Aufgabe 7: DNN-Training II

Nach den Vorbereitungen in Aufgabe 6 besteht das Ziel hier darin, mithilfe von PyTorch ein Modell für die Ausgabeverteilungsdichten des hybriden Spracherkenners zu trainieren.

Erstellung des Hauptskripts

7.1 Erstellen Sie auf Basis Ihrer Datei `uebung5.py` ein neues Hauptskript `uebung7.py` in Ihrem Repository. Geeignete Parameter für die Merkmalsextraktion finden Sie in Aufgabe 6.4. Setzen Sie bitte die Batchgröße auf 1.

Entwurf des DNN

7.2 Nehmen Sie Ihre Datei `model.py` aus Aufgabe 5 als Ausgangspunkt und erstellen Sie ein neues `model.py`-Skript im Verzeichnis `recognizer`.

```
class DNN_Model(torch.nn.Module):
        def __init__(self, idim, odim, hidden_dim):
            ...
        def forward(self, audio_feat):
            ...
```

Die Eingabemerkmale `audio_feat` sollen die Dimension `[BS, f_len, f_dim, c_dim]` haben, wobei `BS` die Batchgröße, `f_len` die Anzahl der Rahmen der Sequenz, `f_dim` die Merkmalsdimension und `c_dim` die Kontextlänge repräsentieren. Implementieren Sie bitte zunächste einen `Flatten`-Layer. Dieser bringt die Merkmals-Frames, welche nach Anhängen des Kontexts eine zusätzliche Dimension haben, in Vektorform. Danach sollen die Merkmale die Dimension `[BS, f_len, (f_dim × c_dim)]` besitzen.

Definieren Sie die einzelnen neuronalen Schichten in der Unterfunktion `__init__()`, dabei bezeichnet `idim = f_dim × c_dim` die Eingabedimension. Der Wert `odim` gibt die Anzahl der Zustände des HMM an. Diese kann mit `HMM.get_num_states()` aus `hmm.py` berechnet werden. Mit `hidden_dim` wird die Anzahl der Neuronen in der verborgenen Schicht bezeichnet. Implementieren Sie ein Feedforward-Netzwerk mit drei vollständig verbundenen Schichten unter Verwendung von `torch.nn.Linear()`.

Auf jede vollständig verbundene Schicht sollte eine ReLU-Aktivierungsfunktion folgen. Implementieren Sie zusätzlich eine vollständig verbundene Klassifizierungsschicht, um die Merkmale in die gewünschte Ausgabedimension umzuwandeln.

Hinweis: Hier benötigen wir keine Softmax-Funktion, um die a-posteriori-Wahrscheinlichkeiten zu ermitteln, da die Pytorch Kreuzentropie-Funktion für die Berechnung des Loss-Wertes verwendet wird. Die Softmax-Funktion wird in der eingebetteten Loss-Funktion berechnet.

DNN-Training

Nehmen Sie Ihre Datei `train.py` aus Aufgabe 5 als Beispiel und erstellen Sie eine neue Datei `train.py` im Verzeichnis `recognizer`. Die folgenden Punkte sind anders als in Aufgabe 5. Darauf sollten Sie achten:

- Bei Aufgabe 5 handelte es sich um eine binäre Klassifikationsaufgabe, deshalb wurde die `torch.nn.BCELoss()` Loss-Funktion verwendet. Im Gegensatz dazu ist Aufgabe 7 eine Multi-Klassen-Klassifikationsaufgabe, deshalb benötigen Sie die Loss-Funktion `torch.nn.CrossEntropyLoss()`.
- In Aufgabe wird die Batchgröße auf 1 festgelegt. Daher ist es nicht erforderlich, eine Padding-Funktion zu implementieren.
- In Aufgabe 7 ist es nicht erforderlich, Zwischenergebnisse in einer `JSON`-Datei zu speichern.

7.3 Nehmen Sie die `train()` Funktion aus Ihrer Datei `train.py` aus Aufgabe 5 als Beispiel und implementieren Sie eine neue `train()` Funktion in der Datei `train.py` für Aufgabe 7

```
def train(dataset, model, odim, optimizer=None, criterion=None):
    ...
```

Bitte versetzen Sie das Modell zunächst mit `model.train()` in den Trainingsmodus. Verwenden Sie das Modell, um die a-Posteriori-Wahrscheinlichkeiten für alle Batches zu berechnen. Ermitteln Sie den jeweiligen Wert der Loss-Funktion `criterion()`, berechnen Sie die Gradienten und aktualisieren Sie die Modellparameter mit `optimizer`.

7.4 Nehmen Sie die `evaluation()` Funktion aus `train.py` aus Aufgabe 5 als Beispiel und implementieren Sie eine neue `evaluation()` Funktion in der Datei `train.py` für Aufgabe 7

```
def evaluation(dataset, odim, model):
    ...
```

Bitte versetzen Sie das Modell in dieser Funktion zunächst mit `model.eval()` in den Evaluierungsmodus. Lassen Sie wieder alle Batches durchlaufen, um mit dem Modell die a-posteriori Wahrscheinlichkeiten $p(k|\mathbf{x})$ zu berechnen. Berechnen Sie nach einem vollständigen Durchlauf bitte auch die Genauigkeit über den gegebenen Development-Set. Speichern Sie bitte **NUR** das Modell, das die beste Genauigkeit über den gegebenen Development-Set erreicht.

Nachdem alle Epochen abgeschlossen sind, evaluieren Sie das beste Modell für das gegebene Test-Set.

Hinweis:

- Da die Kreuzentropie als Lossfunktion verwendet wurde, haben Sie im Modell keine Softmax-Funktion verwendet. Um die geschätzte Verteilungsdichtefunktion (*Prediction*) zu erhalten, sollten Sie hier eine Softmax-Funktion auf den Ausgang des Modelles anwenden.
- Mit einem Desktop-Computer ohne GPU könnte das Training einer Epoche etwa 13 Minuten dauern. Zum Testen ist es also sinnvoll, zunächst nur eine Epoche zu trainieren, und später für bessere Ergebnisse ein längeres Training zu starten. [1]

Visualisierung der a-Posteriori-Wahrscheinlichkeiten

7.5 Implementieren Sie eine Funktion

```
def wav_to_posteriors(model, audio_file, parameters):
    ...
```

in der Datei `train.py`, die für eine Audiodatei `audio_file` mit Hilfe von `model` und `parameters` die Ausgabe des DNNs zurückgibt.

[1] In diesem späteren, längeren Training können Sie den Code für das Early Stopping aus Aufgabe 5 verwenden.

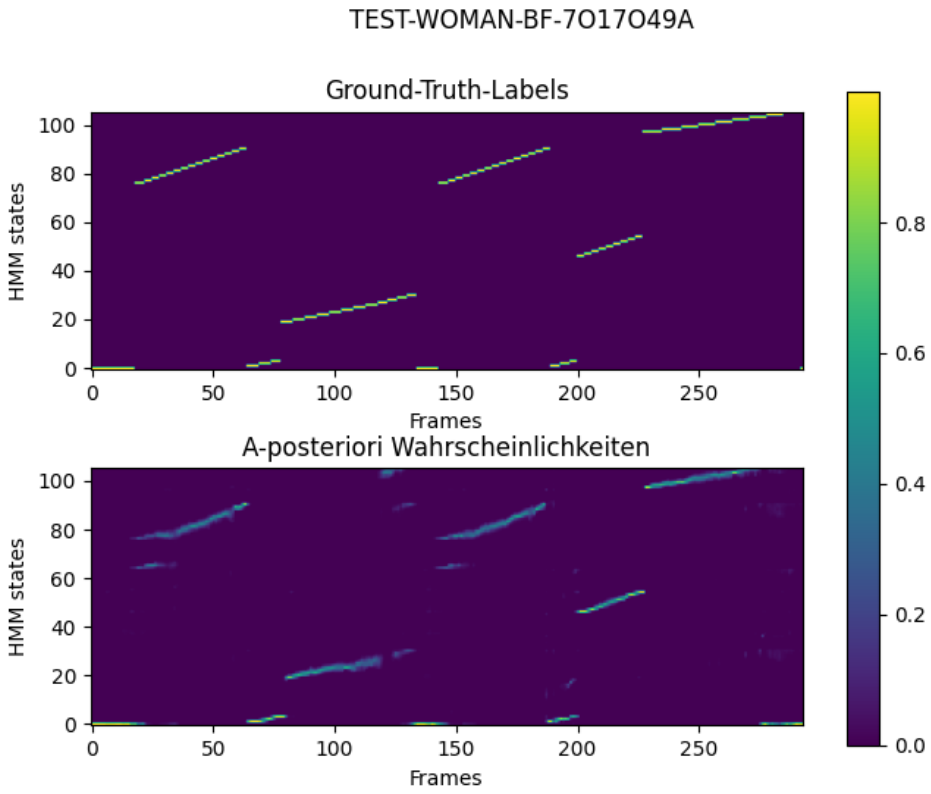

Abbildung 1: Ground-Truth-Labels und die entsprechenden a-posteriori Wahrscheinlichkeiten für die Beispieldatei `TEST-WOMAN-BF-7O17O49A.wav`.

Nachdem das DNN trainiert wurde, kann das DNN mit `TEST-WOMAN-BF-7O17O49A.wav` getestet werden.[2] Plotten Sie am Ende der `run()` Funktion Ihrer Datei `train.py` diese Posteriors für die Beispieldatei zusammen mit den Ground-Truth-Labels (die Sie mit der Funktion `praat_file_to_target()` enthalten). Je nach Netzwerktopologie und Trainingsdauer sollte das Ergebnis in etwas so aussehen wie Abbildung 1.

[2]Diese Audioaufnahme ist im vorhandenen Korpus zu finden.