NLP 2016/2017 - Homework 2

Marco Favorito 1609890

Link: https://drive.google.com/open?id=0B8ywz9q9eB5lbjVGRUE2OEpWMHM
Notice: include always the folders "resources/" in your tests!

Abstract

A POS tagging system with Bidirectional LSTM is implemented with Keras APIs, using as input a combination of a one-hot representation of the words and word features about their case state. The vocabulary is built from training data, after some preprocessing over the seen word. The learning rate is tuned dynamically and an early-stop method is applied. On average, the F1 score over the test dataset is about 94%. For completeness, I have also implemented the model using GloVe and Word2Vec.

Model description

My NN for this task is relatively simple:

- Two inputs:

  one is from the encodings of the words: each instance (i.e. sentence) is seen as a list of one-hot encodings in "integer-form"[1]. This number is computed after the construction of the inverted index over the observed vocabulary;

  the other is taken from a feature extraction[2]: for each word w, a vector of three boolean is computed as the following:

  $$f(w) = \{w.\text{islower}(), w.\text{isupper}(), w.\text{iscapitalized}()\}$$

  respectively, whether every character of w is lowercase, is uppercase, or the word is capitalized.

  The issue about different lengths of the sentences is handled by first computing the maximum length in the corpus and then inserting padding (value 0) in every sentence, in order to fit the data in one `numpy` fixed matrix (which is good for performances).

- Merge of the two inputs, the first passed into an Embedding layer, the other into a fully connected layer (Dense in Keras); then is applied a sum operation.

- The core of the network: a Bidirectional LSTM layer, i.e. two LSTM in cascade, the former goes forward and the latter goes backward. The idea is that, given all the sentence in input, it can catch correlations among all the parts of the sentences, even if they are far from each other;

[1] In fact, one sentence is simply a vector of integers of maximum value equal to `len(vocabulary)+2` (because of the padding tag and the unknown tag). The Keras `Embedding` layer will translate them into vector form of dimension `embedding_size`, a user-defined parameter.

[2] https://arxiv.org/abs/1510.06168

- Another Dense layer, wrapped into a TimeDistributed Keras object (since we are handling sequences), which gives in output a vector of shape (len(sentence), len(nb_classes)), probabilities distributions of tags over each word;
- An activation layer, using the "softmax" function, since we are dealing with multiclass classification.

Optimization

The optimization is done dynamically thanks to the callbacks API functionalities of Keras: the learning process stops when the validation loss does not improve for a number of epochs equal to the "patience" parameter. Moreover, the callback ReduceLROnPlateau tries to recognize when the learning state is on a "plateau": in that case, decrease the learning rate by an user-defined factor. For the batch size: the greater, the better, but strictly depends on the available memory and so also from the number of parameters to train. I've done a grid search for tune the sizes of both the Embedding layer and the LSTM layer. The implementation can be found in `src/other_tasks/grid_search.py`. [3] Other important tunable parameters are the dropout rate and the recurrent dropout rate (in the LSTM): they are fundamental for prevent overfitting on training data.

About the vocabulary: every word is preprocessed in order to achieve less dispersion as possible.

- Every word in lowercase[4] is passed in input to the WordNet Lemmatizer[5];
- A custom fuction is provided, which remove some common repetition that are unambiguos as regards the tag of the word (i.e. numbers, time, URLs, emails).

Experiments

The described model, on average, achieves almost the 94% of F1-score[6]. In the following pages I show some plots documenting my experiments.

Extra points

I've done also the extra-points task, using the same approaches above described. Parameters have been adapted through a grid search. The model performs almost in the same way: it seems to don't complain too much about new italian words. The embedding layer do the rest.

[3] Notice that the script h is not called at every train of the model: it is an "utility" that I've used for understand better the scenario.
[4] The case informations of each word are not lost, by construction.
[5] From `nltk.stem` package.
[6] Which in our case is the same of precision and recall, since there are no "negatives".

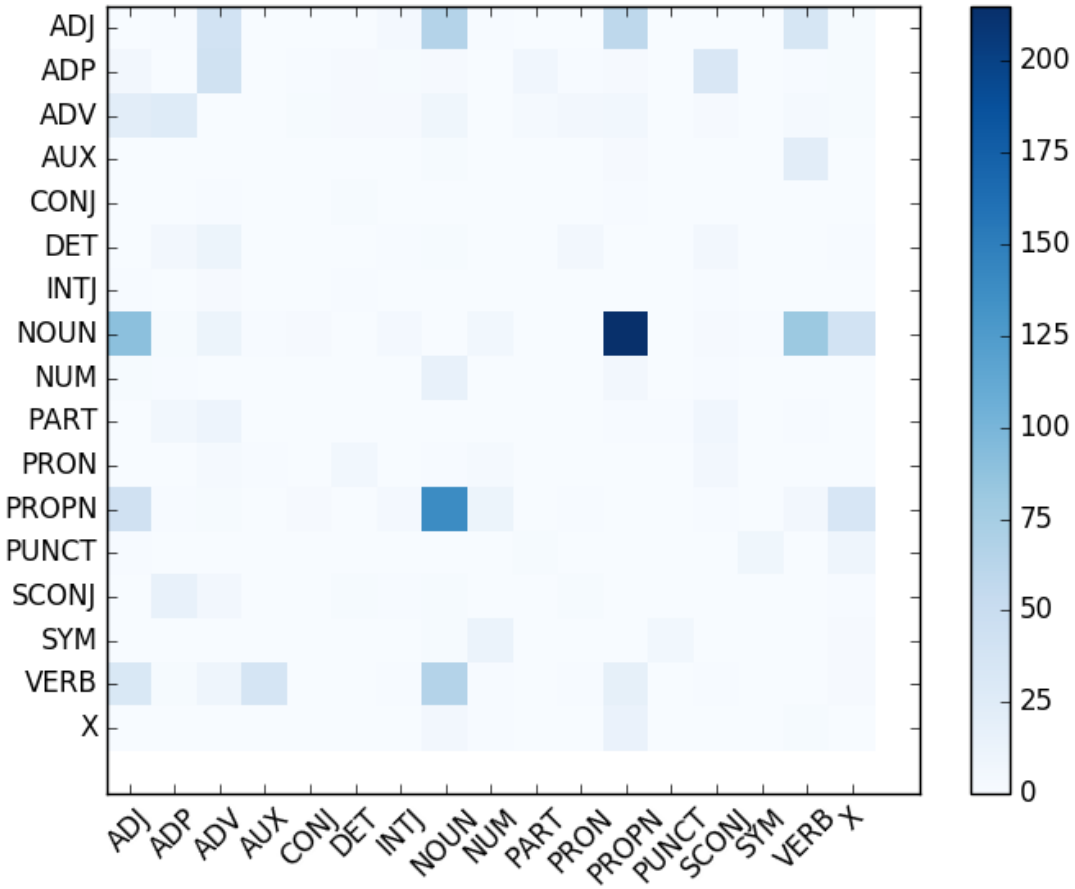

Figure 1: This is the confusion matrix. Nice thing to note is that it is "almost" symmetric, which means that pairs of tags are ambiguous in both directions. The most ambiguous one is the pair (NOUN, PROPN). Some examples are:

- the word "Military" in "This Fallujah operation my turn out to be the most important operation done by the US Military since the end of the war." Here, "Military" is a PROPN, but the model predicted NOUN.
- the word "Project" in "Attached below is Davis Thames' presentation regarding the proposed Project Bruin.". In this case, the predicted tag is "NOUN", while the right one was "PROPN".

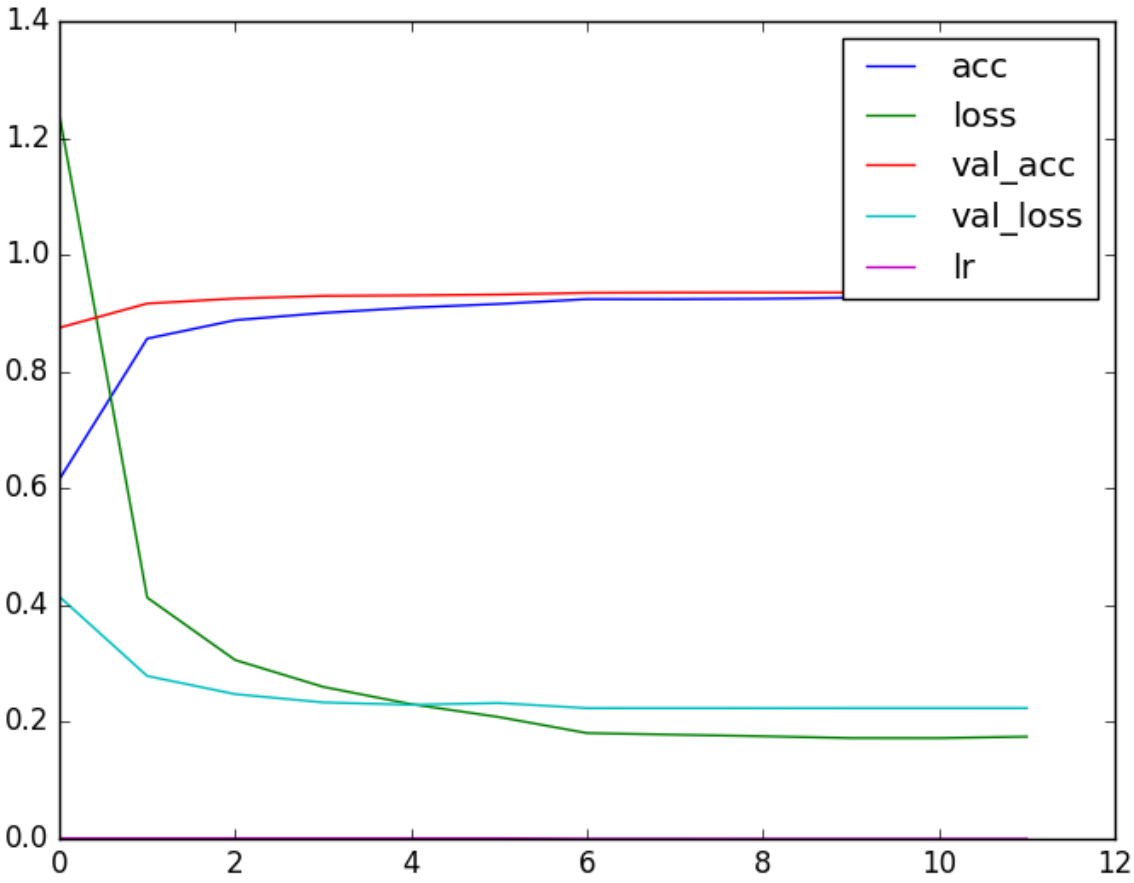

Figure 2: Values of accuracy (acc) and loss (loss) on the training set, validation accuracy (val_acc) and validation loss (val_loss) on the development set in function of the epoch number, during the training of the NN, with parameters:

- 'nb_epochs': 100,
- 'batch_size': 128;
- 'embedding_size': 384;
- 'lstm_size': 256;
- 'dropout_rate': 0.1;
- 'recurrent_dropout_rate': 0.1;
- 'dict_type': 'custom';

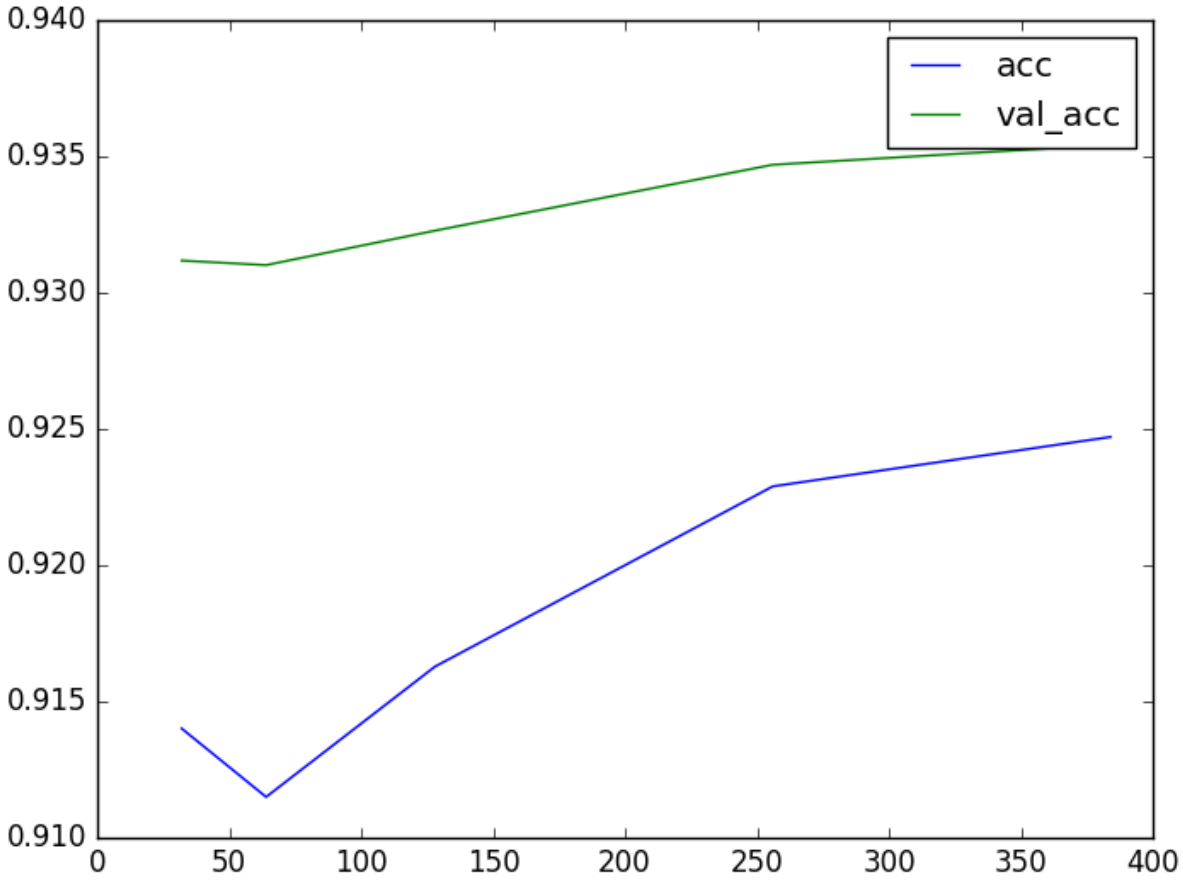

Figure 3: Values of accuracy (acc) and validation accuracy (val_acc) during a grid search with lstm_size fixed at 256 and varying the embedding_size. You can see all the history in `notes/grid_search_english`.

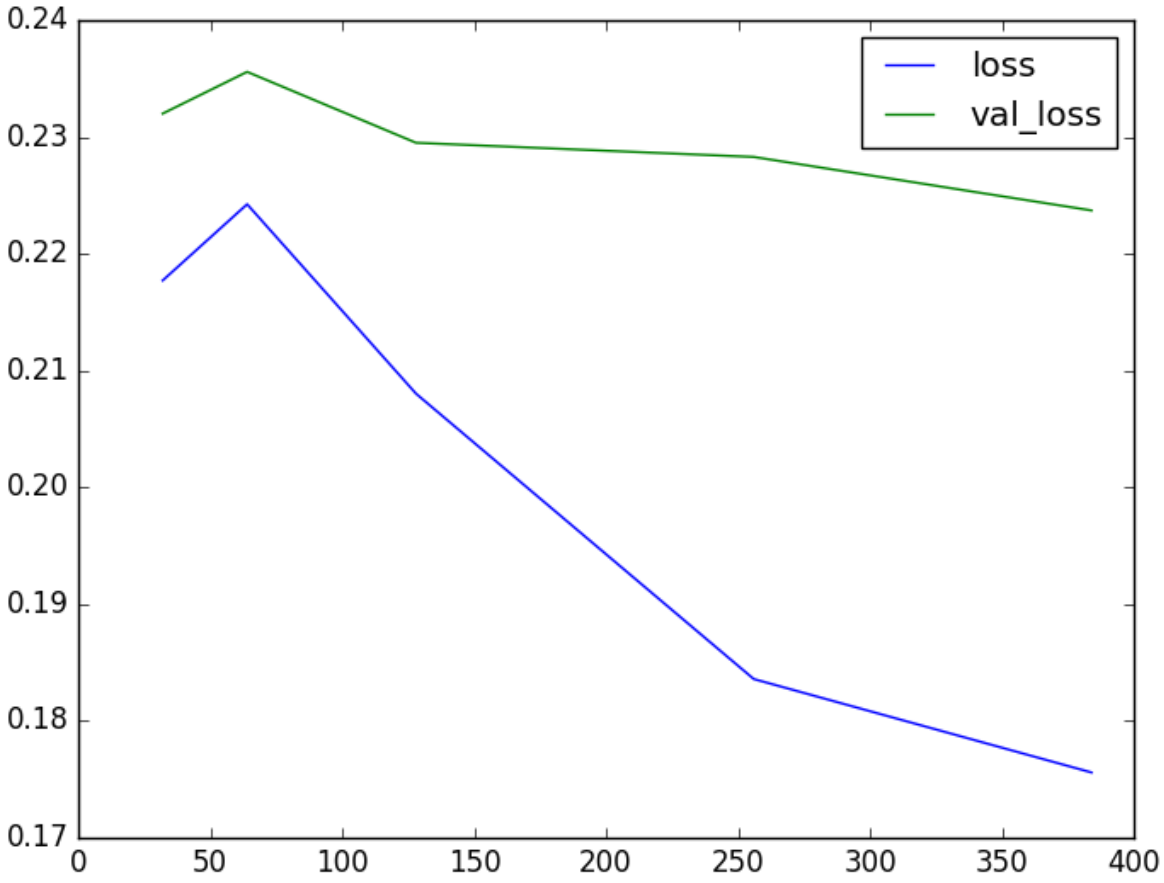

Figure 4: Values of loss and validation loss during a grid search with lstm_size fixed at 256 and varying the embedding_size. You can see all the history in `notes/grid_search_english`