Parsing mit neuronalen Netzen: Training

Nun sollen Sie das Parser-Training implementieren.

Korrigieren Sie zunächst die Fehler in Ihrem Code aus der letzten Aufgabe mit Hilfe der erhaltenen Korrekturvorschläge.

Erstellen Sie dann eine Datei **train-parser.py**, welche eine Klasse `Data` für die Datenvorverarbeitung und Ihre Klasse `Parser` für das Netzwerk importiert. Die Klasse `Data` enthält Funktionen zur Vorverarbeitung der Daten. Sie können den Code an der Adresse `https://www.cis.uni-muenchen.de/~schmid/lehre/Experimente/data/Code-Aufgabe10.zip` herunterladen.

Schreiben Sie nun eine Funktion **train**. Erzeugen Sie darin zunächst ein Objekt der Klasse `Data` mit dem Befehl

```
data = Data(path_train, path_dev)
```

Dann trainieren Sie für n Epochen (bspw. n=50) auf den Trainingsdaten `data.train_parses`, die Sie zu Beginn jeder Epoche mit `random.shuffle` umordnen. `data.train_parses` ist eine Liste von Parsebäumen. Jeder Parsebaum ist ein Paar bestehend aus der Liste der Wörter und der Liste der Konstituenten. Jede Konstituente ist ein Tripel (Label, Startposition, Endposition). Mit der Methode `data.labelID(label)` können Sie die Labels auf IDs abbilden. Mit der Methode `data.words2charIDvec(words)` können Sie eine Liste von Wörtern in eine Liste von Suffixen und eine Liste von Präfixen konvertieren, wobei jedes Suffix/Präfix aus einer Liste von Buchstaben-IDs besteht. In der Datei `Data.py` ist auch der Index der Spanklasse "keine Konstituente" in der Variablen `NoConstLabelID` verfügbar. Die Methoden `data.num_char_types()` und `data.num_label_types()` liefern Ihnen die Zahl der Zeichen und die Zahl der syntaktischen Labels.

Das Netzwerk aus der letzten Aufgabe gibt Ihnen einen 2-dimensionalen Tensor zurück, der für jeden Span eine Liste von Scores liefert. An Position 0 des Vektors befinden sich die Bewertungen für den Span (0,1). Dann folgen (0,2),...,(0,n),(1,2),...,(1,n),...,(n-1,n), wobei n die Zahl der Wörter ist. Um das Loss für diesen Score-Tensor zu berechnen, müssen Sie einen Vektor `labelIDs` erstellen, der für jeden Span die korrekte Label-ID enthält.

Gehen Sie dabei so vor, dass Sie zunächst einen 1-dimensionalen Tensor der richtigen Länge mit `NoConstLabelID` füllen und in `labelIDs` speichern. Dann erzeugen Sie ein Dictionary `spanID`, welches jeden Span (s, e) auf seine Position abbildet, wobei der Span (0, 1) die Position 0 hat. (Am besten erzeugen Sie dazu zunächst mit `combinations` die Liste der Spans.) Dann iterieren Sie über alle Goldstandard-Konstituenten (l, s, e) des aktuellen Parsebaumes und setzen mit dem Befehl `labelIDs[spanID[s,e]] = l` das Label des Spans (s, e) auf die Label-ID l.

Als Trainingskriterium verwenden Sie das `CrossEntropyLoss` von PyTorch. Wenden Sie **Gradient Clipping** (bspw. mit einer Gradient-Norm von 1) an, um extrem große Gradienten zu vermeiden. Nach jeder Epoche berechnen Sie die Zahl der falsch gelabelten Spans in den Development-Daten `data.dev_parses`. Wenn die

Zahl der Fehler die bisher kleinste war, speichern Sie das Netzwerk mit der Methode `torch.save`. Außerdem müssen Sie die Methode `data.store_parameters` aufrufen, um die Tabellen für die Abbildung von Buchstaben und Labels auf Zahlen-IDs zu speichern. Speichern Sie die Tabellen und das Netzwerk in separaten Dateien mit gleichem Basisnamen und unterschiedlichen Dateiendungen (.io bzw. .pt) speichern.

Bei einer guten Implementierung kann die Zahl der falsch gelabelten Konstituenten in den Development-Daten im Laufe des Trainings unter 6000 sinken. Planen Sie genug Zeit für das Training ein, da es mehrere Stunden dauern kann.

Aufruf: python train-parser.py train-parses dev-parses parfile

Bitte schicken Sie mir den kompletten Code, der zur Ausführung notwendig ist, und eine Datei *num-errors.txt* mit der Anzahl der Fehler nach jeder Epoche.