Sentiment-Analyse mit LSTMs

Laden Sie die Sentimentanalyse-Daten an der Adresse `http://www.cis.uni-muenchen.de/~schmid/lehre/Experimente/data/sentiment-data.zip` herunter und trainieren Sie damit einen Klassifizierer auf Basis von LSTMs. Die Zahl vor dem Tabulator gibt jeweils die Bewertung des nachfolgenden Satzes an, die das System lernen soll.

Schreiben Sie eine Funktion `read_data`, welche Trainingsdaten (oder Development-Daten) aus einer Datei einliest und eine Liste von Paaren zurückgibt, wobei jedes Paar aus einem Klassen-Label (als Integer) und einer Liste von Tokens besteht.

Schreiben Sie eine Klasse `Vocabulary` mit den Methoden `__init__(self, data, min_freq)` und `lookup(self, tokens)`. Die init-Funktion extrahiert aus `data` (einer Liste von Label-Tokens-Paaren) die Tokenhäufigkeiten und weist allen Tokens mit der Mindesthäufigkeit `min_freq` einen eigenen Index zu. Die Indizes beginnen mit 2. Die Indizes 0 und 1 sind für das unbekannte Token und das Padding-Symbol reserviert. Die lookup-Methode erhält eine Liste von Tokens als Argument, schlägt den Index jedes Tokens nach und gibt die Liste der Indizes zurück. Unbekannte Wörter erhalten den Index 0.

Schreiben Sie dann eine Funktion `collate`, welche ein Batch von Datenpaaren als Argument erhält und die Wortfolgen mit Hilfe des Vokabulars auf Zahlen abbildet und drei Tensoren mit den Labels, den Wort-IDs und den Textlängen zurückgibt. Die Wort-IDs werden mit der PyTorch-Methode `pad_sequence` "gepaddet".

Erzeugen Sie nun PyTorch-Objekte des Typs `DataLoader` für die Trainingsdaten und die Development-Daten, welche die Funktion `collate` verwenden, um Beispiele zu einem Batch zusammenzufassen.

Schreiben Sie eine Funktion `train`, welche eine Trainingsepoche durchführt, und eine Funktion `evaluate`, welche die Genauigkeit auf den Developmentdaten berechnet. Die Funktionen erhalten jeweils einen DataLoader als Argument.

Dann schreiben Sie eine Klasse `TextClassifier`, welche das neuronale Netz implementiert, und schließlich das restliche Hauptprogramm.

Zur effizienteren Verarbeitung im LSTM transformieren Sie den gepaddeten Eingabetensor mit `pack_padded_sequence` und transformieren den Ausgabetensor des LSTMs mit `pad_packed_sequence`.

Im Training minimieren Sie die negative Loglikelihood mit dem `CrossEntropyLoss` von PyTorch. Nach jeder Epoche evaluieren Sie das System auf den Entwicklungsdaten und geben die erzielte Genauigkeit aus. (39% Genauigkeit sollten hier erreichbar sein.) Das Modell aus der besten Trainingsiteration speichern Sie in einer Datei. Verwenden Sie keine vortrainierten Embeddings.

Das Trainingsprogramm rufen Sie so auf:

```
python3 lstm-classifier training-data dev-data parfile
```

Die Hyperparameter Ihres Modelles übergeben Sie als optionale Kommandozeilen-Argumente, die sie mit dem Python-Modul `argparse` verarbeiten. Definieren Sie auch Defaultwerte für die optionalen Argumente.

Vorüberlegungen

- Welche Ebenen sollte Ihr neuronales Netz umfassen?
- Wie vermeiden Sie Overfitting?

Schicken Sie mir Ihr Programm und die Ausgabe mit den erzielten **Genauigkeiten**.

Technische Hinweise:

Installieren Sie zunächst PyTorch von der Seite `https://pytorch.org/`.

Wenn Ihr Rechner eine Nvidia-Grafikkarte besitzt, können Sie auch darauf arbeiten. Sie müssen dann aber noch zunächst CUDA von Nvidia installieren.

Achtung: Wenn Sie sich remote auf einem CIP-Pool-Rechner einloggen, sollten Sie mit dem Befehl `ssh -X rechner` anschließend auf einen Rechner "rechner" mit guter Grafikkarte einloggen. Die Rechner im Gobi-Pool sind gut geeignet.

Bevor Sie ein PyTorch-Programm remote auf einem CIP-Pool-Rechner starten, sollten Sie sicherstellen, dass kein anderer eingeloggt ist. Am besten starten Sie Rechenjobs remote mit Hilfe von Slurm (`https://www.rz.ifi.lmu.de/infos/slurm_de.html`). Dann sucht das System selbst einen der verfügbaren Rechner aus.

Der Befehl `nvidia-smi` sagt Ihnen, was gerade auf Ihrer GPU gerechnet wird.