Aufgabe 6: DNN-Training I - Vorbereitung der Daten

Mit Hilfe der in Aufgabe 1–4 berechneten Features soll in dieser und der nächsten Aufgabe ein Modell für den Spracherkenner mit PyTorch trainiert werden.

Kontextinformation zu den Features hinzufügen

6.1 Implementieren Sie eine Funktion

```
def add_context(feats, left_context=6, right_context=6):
    ...
```

in der Datei `feature_extraction.py`, die an die Features für jedes Frame jeweils die gegebene Anzahl von `left_context` und `right_context` vielen Vorgänger- bzw. Nachfolger-Frame-Features anhängt.

Die ersten beiden Dimensionen bleiben dabei gleich, nämlich die Sequenzlänge und die Merkmalsdimension. Die neue, dritte Dimension bekommt nach Anhängen des Kontexts den Wert: `c_dim` = `left_context` + `right_context` + 1.

Für den Kontext der Frames am Anfang und Ende der Sequenzen soll dabei mit Kopien des ersten bzw. letzten Frames aufgefüllt werden. Beachten Sie auch den Sonderfall, dass sowohl `left_context` als auch `right_context` Null sein können.

Hilfreiche Funktionen: `numpy.roll()`, `numpy.expand_dims()`, `numpy.moveaxis()`, `numpy.dstack()`

6.2 Implementieren Sie eine Funktion

```
def compute_features_with_context(audio_file, window_size=25e-3,
                                  hop_size=10e-3, feature_type='STFT',
                                  n_filters=24, fbank_fmin=0,
                                  fbank_fmax=8000, num_ceps=13,
                                  left_context=4, right_context=4):
    ...
```

in der Datei `feature_extraction.py`, die die jeweilige Merkmalsextraktion durchführt, einen entsprechenden Kontext anhängt und die extrahierten Features mit Kontext zurückgibt. Die extrahierte Merkmale sollen die Dimension `[f_len, f_dim, c_dim]` haben, wobei `f_len` die Anzahl der Rahmen der Sequenz, `f_dim` die Merkmalsdimension und `c_dim` die Kontextlänge repräsentieren.

Vorbereitung der Trainingsdaten

6.3 Laden Sie die Trainings- und Test-Daten aus der TUB-Cloud herunter: `https://tubcloud.tu-berlin.de/s/y3RDt5x9JGmmrRB` (Passwort: `ASE2324TUB`). Der Datensatz enthält einen `TRAIN` und einen `TEST` Ordner. Beide Ordner enthalten jeweils die drei Ordner `wav`, `lab` und `TextGrid`. Der `wav`-Ordner enthält alle Audiodateien, der `lab`-Ordner die korrekte Transkription zu der jeweiligen Audiodatei und der `TextGrid`-Ordner Praat-Dateien[1] mit Information über Anfangs- und Endzeiten jedes Wortes/Phonems.

Bitte beachten Sie, dass der Korpus **NICHT** im `./my-repository`-Ordner gespeichert werden sollte. Legen Sie bitte die Datenbank in einen lokalen Ordner.

6.4 Neuer Data-Loader: Nehmen Sie Ihre Datei `utils.py` aus Aufgabe 5 als Beispiel und erstellen Sie ein neues `utils.py` Skript im Verzeichnis `recognizer`. Implementieren Sie einen neuen Data-Loader in der Datei `utils.py` für den gegebenen Datensatz.

Der Data-Loader muss für die Daten

- Die Merkmale extrahieren und
- die Labels extrahieren,
- die Merkmale und Labels in die richtige Form bringen (Tensoren für PyTorch)
- und dafür zunächst alle Dateien finden,

Finden der Dateien

Im ISIS-Kurs finden Sie `Aufgabe6.zip`. Um die Data-Loader für die Trainings-, Development- und Testsets zu erstellen, benötigen Sie aus diesem File die Metainformationen zum Datensatz. Bitte kopieren Sie daher den `dataset`-Ordner aus `Aufgabe6.zip` in Ihr lokales Repository: `./my_repository/`.

Der `dataset`-Ordner enthält danach drei `JSON`-Dateien mit allen Metainformationen. Dazu zählen der Pfad der akustischen Aufnahme sowie die Ground-Truth-HMM-Zustände für die Trainings-, Development- und Testsets.

Aus diesen JSON-Dateien erhalten Sie die Pfade zu den Audiodateien (`audiodir`) und den Label-Dateien (`targetdir`). Passen Sie Ihre `__getitem__`-Funktion im Data-Loader an, sodass diese Pfadinformationen benutzt werden, um auf die Dictionaries für die Audiodaten und Labelinformationen zuzugreifen.

[1] Siehe Aufgabe 6.4

Berechnung der Merkmale Berechnen Sie in Ihrem Data-Loader die Features jeweils mit der zuvor definierten Funktion

```
compute_features_with_context(audio_dir, window_size,
                              hop_size, feature_type,
                              n_filters, fbank_fmin,
                              fbank_fmax, num_ceps,
                              left_context, right_context)
```

Mithilfe des Dictionaries `parameters` werden dabei alle für die Merkmalsextraktion benötigten Parameter übergeben. Mit Hilfe von `**parameters` können Sie die Parameter einfach 'entpackt' an eine Funktion übergeben, die auf die einzelnen Parameter zugreift. Setzen Sie `feature_type=MFCC_D_DD`, `window_size=25e-3`, `hop_size=10e-3`, `n_filters=40`, `fbank_fmin=0`, `fbank_fmax=8000`, `num_ceps=13`, `left_context=10`, und `right_context=10`.

Berechnung der Labels In `Aufgabe6.zip` finden Sie auch die Datei `recognizer/tools.py` beinhaltet vier Funktionen. Eine davon, die `praat_file_to_target`-Funktion, ermöglicht es Ihnen, schnell die Labels aus Praat-Dateien zu berechnen. Um diese Funktion verwenden zu können, kopieren Sie bitte diese vier Funktionen in Ihre eigene `recognizer/tools.py`-Datei.

Für die Berechnung der Ground-Truth-Labels benötigen Sie die Bibliothek `praatio`, die Sie via `pip install praatio` in Ihrer virtuellen Umgebung installieren können. Wenn Sie Probleme bei der Installation der `praatio`-Bibliothek haben, lesen Sie bitte die Installationsanleitung am Ende dieses Übungsblatts.

Außerdem enthält die `Aufgabe6.zip`-Datei eine Version der HMM-Klasse[2] namens `recognizer/hmm.py`, die nur die notwendigen Funktionen zur Berechnung der Labels enthält. In den nächsten Übungen wird diese Klasse durch Ihre eigenen Funktionen erweitert. Kopieren Sie `recognizer/hmm.py` in Ihr eigenes `recognizer`-Verzeichnis.

Um ein `hmm` unter Nutzung von `hmm.py` zu konstruieren, fügen Sie zunächst die Importanweisungen `import recognizer.hmm as HMM` ein. Erstellen Sie dann das `hmm` mit `hmm=HMM.HMM()`.

Die Ground-Truth Labels können schließlich mit

[2]HMM: Hidden Markov Model. HMMs sind das Thema der ersten Vorlesung im neuen Jahr.

```
tools.praat_file_to_target(target_dir, sampling_rate,
                           window_size_samples, hop_size_samples, hmm)
```

berechnet werden.

Merkmale und Labels in die richtige Form bringen Die extrahierten Merkmale und die Ground-Truth-Labels müssen abschließend mit Hilfe von `torch.FloatTensor()` in Tensoren umgewandelt werden.

Hinweis:

- Im Gegensatz zu Aufgabe 5 sollten akustische Merkmale auf keinen Fall gemäß der Variable `max_len` beschnitten werden.
- Eine Padding-Funktion ist nicht erforderlich, weil die Batchgröße auf 1 festgelegt wird.

6.5 Überprüfung und Darstellung der Ergebnisse: Erstellen Sie ein `uebung6.py`-Skript, in dem Sie einen Data-Loader für das Development-Set erstellen. Setzen Sie die Batchgröße bitte auf 1. Iterieren Sie durch Ihren Data-Loader, um sicherzustellen, dass dieser die extrahierten Merkmale und die entsprechenden Ground-Truth-Labels zurückgibt. Plotten Sie die Ground-Truth-Labels der ersten beiden Samples, die vom Data-Loader zurückgegeben wurden, und überprüfen Sie, ob sie unterschiedlich sind. Falls sie gleich aussehen, könnte Ihr Data-Loader einen Fehler aufweisen. Setzen Sie `sampling_rate=16000`. Das Ergebnis sollte prinzipiell so aussehen wie in Abbildung 1.

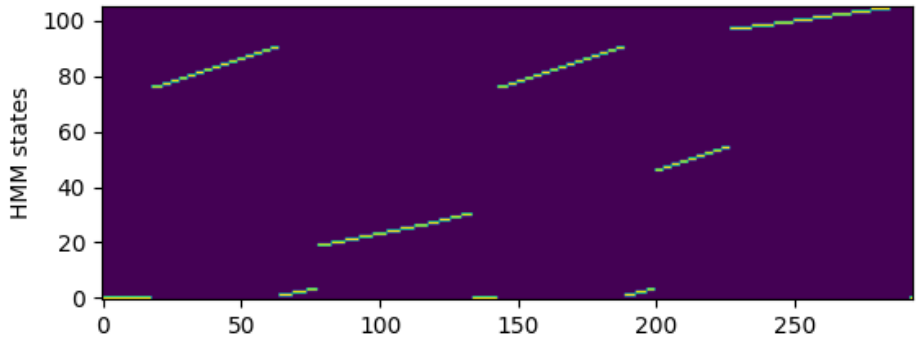

Abbildung 1: Ground-Truth-Labels für das Beispiel `TEST-WOMAN-BF-7017049A.TextGrid`.

Hinweis: Achten Sie bitte auf den Unterschied zwischen `window_size` und `window_size_samples`, sowie zwischen `hop_size` und `hop_size_samples`.

Repository-Struktur

Nach dem heutigen Praktikum sollte Ihr Repository die folgende Struktur haben.

Spracherkenner:

```
./my-repository/data/TEST-MAN-AH-3033951A.wav
./my-repository/dataset/train.json
./my-repository/dataset/dev.json
./my-repository/dataset/test.json
./my-repository/recognizer/__init__.py
./my-repository/recognizer/hmm.py
./my-repository/recognizer /tools.py
./my-repository/recognizer/feature-extraction.py
./my-repository/recognizer/utils.py
```

PyTorch-Einführung:

```
./my-repository/torch_intro/dataset/train.json
./my-repository/torch_intro/dataset/dev.json
./my-repository/torch_intro/dataset/test.json
./my-repository/torch_intro/local/feature-extraction.py
./my-repository/torch_intro/local/model.py
./my-repository/torch_intro/local/utils.py
./my-repository/torch_intro/local/train.py
```

Hauptskripte:

```
./my-repository/uebung1.py
./my-repository/uebung2.py
./my-repository/uebung3.py
./my-repository/uebung4.py
./my-repository/uebung5.py
./my-repository/uebung6.py
```

Installation der `praatio`-Bibliothek

Wenn Sie versuchen, die `praatio`-Bibliothek in der Python-3.6-Anaconda-Umgebung zu installieren, kann es vorkommen, dass Python die Bibliothek nicht findet, wie in Abbildung 2 gezeigt. Um dieses Problem zu lösen, können Sie den "Conda Package Manager" verwenden. Dadurch können Sie die `praatio`-Bibliothek einbinden, wie in Abbildung 3 dargestellt. Für unsere Aufgabe brauchen wir die `praatio<5.0`-Version.

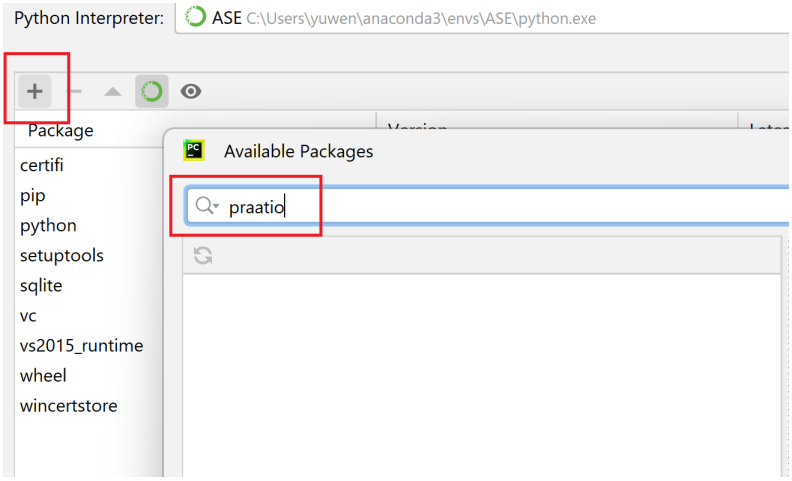

Abbildung 2: Es wurde keine `praatio`-Bibliothek gefunden.

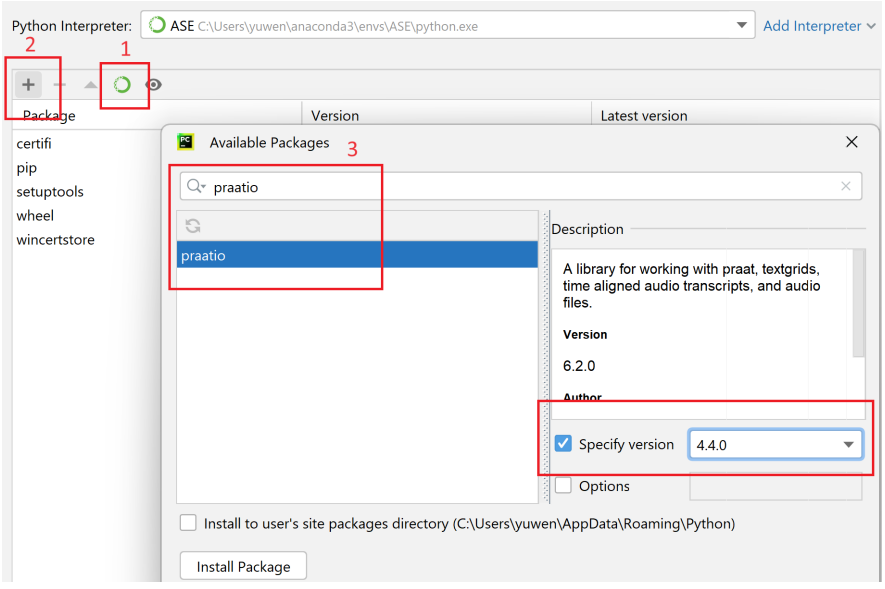

Abbildung 3: `praatio`-Bibliothek installieren.