Parsing mit neuronalen Netzen: ???

In den folgenden Übungen werden Sie einen Parser auf Basis von neuronalen Netzwerken implementieren. Der Parser orientiert sich an dem folgenden Artikel von Gaddy et al. (2018): `https://arxiv.org/pdf/1804.07853.pdf`

Vorverarbeitung

Sie finden die Parsebäume, mit denen der Parser trainiert werden soll in `http://www.cis.uni-muenchen.de/~schmid/lehre/data/PennTreebank.zip`.

Wir müssen diese Parsebäume zunächst vorverarbeiten.

Beispiel:

Für die Eingabe

```
(TOP(S(NP(NNP Ms.)(NNP Haag))(VP(VBZ plays)(NP(NNP Elianti)))(. .)))
```

soll zurückgegeben werden:

- die **Wortliste**: `["Ms.", "Haag", "plays", "Elianti", "."]`
- die **Konstituentenliste**: `[("TOP=S",0,5), ("NP",0,2), ("NNP",0,1), ("NNP",1,2), ("VP",2,4), ("VBZ",2,3), ("NP=NNP",3,4), (".",4,5)]`

  `TOP=S` und `NP=NNP` ergeben sich durch Eliminierung von Kettenregeln, die der Parser nicht verarbeiten kann.

Die folgenden Grammatikregeln erzeugen alle wohlgeformten Parsebäume:

tree → '**(**' **label** (**tree**+ | ' ' **label**) '**)**'
label → [**A-Za-z0-9...**]+ *(Folge beliebiger Zeichen außer '(', ')' und Whitespace)*

Schreiben Sie eine Funktion `read_tree` für die erste Regel und eine Funktion `read_label` für die zweite Regel im Stil eines Recursive-Descent-Parsers. `read_tree` analysiert den Parsebaum und ruft dabei `read_label` und rekursiv sich selbst auf. Dabei erstellt `read_tree` die Wortliste und die Konstituentenliste. Anschließend fasst es noch die Konstituenten mit gleichem Span zusammen.

Vorüberlegungen

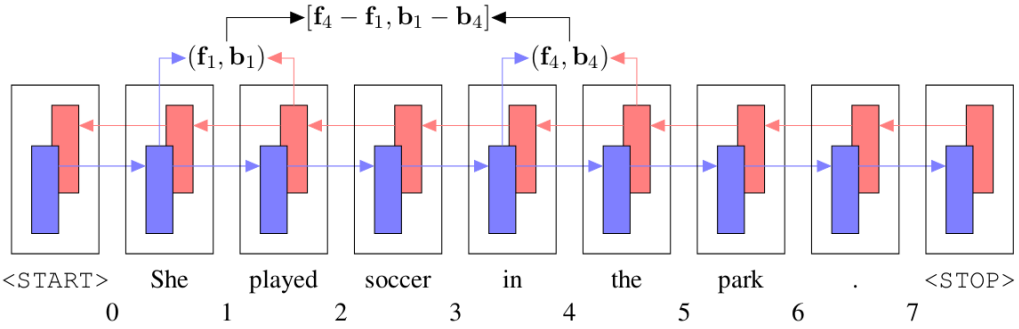
- Welche Argumente benötigen die Funktionen und was liefern sie zurück?
- Wie können Sie die Start- und Endposition einer Konstituente ermitteln?
- Wie können Sie mit zusätzlichen Leerzeichen in der Eingabe umgehen und Indexfehler beim Zugriff auf den Eingabestring vermeiden?
- Wie können Sie die Kettenregeln entfernen?

- Wie gehen Sie mit Fehlern in der Eingabe um?
- Wie kann der Parsebaum rekonstruiert werden?

Neuronales Netz

Das neuronale **Netzwerk** erledigt folgende Aufgaben:

- Berechnung der **Wortrepräsentationen**: Das neuronale Netz berechnet zunächst buchstabenbasierte Wortrepräsentationen mit einem BiLSTM und dann Repräsentationen für alle Satzpositionen mit einem zweiten BiLSTM. Sie können dafür den Code aus der Aufgabe *LSTM-Tagger* wiederverwenden.
- Den Ausgabetensor des zweitem BiLSTMs teilen Sie mit der `chunk`-Methode in zwei Hälften. Die erste Hälfte liefert die Forward-Repräsentation, die zweite Hälfte die Backward-Repräsentation. Damit Sie mit diesen beiden neuen Tensoren – wie im Schaubild gezeigt – die Span-Repräsentationen berechnen können, entfernen Sie beim forward-Tensor das letzte Element und beim backward-Tensor das erste Element.



  Nun berechnen Sie durch Subtraktion der Startrepräsentationen aller Spans von ihren Endrepräsentationen die Spanrepräsentationen. Die PyTorch-Methode `triu_indices` mit Offset 1 liefert Ihnen die Start- und Endpositionen aller Spans. Mit `forward[endpositions] - forward[startpositions]` und `backward[startpositions] - backward[endpositions]` können Sie dann alle forward- und backward-Differenzvektoren parallel berechnen. Die Differenzvektoren konkatenieren Sie.

  Da Sie für die Spanberechnungen auch Repräsentationen für die Positionen 0 und Satzlänge+1 brauchen, müssen Sie am Anfang und Ende des Eingabe-Tensors des zweiten BiLSTMs einen Nullvektor als Dummy-Eingabe hinzufügen. Dafür verwenden Sie am besten die PyTorch-Funktion `pad`.

- Berechnung der **Bewertungen** der Kategorien: Ein Feedforward-Netzwerk mit einer Hidden Layer (analog dem des LSTM-Taggers) transformiert jede Spanrepräsentation in einen Vektor, der für jede syntaktische Kategorie einen Score liefert. Die Größe der Hidden Layer sollte frei wählbar sein. Die Bewertung des Labels "keine Konstituente" (mit der ID 0) sollte überall auf 0 gesetzt werden.

In dieser Übungsaufgabe implementieren Sie noch nicht das Gesamtsystem, sondern nur das neuronale Netzwerk. Sie können also noch nicht auf realen Daten trainieren.

Aufruf: python model.py

Vorüberlegungen

- Wie **gliedern** Sie das Netzwerk sinnvoll in Teilnetzwerke?
- Welche Argumente sollten die **Konstruktoren** der Teilnetzwerke erhalten?
- Welche Argumente sollten die **Forward**-Funktionen der Teilnetzwerke erhalten und welche Rückgabewerte sollten sie liefern?
- Wo sollte **Dropout** angewendet werden?
- Wie können Sie ihr Netzwerk separat **testen**?

Anmerkung: Aus Effizienzgründen sollten Sie möglichst viel durch Verwendung von Tensor-Operationen parallel verarbeiten. (Eine parallele Verarbeitung mehrerer Sätze müssen Sie aber nicht versuchen.)