Aufgabe 10: Fertigstellung des Erkenners und Bestimmung der Wortfehlerrate

In dieser Übung sollen die Erkennung einer Audiodatei sowie die Berechnung der Wortfehlerrate (WER) mithilfe des DNNs, des Viterbi-Algorithmus und des Verbundworterkenners durchgeführt werden.

Erkennung auf geladener Posterior-Matrix

10.1 Implementieren Sie eine Funktion

```
def posteriors_to_transcription(posteriors):
    ...
```

in der Datei `hmm.py`. Diese Funktion soll die mithilfe der Funktionen `viterbi()` und `getTranscription()` aus einer Matrix von `posteriors` eine Transkription berechnen und zurückgeben. Die Matrix der `posteriors` enthält dabei die Zustandswahrscheinlichkeiten über der Zeit, entsprechend der Ausgabe des DNNs.

Zum Testen der Funktion `posteriors_to_transcription()` wird in `uebung10.py` zunächst eine Matrix mit (nicht logarithmierten) Zustandswahrscheinlichkeiten von `./data/TEST-WOMAN-BF-7O17O49A.npy` geladen, die bei richtiger Implementierung der nötigen Funktionen die Trankription `['SEVEN', 'OH', 'ONE', 'SEVEN', 'OH', 'FOUR', 'NINE']` ausgibt.

Achten Sie darauf, dass wir die Posterior-Matrix mit der Topologie des Default-HMMs erstellt haben und deswegen die Erkennung nur dann funktioniert, wenn Sie ebenfalls das Default-HMM aus Übung 7 benutzen, wenn Sie also die Anzahl der Zustände pro Wort und die Reihenfolge der Worte nicht verändert haben.

Erkennung auf selbst berechneter Posterior-Matrix

10.2 Nutzen Sie nun Ihr in Übung 7 trainiertes DNN, um die Erkennung mit einer selbst berechneten Posterior-Matrix zu der Datei `./TIDIGITS-ASE/TEST/wav/TEST-WOMAN-BF-7O17O49A.wav` durchzuführen. Verwenden Sie dabei die Funktion `wav_to_posteriors()` aus Übung 7, um die Eingabe für

die Funktion `posteriors_to_transcription()` zu erzeugen. Die Transkription sollte dann nicht oder nur geringfügig von der richtigen Transkription abweichen.

Wortfehlerrate auf Testdatensatz

10.3 Implementieren Sie eine Funktion

```
def test_model(datadir, hmm, model, parameters):
    ...
```

in der Datei `uebung10.py`, die die Wortfehlerrate für den gesamten Testdatensatz (siehe die Datei `dataset/test.json`) berechnet. Die Audiodatei können Sie in dem entsprechenden `wav`-Ordner in `<datadir>/TEST` finden. Für die Referenz, also den wirklich gesprochenen Text, können die Dateien aus dem `lab`-Ordner verwendet werden.

Öffnen Sie dazu die jeweiligen Referenzdateien mithilfe von `open()` und lesen Sie den Inhalt mit `read()` und `split()` ein.

Lassen Sie den Erkenner, inklusive DNN, Viterbi und Verbundworterkenner, auf jeder Audiodatei aus dem Testset ausführen.

Mit der Funktion

```
N, D, I, S = tools.needlemann_wunsch(ref_seq, word_seq),
```

können die Anzahl der Deletions `D`, Insertions `I`, Substitutions `S` und die Gesamtzahl der Worte `N` berechnet werden. `ref_seq` ist die Referenzsequenz, die aus der `lab`-Datei eingelesen wurde. `word_seq` is das Ergebnis Ihres Erkenners. Die Funktion `needlemann_wunsch` finden Sie in der aktuellen Übung. Kopieren Sie diese in Ihre eigene `tools.py`.

Berechnen Sie schließlich die Wortfehlerrate für den gesamten Testdatensatz mit

$$WER = 100 \cdot \left(\frac{D_{ges} + I_{ges} + S_{ges}}{N_{ges}} \right),$$

indem Sie alle Ausgaben von `tools.needlemann_wunsch()` in D_{ges}, I_{ges}, S_{ges} und N_{ges} akkumulieren.

Hinweis: Sie können zunächst nur einen kleinen Teil der Testdaten verwenden, z. B. 50 Dateien, da die Berechnung sonst lange dauern kann. Stellen Sie dafür aber sicher, dass die Testdateien nicht alle von einem Sprecher kommen, indem Sie die Liste von Audio- und Referenzdateien vorher z. B. zufällig (aber gleich!) permutieren.

Geben Sie zur Übersicht die aktuelle Wortfehlerrate nach jeder Datei aus. Das kann dann z. B. so aussehen:

```
------------------------------------------------------------
/media/public/TIDIGITS-ASE/TEST/wav/TEST-MAN-HR-97A.wav
REF: ['NINE', 'SEVEN']
OUT: ['NINE', 'SEVEN']
I: 0    D: 0    S: 0    N: 2
Current Total WER: 3.4100596760443302
------------------------------------------------------------
/media/public/TIDIGITS-ASE/TEST/wav/TEST-WOMAN-BJ-6Z38ZA.wav
REF: ['SIX', 'ZERO', 'THREE', 'EIGHT', 'ZERO']
OUT: ['SIX', 'ZERO', 'THREE', 'EIGHT', 'ZERO', 'OH']
I: 1    D: 0    S: 0    N: 5
Current Total WER: 3.4804753820034007
------------------------------------------------------------
/media/public/TIDIGITS-ASE/TEST/wav/TEST-MAN-HM-25ZZ8A.wav
REF: ['TWO', 'FIVE', 'ZERO', 'ZERO', 'EIGHT']
OUT: ['TWO', 'FIVE', 'ZERO', 'OH', 'ZERO', 'OH', 'EIGHT']
I: 2    D: 0    S: 0    N: 5
Current Total WER: 3.6348267117497834
------------------------------------------------------------
```

Allgemeine Tipps zum Verbessern des Erkenners

Sollten Sie mit der Wortfehlerrate Ihres Spracherkenners noch unzufrieden sein, dann finden Sie im Folgenden einige Vorschläge für das Performance-Tuning Ihres Systems, sortiert nach aufsteigendem Schwierigkeitsgrad.

Bei einer Wortfehlerrate über 10% haben Sie vermutlich noch Fehler in Ihrer Implementierung oder Ihr DNN ist nicht genügend trainiert worden. Benutzen Sie den Programmcode aus den vorherigen Übungen, um Ihre Implementierung gründlich zu testen.

Ist Ihre Implementierung korrekt, dann können Sie versuchen, in Abhängigkeit der von Ihnen beobachteten Wortfehler den Grammatik-Score der TIDIGITS-Worte anzupassen. Ändern Sie dazu die Eintrittswahrscheinlichkeiten einzelner Worte, z.B. wenn das kurze Wort `"oh"` zu häufig erkannt wird, dann senken Sie die Eintrittswahrscheinlichkeit z.B. von 1/11 auf einen kleineren Wert relativ zu den verbleibenden Worten.

Neben den Maßnahmen zur Anpassung der Topologie und der Werte der Übergangsmatrix kann man auch versuchen, die Log-Likelihood-Berechnung zu optimieren. Dazu wird das künstliche neuronale Netz angepasst, z.B. kann man die Anzahl Schichten und die Anzahl Neuronen pro Schicht vergrößern oder die Aktivierungsfunktion ändern. Zusätzlich kann das DNN durch Vergrößern der Anzahl Trainingsepochen noch weiter trainiert werden.