Assignment 1: Part-of Speech (POS) tagging as Sequence Labelling using Recurrent Neural Architectures

Mohammadreza Hosseini, Parvaneh Soleimanybaraijany, Fatemeh Rajbaran and **Shakiba sadat Mirbagheri**
Master's Degree in Artificial Intelligence, University of Bologna
{ mohammadrez.hossein2, p.soleimanybaraijny, fatemeh.ranjbaran, shakiba.mirbagheri }@studio.unibo.it

Abstract

This report consists of showing different methodologies used to address a common NLP task called POS tagging using Recurrent architectures. A POS tag (or part-of-speech tag) is a special label assigned to each token (word) in a text corpus to indicate its part of speech in a sentence and often also other grammatical categories. What we are asked in this assignment is to implement four neural network structures starting from a baseline model and making some little changes in the network layers and find the best two ones according to their performance on the validation data. The overal procedure we followed is first to do some precess on data to change our input into a network-feedable format. For handling the out of vocabulary terms (OOV) we decided to create a random embedding for each of these words. Before starting to fit the architectures on our data, in order to determine the right combination of hyperparameters that maximizes the performance the models, we used KerasTuner library. Now it is the time to train our models using the setting on the validation set to find the best two architectures based of F1-score metric. In the end, only the two best models were evaluated on test set and to see the performance of the networks and error analysis, we used classification report and confusion matrix to get experimental results.

1 System description

1.1 Data preparation

In this experiment we use 'dependency treebank' as our dataset from NLTK data module. To prepare our data to be ready for further manipulation we convert all the words to their lower forms. We do not apply any other kind of preprocessing like removing stopwords or punctuation marks according to the problem definition. Mapping to lowercase is important to avoid the majority of the OOV. The next step is to tokenize each sentence to provide an appropriate representation for the terms. We fit Keras Tokenizer on our entire dataset that basically replaces each word with its corresponding integer value from the word index dictionary. As of now, the sentences present in the data are of various lengths. We need to either pad short sentences or truncate long sentences to a fixed length. This fixed length, however, is a hyperparameter. The maximum length is chosen 89 in order to cover approximately 99.9 percent of the sentences. We found that 99.9 percent of the sentences contain 89 words or fewer in our dataset. The resulting sentences were then one-hot encoded, resulting in a tensor of size N × max length × C in which N is the number of phrases and C is the number of unique tags plus 1 for the padded tokens which is equal to 46 for our data. Afterwards we split data in training, validation and test sets. The input features to the model must be pre-trained word vectors from GloVe embeddings.

1.2 Out of vocabulary

A key step is to define a procedure to build an embedding matrix with the size equal to V * d, where V is the set containing all the terms presented in the corpus and d is the dimension associated to the word-embedding vectors. The vocabulary provided by GloVe contains 400k unique words, but anyway there may be words that are not present in such dictionary but that appear in our dataset (OOV). What we have done in odrer to handle this out of vocabulary words is to check the existence of the word in GloVe embedding. If it is presented, its embedding vector would be added to our embedding matrix. Otherwise we ssign a randomly generated vector of dimension d sampled from a uniform distribution ranging in [-0.5 , 0.5]. Finally for all of the words presented in our dataset, there is an equivalent embedding.

	Unit number	LR
Bi-direct	96	0.0333
GRU	64	0.0334
Two-Bi-Direct	64/64	0.0333
Two-dense	64	0.0332

Table 1: Obtained Hyper-parameter after tuning

	F1-score(validation)	F1-score(test)
Bi-direct	0.80	0.77
GRU	0.78	-
Two-Bi-Direct	0.82	0.74
Two-dense	0.79	-

Table 2: Evaluation result on validation and test sets

1.3 Designing the networks

Four networks were implemented using the Tensorflow/Keras framework. All the models have as first layer a non-trainable Embedding layer which transforms the words into their corresponding embeddings. Here we report a brief description of their structure:

- 1. Baseline model: Consisting a Bidirectional LSTM layer and a Fully Connected (FC) Dense layer.

- 2. GRU+FC: The second model is a variation of the baseline, consisting of a GRU layer instead of the LSTM one.

- 3. Two-Bi-Deirect+FC: The third model tries to add more complexity to the baseline model by adding another Bidirectional LSTM layer on top the first one.

- 4. Bi-direct+two-FC: Similarly, the fourth model takes the baseline model and it adds after the Bidirectional LSTM a FC layer.

1.4 Hyperparameter tuning

One of the most important steps in constructing a neural network model is to find the optimal model architecture. In this experiment we decided to make automatic the process of exploring a range of possibilities to select the most promising architectue by using KerasTuner library which is a scalable hyperparameter optimization framework. The result of tuning phase is to find the optimal number of units in the hidden layers and the optimal learning rate for the optimizer of the models which is Adam in our experiment. The detail of obtained hyperparameters is shown in table 1. We train the models on the training set with the tunned parameter and in order to avoid overfitting and stop training once the model performance stops improving, we used early stopping method.

2 Experimental setup and results

Once the training was completed, the two best models are chosen by looking on the macro-F1 scores on the validation set. The two mentioned models are then evaluated on the test set. The macro-F1 score is computed on the ground truth labels and the predicted ones by looking token by token. In the evaluation, all the punctuation tags and symbols tags are not taken in consideration. we report the best results on Macro-F1 obtained in the experimentation in Table 2. It seems that the models are able to generalize well as they achieve F1-scores on test set comparable the ones obtained on the validation set.

3 Discussion

In order to analyze the prediction errors that our best models made, we generated classification report and confusion matrices. The report tells us the fact that our classifier does not work well while predicting a tag related to the class with small number of supprts. We have also noticed that the dataset is greatly unbalanced as the target variable has more observations in some specific classes like 'vb', 'NNP' than the others. By looking at the result of confusion matrix we can simply find which tags we get wrong the most and what kind of errors our classifier makes.

4 Conclusion

As mentioned, the architectures that have reached the best performances in terms of F1-score are: the baseline model and the two-Bi-direct + FC model. Possible extensions of this work could be:

- Stratified sampling or rebalancing as unbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier, as we saw its effect.

- The addition of learneable embeddings, in order to be able to fine-tune them to our domain.