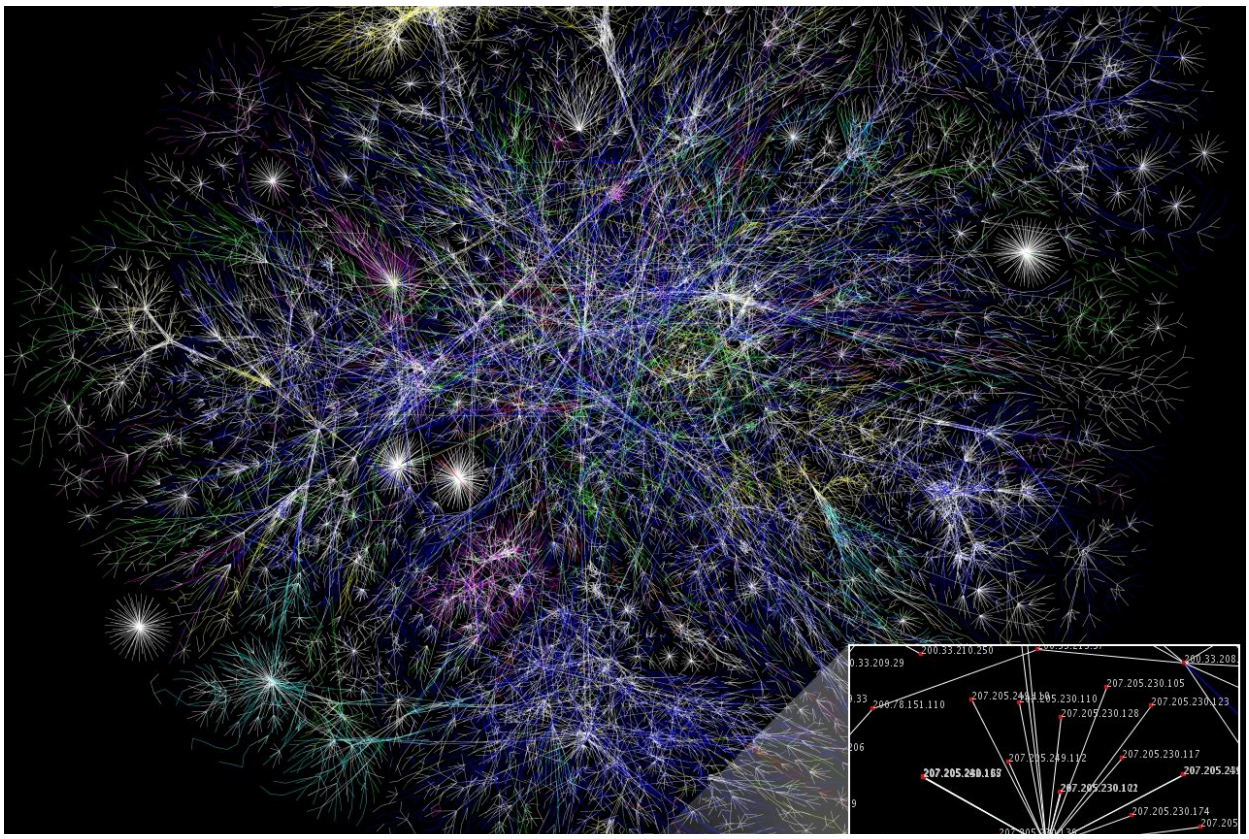


Exploring Semantic role labeling and Predicate Disambiguation with Graph Convolutional Neural Network layer

09/04/2018

Umberto Junior Mele

Sapienza, NLP project
matricola: 1388371

Goal

The aim of this project is to build a deep learning model that automatically make **predicate disambiguation** (ex the cat plays with the ball -> plays.01) and **semantic role labeling** (ex. “the cat plays with the ball -> arg0 [cat]; arg1[ball]).

My proposal models is based on three layers of BiLSTM and one layer of Graph Convolutional Neural Network (following the instruction of Marcheggiani et al. [1]) on top of the BiLSTM last layer concatenation

1) **First deep NN model:** this network has the task to do **predicate disambiguation** and **predicate detection**

2) **Second deep NN Model:** aims to predict the semantic role labeling given the predicate.

Inputs

Using CoNLL 2009 as training data and CoNLL 2009 development for evaluation, I choose to give like inputs of my models GloVe words embeddings, a random initialized lemma, a random initialized part of speech embeddings and two matrices that represent the syntactic parsing of each sentence L and S.

1. **GloVe:** Global Vectors for Word Representation are pre-trained lemma embeddings
2. **Random Lemma:** from an one hot encoded vector is random initialized an embeddings that will be trained during the train session
3. **Random Lemma:** same procedure than random lemma
4. **matrix L:** is the adjacency matrix n by n, with n the maximum number of words in a sentence. This matrix has 1 value in row i and column j if there is a syntactic relation between word[i] and word[j] in the sentence.
5. **matrix S:** is a n by v matrix that count for each word in the sentence how many kind of dependency relationship has word i with label j. (next in the GCN paragraph it will be explained better the implementation).
6. **Sequence length and Predicate counts:** this two are used to make nice filters and reduce the effort of computation.

BiLSTM

To encode the sentence with the aim to preserve neighbors information, I used three layers of Bi-LSTM with dropout 0.2 and 150 units for each. For reasons of speed I choose the class Cudnn LSTM [3] that performs very well on GPUs on the cloud.

The maximum length for a sentence in the train is 144, and the batch size was fixed to 500 (to improve parallelization on gpu).

The outputs of the Bi-Lstm is the concatenation of the forward and the backward outputs.

Graph Convolutional Network layer

The implementation of this layer takes inspiration from Marcheggiani et al. [1], and after some simple counts, is possible to construct a GCN layer using tensor representation.

Lets us see before the node activation function suggested by the paper:

$$h_v^{(k+1)} = ReLU\Big(\sum_{j \in N(i)} W^k_{d(j,i)} \cdot h_j^k + b_{L(i,j)}\Big)$$

where **h** is the vector encoding of the word j on layer k, d(i,j) is the classifier of the direction of the relationship between j and i could be 1 if there is a directed edge, -1 for the inverse and 0 for the loop. So at the end we have just three different **W** : W_1 , W_{-1} , W_0 .

L(i,j) instead is the classifier of the dependency relation between the nodes and can take value from more than 80 different possibilities (ex. NAME, SBJ, P ...), so this means that we add bias for all possible labels.

So this model is amazing! It's simply encode information from neighbors words on the tree syntactic parsing!

But, in this format is difficult to implement the GCN on tensorflow, since it requires tensors!

Then I applied some simple math on that. Since **W** is constant for all nodes given the direction we can use a similar laplacian matrix **L** to encode just the nodes with a directed edge from row i to column j, and simply with the transpose of that we have the oppose directions so: $L^T \cdot H^T$ is a vector that encodes the sum of all features of nodes that come from the node_i, whereas $L \cdot H^T$ encode the sum of all features of nodes that go to the node_i, then $\sum_{j \in N(i)} W^k_{d(i,j)} \cdot h_j^k$ it becomes $L^T H^T W_1 + L H^T W_{-1} + H^T W_0$.

Now it remains the second part, and since $\sum_{j \in N(i)} b_{L(j,i)}$ is simple the sum of all nodes with label L that reach i, we can encode this information in a matrix S with dimension n by 2*max_number_of_l since we want to encode both direction information, and that sum it becomes $S \cdot B$ where B is the variable for bias.

At finally I just did a simple Relu activation function on:

$$H^{(k+1)} = ReLU(L^T H^T W_1 + L H^T W_{-1} + H^T W_0 + S \cdot B)$$

must be mentioned that in the paper there's a further passage using gating constant that aims to reduce the wrongs connections and improve good ones in a very smart way!

Differences between models

I. Predicate Disambiguation System

This model has a multi-task learning goal, one is to detect if the word is or not a predicate and the second one is the predicate disambiguation, my implemented model consists on three BiLSTM layers with the predicate detection task above of them, and a Graph Convolutional Layer over the Bi-LSTMs that produces hidden representation for the predicate disambiguation task.

II. Semantic Role Labeling System

For the Semantic Role Labeling system I trained the same model used for the Predicate tasks but with the additional task of argument detection, more like additional feature is the fact that I used the **concatenation of predicates and the words** hidden representation after the GCN layer, this to make possible to teach to the model the conditional probability of argument given the predicate.

Evaluation metrics

PREDICATE DETECTION		PREDICATE DISAMBIGUATION	
Accuracy	0.9865	Accuracy	0.73899
F1 score	0.7934	F1 score	0.45653
Recall	0.7819	Recall	0.46644
Precision	0.8051	Precision	0.46810

From this tables we can see that predicate detection it works well and also predicate disambiguation it works even if i didn't achieve great results.

SEMANTIC ROLE LABELING	
Accuracy	**0.68**
F1 score	**0.27**
Recall	**0.27**
Precision	**0.27**

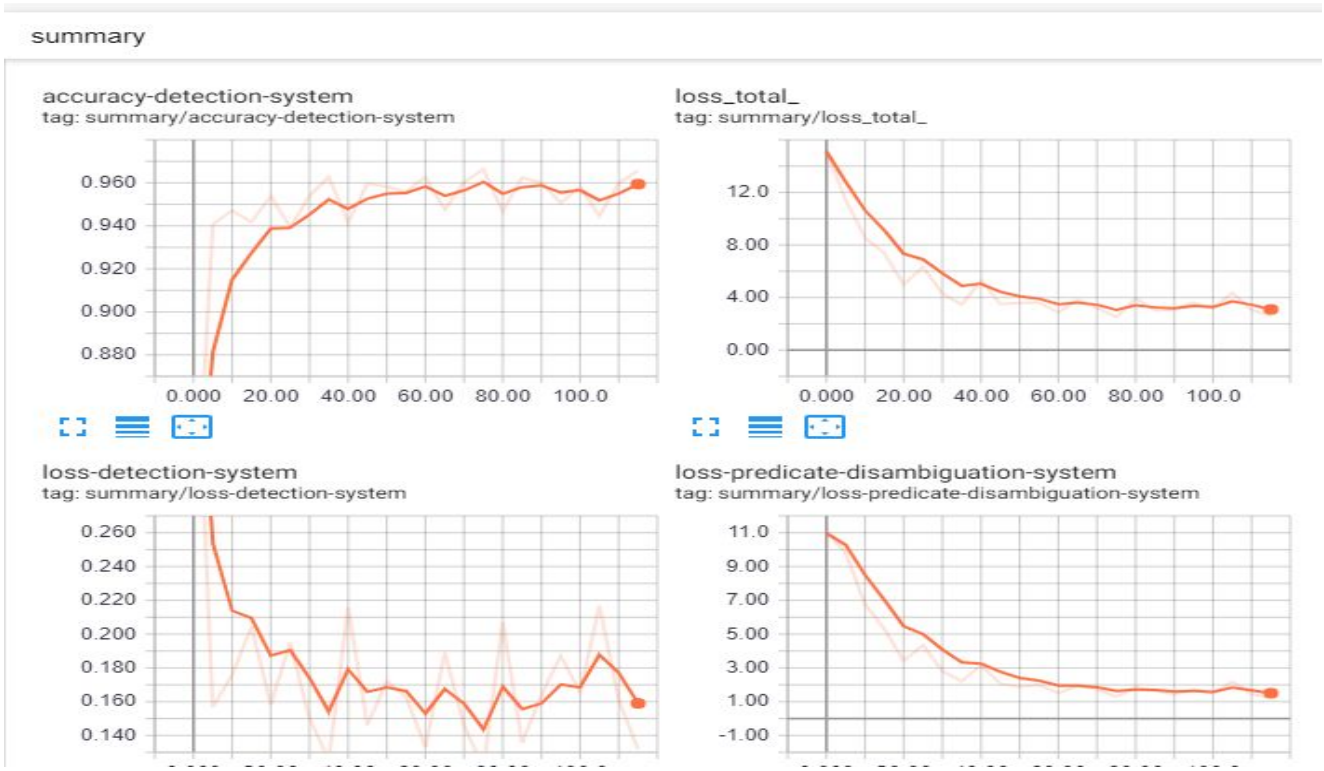
Instead SRL system doesn't work so well, my model has a very low F1 score this means that there is some errors on my model implementation, since I used some tensor manipulation tensorflow function for the concatenation task I think that I made errors there or probably I made some mistakes on the implementation of GCN layer, or in the data processing.

Conclusion and Improvements

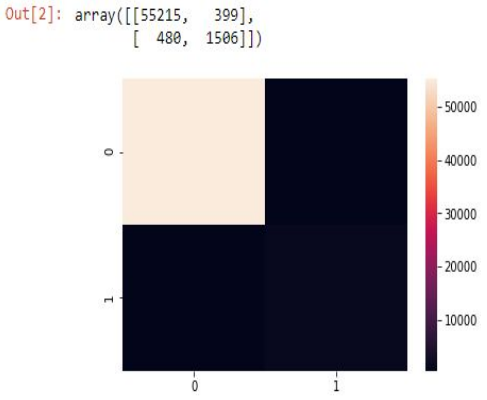
To improve this models it could be done many things, first of all fix the implementation of the GCN layer. Then could better to increase the dataset using any tools we have for example predicate alignment with other datasets using WSD models trained on BabelSynsets to increase the dataset for predicate disambiguation. Another improvement could be adding on input sense embeddings and synsets embeddings. Also interesting is to use edge-wise gating on the GCN to reduce the noise due to automatic dependency relation on syntactic parsing.

Images, Tables and Graphs

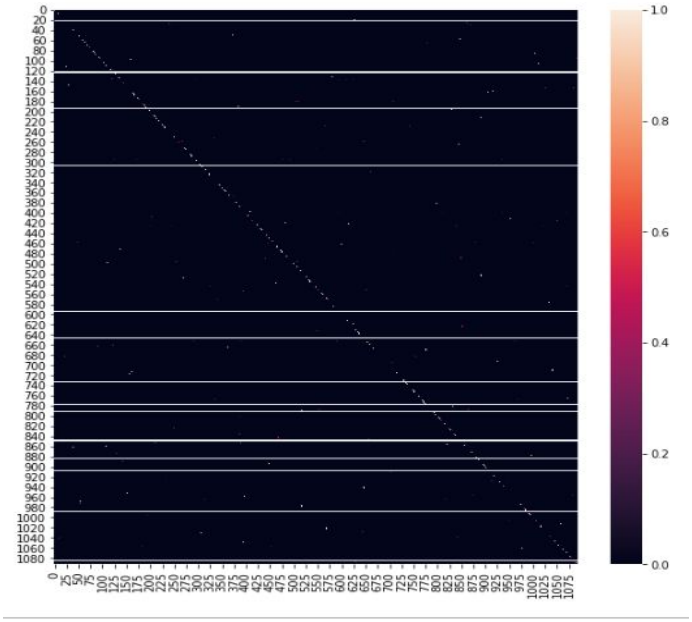
#tensorboard screenshot for first model metrics on evaluation dataset



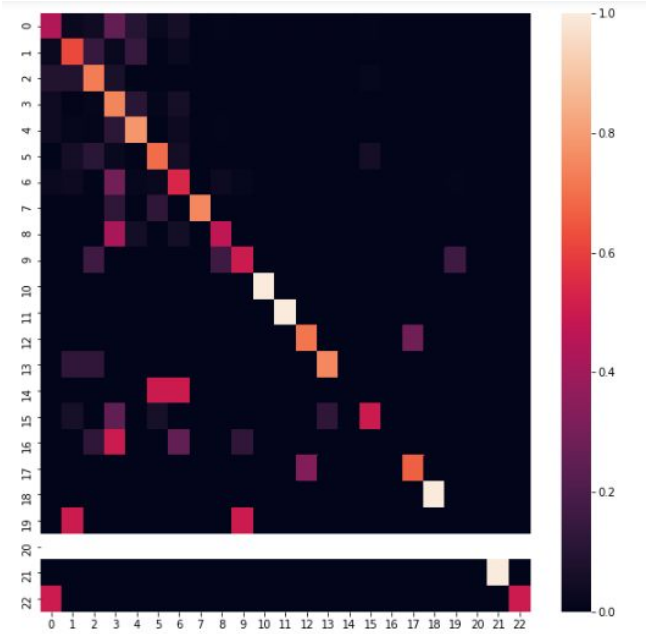
predicate detection confusion matrix



predicate disambiguation cm



Confusion matrix on arguments



References

[1] Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling

[2] Global Vectors for Word Representation

[3] Cudnn LSTM