POS Tagging using Recurrent Neural Network (LSTM) Homework2

Marco Treglia
University of Sapienza
Master in Artificial Inteligents and Robotics
Natural Leanguage Processing

***Abstract*—In this homework we have to create a neural network using the LSTM (Long short time memory) [1] architecture that given in input a sequence of word return the respective POS (part of speech) for each word.**

1. Introduction

At the beginning was very agog to start to work on this NN architecture, the power of learning probability distribution between sequence, who would not be excited. Instead after some week, meanwhile my work was going on, I realized that this would have been a pity. Of course, i'm referring that all the issue I encountered was in memory and heavily in computational time. I don't possess a GPU, and "no GPU no party !", but cut the cackle and let's move on.

2. LSTM

Recurrent neural networks (RNNs) are a powerful family of connectionist models that capture time dynamics via cycles in the graph. Though in theory, RNNs are capable of capturing long-distance dependencies, in practice, they fail due to the gradient vanishing/exploding problems [2]. LSTMs are variants of RNNs designed to cope with these gradient vanishing problems. An LSTM network is formed like the standard RNN except that the self-connected hidden units are replaced by specially designed units called memory blocks [3] as illustrated in Figure 1:

2.1. LSTM Cell layer

The first step in our LSTM is to decide what information we are going to throw away from the cell state.

$$f_t \;=\; \sigma\left(W_f \;\cdot\; [h_{t-1}, x_t] \;+\; b_f\right)$$

The next step is to decide what new information we are going to store in the cell state.

$$i_t \;=\; \sigma\left(W_i \;\cdot\; [h_{t-1}, x_t] \;+\; b_i\right)$$

Next, with the new information, we are going to update cell state.

$$\mathrm{C}_t \;=\; \sigma\left(W_{\mathbf{C}} \;\cdot\; [h_{t-1}, x_t] \;+\; b_{\mathbf{C}}\right)$$

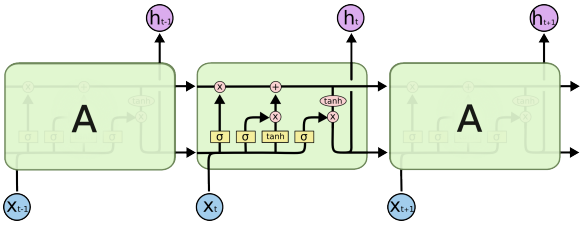

Figure 1. LSTM Cell

Finally, we need to decide what were going to output. This output will be based on our cell state but will be a filtered version of the hidden layer.

$$o_t \;=\; \sigma\left(W_o \;\cdot\; [h_{t-1}, x_t] \;+\; b_o\right)$$

Then, we put the cell state through tanh (to push the values to be between 1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

$$h_t \;=\; o_t \;\tanh\; (\mathrm{C}_t)$$

2.2. How predict

An input vector sequence $x \;=\; (x_1, ..., x_T)$ is passed throught a weighted connections to stack of N recurrently connected hidden layers to compute first the hidden vector sequence $h^n \;=\; (h_1^n, ..., h_T^n)$ and then the output sequence $y \;=\; (y_1, ..., y_T)$. Each output vector y_t is used to parameterise a predictive distribution $Pr(x_{t+t}|y_t)$ over the possible next inputs x_{t+1}.

The probability given by the network to the input sequence x is :

$$Pr\left(x\right) \;=\; \prod_{t=1}^{T} Pr\left(x_{t+1}|\; y_t\right)$$

and then the Loss $L()x$ used to train the network is the negative logarithm of $Pr(x)$:

$$L\left(x\right) \;=\; -\;\sum_{t=1}^{T} \log\; Pr\left(x_{t+1} \mid y_t\right)$$

3. Model

I tested different structure of models but the expectancy of each test bring me away to test each one deeply. The common layer that I used is explained here. the Input layer, used just for convenience because it could be integrated into the embedding layer. Then the Embedding layer wich transforms each integer representations of the words in a new dimension space. For my experiment, I don't go over one hundred and fifty dimensions. Then the core of the model, the Lstm layer. Takes as input a sequence of word converted from the embedding layer and using the return sequences to true return as output a tensor wich represent each outcome of the cell state given that input sequence. For managing the outcome tensor it's necessary to use the TimeDistributed layer called before the Dense layer, wich link all neurons units to the states of the lstm. It returns two dimension data. Finally the Dense layer with as activation function the softmax for mutually exclusive classification.
For computational time issue, I trained the model in Figure2.

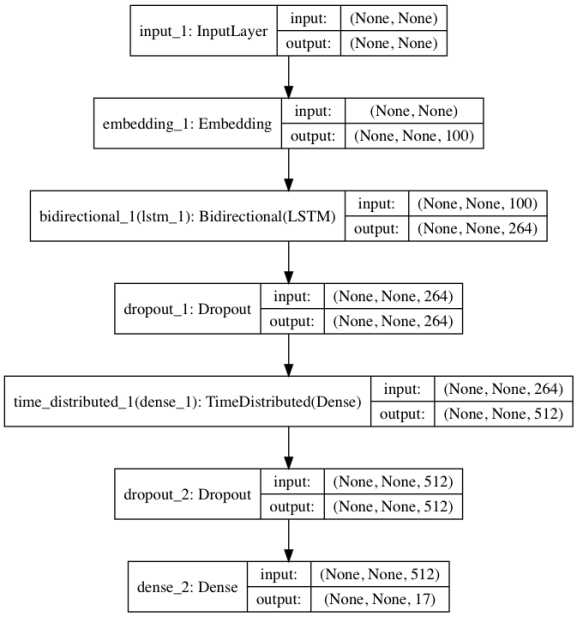

Figure 2. Keras model, Total params: $2,355,749.0$

4. Optimization

The first optimization is the initialization of the weights and bias using the Xavier initializer called in Keras glorot uniform. This provides a better starting point for our learning parameters. After I used a lot of dropouts which drop some neurons connections with some assigned probability, almost I used it after each layer, for decrease the overfitting on the training data and increase the testing accuracy. As last thing i used L2 weights regularization.

5. Exmperiment

For my first experiment, I trained the model learning my embeddings. For it, i used the vocabulary dimension of the train data set and the unknown words have been substituted with the token $< UNK >$. For the input sequences I started from the one hot representation of the words and creating the sequences as moving windows of context word of all the sentences raveled, with this solution I encountered in memory consumption problems(solvable with HDF5 matrix) but more important the sequences presented also word from other sequences. Then I realized that with the Keras's embeddings layer was easy solvable the problem of the large matrix and for the input, I used the padding solution. However with padding though i had all the context words from the sequence i encountered in logic resoning in the back propagation. So I went further and, in my opinion, I found one of the best possible solution using the train on batch and given a sentence at a time, without worry about padding, but taking care to reshape each sentence as tensor and to have not fixed shape as keras's input.
Achieved this there was nothing but that training. I tried also to load word2vec weights using gensim, but I had no time for testing other types of structure models for getting better results.
Therefore the model I present is trained on my embeddings.

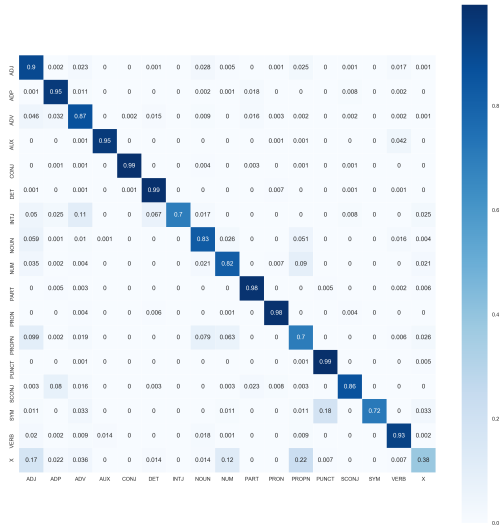

Figure 3. Confusion Matrix

The result in terms of precision, recall and F1 : 0.9037

References

[1] Hochreiter S., Schmidhuber J. (1997) *LSTM.*

[2] Bengio et al., 1994; Pascanu et al., 2012

[3] Graves et al., 2013