathematics*, 20:701–710.

Fred Karlsson. 1990. Constraint grammar as a framework for parsing running text. In *COLING 1990 Volume 3: Papers presented to the 13th International Conference on Computational Linguistics*.

Henry Kucera and W. Nelson Francis. 1967. *Computational Analysis of Present-Day American English*. Brown University Press, Providence, RI, USA.

Geoffrey Leech, Roger Garside, and Michael Bryant. 1994. Claws4: the tagging of the british national corpus. In *COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics*.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52.

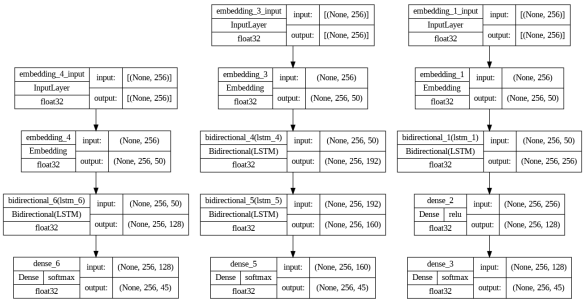
J. Lyons. 1977. *Semantics: Volume 1*. ACLS Humanities E-Book. Cambridge University Press.

Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. 2019. Kerastuner. https://github.com/keras-team/keras-tuner.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Radim Rehurek and Petr Sojka. 2011. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2).

Appendix



(a) Baseline (b) Model 1 (c) Model 2

Figure 1: The Architectures of the Three Required Models

Model	No. of Units	Activation Function	Learning Rate	No. of Epochs
Baseline	Layer 1: 64	Tanh	0.0038720843	10
Model 1	Layer 1: 96 Layer 2: 80	Tanh	0.0091247067	10
Model 2	Layer 1: 128 Layer 2: 128	Tanh	0.0096679321	10

Table 1: Models with tuned hyperparameters.

Model	Seed	Test Loss	Test Accuracy	Test F1 Score
Baseline	42	0.2717678	0.9215181	0.8323744
	123	0.2871494	0.9250562	0.8480362
	999	0.2707961	0.9205532	0.8103828
Model 1	42	0.2971650	0.9268575	0.8552614
	123	0.2703231	0.9219041	0.8365931
	999	0.2704216	0.9248632	0.8533010
Model 2	42	0.3340029	0.9243486	0.8463044
	123	0.3799466	0.9224830	0.8576753
	999	0.2885655	0.9249919	0.8403223

Table 2: Model results for each seed.

	Baseline Model		Model1		Model1_TimeDist		Model2	
	Test Set	Val Set	Test Set	Val Set	Test Set	Val Set	Test Set	Val Set
Accuracy	0.9202	0.9186	0.9258	0.9212	0.9223	0.9211	0.9247	0.9188
Loss	0.2767	0.2900	0.2830	0.3083	0.2542	0.3114	0.2960	0.3394
f1 score	0.8419	0.7579	0.8268	0.8013	0.8430	0.7889	0.8624	0.8072

Table 3: Quantitative Results

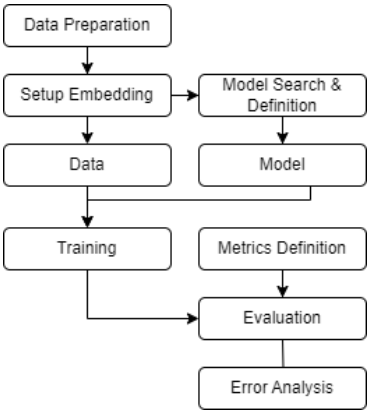


Figure 2: Solution Development Pipeline