Neural Networks Project
A sequence Part Of Speech tagger with Bidirectional LSTM

Domenico Alfano

1. Introduction

The main goal of the project is to implement a sequence POS tagger with a Bidirectional LSTM. Before starting with the implementation of the project, I would like to start with a theoretical framework. In fact in this section I will talk about Part Of Speech Tagging, Keras, Bidirectional LSTM and how the model can be evaluated.

1.1. Part Of Speech Tagging

The grammatical analysis of the text is the association of a part of the speech to each word of a sentence. It provides to establish the relationship between the words of a sentence, the adjacent ones and those contained within the same proposition. Part of Speech Tagging (POS) is a method for grammatically analyzing text automatically, using a software or computer algorithm, and facilitating understanding of the information. It is a method used in Information Retrieval (IR), Text Mining, and Artificial Intelligence. The text is first divided into phrases and propositions. Propositions are, in turn, transformed into sequential word vectors. Each word is compared with an external data base, which contains the possible terms of the speech. A term is a set of words (token) or root (stem). The elements of the set can be ordered or not into a group. If the group is ordered, the words are subdivided according to the function they perform in a sentence. For example, noun or adjective. In this case, the order of tokens in the list carries an additional information. If the group is not ordered, the words are contained in the term list without any additional indication of their function and without any particular order. The token management allows the labeling of words contained in a sentence or the main parts or fragments of the speech (part of speech). Once a possible lexical function is associated with every word or segment of the text, it is easier to trace back to the grammatical structure of the text. Unfortunately, the same word can be tagged with different Part Of Speech tags, so for this reason, in Natural Language Processing, Part-of-Speech tagging is considered a disambiguation task.

1.2. Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow or Theano. The core data structure of Keras is a model, a way to organize layers. A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.

1.3. Bidirectional LSTM

The method that I used to solve the speech recognition task is the Recurrent Neural Networks (RNN). Several approaches to this task has been used in the past with poor results, instead, RNN produces promising results. The RNN structure has an input layer, an hidden layer and an output layer. In named entity tagging context, the input layer represents input features and the hidden layer represents tags.
In this project, I will apply Bidirectional Long Short-Term Memory for sequence tagging, to have access to both past and future input features for a given time. The Bidirectional RNN allows to split the neurons of a regular RNN into two directions, one for forward states, and another for backward states.

1.4. Evaluation

In Natural Language Processing, the F1 score is a measure of a test's accuracy. It considers both the precision and the recall of the test to compute the score. Precision is the number of correct positive results divided by the number of all positive results. Recall is the number of correct positive results divided by the number of positive results that should have been returned. The traditional F1 score is the harmonic mean of precision and recall.
The F1 score is often used in the field of information retrieval for measuring search, document classification, and query classification performance and is also used in machine learning.

2. Implementation

In this section I will talk about the implementation, explaining in detail: the type of data founded, the vectorization used for words and tags, the description of the model and finally the discussion of the results. Before doing this, it is necessary explain how the project has been implemented.
Firstly it has been created the file Project.py that defines 3 abstract classes: AbstractPOSTaggerTrainer, AbstractPOSTaggerTester and AbstractLSTMPOSTagger.
AbstractPOSTaggerTrainer represents a class to train a model. It has one abstract method which takes as input the path to the training data and outputs the model.
AbstractPOSTaggerTester represents a class to test a trained model. It has one abstract method which takes as input a trained model and the path to the gold standard data. It outputs f1 score.
AbstractLSTMPOSTagger is an abstract pos tagger that can predict the pos tags given a tokenized sentence. It has one abstract method which given a list of tokens outputs a list

of same length with the pos tags for each word.
Project.py also contains an utility class, ModelIO, thath implements two methods: $save(model, path)$ that save the model on the specified path and $load(path)$ that load a keras model from the path.
Finally, Project.py define a Test class that implements a test method which ensures the good performance of the all classes.

2.1. Type of data

The format used is the CoNLL format where each line of this files is divided in 10 column tab separated.
The first five columns contain the main information:

- The first line contains the Word ID: starts from 1 for each sentence and may be a range for multiwords
- The second line contains the FORM: the word form or the punctuation symbol.
- The third line contains the LEMMA: the lemma of the word form.
- The fourth line contains the U-POSTAG: universal part of speech.
- The fifth line contains the X-POSTAG: language specific pos tag.

The tag set of each file contains 17 POS: ADJ: adjective, ADP: adposition, ADV: adverb, AUX: auxiliary, CONJ: coordinating conjunction, DET: determiner, INTJ: interjection, NOUN: noun, NUM: numeral, PART: particle, PRON: pronoun, PROPN: proper noun, PUNCT: punctuation, SCONJ: subordinating conjunction, SYM: symbol, VERB: verb, X: other.

2.2. Vectorization

The word-vector representations that I used is the One-Hot encoding. In natural language processing, a one-hot vector is a 1 $\times$ N matrix (vector) used to distinguish each word in a vocabulary from every other word in the vocabulary. The vector consists of 0s in all cells with the exception of a single 1 in a cell used uniquely to identify the word. To do this, firstly I create from the train file a list of all words contained in it and then I create a dictionary that associates an index with each word. In the same way I create an other dictionary that associates each tag to an index. After doing this, I create the features vector and the labels vector. For the features vector, as required by the Embedding layer, I created a two-dimensional vector and I linked the index of the corresponding word to each cell. It will then be the task of the Embedding layer to convert the indices into One-Hot vector. For the labels vector, instead, I've created a two-dimensional vector that consists of 0s in all cells with the exception of a single 1 in a cell used uniquely to identify the tag associated to the word. Also, the management of unknown words during the test phase has been resolved by associating a random index to these words.

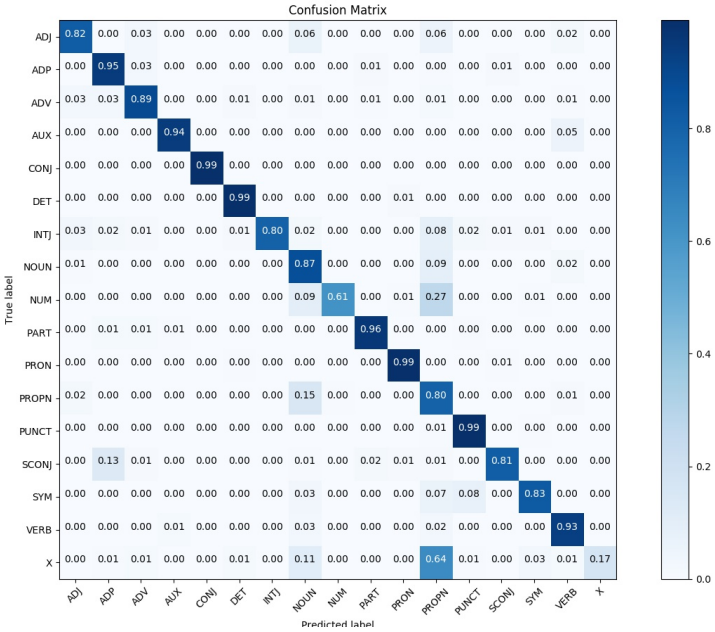
2.3. Model description

The simplest model is defined in the Sequential class which is a linear stack of Layers. There are two ways to create a Sequential model: the first one, is to create a Sequential model and define all of the layers in the constructor, the second one, is to create a Sequential model and add the layers in the order of the computation you wish to perform. As we can see in the file POSTaggerTrainer.py I have choosen the second way to build my model. The first layer in the model must specify the shape of the input. As first layer I choose the Embedding layer that turns positive integers (indexes) into dense vectors of fixed size. The Embedding layer takes in input a 2D tensor and provides as output a 3D tensor. This layer requires an input dimension that in my case is the lenght of my dictionary, the output dimension that is an arbitrary integer and an inpunt lenght that is the lenght of how many words I consider in a sentence.
After the Embedding layer I add to my model a recurrent layer: The LSTM layer. An important benefit of recurrent neural networks is their ability to use contextual information when mapping between input and output sequences. In my model the LSTM layer has only one argument: the units, a positive integer that represent the dimensionality of the output space.
After this, I need to add to my model a core Layer. There are a large number of core Layer types for standard neural networks and I have choosen a Dense layer. The Dense layer is a fully connected layer and the most common type of layer used on multi-layer perceptron models. In my model the Dense layer, as in the case of LSTM layer, has only one argument: the units, a positive integer that represent the dimensionality of the output space.
Finally, to complete my model, I have added the activation function Softmax. Softmax Regression is a generalization of logistic regression that can be use for multi-class classification.
Once I have defined my model, it needs to be compiled. This creates the efficient structures used by the underlying backend, in my case TensorFlow, in order to efficiently execute my model during training. I compile the model using the compile() function and it accepts three important attributes:

- Model optimizer: the search technique used to update weights in the model.
  For my model I used Adam, an adaptive learning rate optimization method.
- Loss function: the evaluation of the model used by the optimizer to navigate the weight space. For my model I used the categorical crossentropy, used for multi-class logarithmic loss.
- Metrics: are evaluated by the model during training. For my model I used the categorical accuracy, used for multi-class logarithmic loss.

3. Results

The results in terms of Precision, Recall and F1 score are obtained by creating a Confusion Matrix given the true tag and the predicted tag according to the tag set.
For this reason, as we can see in the next figure, the Confusion Matrix is composed by 17 rows and 17 columns.
Precision has to be computed as the number of correctly predicted pos tag over the number of predicted pos tags.
Recall has to be computed as the number of correctly predicted pos tag over the number of items in the gold standard.
F1 has to be computed as the armonic mean between Precision and Recall (2* P * R / (P + R)) and it is equal to 91%.



Finally, the next two figure represents the model during the training phase in terms of accuracy and loss.