Assignment 1

Elisa Ancarani, Enrico Benedetti, Stefano Fantazzini, and **Irene Gentilini**
Master's Degree in Artificial Intelligence, University of Bologna
{ elisa.ancarani4, enrico.benedetti5, stefano.fantazzini, irene.gentilini2 }@studio.unibo.it

Abstract

In this work we implement, train and evaluate four variations of Recurrent Neural Network architectures to tackle the part-of-speech (POS) tagging sequence labeling task. Results show that the considered models achieve similar performances, which do not vary significantly with respect to the most frequent tag baseline.

1 Introduction

The most common approach for part of speech (POS) tagging, is to use neural networks, along with machine learning and hybrid approaches. The Hidden Markov Model is the most used machine learning algorithm for POS tagging, regardless of the Conditional Random Field method outperforming it. However, deep learning algorithms are vastly more common, because of their efficiency in learning from large-size corpus of text. In particular, Recurrent Neural Networks (RNN) such as the ones we evaluated, are preferred (Zewdu and Yitagesu, 2022).
In this paper we tested four different Recurrent Neural Network (RNN) architectures. Our initial baseline architecture is a bidirectional LSTM. We also tried using a Gated Recurrent Unit (GRU) layer instead of the BiLSTM one, as well as adding an additional BiLSTM layer, and adding a fully connected layer (FC), in three separate models.
The training, validation and test sets were built using the Penn Treebank WSJ dataset (Marcus et al., 1993). This dataset is a text corpus containing a set of sentences, divided into 200 separate files. Each file is made up of rows, each containing a token (word, symbol or punctuation mark) next to the tag of the part of speech it belongs to. Each entry in our datasets consisted of the token, the tag, and the file and sentence they were taken from.

2 System description

We have implemented, using the PyTorch framework (Paszke et al., 2017), four recurrent neural network models to experiment with. Their structure comes from the assigment directions provided to us. We defined those model classes from scratch using the PyTorch API.

We prepared functions to aggregate the word tokens into padded input sequences. Each sequence corresponds to a sentence from the dataset. Then, multiple sequences are put into mini-batches, where padding is added to achieve uniform length. The training and evaluation loops were adapted from tutorials on the PyTorch website. The validation metric was the accuracy, while the testing metric used was the macro F1.

The four model architectures contain, in addition to the required layers, an embedding layer. Its weights are the vector embeddings from the GloVe (Pennington et al., 2014) vocabulary, which are frozen during training. Hyper-parameter tuning was implemented using the Ray Tune library (Liaw et al., 2018).

To facilitate analysis of results, the best model weights from each architecture type was saved, along with its training history.

Table 1: The four models' layer sequence (from top to bottom). The input dimension is a one-hot encoded tensor which identifies the input token. The output consists of one activation for each POS to predict. All recurrent layers are bidirectional.

Baseline	**GRU**	**Additional BiLSTM**	**Additional FC**
Embedding	Embedding	Embedding	Embedding
LSTM	GRU	LSTM (2 layers)	LSTM
FC	FC	FC	FC (2 layers)

3 Experimental setup and results

We set up a pipeline where we trained extensively each of the four models, for up to 15 epochs. We use a borrowed Early Stopper object to end training early in case the validation loss does not decrease by a significant amount (10^{-2}, which was chosen empirically after monitoring some early training experiments).

We performed a grid search over the α hyper-parameter, which determines the hidden layer dimension. The formula we used is based on (Lane, 2015):

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))},$$

where N_s is the number of samples in the training set, α is the scaling factor, N_i is the number of input neurons to the RNN (the GloVe embedding dimension, 100) , N_o is the number of output neurons (the total number of classes, 45).

The other hyper-parameters tuned were the training batch size and the Adam Optimizer learning rate. The parameter combinations we tried in our experiments are outlined in Table 2.

All our tested architectures optimized the same loss, multiclass crossentropy. The metrics considered were categorical accuracy for training/validation, and macro F1 for the evaluation on test data.

Table 2: Hyper-parameters combinations and best configuration for each model.

Name	**Values**	**Baseline**	**Gru**	**Additional fc**	**Additional lstm**
α	1,2,10	1	1	1	1
batch size	1,16,128	16	16	16	16
learning rate	$10^{-3}, 3 \cdot 10^{-3}$	0.03	0.03	0.03	0.03

All the results for the best configuration for each architecture are outlined in Table 3. The model parameters are sometimes recovered from a checkpoint and the scores are computed on the best epoch, based on validation accuracy. The table also contains the metrics from the most frequent class baseline, obtained by counting the word-tag frequencies in the training set.

Table 3: Total number of training epochs and validation accuracy for each architecture, and macro F1 scores for the two best models (selected based on val. accuracy).

Name	**# Epochs**	**Val. accuracy**	**Test macro F1**
Baseline	8	0.823	—
GRU	8	0.817	—
Additional BiLSTM	6	**0.834**	0.68
Additional FC	6	**0.831**	0.69
Most freq. class baseline	—	0.82 (test set)	0.70

4 Discussion

The results for the experiments with randomly sampled versus 0 vector embeddings for out of vocabulary words are similar, as the first method has only marginally better accuracy but slightly lower F1 score than the second. The absence of a clear winner may be because the number of tokens without a GloVe embedding is relatively small compared to that of the rest of tokens in the dataset.

In all cases, the results seem to be better with a smaller α value, which corresponds to a larger hidden layer dimension. In addition, by comparing our models with the most frequent tag baseline it turns out that overall results are very similar.

By looking at the per-tag metrics, excluding punctuation and symbols, we noticed that all the considered models predicted some classes better than others. Another observation is that our models confuse tags that have similar functions.

In linguistics, a class of words can be categorized as open or closed. An open class commonly accepts new words, whereas a closed class rarely does. It was be observed that words that belong to the closed class like prepositions, determiners, conjunctions, and pronouns, are identified with near-perfect F1 ($\geq 97\%$). The tags that confuse models the most are part of the open class. The models also mislabel parts of speech that are similar such as comparative adverbs with comparative adjectives, superlative adverbs with superlative adjectives, or plural proper nouns with singular proper nouns.

It should be noted that the per-token metrics can only go so far. Per-sentence accuracy gets quite low, even for the current state-of-the-art techniques (around 56%) (Manning, 2011). The work of Manning on the same dataset highlights more in depth some possible causes of tag prediction error.

In future developments, more work on the data could be done, for instance by removing punctuation also at traning time, and applying different preprocessing to the input tokens. The best model could be expanded with more layers, or the number of parameters could be changed. The training recipe could be improved by employing more regularization techniques like weight decay and dropout. More extensive tuning should be considered. Covering a larger range of hyper-parameters values could improve performance at the cost of time and computation during search.

5 Conclusion

To conclude, the most common approach when tackling the POS tagging problem is to use RNNs. We tested four of them, and regardless of their differences their performance was similar, with a mild improvement occurring when using additional layers, whether recurrent or dense ones. Surprisingly, the result of the best two models was virtually the same as that of the most frequent class baseline model, showing that a simple machine learning model can work just as well as a small neural network. All our models produced a validation accuracy smaller than the training accuracy, and that could be due to the size of the neural network and the lack of dropout layers. In the future the network could be expanded, and the data could be preprocessed differently.

6 Links to external resources

References

Hobson Lane. 2015. How to choose the number of hidden layers and nodes in a feed-forward neural network? Cross Validated. URL:https://stats.stackexchange.com/q/136542 (version: 2021-05-05).

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. 2018. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Computational Linguistics and Intelligent Text Processing*, pages 171–189, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

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.

Alebachew Zewdu and Betselot Yitagesu. 2022. Part of speech tagging: a systematic review of deep learning and machine learning approaches. *Journal of Big Data*, 9.