LSTM based Part-of-Speech Tagger

Natural Language Processing Assignment II

Kamil Akhmetov
Innopolis University
Innopolis, Republic of Tatarstan, Russia
k.ahmetov@innopolis.ru

ABSTRACT

Part-of-Speech tagging is one of the important tasks of Natural Language Processing. In this work, I have tried to implement a tagger using Long Short-Term Memory[4] RNN. The details of the process are described in the further sections.

1 MODEL ARCHITECTURE

The model is a Neural Network which uses several layers. First, numerical representations of words in a sentence are converted to sparse embeddings which are then passed to the LSTM module. After that, a fully connected layer passes LSTM outputs to Part-of-Speech Tags log-probability space using log-softmax function.

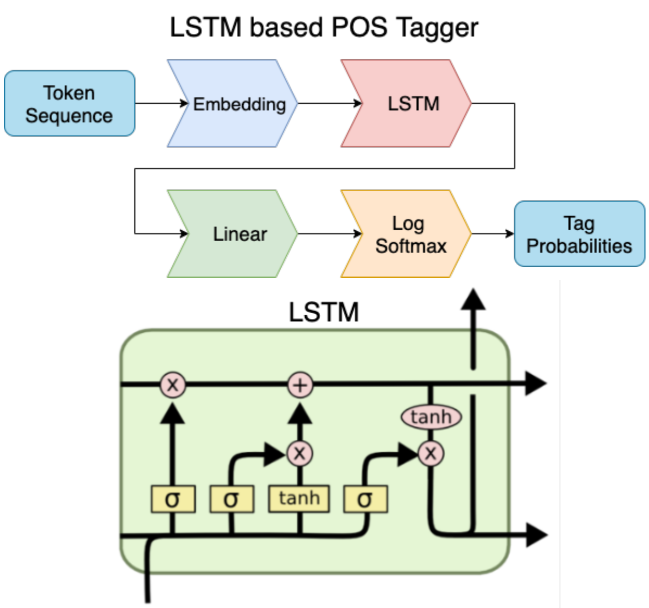

Figure 1: Model Architecture

2 DATASET

For training and testing the model we have used the Russian Taiga Corpus from Universal Dependencies data [3] The selected dataset [2] consists of 3,264 sentences (38,555 tokens) on blog, news, poetry and social topics.

3 TRAINING

Here we discuss details of how the training process is done.

3.1 LOSS

For the loss function we have used Negative Log Likelihood Loss. Essentially this method estimates the likelihood of tag observation given the tag probability distribution for a particular token. By negating this value we obtain the loss, which is back-propagated to train the network.

3.2 Optimizer

As soon as we use sparse representation of a word, we are limited in choice of optimizer. Currently PyTorch supports several optimizers to work with our embeddings. I have chosen Stochastic Gradient Descent. It is a trade-off between computational resources and convergence rate. By using only a subset of data every time algorithm does several less accurate steps instead of doing one but very careful at the same time.

4 HYPER-PARAMETER TUNING & RESULTS

The described model has the following hyper-parameters:

- **E_DIM** defines the word embedding dimensionality, in other words length of the LSTM component input vector for a single word. *[64 by default in our model]*
- **H_DIM** defines the dimensionality of the hidden state of LSTM module. *[64 by default in our model]*
- **Learning Rate** is very important as soon as it defines how fast the optimizer converges.
- **Token Mode** defines what form of the token in a sequence is used. We might take words as they are (the *form* we has observed in a sequence) or try to work with *lemmatized* versions.

As soon as our model has several hyper-parameters, we have defined a number of configurations in the search space. By doing a simple grid search regarding the loss we can fine tune the defined parameters.

conf	lr	token mode	epoch 50 loss
0	0.01	form	0.57026
1	0.01	lemma	0.55938
2	0.1	form	0.00545
3	0.1	lemma	0.00535

Table 1: Configurations on Taiga dataset

sentence accuracy	14.37 %
token accuracy	75.69 %

Table 2: Accuracies on 50 epochs on original split of Taiga

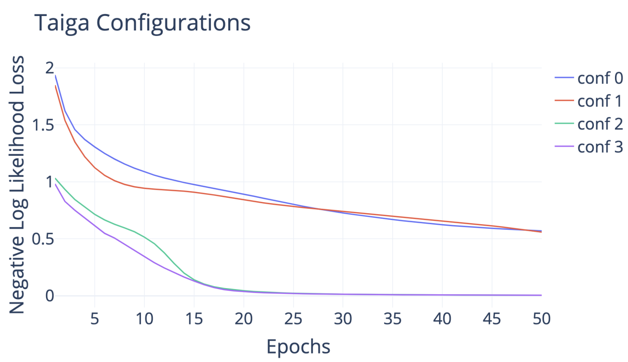

Figure 2: Losses

Deriving from the above the best explored configuration, let us have a closer look to the configuration #3. It had higher learning rate and used lemmas, that is why it has shown a better result on 50 epochs with Russian Dataset.

Here **sentence accuracy** means that each of the tokens within a single sentence were predicted correctly.

In order to understand what are the mistakes the model does most common, observe the confusion matrix:

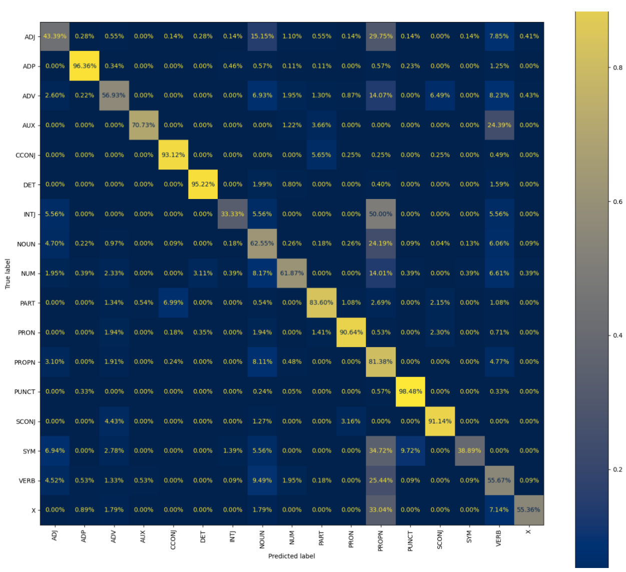

Figure 3: Confusion matrix

As we can see there still exist issues with Adjectives, Adverbs, Interjections, Verbs and other classes. Using the matrix one can determine what are the predictions for objects of these problematic classes.

5 ISSUES

One of the issue to think while designing the tagger is how to treat the **unknown words**. In the process of model evaluation there occurs a situation when some of the observed words might be not seen before during the learning process. Several workarounds exist: having fixed vocabulary, etc. I have chosen to substitute every unknown word by a special marker token, so that such words are indistinguishable by the model.

6 CODE

All the code mostly self-explanatory and is commented where needed. Located at GitHub [1]. For the implementation I have used Python 3 with PyTorch Neural Network Framework. Structure of the project and brief explanation are presented here. We have impelemented a Dataset class for the Corpus which conforms PyTorch. Also for managing the training process and holding different model-related or data-related parameters and options there exists a class named **Solver**. Some other code, containing utility methods, for example, is also present.

7 CONCLUSIONS

To conclude, we have implemented a LSTM RNN based Part-of-Speech tagger using Neural Networks. In order to assess performance we report sentence and token accuracies and plot the confusion matrix. Also we have done the hyper-parameter tuning and **gridsearch** in order to extract the best configuration we have seen so far.

Personally, I think, that still the results should be much better if we train the model for a longer time period, using a bigger dataset and having a more powerful environment. But for now, we have explored the technology of making a POS tagger based on LSTM and tried to run it on a low scaled environment in order to test the main functionalities.

During the assignment I have learned how the Recurrent Neural Networks such as LSTM use both long-term and short-term memory and, solving some of the problems of previous architectures, they allow for building a sequence-to-sequence prediction models for many Natural Language Processing tasks, including POS tagging algorithms.

Thank you!

REFERENCES

[1] Kamil Akhmetov. [n.d.]. *LSTM POS Tagger*. https://github.com/kamilkoduo/lstm-pos-tagger
[2] Taiga Dataset. [n.d.]. . https://tatianashavrina.github.io/taiga_site
[3] Universal Dependencies. [n.d.]. . www.universaldependencies.org
[4] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.