SAPIENZA UNIVERSITY OF ROME

Natural Language Processing

Third Homework Assignment

Author:
Giorgio Mariani

1 Problem Description

The *Semantic Role Labeling* (**SRL**) problem [1] consists in the identification, given a predicate in a sentence, of its semantic arguments (i.e. the words in the sentence which play an important *semantic role* for the predicate, like the *subject* of a certain predicate). More precisely, the **SRL** problem consists in the identification and subsequent role annotation of predicate arguments, having a sentence and a semantically disambiguated predicate contained in it. Obviously achieving a good solution for such a problem can open new possibilities for more semantically capable softwares. It can be expecially useful for tasks like *Question-Answering* and *Information Extraction*. The goal of this assignment is the implementation of three deep neural network architecture for **SRL**, heavily inspired by the work of *D. Marcheggiani* and *I. Titov* [2] [3].

Example of *semantic role labeling* of a predicate:

The	cat	is	on	the	table.
	A0	**be.01**			**A1**

1.1 State of the Art for SRL

At first, when this problem was first proposed, a great deal of importance was given to the syntactic features of words and sentences, labeling syntactic chunks of sentences with the respective semantic role. However, lately the state of the art solutions for this problem have been progressively moving towards more deep neural network oriented approaches, giving less focus to syntactic information and parsing (leaving responsibility for pattern recognition in the sentence to the network). Specifically, in this assignment two of such models were taken into account: A simple syntactic-agnostic neural model and a more complex one built on top of the first which takes advantage of the sentence's *dependency tree*.

2 Training and Evaluation Data

The data used for training the neural networks is the **CoNLL 2009** English *training* dataset.
This dataset contains 39,279 sentences, and for each word its *Part-of-Speech*, *lemmatization* and dependency info (*dependency head* word and *dependency type*) are shown, together with eventual predicate-argument information.

In order to measure the performance of each neural model the **CoNLL 2009** English *development* data-set is used. Such data-set contains 1,334 sentences and is structured in the same fashion as the *training* one.

3 Used Approach

During the execution of this assignment mainly three neural network models were considered and implemented:

- A relatively simple neural network, mostly composed by a *BiLSTM* (see section 3.1). It is used as a baseline. This model is a scaled down version of the model presented in [2].
- A deep neural network (built on top of the first one) which exploits dependency information through a *GCN* (see section 3.2). This model, like the previous one, is a simplified and scaled implementation of an already existing model [3].
- An alternative neural network architecture exploiting dependency information through a *GCN*.

3.1 Long Short-Term Memory Neural Networks

Long Short-Term Memory[1] (*LSTM*) neural networks are a useful type of neural networks which is able to work relatively well (compared to simple *RNNs*) over sequences of input data. They are able to do so by utilizing a memory, which is able to propagate information along the sequence (that is the reason of the "Long" part of the name, since information in the first sequence element can effect even the last one).

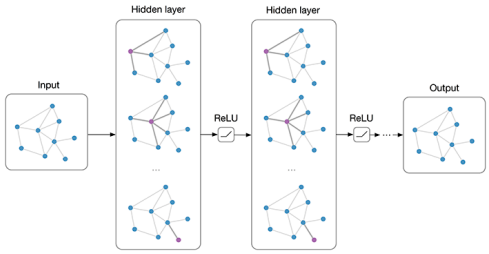
It is common to stack up different *LSTM* layers, in order to obtain a more expressive network, and it is especially useful to stack two *LSTM* and organize them in such a way that the top one receive the reversed output of the bottom one (this type of architecture is called *Binary LSTM* (*BiLSTM*)): the main advantage of *BiLSTM*s is that when computing a word's output both its left and right context (previous and future words) are considered during the processing, while with standard *LSTM*s only information coming from the left context can effect the output.

[1] For a brief explanation follow this link.

3.2 Graph Convolutional Neural Networks

Graph Convolutional Networks[2] (*GCN*) are a class of neural networks designed for a specific goal: the elaboration of graph data through a neural network.

Figure 1: *GCN* Example.



*GCN*s work in the following manner: consider the graph $G = (V, E)$, with $n = |V|$, and with each vertex in the graph having an associated feature vector x_i (with d dimensions), then for all x_i a corresponding output feature vector z_i (with d' dimensions) exists and it is expressed as function of i's neighborhood (N_i) and the vector x_i itself.

There is no golden rule about the shape of such function, however a simple yet effective one could be something similar to

$$z_i = \text{ReLU}\left(\sum_{j \in N_i \cup \{i\}} W x_j + b\right)$$

with W an $n \times d'$ weight matrix and b a bias with d' dimensions.

Since it uses data from all the node's neighbors and is a non-linear function.

Usually however, more complex operations are also involved in the computation, operators like *gates* and *normalization factors*.

Obviously, if only one *GCN* layer is used then it is not possible, when computing a node's output value, to use information from non-adjacent nodes. However, by stacking more layers it is possible to reach more and more nodes: with two layers, information between nodes that have distance two or less can be used, with three, nodes at distance three or less can be used and so on. s

4 Simple Architecture

This model is a simplified and smaller implementation of the neural network described in [2]. It is composed by mainly three components: a *BiLSTM* layer, a predicate encoding component and finally a simple *Softmax* classifier layer. The network takes as input a word sequence[3] (representing a sentence), each input word is expressed in the network through the concatenation of the following word-specific data:

- A pre-trained, non-trainable word embedding of the *lemma* associated to the word (**GloVe** embeddings with 100 features).
- A trainable randomly initialized word embedding of the same *lemma* (with 100 dimensions).
- A trainable randomly initialized embedding of the word's *Part-of-Speech* using 16 features.
- A trainable randomly initialized embedding of the word's *dependency-type* using 32 dimensions.
- If the word is a predicate in the sentence (not necessarily the one whose arguments we are predicting) then a trainable randomly initialized embedding of the predicate (64 dimensions). If the word is not a predicate then a default, neutral embedding is used.
- A single boolean field with value equals to 1 if the word is the predicate whose semantic roles are being predicted, else equals to 0.

The sentence is then given in input to a single *BiLSTM* layer with two memory cells (one for the forward pass and one for the backward pass) of 150 elements each. After, the resulting output is "enhanced" with predicate information and used by a classification layer.
This predicate "enhancing" works as follows: the *BiLSTM* predicate word representation is simply concatenated to the each word in the *BiLSTM* output sequence, so to encode the predicate information with the hidden sentence representation. Finally, this encoded sequence is feed to a standard *softmax* classifier in order to predict (for each word) it's role.

4.1 Results

Despite its simplicity, this approach is still able to achieve a relatively adeguate performance (table 1), and the original implementation [2] (which

[2]More info about *GCN* here.

[3]For sake of simplicity we consider the input to be a single sequence: in truth the network works over batches of such sequences (padded w.r.t. the longest in the batch).

is slightly more complex and with a lot more parameters) is essentially on-par with the current state of the art.

Precision	Recall	F1-measure
0.8411	0.7576	0.7972

Table 1: Performance of the simple model over the **CoNLL 2009** *development* data.

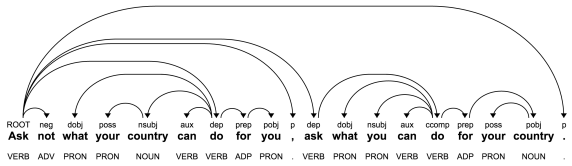
		Predicted Values: *Positive*	Predicted Values: *Negative*
Actual Values	*Positive*	10504	1259
	Negative	1983	180314

Table 2: Confusion Matrix

5 Dependency GCN Architecture

Despite the relatively good results achieved by the model described in section 4, an improvement to the network can still be made by exploiting information about the sentences' *dependency trees* (fig. 2): these are structures that implicitly incorporate a important syntactic and (consequently) semantic information.

Figure 2: Dependency Tree Example.



For this reason a model using a specially designed *GCN*[4] was developed by **Marcheggiani** and **Titov**, built on top of the model presented in section 4:

This *GCN* layer is inserted between the output of the *BiLSTM* (which is now the *GCN*'s input) and the predicate extraction/concatenation phase previously described, now working on the *GCN*'s output.

However, there are some minor problems, namely: how can dependency info be embedded into the network? And what can be made in order to better control the flow of information between edges?

- The solution to the first question consists in the use of *dependency type* specific biases when computing the output of a *GCN* layer.
- The addition of edge-specific gates to the *GCN* answer the second need.

We indicate the *dependency type* associated with an edge (i, j) using the notation $L(i, j)$. It is also useful to differentiate w.r.t. the direction of an edge, due to information flowing in the opposite direction of a syntactic dependency carrying different information than information going in the same direction, so the edge label is direction-dependant. For example, consider the arc between **Ask** and **not** seen in fig. 2:

$$L(\text{Ask}, \text{not}) = \text{neg}_{\rightarrow}$$
$$L(\text{not}, \text{Ask}) = \text{neg}_{\leftarrow}$$

Generally speaking, $L(i, j) = \text{dep-type}_{dir(i,j)}$ with

$$dir(i, j) = \begin{cases} \rightarrow & \text{if } i \text{ is the dep. head} \\ \leftarrow & \text{if } i \text{ is the dependant} \end{cases}$$

The final formulation for the *GCN* layer function used by is then the following:

$$z_i = \text{ReLU}\left(\sum_{j \in N_i \cup \{i\}} g_{j,i}\left(W_{dir(i,j)} x_j + b_{L(j,i)}\right)\right)$$

Where $g_{j,i}$ is a gate that uses edge and vertex information, in order to decide whether to block or allow a certain node. The precise gate math formulation is:

$$g_{j,i} = \sigma\left(\hat{w}_{dir(j,i)} \cdot x_j + \hat{b}_{L(j,i)}\right)$$

5.1 Results

The addition of this supplementary layer allows for better role predictions when using the network, with a drastic increment of the network's *Recall*.

Precision	Recall	F1-measure
0.8524	0.82	0.8359

Table 3: Performance of the *GCN* extended architecture over the **CoNLL 2009** *development* data.

[4] Since *dependency trees* are essentially labeled trees, they can be represented through a *GCN* architecture.

		Predicted Values	
		Positive	*Negative*
Actual Values	*Positive*	11370	1474
	Negative	1968	180217

Table 4: Confusion Matrix

6 Alternative Dependency GCN Architecture

The model described in [3] (section 5) works by creating a deep network composed firstly by a *BiLSTM* layer, then a *GCN* layer and finally a classifier one. Such network exploits the different nature of *BiLSTM* and *GCN* in order to achieve better performance (by complementing layer-specific advantages).

However, in [3] it is never explicitly expressed why the two layer should be stacked together, for example: why not use these two layers parallelly and then leave to the classifier the decision to whether use the information from the *BiLSTM* or from the *GCN*? That is exactly what was experimented during the development of this assignment: an alternative *GCN-BiLSTM* architecture was created and tested, with relatively good results.

The input representation is identical to one described in section 4. However, after the translation into feature vectors the input sequence is given as input (parallelly) to both a *BiLSTM* layer (whose output sequence is indicated by h_{lstm}) and a *GCN* layer (having as output the sequence h_{gcn}). The same predicate extraction and concatenation shown in section 4 is then applied over both h^{lstm} and h^{gcn}.

The major difference of this model is the classifier layer, since it must be able to process the information from both h^{lstm} and h^{gcn}. Indeed, instead of having a single classification layer, two are used (one for the *BiLSTM* and one for th *GCN*) and then a sum (controlled through gates) is applied in order to compute the definite *logits* of the model.

$$z_i^{lstm} = h_i^{lstm} W_{lstm} + b_{lstm}$$
$$z_i^{gcn} = h_i^{gcn} W_{gcn} + b_{gcn}$$

The actual output of the network (i.e. the sequence of *semantic role* probabilities z) is then equal to

$$z_i = \text{Softmax}\big(g_i^{lstm} \cdot z_i^{lstm} + g_i^{gcn} \cdot z_i^{gcn}\big)$$

with g_i^{α} word-specific gates such that

$$g_i^{\alpha} = \sigma(h_i^{\alpha} \hat{w}_{\alpha} + \hat{b}_{\alpha})$$

6.1 Results

This novel architecture achieves even better results than the the previous one (section 5), improving both *Precision* and*Recall* of the system. However, is still quite far from achieving *state-of-the-art* performances.

Precision	Recall	F1-measure
0.861	0.8323	0.8464

Table 5: Performance of the alternative dependency *GCN* model over the **CoNLL 2009** *development* data.

		Predicted Values	
		Positive	*Negative*
Actual Values	*Positive*	11541	1334
	Negative	1863	180291

Table 6: Confusion Matrix

References

[1] Daniel Gildea and Daniel Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288, 2002.

[2] Diego Marcheggiani, Anton Frolov, and Ivan Titov. A simple and accurate syntax-agnostic neural model for dependency-based semantic role labeling. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), Vancouver, Canada, August 3-4, 2017* [2], pages 411–420.

[3] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017* [3], pages 1506–1515.