LSTM-Wortart-Tagger (Teil 2)

In dieser Übung werden Sie den Wortart-Tagger fertigstellen.

Schreiben Sie ein Programm `rnn-train.py`, welches das Training implementiert. Es erzeugt ein Objekt der Klasse `Data`, um die Daten einzulesen, und ein Objekt der Klasse `TaggerModell`. Dann iteriert es `numEpochs`-mal über die Trainingsdaten, und trainiert nacheinander auf jedem Trainingssatz. Sie sollten die Trainingsdaten vor jeder neuen Epoche zufällig umordnen mit dem Befehl: `random.shuffle(data.train_sentences)`

Nach jeder Epoche wird über alle Entwicklungsdaten iteriert, um die Tagging-Genauigkeit zu berechnen und auszugeben. Da Sie Dropout verwenden, müssen Sie mit Hilfe der Methode `model.train(True/False)` zwischen Trainings- und Evaluierungsmodus umschalten. Wenn die neue Genauigkeit höher als alle bisher erzielten Genauigkeiten ist, wird das Modell mit dem Befehl

```
torch.save(model, args.parfile+'.rnn')
```

gespeichert.

Fügen Sie außerdem eine Methode `store_parameters(args.parfile+'.io)` zum Modul `Data.py` hinzu, welche die Tabellen speichert, die für die Abbildung zwischen Wörtern und Zahlen benötigt werden. Erweitern Sie die Konstruktor-Methode `__init__` so, dass sie im Falle des Aufrufes mit nur einem Argument (Taggingmodus) mit der Methode `init_test` die gespeicherten Tabellen einliest, und andernfalls den ursprünglichen Konstruktor aufruft, der nun `init_train` heißt:

```
def __init__(self, *args):
   if len(args) == 1:
      self.init_test(*args)
   else:
      self.init_train(*args)
```

Programmaufruf:

```
./rnn-train.py trainfile devfile paramfile --num_epochs=20 --num_words=10000
      --emb_size=200 --rnn_size=200 --dropout_rate=0.5 --learning_rate=0.5
```

Die Kommandozeilenargumente verarbeiten Sie am besten mit dem Modul `argparse`.

Wenn Ihr Rechner eine gute Grafikkarte besitzt, können Sie diese nutzen, indem Sie mit dem Befehl `model = model.to(device)` das Modell auf GPU verschieben. Die globale Variable `device` definieren Sie einmal zu Beginn mit dem Befehl

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Schließlich müssen Sie noch ein Programm `rnn_annotate.py` schreiben, welches mit den Befehlen

```
data  = Data(args.path_param+".io")   # read the symbol mapping tables
model = torch.load(args.path_param+".rnn")   # read the model
```

die Abbildungstabellen und das neuronale Netzwerk einliest und dann die zu annotierenden Daten Satz für Satz einliest (Methode `sentences` unten) und die Wörter und ihre besten Tags ausgibt.

Zum Modul `Data.py` fügen Sie zwei neue Methoden hinzu:

`sentences(filename)` ist ein Generator, der eine Datei einliest und mit dem Befehl `yield` nach dem Einlesen eines Satzes die Liste der Wörter zurückgibt. Jede Zeile der Datei enthält einen bereits tokenisierten Satz.

mögliche Parameterwerte:

Vokabulargröße: 10000-50000

Embeddings-Größe: 100-300

RNN-Größe: 200-500

Optimierer: Adam mit Lernrate circa 0.001 oder SGD mit Lernrate circa 0.5

Bitte geben Sie Ihren Code und eine Datei mit Angaben zur erzielten Genauigkeit auf den Development-Daten ab.

Vorüberlegungen:

- Welche Schritte müssen in den Programmen `rnn-train.py` und `rnn-annotate.py` ausgeführt werden?

Debugging:

Wenn Sie die Ursache eines PyTorch-Fehlers nicht finden, können Sie so vorgehen:

- Prüfen Sie die Fehlermeldung. Oft weist ein Eingabetensor falsche Dimensionen oder einen falschen Datentyp auf oder die Argumente befinden sich nicht alle auf der GPU. PyTorch sagt Ihnen dies dann. Wenn sich die Fehlermeldung auf einen Befehl in einer PyTorch-Bibliothek bezieht, müssen Sie nachschauen, welcher Befehl Ihres Codes zuletzt ausgeführt wurde. Prüfen Sie die Argumente dieses Befehles.
- Wenn das Programm einfach abstürzt, dann müssen Sie zuerst herausfinden, an welcher Stelle Ihres Programms das Problem auftaucht (z.B. durch Kontrollausgaben).
- Prüfen Sie dann, ob die Argumente der Operation die richtigen Tensorgrößen aufweisen.
- Bei Indexoperationen auf Tensoren sollten Sie prüfen, ob der Index kleiner als die Tensorgröße ist. Dasselbe gilt für Embedding-Operationen. Ungültige Indizes können vor allem in Verbindung mit der GPU zu großen Debuggingproblemen führen.
- Wenn Sie Fehlermeldungen von Cuda oder CuDNN bekommen, hilft Ihnen vielleicht auch ein Test auf CPU weiter.