KU Leuven - CSAI

Exercise Session 2
Syntactic Parsing and Semantic Role Labeling with a GNN

For questions contact: Toledo Forum for Questions (under Discussions) or nlp@ls.kuleuven.be

Ruben Cartuyvels, Victor Milewski and Marie-Francine Moens

In this session, we discuss syntactic constituency parsing and dependency parsing as well as Semantic Role Labelling (SRL). Background can be found in chapters 17, 18 and 19 of ”Speech and Language Processing” by Dan Jurafsky and James H. Martin. The 2023 draft is available online at https://web.stanford.edu/~jurafsky/slp3/. You can also refer to chapters 10 and 11 of the book “Introduction to Natural Language Processing” by Jacob Eisenstein.
While Jurafsky and Martin explain context-free grammars and CKY parsing, they do not explain the inside/outside algorithms for their unsupervised induction. For that, we recommend you to read chapter 11 of the book by Manning and Schütze: Foundations of Statistical Natural Language Processing (https://nlp.stanford.edu/fsnlp/).
For the GNN and more specifically GCNs, you can look at the original blog post by the author of the GCN paper: https://tkipf.github.io/graph-convolutional-networks/.
Here is also a short youtube video describing and visualizing the basics of a GCN: https://youtu.be/2KRAOZIULzw. Note that his aggregation and update of the layers are a bit different. But the visualization is great for understanding the concept of Graph Neural Networks. Read the questions carefully so you use the correct inputs and outputs and do the correct updates.

1 Parsing Context-Free Grammars

In this exercise, we cover the classical CKY (Cocke-Kasami-Younger, sometimes also reffered as CYK) algorithm for parsing a context-free grammar (CFG).

Question 1.A

We consider the following CFG:

$$\begin{array}{rl}
\text{S} & \rightarrow \text{NP} \quad \text{VP} \\
\text{NP} & \rightarrow \text{D} \quad \text{N} \quad | \quad \text{NP} \quad \text{PP} \quad | \quad \text{Pro} \\
\text{VP} & \rightarrow \text{V} \quad \text{NP} \quad | \quad \text{V} \quad \text{NP} \quad \text{PP} \\
\text{PP} & \rightarrow \text{P} \quad \text{NP} \\
\text{D} & \rightarrow \textit{the} \quad | \quad \textit{a} \\
\text{P} & \rightarrow \textit{with} \\
\text{Pro} & \rightarrow \textit{john} \\
\text{N} & \rightarrow \textit{cat} \quad | \quad \textit{tree} \quad | \quad \textit{binoculars} \\
\text{V} & \rightarrow \textit{sees} \quad | \quad \textit{climbs}
\end{array}$$

1. Think of two sentences that would be grammatical according to this grammar.

Question 1.B

We would like to use this grammar to parse sentences using the CKY algorithm. However, for CKY the grammar needs to be in Chomsky Normal Form (CNF). Convert the grammar given above into CNF.

Question 1.C

Now that we have our grammar in CNF, we could parse sentences using CKY. Find **all the trees** for the following sentences:

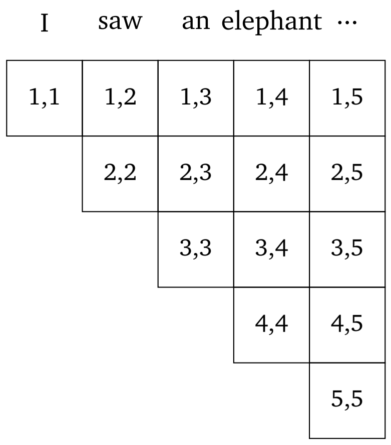


Figure 1: Chart for exercise in section 2

1. John climbs the tree.
2. The cat sees the tree with the binoculars.

Question 1.D

If everything went correctly, you found two possible trees for sentence 2.
This type of ambiguity is called PP-attachment ambiguity. Make use of the difference in trees to describe what this ambiguity is and when it occurs.

2 Self-supervised latent trees

The algorithm seen in the lecture slides (slides 38-48, as proposed by Drozdov et al. in https://aclanthology.org/N19-1116/) is a modified version of the inside/outside algorithm for probabilistic context-free grammars as explained in Manning and Schütze. First, the algorithm in the slides models a simplified PCFG, with no more than 1 nonterminal symbol. Second, the algorithm in the slides deals with additive scores, which we can see as the logarithms of the probabilities that Manning and Schütze work with, so the products of probabilities in their book become additions of log-probs in the slides. Finally, the algorithm in the slides does not only apply the recursion to scalar scores, but runs in parallel a similar recursion on vector representations of tokens and spans of tokens. It also uses these representations as input for the score calculations (which has a computational cost, of course). As a result, the specific computations in an iteration of the recursion are different, but the structure of the recursion remains identical to the well known inside/outside algorithms explained by Manning and Schütze.

2.1 Inside pass

Starting from the following log-probs, compute the inside (log-)scores for every span in the sentence, according to the algorithm in slide 44. Use a chart like the shown in figure 1, and iterate over the cells as indicated by the right half of the figure on slide 21. The scores $s^{in}_{i,i}$, so for single token spans, are given by the log-probs below. Ignore the vector representations ($h^{in}_{i,j,k}$) as well as the term that these representations occur in, in the formula for the scores ($(h^{in}_{i,k})^T W h^{in}_{k+1,j}$). In every iteration, use $s^{in}_{i,j} = \sum_k a_{i,j,k} \cdot s^{in}_{i,j,k}$, so a subtree's score is the weighted sum of the scores of the split points, and

$$a_{i,j,k} = \frac{s^{in}_{i,j,k}}{\sum_l s^{in}_{i,j,l}} .$$

Sentence: “I saw an elephant”. Log-probs:

I	-1
saw	-2
an	-5
elephant	-7

2.2 Outside pass

Now compute the outside scores for every span, starting from the top-right cell (filled with value $= 0$, representing the entire sentence) of a new chart, using the formula on slide 45. Proceed in the opposite order of the order you used for CKY and the inside pass. Again, ignore the vector representations and the terms where they occur. You'll need to make use of the filled in chart of inside scores. Give the outside scores for each of the tokens of the sentence.
This time, in every iteration, use $s_{i,j}^{out} = \sum_k s_{i,j,k}^{out}$ (so ignore the weighting factors $a_{i,j,k}$).

2.3 Maximum score tree

Now find the tree with the maximum inside score, by running the CKY recursion on inside scores. Instead of $s_{i,j}^{in} = \sum_k s_{i,j,k}^{in}$, use $s_{i,j}^{in} = \max_k s_{i,j,k}^{in}$. What do you notice? What prevents this from happening in the algorithm on the slides?

3 Using a dependency tree in a Graph Neural Network

In this exercise we work with a simplified model from the "Graph Convolution over Pruned Dependency Trees Improves Relation Extraction" paper [1]: `https://aclanthology.org/D18-1244/`.

We will see how to predict a dependency tree in Question 4. Such a tree allows us to get insight into the meaning of the sentence. In this question, we will see how we can make use of a dependency tree to make other predictions for the sentence.
We are going to do semantic role labeling (SRL). We define a very simplistic model. The definition of the model is:

$$\hat{\boldsymbol{y}} = \text{softmax}(F_{pred}(\boldsymbol{h}))$$

Where $\boldsymbol{h} \in \mathbb{R}^e$ is the vector representation of a word, $\hat{\boldsymbol{y}} \in \mathbb{R}^l$ the predicted label probabilities, p the boolean value indicating if the word is the predicate or not, and F_{pred} is the prediction function or model.
Our model consists of a GCN and as the final step we use a linear layer function with weight matrix $\boldsymbol{W}_{pred} \in \mathbb{R}^{m \times l}$ and biases $\boldsymbol{b}_{pred} \in \mathbb{R}^m$.
We define e as the embedding size, l as the number of labels, and m as the number of hidden dimensions.
The possible labels are a reduced version of the prob bank: `<ARG0, ARG1, O>` with ARG0 the agent, ARG1 the patient, and O not part of a role.

There are many options for $\boldsymbol{h}$, such as pretrained word embeddings (word2vec, BERT) or a model like a bi-LSTM. However, we want the features to be strongly informed by grammatical relations between words. So we use a GCN over the dependency relations, to update the hidden states before passing them through the prediction layer. In addition to the dependency relations, we **also use bidirectional sequence relations**: each word has an additional incoming edge from the previous word and from the following word.

We don't have edge labels, so we don't use them. We define the **direction of the edge as two different labels** (incoming/outgoing), so the model can learn a distinction between incoming and outgoing dependencies.

Here is the definition of the GCN (taken and adapted from Equation 1 from the paper mentioned above):

$$\boldsymbol{h}_i^{(t+1)} = \text{ReLU}\big(\boldsymbol{h}_i^{(t)} + \sum_{n \in \mathbb{N}(i)} (\boldsymbol{W}_{in}\boldsymbol{h}_n^{(t)} + \boldsymbol{b}_{in}) + \sum_{m \in \mathbb{M}(i)} (\boldsymbol{W}_{out}\boldsymbol{h}_m^{(t)} + \boldsymbol{b}_{out})\big)$$

With $\mathbb{N}(i)$ and $\mathbb{M}(i)$ the set of nodes with incoming and outgoing edges to and from the current node i, respectively. $\boldsymbol{W}_{in} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{W}_{out} \in \mathbb{R}^{m \times m}$ are the weight matrices for these incoming and outgoing

neighbors, and $\boldsymbol{b}_{in} \in \mathbb{R}^m$ and $\boldsymbol{b}_{out} \in \mathbb{R}^m$ are the biases for incoming and outgoing neighbors. m is the hidden size.
We chose the hidden size between timesteps (or GCN layers) to be the same. This makes it possible to reuse the same weight matrices for multiple layers. It is not mandatory to keep the hidden size equal. In fact, for convolutions, it is common to reduce the size after every layer. In the literature, you also see methods using different weight matrices for every layer. In this question, we simply use a single timestep (or a single application of a GCN layer).

NOTE: In this exercise we have multiple types of connections, without distinction between them (so we don't use their labels and don't apply different weights). This can result in multiple identical connections between words.

Question:

Predict the labels of each word using the SRL model with GCN for the sentence:

the cat sees the tree with the binoculars

As a dependency tree, use the prediction from Question 4. The predicate of this sentence is **sees**.

Use the given weights and biases below. We also provide the initial embeddings for the words in the sentence in matrix $\boldsymbol{E} \in \mathbb{R}^{|V| \times 2}$, where $|V|$ is the size of the vocabulary.
The vocabulary is:

```
['binoculars', 'cat', 'sees', 'the', 'tree', 'with']
```

Use the following steps to solve the question:

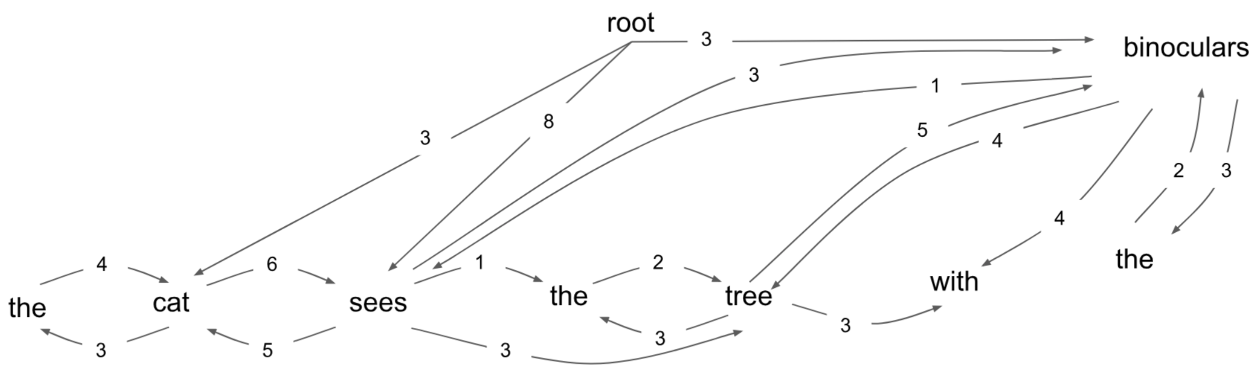
1. For each node, define the two sets of neighbors, incoming and outgoing. Duplicates are allowed.
2. Define $\boldsymbol{h}^{(0)}$ as the 8×3 matrix that gathers every word's (node's) embedding from $\boldsymbol{E}$ and appends the binary predicate value p as the last column. Do one GCN step and update the hidden state for all the nodes in the graph to $\boldsymbol{h}^{(1)}$.
3. Use $\boldsymbol{h}^{(1)}$ to predict the role label for each of the words in the sentence.

$$\boldsymbol{E} = \begin{bmatrix} -0.5 & 0.7 \\ 1.4 & 1.1 \\ 0.8 & -0.5 \\ -0.3 & 0.1 \\ 1.3 & 0.8 \\ 0.8 & -0.8 \end{bmatrix} \quad \boldsymbol{b}_{in} = \begin{bmatrix} 0.6 & 3. & -1. \end{bmatrix}^T \quad \boldsymbol{b}_{out} = \begin{bmatrix} -0.7 & 1. & 0.2 \end{bmatrix}^T \quad \boldsymbol{b}_{pred} = \begin{bmatrix} 1. & -0.5 & -0.5 \end{bmatrix}^T$$

$$\boldsymbol{W}_{in} = \begin{bmatrix} -0.5 & 0. & 0.6 \\ -2. & 0.5 & 1. \\ 2. & -0.7 & -0.8 \end{bmatrix} \quad \boldsymbol{W}_{out} = \begin{bmatrix} 0.1 & -1. & 1. \\ -0.7 & -2. & 2. \\ 1. & 0.3 & -0.8 \end{bmatrix} \quad \boldsymbol{W}_{pred} = \begin{bmatrix} 3. & -2. & 0.2 \\ 0.1 & 2. & -3. \\ 0.7 & -2. & 2. \end{bmatrix}$$

4 Dependency Parsing (optional)

In this question, we cover a classical algorithm for edge-factored dependency parsing models. We assume we already have an edge scoring model (e.g., a neural network). Consider the following scores given by our model:



We get a dependency tree by computing the *maximum spanning tree* for the graph above.

a. Use the Chu-Liu Edmonds (CLE) algorithm to get the maximum spanning tree. This algorithm is explained in the exercise session slides (not the lecture slides).

b. To which of the two meanings (cfr. exercise 1) does this dependency parse correspond?

5 Complexity (optional)

Let us consider the CLE algorithm from the previous question. We were able to compute the correct dependency tree using this method for the example sentence. But what is the complexity of this algorithm? Use the big O notation to define this complexity.
We assume that we always start from a fully connected graph, which has M nodes. We also assume that there are R types of relations possible. Before you start, consider what such a graph will look like.
We will do this in steps:

a. First define the time complexity to find all the best incoming edges for all nodes.

b. Next define the complexity for the number of cycles needed.

c. What is the overall time complexity?

References

[1] Yuhao Zhang, Peng Qi, and Christopher D. Manning. "Graph Convolution over Pruned Dependency Trees Improves Relation Extraction". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. by Ellen Riloff et al. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 2205–2215. DOI: 10.18653/v1/D18-1244. URL: https://aclanthology.org/D18-1244.