Lemmatisierung mit einem Encoder-Decoder-Modell: Vorverarbeitung

In den kommenden drei Wochen werden Sie ein Encoder-Decoder-Modell mit Attention implementieren und für die Lemmatisierung von Wörtern verwenden. In der aktuellen Aufgabe schreiben Sie eine Klasse `Data` für die Vorverarbeitung der Lemmatisiererdaten. Die Klasse liest die Daten ein, wandelt Buchstaben in Zahlen um und fasst Beispiele zu Batches zusammen. Die Klasse besitzt folgende Methoden:

- `read_data(filename)`: Diese Methode liest eine Datei ein, in der jede Zeile ein Wort und ein Lemma enthält, die durch ein Tabulatorzeichen getrennt sind. Die Liste der Paare wird zurückgegeben.

- `make_table(words)`: Diese Methode zählt die Buchstaben in der gegebenen Liste von Wörtern, extrahiert alle Buchstaben, die mindestens zweimal aufgetreten sind, und weist ihnen fortlaufende Nummern ab 2 zu. Die Nummern 0 und 1 werden dem unbekannten Zeichen und dem Paddingsymbol zugewiesen. Die Methode gibt ein Dictionary zurück, welches die Symbole auf ihre Nummern abbildet sowie den maximalen Index plus 1.

- `init_train(self, traindata, devdata)`: Diese Methode liest die beiden Argumentdateien mit der Methode `read_data` in `self.train_data` und `self.dev_data` ein. Sie extrahiert mit einem zip-Befehl die Wörter und Lemmata aus den Trainingsdaten. Sie ruft die Methode `make_table` mit der Liste der Wörter auf und speichert die Rückgabewerte in `self.src_char_to_id` und `self.num_src_chars`. Analog werden mit der Liste der Lemmata `self.tgt_char_to_id` und `self.num_tgt_chars` erzeugt. Zusätzlich wird ein Dictionary `self.id_to_tgt_char` erzeugt, welches die Nummern wieder auf die Lemma-Symbole abbildet. Schließlich speichern Sie noch die Nummer des Paddingsymbols in `self.pad_id` und die Nummer des unbekannten Zeichens in `self.unk_id`.

  Zum Schluss iteriert die Methode über alle Wort-Lemma-Paare in den Trainingsdaten, berechnet das maximale Verhältnis der Lemmalänge zur Wortlänge und speichert es in `self.max_len_factor`.

- `save(self, paramfile)`: Diese Methode speichert mindestens die Attribute `self.src_char_to_id`, `self.id_to_tgt_char`, und `self.max_len_factor` mit Pickle in der Argumentdatei.

- `init_test(self, filename)`: Diese Datei liest die mit `save` gespeicherten Parameter wieder aus einer Datei ein.

- `__init__(self, *args)`: Die Konstruktormethode ruft `init_test(*args)` auf, falls die Länge von `args` gleich 1 ist, und anderenfalls `init_train(*args)`.

- `train_batches(self, max_batch_size)`: Diese Methode ordnet `self.train_data` zufällig mit dem Befehl `random.shuffle` um und ruft dann die Methode `batches` (s.u.) auf, um eine Folge von Batches zu generieren, die zurückgegeben werden.

- `dev_batches(self, max_batch_size)`: analog zu `train_batches`, aber für Developmentdaten und ohne Shuffling.

- `batches(self, data, max_batch_size)`: Diese Methode erzeugt aus den Eingabedaten `data` eine Folge von Batches für die Verarbeitung mit dem neuronalen Netz. Die Summe der Längen aller Wörter und Lemmata in einem Batch sollte dabei den Wert `max_batch_size` nicht übersteigen. Für jedes Batch wird die (innerhalb der Methode `batches` definierte) Unter-Funktion `process_batch` aufgerufen. Diese Funktion wandelt die Liste der Wörter mit Hilfe der Abbildungstabelle `self.src_char_to_id` in Listen von Buchstaben-Nummern um, merkt sich die Längen der Wörter in `src_lengths` und erzeugt mit dem Torch-Befehl `pad_sequence` einen gepaddeten Tensor `src_id_vecs`.

  Die Lemma-Buchstaben werden mit `self.tgt_char_to_id` auf Nummern abgebildet. Dann wird am Anfang und Ende jeder Folge ein Paddingsymbol hinzugefügt und ein gepaddeter Tensor `tgt_id_vecs` erzeugt. Das Tripel `(src_id_vecs, src_lengths, tgt_id_vecs)` wird mit yield zurückgegeben. Vergessen Sie nicht, auch das letzte Batch auszugeben, das kleiner sein kann.

- `input_batches(self, file, max_batch_size)`: Diese Methode erhält eine Datei nur mit Wörtern als Eingabe und erzeugt analog zu der Methode `batches` eine Folge von Batch-Tensoren. Mit `yield` wird eine Folge von Quadrupeln (`srcs`, `src_id_vecs`, `src_lengths`, `max_tgt_len`) generiert. Dabei ist `srcs` die Liste der Wörter im Batch und

  `max_tgt_len = max(src_lengths) * self.max_len_factor + 4`

  Die Gesamtlänge der Wörter plus `max_tgt_len` mal der Batchgröße sollte `max_batch_size` nicht übersteigen. Lesen Sie nicht die ganze Datei auf einmal ein, sondern geben Sie das nächste Batch aus, sobald es voll ist.

- `tgt_ids_to_chars(self, tgt_char_ids)`: Diese Methode erhält eine Folge von Buchstaben-Nummern als Eingabe und entfernt alle Nummern ab dem (nicht immer vorhandenen) pad_id-Element. Die übrigen Nummern werden mit `self.id_to_tgt_char` auf Buchstaben abgebildet und zurückgegeben.

Schreiben Sie außerdem noch Testcode, welcher beginnt mit:

```
if __name__ == '__main__':
```

Hier rufen Sie `Data('train.txt', 'dev.txt')` auf und prüfen dann die korrekte Funktionsweise des Modules. Die beiden Dateien können Sie an der Adresse `https://www.cis.lmu.de/~schmid/lehre/data/Lemmatizer-Data.zip` herunterladen. In diesen Dateien umfasst die Eingabe neben dem Wort auch noch seine Wortart, um bei Homonymen (bspw. `sichere`) das Lemma (`sichern` vs. `sicher`) besser desambiguieren zu können.

Geben Sie eine Datei `data.py` mit gesamten Code ab.