Deep Learning in NLP

Practical Session P12

Solution to be sent (zip archive with the code, PDF with additional notes when needed) to `waszczuk@phil.hhu.de` and `cwurm@phil.hhu.de` by the end of January.

Introduction

Our next goal is to extend the implemented LSTM-based tagger to predict POS tags jointly with universal dependencies. The plan is to implement a simplified version of the parser described in `https://arxiv.org/pdf/1611.01734.pdf`.

As usual, you can download the current version of the code (including the English dataset and the corresponding fastText embeddings) from the course's webpage. Additionally, have a look at the description of the most recent code modifications, introduced in order to enable the prediction of dependency heads.

Model

The POS tagging model remains the same: (i) each word is mapped to the corresponding (fastText) word embedding, (ii) BiLSTM is used to contextualize the word embeddings, and (iii) a linear layer is used to score the contextualized embeddings.

We add a dependency parsing component on top of the BiLSTM embeddings. This in particular means that the same contextualized representations will be used for both POS tagging and dependency parsing. This is, by the way, an example of multi-task learning.

Each dependency structure in UD is a connected tree with a single root.[1] This means that each word has exactly one dependency head (in the conllu files, the heads are specified in the 7-th column). Therefore, the task of the dependency tree prediction is not unlike POS tagging: for each word in the sentence, we have to predict (instead of the corresponding POS tag) the index of the corresponding head word.

Scoring. Predicting the head indices in exactly the same way as POS tags wouldn't work particularly well for several reasons (can you think of any?). We will use the following model instead. Let $(x_i \in \mathbb{R}^d)_{i=0}^n$ be a sequence of contextualized word embeddings (i.e.,

[1] There are also the so-called enhanced dependencies, but we won't work with them.

BiLSTM output), where n is the sentence length and d is the embedding size.[2] We define two functions:

- $H : \mathbb{R}^d \to \mathbb{R}^d$: the *head representation* of the given word embedding
- $D : \mathbb{R}^d \to \mathbb{R}^d$: the *dependent representation* of the given word embedding

Both functions keep the embedding size d of the input vectors for simplicity. We then define the *score* of the word at position j being the head of the word at position i using dot product $(\cdot)$:

$$score(i, j) = D(x_i) \cdot H(x_j) \tag{1}$$

Since $v \cdot w = \|v\| \, \|w\| \cos(\theta)$, where θ is the angle between v and w, the score is proportional to the cosine of the angle between the head representation of j and the dependent representation of i. Put differently: the closer (and larger) the two vector representations, the higher the resulting score.[3]

Prediction. Once we determine the head/dependent scores for a given sentence of length n, we can simply pick the head for each word $i = 1..n$ so as to maximize the scores:

$$head(i) = \arg\max_{j=0..n} score(i, j) \tag{2}$$

Our goal is to implement the above-described model. Consider the following questions:

- The description doesn't specify the form of the functions H and D. Clearly, we will implement them as neural modules. What could be a simple/reasonable choice?
- Eq. 1 is defined for a single dependent i and a single head j. Is it possible to vectorize this equation so that it applies to all $i = 1..n$ and $j = 0..n$ at once?
- **NEW 21.02.2020**: The two points above relate to the extention of the model to a joint POS tagger and dependency parser. Is there anything else that needs to be adapted in the code in order to make it work on the given English dataset? Consider the current training procedure invocation.
- For a given head representation $H(x_j)$, we may want to introduce the bias of the word j being a head, regardeless of the dependent we are trying to match it with. This could help to learn that, e.g., *the* is virtually never a head of any other word. How could we modify Eq. 1 to introduce such a bias?
- Does prediction with this model guarantee that the output dependency structure is a tree in the graph-theoretical sense? If not, what can we do about this?

[2] Note the special embedding x_0, which corresponds to the dummy root at position 0. We use it in order to transparently handle the root of the tree.

[3] We could also enforce that both representations have unit length. Only the angle between the representations would matter then.