ΕΠΕΞΕΡΓΑΣΙΑ ΦΩΝΗΣ & ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ

8ο Εξάμηνο 2022 – 2023

Lab Assignment 2 – Solutions

Ζαρίφης Στέλιος – el20435 Email: el20435@mail.ntua.gr

Αστρεινίδης Ζαφείριος – el20435 Email: el20435@mail.ntua.gr

Contents

1. Περιγραφή

Στην παρούσα εργασία θα υλοποιήσουμε ένα σύστημα επεξεργασίας και αναγνώρισης φωνής χρησιμοποιώντας το εργαλείο Kaldi. Συγκεκριμένα, το σύστημα θα αφορά σε αναγνώριση φωνημάτων από ηχογραφήσεις της USC – TIMIT από τέσσερεις διαφορετικούς ομιλητές, για εκπαίδευση και αξιολόγηση.

Ο σχεδιασμός του συστήματος συμβαίνει σε τέσσερεις φάσεις:

1. Εξαγωγή ακουστικών χαρακτηριστικών από τα δεδομένα φωνής με χρήση των συντελεστών Mel – Frequency Cepstral Coefficients (MFCC).
   Στην επεξεργασία φωνής, το mel-frequency cepstrum είναι μια αναπαράσταση του power spectrum βραχέως χρόνου ενός ηχητικού σήματος[1].
   Η προσέγγιση αυτή είναι προσαρμοσμένη στη (μη γραμμική) ανθρώπινη αντίληψη του ήχου, βάσει ψυχοακουστικών μελετών.
2. Δημιουργία γλωσσικών μοντέλων χρησιμοποιώντας τα transcriptions του dataset. Από αυτά παίρνουμε την a priori πιθανότητα για το τελικό σύστημα.
3. Εκπαιδεύουμε ακουστικά μοντέλα μέσω των ακουστικών χαρακτηριστικών που εξάγαμε.
4. Το τελικό σύστημα προκύπτει από το συνδυασμό των ανωτέρω σημείων, το οποίο με είσοδο ένα σήμα φωνής μπορεί να εξάγει τα ακουστικά χαρακτηριστικά και να διαχωρίσει το σήμα σε ακολουθία φωνημάτων ή λέξεων

2. Θεωρητικό Υπόβαθρο

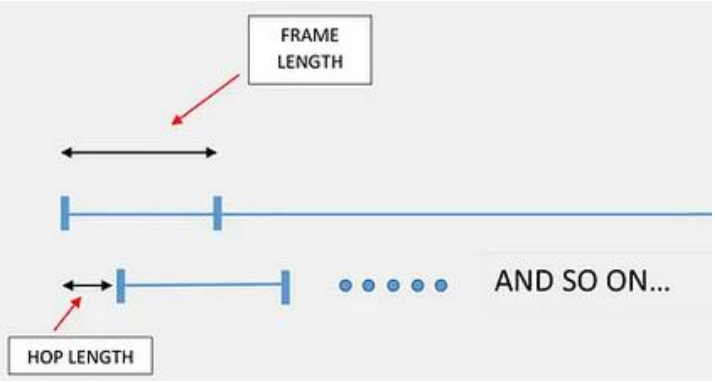
Βασικές έννοιες:

A. Mel – frequency Cepstral Coefficients (MFCCs):
Πριν αναπτύξουμε τη λειτουργία των MFCCs, πρέπει να εξηγήσουμε πρώτα τι είναι το Mel – frequency cepstrum (MFC). Όπως αναφέραμε σύντομα στο πρώτο μέρος της αναφοράς, το MFC είναι η αναπαράσταση του short – term power spectrum ενός ηχητικού σήματος, η οποία βασίζεται στο μετασχηματισμό συνημιτόνου της φασματικής πυκνότητας ισχύος του σήματος χρησιμοποιώντας τη κλίμακα mel.

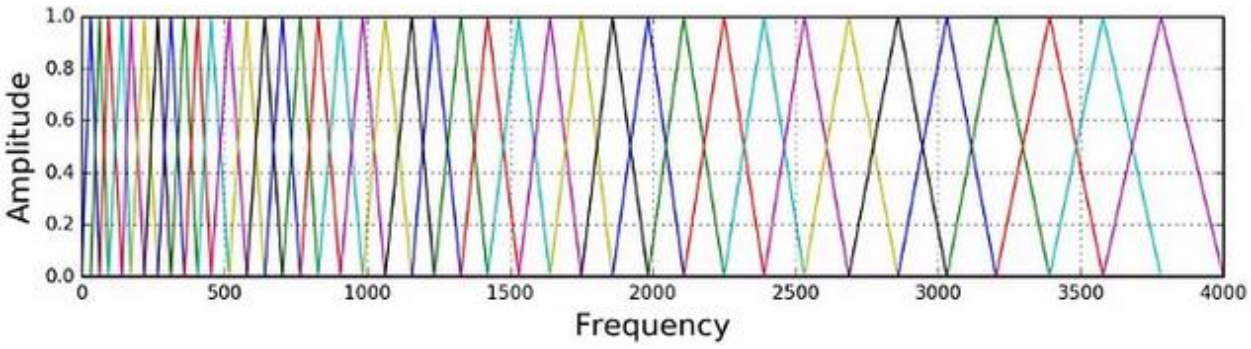
Η κλίμακα mel (της οποίας το όνομα προέρχεται από τη λέξη melody) είναι μια κλίμακα τόνων που ο μέσος άνθρωπος αντιλαμβάνεται ότι οι τόνοι αυτοί απέχουν ίσες αποστάσεις μεταξύ τους[2].

Οι MFCCs είναι οι συντελεστές που συγκροτούν το MFC ενός ηχητικού σήματος. Η εξαγωγή αυτών από ένα σήμα προκύπτει από τα εξής βήματα[3]:

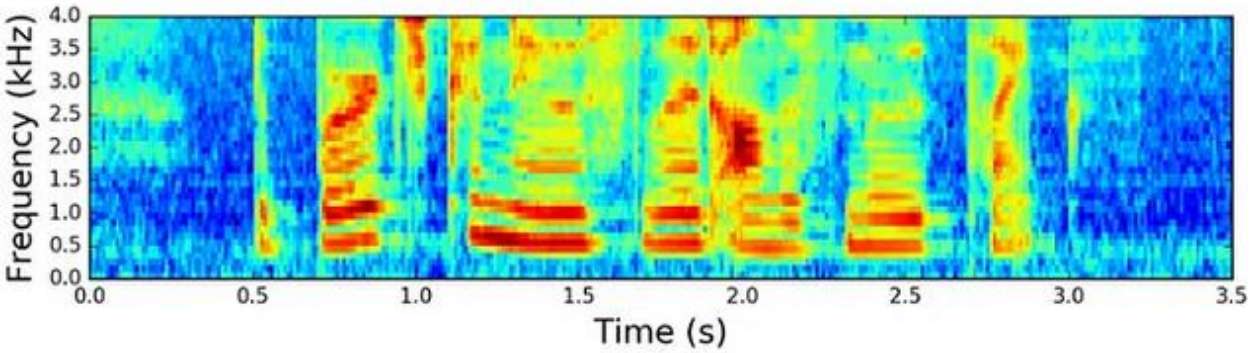
1. Υπολογίζουμε το μετασχηματισμό Fourier για κάθε παράθυρο του σήματος (κάθε παράθυρο έχει διάρκεια περίπου $20 - 40\ ms$).
2. Απεικονίζουμε το φάσμα ισχύος στην κλίμακα mel, χρησιμοποιώντας τριγωνικά (ή συνημιτονοειδή κάποιες φορές) επικαλυπτόμενα παράθυρα. Οι επικαλυπτόμενες περιοχές καταλαμβάνουν 160 δείγματα (hop length).
3. Εφαρμόζουμε τα Filter Banks. Τα Filter Banks για την κλίμακα mel είναι ένα σύνολο από $20 - 40$ τριγωνικά φίλτρα. Καταλήγουμε με ένα spectrogram σαν της εικόνας.
4. Περνάμε τις τιμές του spectrogram από μια log function και παίρνουμε τους MFCCs.
5. Εξάγουμε επιπλέον τιμές σχετικά με το πόσο μεταβάλλονται κάποια στατιστικά χαρακτηριστικά του cepstrum στο παράθυρο, ακόμα και πόσο μεταβάλλονται οι μεταβολές αυτές (deltas & double – deltas in delta training).



Απεικόνιση της επικάλυψης των παραθύρων που αναφέρονται στο βήμα 1



Απεικόνιση των τριγωνικών παραθύρων που αναφέρονται στο βήμα 2



Παράδειγμα του προκύπτοντος spectrogram στο βήμα 3

Β. Γλωσσικά Μοντέλα (Language Models):

Τα γλωσσικά μοντέλα είναι κατανομές πιθανοτήτων πάνω σε λέξεις, δεδομένων κάποιων συμφραζομένων.

Προσεγγίσεις – υλοποιήσεις:

1. n – gram model: Η φυσική γλώσσα μοντελοποιείται ως μαρκοβιανή αλυσίδα. Ο χώρος καταστάσεων αποτελείται από ακολουθίες λέξεων μήκους n. Οι πιθανότητες μετάβασης εκπαιδεύονται ως φυσικές συχνότητες με βάση ένα επαρκώς μεγάλο text corpus.
   Υπάρχει, βέβαια, ο κίνδυνος να μην εκπαιδευθεί καλά μια τέτοια πιθανότητα μετάβασης, αν η n – άδα ή η μετάβαση παρατηρηθούν λίγες φορές. Μπορεί να αποδοθεί μηδενική πιθανότητα σε γλωσσικά έγκυρη μετάβαση.

2. Νευρωνικά γλωσσικά μοντέλα: Οι λέξεις μοντελοποιούνται ως word embeddings, διανύσματα διαστάσεων 50 – 1000, τυπικά. Τα word embeddings έχουν πολύ καλύτερες επιδόσεις στην αναπαράσταση σημασιολογίας λέξεων, μπορούν να κωδικοποιήσουν αποτελεσματικότερα την ομοιότητα δύο λέξεων και ως εκ τούτου να θεωρήσουν πιθανές (γλωσσικά έγκυρες) μεταβάσεις οι οποίες δεν εμφανίζονται στο text corpus. Τα word embeddings δίνονται ως

είσοδοι σε δίκτυα πρόσθιας τροφοδότησης (feedforward neural networks) ή σε αναδρομικά νευρωνικά δίκτυα (RNNs). Προσφέρονται για ανάλυση συναισθήματος, ενώ με τα αναδρομικά νευρωνικά δίκτυα έχουμε καταφέρει να συνοψίσουμε ολόκληρη την πληροφορία από το προηγούμενο κείμενο στην εσωτερική κατάσταση ($h(t-1)$ στην αρχιτεκτονική Elman). Έτσι, δεν χρειάζεται να προβληματιστούμε για τη επιλογή μεγέθους του παραθύρου, ούτε να υποστούμε τον περιορισμό που εισάγει!

C. Φωνητικά Μοντέλα (Acoustic Models):

Τα ακουστικά μοντέλα χρησιμοποιούνται στην αυτόματη αναγνώριση ομιλίας και έχουν σκοπό τους την αναπαράσταση της συσχέτισης μεταξύ του σήματος φωνής και των γλωσσικών φωνημάτων. Εκπαιδεύονται με χρήση των acoustic recordings και των αντίστοιχων transcripts[4].

Από κοινού με ένα γλωσσικό μοντέλο, μοντελοποιούν τις στατιστικές ιδιότητες του λόγου.

Τα περισσότερα συστήματα αναγνώρισης φωνής, επεξεργάζονται το ακουστικό σήμα σε μικρά frames, από τα οποία εξάγει τα MFCCs. Τα MFCCs μπορούν να εξαχθούν και από συνελικτικό νευρωνικό δίκτυο με επεξεργασία της γραφικής παράστασης του ηχητικού σήματος.

3. Βήματα Προπαρασκευής

Σε αυτήν την ενότητα της αναφοράς παρουσιάζουμε συνοπτικά τα βήματα που ακολουθήσαμε για την προπαρασκευή. Παράγουμε τα απαραίτηταρχεία για τα datasets (train, evaluation , test) για την εκπαίδευση των μοντέλων στο κυρίως μέρος.

1. Imports: Εισάγουμε τις απαραίτητες βιβλιοθήκες: os, re, sys και shutil.

2. create_uttids function: Η συνάρτηση create_uttids δημιουργεί ένα αρχείο με όνομα "uttids" στο κατάλληλο directory (με βάση την παράμετρο name). Αντιγράφουμε τα περιεχόμενα ενός από τα αρχεία training.txt, testing.txt και validation.txt στα destinations data/train/uttids, data/test/uttids και data/dev/uttids, αντίστοιχα.

3. create_utt2spk function: Η συνάρτηση create_utt2spk δημιουργεί το αρχείο "utt2spk". Αυτό το αρχείο αντιστοιχίζει τα ids των δειγμάτων στους ομιλητές.

4. create_wavscp function: Η συνάρτηση create_wavscp δημιουργεί το αρχείο wav.scp που αντιστοιχίζει το id κάθε δείγματος με το αρχείο ήχου που το περιέχει.

5. create_text function: Η συνάρτηση create_text δημιουργεί το αρχείο text το οποίο αντιστοιχίζει τα transcriptions στις προτάσεις τους.

6. create_textPronunciation: Η συνάρτηση create_textPronunciation δημιουργεί το αρχείο textPronunciation. Αυτό αντιστοιχίζει τουτα ids των δειγμάτων στο pronounciation αυτών (των προτάσεων που εκφωνούνται).

7. Data preparation lists: Ορίζουμε τις λίστες: names, lines και writeNames. Αυτές οι λίστες περιέχουν τις ονομασίες (train, test, dev), τον αριθμό των γραμμών για κάθε dataset και τα αντίστοιχα ονόματα αρχείων (training, testing, validation), για να τρέξουμε παραμετρικά τις συναρτήσεις για τα directories train, test, dev.

```
import os
import re
import sys
import shutil

# function to create uttids file
def create_uttids(name, lines):
    if name == "train":
        source = "filesets/training.txt"
        dest = "data/train/uttids"
        # Copy file from path1 to path2
        shutil.copyfile(source, dest)
    elif name == "test":
        source = "filesets/testing.txt"
        dest = "data/test/uttids"
        # Copy file from path1 to path2
        shutil.copyfile(source, dest)
    else:
        source = "filesets/validation.txt"
```

```
        dest = "data/dev/uttids"
        # Copy file from path1 to path2
        shutil.copyfile(source, dest)

# function to create utt2spk file
def create_utt2spk(name, lines, writeName):
    with open('data/' + name + '/utt2spk', 'w', newline="") as utt2spkFile:
        with open('filesets/' + writeName + '.txt') as f:
            for line,i in zip(f, range(1, lines + 1)):
                utt2spkFile.write(line[:-1] + ' ' + line[:2] + '\n')

# function to create wav.scp file
def create_wavscp(name, lines, writeName):
    with open('data/' + name + '/wav.scp', 'w', newline="") as wavscpFile:
        with open('filesets/' + writeName + '.txt') as f:
            for line,i in zip(f, range(1, lines + 1)):
                wavscpFile.write(line[:-1] + ' wav/' + line[:-1] + '.wav' +
'\n')

# function to create text file
def create_text(name, lines, writeName):
    utt2spk = {}
    with open('data/' + name + '/utt2spk', 'r') as f:
        for line in f:
            key, value = line.strip().split()
            utt2spk[key] = value

    transcriptions = {}
    with open('transcriptions.txt', 'r') as f:
        for line in f:
            line = line.strip().split('\t')
            key = line[0]
            value = line[1]
            transcriptions[key] = value

    with open('data/' + name + '/text', 'w', newline="") as textFile:
        with open('filesets/' + writeName + '.txt') as f:
            for line,i in zip(f, range(1, lines + 1)):
                textFile.write(line[:-1] + ' ' + transcriptions[line[3:6]] +
'\n')

# function to create textPronunciation file
def create_textPronunciation(name):
    with open('lexicon.txt', 'r') as f:
        lexicon = {}
        for line in f:
            lexicon[str(line[:line.find(' ')-1])] = line[line.find(' ')+1:-1]

    with open('data/' + name + '/text', 'r') as textFile:
        with open('data/' + name + '/textPronunciation', 'w', newline="") as
pronFile:
            for line in textFile:
                pronFile.write(line[:line.find(' ')])
                # write 'sil' 2 times for each line
                pronFile.write(' sil')
                # words separated by <space> or <->
                for word in re.split(' |-', line[line.find(' ')+1:-1]):
```

```
                    # uppercase
                    word = word.upper()
                    # remove punctuation exept <'>
                    word = re.sub(r'[^\w\s\']','',word)
                    pronFile.write(' ' + lexicon[word])
                pronFile.write(' sil\n')

# define names, lines, and writeNames
names = ['train', 'test', 'dev']
lines = [1320, 367, 149]
writeNames = ['training', 'testing', 'validation']

for i in range(len(names)):
    create_uttids(names[i], lines[i])
    create_utt2spk(names[i], lines[i], writeNames[i])
    create_wavscp(names[i], lines[i], writeNames[i])
    create_text(names[i], lines[i], writeNames[i])
    create_textPronunciation(names[i])

with open("data/dev/text", "w", newline = "") as textFile:
    with open("data/dev/textPronunciation", "r", newline = "") as
textPronFile:
        for line in textPronFile:
            textFile.write(line)

with open("data/test/text", "w", newline = "") as textFile:
    with open("data/test/textPronunciation", "r", newline = "") as
textPronFile:
        for line in textPronFile:
            textFile.write(line)

with open("data/train/text", "w", newline = "") as textFile:
    with open("data/train/textPronunciation", "r", newline = "") as
textPronFile:
        for line in textPronFile:
            textFile.write(line)
```

4. Βήματα Κυρίως Μέρους

4.1. Προετοιμασία Διαδικασίας Αναγνώρισης Φωνής για τη USC-TIMIT

4.1.1. Setting Up Kaldi Environment for Wall Street Journal (WSJ) Procedure.

Χρησιμοποιώντας έναν editor, ορίζουμε το KALDI_ROOT=`pwd`/kaldi ώστε να δείχνει στον φάκελο του Kaldi και αλλάζουμε τις τιμές των μεταβλητών train_cmd, decode_cmd και cuda_cmd σε run.pl.

Για τη μεταβλητή KALDI_ROOT:

```
vi kaldi/egs/wsj/s5/path.sh
export KALDI_ROOT="/mnt/e/HMMY/github/NLP-lab2/kaldi"
[ -f $KALDI_ROOT/tools/env.sh ] && . $KALDI_ROOT/tools/env.sh
export PATH=$PWD/utils/:$KALDI_ROOT/tools/openfst/bin:$PWD:$PATH
[ ! -f $KALDI_ROOT/tools/config/common_path.sh ] && echo >&2 "The standard
file $KALDI_ROOT/tools/config/common_path.sh is not present -> Exit!" && exit
1
. $KALDI_ROOT/tools/config/common_path.sh
export LC_ALL=C

export PYTHONUNBUFFERED=1
```

Για τις μεταβλητές train_cmd, decode_cmd και cuda_cmd:

```
vi kaldi/egs/wsj/s5/cmd.sh

export train_cmd=run.pl
export decode_cmd="run.pl --mem 2G"
# the use of cuda_cmd is deprecated, used only in 'nnet1',
export cuda_cmd="run.pl --gpu 1"

if [ "$(hostname -d)" == "fit.vutbr.cz" ]; then
  queue_conf=$HOME/queue_conf/default.conf # see example
/homes/kazi/iveselyk/queue_conf/default.conf,
  export train_cmd="queue.pl --config $queue_conf --mem 2G --matylda 0.2"
  export decode_cmd="queue.pl --config $queue_conf --mem 3G --matylda 0.1"
  export cuda_cmd="queue.pl --config $queue_conf --gpu 1 --mem 10G --tmp 40G"
fi
```

4.1.2. Creating Soft Links for Kaldi Process Folder.

Με τις ακόλουθες εντολές bash δημιουργούμε:

- ένα symbolic link με όνομα steps που δείχνει στο directory steps που βρίσκεται στο path kaldi/egs/wsj/s5.
- ένα symbolic link με όνομα utils που δείχνει στο directory utils που βρίσκεται επίσης στο path kaldi/egs/wsj/s5.

Αυτοί οι σύνδεσμοι δημιουργούν ένα shortcut για τα αρχεία των φακέλων steps και utils.

```
ln -s /HMMY/github/NLP-lab2/kaldi/egs/wsj/s5/steps steps

ln -s /HMMY/github/NLP-lab2/kaldi/egs/wsj/s5/utils utils
```

Το αποτέλεσμα φαίνεται με την εντολή `ls -l`:

```
ls -l
```

```
- steps -> /HMMY/github/NLP-lab2/kaldi/egs/wsj/s5/steps
- utils -> /HMMY/github/NLP-lab2/kaldi/egs/wsj/s5/utils
```

4.1.3. Creating Soft Link to File `score_kaldi.sh`.

Με τις ακόλουθες εντολές bash δημιουργούμε στον φάκελο `local` ένα symbolic link με όνομα `score_kaldi.sh` που δείχνει στο αρχείο `score_kaldi.sh` το οποίο βρίσκεται στο path `kaldi/egs/wsj/s5/steps`.

```
mkdir local
```

```
ln -s /kaldi/egs/wsj/s5/steps/score_kaldi.sh local/score_kaldi.sh
```

Το αποτέλεσμα φαίνεται με την εντολή `ls -l local`:

```
ls -l local
```

```
- score_kaldi.sh -> /kaldi/egs/wsj/s5/steps/score_kaldi.sh
```

4.1.4. Creating the `conf` Directory and Copying the `mfcc.conf` File.

Με τις ακόλουθες εντολές bash αντιγράφουμε στον φάκελο `conf` το αρχείο `mfcc.conf`.

```
mkdir conf
```

```
cp ../wsj/s5/conf/mfcc.conf conf/
```

Το αποτέλεσμα φαίνεται με την εντολή `ls -l conf`:

```
ls -l conf/
```

```
- mfcc.conf
```

4.1.5. Creating Data Folders.

Με τις ακόλουθες εντολές δημιουργούμε τα directories `data/lang`, `data/local/dict`, `data/local/lm_tmp`, `data/local/nist_lm`:

```
mkdir -p data/lang
mkdir -p data/local/dict
mkdir -p data/local/lm_tmp
mkdir -p data/local/nist_lm
```

Η δομή των directories κάτω από το directory `data` φαίνεται με την εντολή `tree data`:

```
data/
|-- lang
`-- local
    |-- dict
    |-- lm_tmp
    `-- nist_lm
5 directories, 0 files
```

Το directory `usc` έχει την εξής μορφή στο τέλος του βήματος 4.1:

```
usc/
|-- conf
|   `-- mfcc.conf
|-- data
|   |-- dev
|   |   |-- text
|   |   |-- textPronunciation
|   |   |-- utt2spk
|   |   |-- uttids
|   |   `-- wav.scp
|   |-- lang
|   |-- local
|   |   |-- dict
|   |   |-- lm_tmp
|   |   `-- nist_lm
|   |-- test
|   |   |-- text
|   |   |-- textPronunciation
|   |   |-- utt2spk
|   |   |-- uttids
|   |   `-- wav.scp
|   `-- train
|       |-- text
|       |-- textPronunciation
|       |-- utt2spk
|       |-- uttids
|       `-- wav.scp
|-- filesets
|   |-- testing.txt
|   |-- training.txt
|   `-- validation.txt
|-- lab2.ipynb
|-- lexicon.txt
|-- local
|   `-- score_kaldi.sh -> /kaldi/egs/wsj/s5/steps/score_kaldi.sh
|-- preLabFiles.py
|-- steps -> /HMMY/github/NLP-lab2/kaldi/egs/wsj/s5/steps
|-- transcriptions.txt
|-- utils -> /HMMY/github/NLP-lab2/kaldi/egs/wsj/s5/utils
`-- wav
    |-- f1_001.wav
    |-- f1_002.wav
    |-- f1_003.wav
    ...
```

4.2. Προετοιμασία γλωσσικού μοντέλου

4.2.1. Creating Language Model Files and Directories

Εξηγούμε τα βήματα:

1. Δημιουργία των αρχείων `silence_phones.txt` και `optional_silence.txt`: Μέσα στο φάκελο `dict` δημιουργούνται τα δύο αρχεία `silence_phones.txt` και `optional_silence.txt`. Τα αρχεία `silence_phones.txt`, `optional_silence.txt` περιέχουν μόνο το φώνημα silence (`sil`).

2. Δημιουργία του αρχείου `nonsilence_phones.txt`: Μέσα στο φάκελο `dict`, δημιουργείται το αρχείο `nonsilence_phones.txt`. Αυτό, περιέχει έναν κατάλογο όλων των υπόλοιπων φωνημάτων (εκτός του `sil`) που χρησιμοποιούνται στο γλωσσικό μοντέλο. Κάθε φώνημα γράφεται σε ξεχωριστή γραμμή και ταξινομημένα.

3. Δημιουργία του αρχείου `lexicon.txt`: Μέσα στο φάκελο `dict` δημιουργείται το αρχείο `lexicon.txt`. Το αρχείο `lexicon` θα είναι το λεξικό του γλωσσικού μοντέλου. Το γλωσσικό μοντέλο ασχολείται με την αναγνώριση φωνημάτων και όχι λέξεων, άρα κάθε γραμμή του αρχείου αντιπροσωπεύει ένα φώνημα. Το φώνημα σιωπής "sil" πρέπει επίσης να περιλαμβάνεται στο λεξικό.

4. Δημιουργία του `lm_train.text`: Το αρχείο `lm_train.text` δημιουργείται με την προσθήκη των units <s> (αρχή πρότασης) και </s> (τέλος πρότασης). Με παρόμοιο τρόπο δημιουργούμε και τα αρχεία `lm_test.text` και `lm_dev.text`, για τα test set και validation set.

5. Δημιουργία του αρχείου `extra_questions.txt`: Μέσα στο φάκελο `dict` δημιουργείται και το αρχείο `extra_questions.txt`. Αυτό μένει κενό, καθώς δε θα το χρησιμοποιήσουμε στο γλωσσικό μοντέλο.

```
with open("data/local/dict/silence_phones.txt",'w', newline="") as f:
    f.write("sil\n")
with open("data/local/dict/optional_silence.txt",'w', newline="") as f:
    f.write("sil\n")
phonemesList = []
with open("lexicon.txt", "r") as lexiconFile:
    with open("data/local/dict/nonsilence_phones.txt", "w", newline="") as
nonsilence_phonesFile:
        for line in lexiconFile.readlines():
            toAdd = line.split()[1:]
            for elem in toAdd:
                if elem not in phonemesList:
                    phonemesList.append(elem)
        phonemesList = sorted(phonemesList)
        for phonem in phonemesList:
            if phonem != "sil":
                nonsilence_phonesFile.write(phonem + "\n")
with open("data/local/dict/lexicon.txt", "w", newline="") as lexiconFile:
    for phonem in phonemesList:
        lexiconFile.write(phonem + " " + phonem + "\n")
with open("data/dev/text", "r") as devTextFile:
```

```
    with open("data/local/dict/lm_dev", "w", newline="") as lm_devFile:
        for line in devTextFile:
            idx = line.find(' ')
            lm_devFile.write("<s> " + line[idx+1:-1] + " </s>\n")

with open("data/test/text", "r") as testTextFile:
    with open("data/local/dict/lm_test", "w", newline="") as lm_testFile:
        for line in testTextFile:
            idx = line.find(' ')
            lm_testFile.write("<s> " + line[idx+1:-1] + " </s>\n")

with open("data/train/text", "r") as trainTextFile:
    with open("data/local/dict/lm_train", "w", newline="") as lm_trainFile:
        for line in trainTextFile:
            idx = line.find(' ')
            lm_trainFile.write("<s> " + line[idx+1:-1] + " </s>\n")
with open("data/local/dict/extra_questions.txt", "w+", newline="") as f:
    pass
```

4.2.2. Creating Intermediate Language Model

Χρησιμοποιούμε την εντολή `build-lm.sh` για τη δημιουργία του γλωσσικού μοντέλου. Χρησιμοποιούνται οι ακόλουθες παράμετροι:

- `-i "lm_{train, test, dev}"`: Καθορίζει το αρχείο εισόδου για το γλωσσικό μοντέλο, το οποίο είναι το αρχείο lm_train.text που δημιουργήσαμε νωρίτερα.
- `-n {1, 2}`: Καθορίζει την κλάση του γλωσσικού μοντέλου (unigram/bigram).
- `-o lm_{train, test, dev}{Uni, Bi}.ilm.gz`: Ορίζει το όνομα και τη μορφή του αρχείου εξόδου για το ενδιάμεσο γλωσσικό μοντέλο.

Η εντολή αυτή χρησιμοποιώντας το αρχείο `lm_{train, test, dev}.text` δημιουργεί το ενδιάμεσο γλωσσικό μοντέλο.

Το shell script έχει ως εξής:

```
source ./path.sh

export IRSTLM=$KALDI_ROOT/tools/irstlm/
export PATH=${PATH}:$IRSTLM/bin

cd "data/local/dict"

build-lm.sh -i "lm_train" -n 1 -o lm_trainUni.ilm.gz
build-lm.sh -i "lm_train" -n 2 -o lm_trainBi.ilm.gz
build-lm.sh -i "lm_test" -n 1 -o lm_testUni.ilm.gz
build-lm.sh -i "lm_test" -n 2 -o lm_testBi.ilm.gz
build-lm.sh -i "lm_dev" -n 1 -o lm_devUni.ilm.gz
build-lm.sh -i "lm_dev" -n 2 -o lm_devBi.ilm.gz
```

Και το output φαίνεται εδώ:

```
./getIntermediateLangModel.sh
LOGFILE:/dev/null
```

```
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c >
$tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary
$additional_parameters >> $logfile 2>&1
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c >
$tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary
$additional_parameters >> $logfile 2>&1
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing
"$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c
$tmpdir/ngram.${sdict}.gz" -sublm $tmpdir/lm.$sdict $additional_parameters >>
$logfile 2>&1
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c >
$tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary
$additional_parameters >> $logfile 2>&1
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing
"$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c
$tmpdir/ngram.${sdict}.gz" -sublm $tmpdir/lm.$sdict $additional_parameters >>
$logfile 2>&1
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c >
$tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary
$additional_parameters >> $logfile 2>&1
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing
"$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c
$tmpdir/ngram.${sdict}.gz" -sublm $tmpdir/lm.$sdict $additional_parameters >>
$logfile 2>&1
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c >
$tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary
$additional_parameters >> $logfile 2>&1
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing
"$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c
$tmpdir/ngram.${sdict}.gz" -sublm $tmpdir/lm.$sdict $additional_parameters >>
$logfile 2>&1
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c >
$tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary
$additional_parameters >> $logfile 2>&1
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing
"$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c
$tmpdir/ngram.${sdict}.gz" -sublm $tmpdir/lm.$sdict $additional_parameters >>
$logfile 2>&1
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c >
$tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary
$additional_parameters >> $logfile 2>&1
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing
"$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c
$tmpdir/ngram.${sdict}.gz" -sublm $tmpdir/lm.$sdict $additional_parameters >>
$logfile 2>&1
```

4.2.3. Compiling Language Model in ARPA Format

Χρησιμοποιούμε την εντολή `compile-lm` για τη μεταγλώττιση του ενδιάμεσου γλωσσικού μοντέλου στη μορφή ARPA. Έχουμε τις ακόλουθες παραμέτρους:

- `lm_{train, test, dev}{Uni, Bi}.ilm.gz`: Είναι το αρχείο του ενδιάμεσου γλωσσικού μοντέλου που δημιουργήθηκε στο προηγούμενο βήμα και είναι η είσοδος της συνάρτησης.
- `/dev/stdout`: Τυπώνει την έξοδο του μεταγλωττισμένου Language Model.
- `grep -v unk`: Φιλτράρει τις γραμμές που περιέχουν το σύμβολο unk, δηλαδή τις άγνωστες λέξεις.
- `gzip -c > ../nist_lm/lm_{train, test, dev}{Uni, Bi}.arpa.gz`: Αποθηκεύει τη μεταγλωττισμένη έξοδο του LM.

Το script που γράψαμε είναι:

```
source ./path.sh

export IRSTLM=$KALDI_ROOT/tools/irstlm/
export PATH=${PATH}:$IRSTLM/bin

cd "data/local/lm_tmp"

compile-lm lm_trainUni.ilm.gz --text=yes lm_trainUni.lm
compile-lm lm_trainUni.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
../nist_lm/lm_trainUni.arpa.gz

compile-lm lm_trainBi.ilm.gz --text=yes lm_trainBi.lm
compile-lm lm_trainBi.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
../nist_lm/lm_trainBi.arpa.gz

compile-lm lm_testUni.ilm.gz --text=yes lm_testUni.lm
compile-lm lm_testUni.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
../nist_lm/lm_testUni.arpa.gz

compile-lm lm_testBi.ilm.gz --text=yes lm_testBi.lm
compile-lm lm_testBi.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
../nist_lm/lm_testBi.arpa.gz

compile-lm lm_devUni.ilm.gz --text=yes lm_devUni.lm
compile-lm lm_devUni.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
../nist_lm/lm_devUni.arpa.gz

compile-lm lm_devBi.ilm.gz --text=yes lm_devBi.lm
compile-lm lm_devBi.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
../nist_lm/lm_devBi.arpa.gz
```

Και η έξοδός του φαίνεται κάτω:

```
./getARPAmodel.sh
inpfile: lm_trainUni.ilm.gz
outfile: lm_trainUni.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 1600
OOV code is 1600
Saving in txt format to lm_trainUni.lm
inpfile: lm_trainUni.ilm.gz
outfile: /dev/stdout
```

```
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 1600
OOV code is 1600
Saving in txt format to /dev/stdout
inpfile: lm_trainBi.ilm.gz
outfile: lm_trainBi.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 1600
OOV code is 1600
Saving in txt format to lm_trainBi.lm
inpfile: lm_trainBi.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 1600
OOV code is 1600
Saving in txt format to /dev/stdout
inpfile: lm_testUni.ilm.gz
outfile: lm_testUni.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 499
OOV code is 499
Saving in txt format to lm_testUni.lm
inpfile: lm_testUni.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 499
OOV code is 499
Saving in txt format to /dev/stdout
inpfile: lm_testBi.ilm.gz
outfile: lm_testBi.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 499
OOV code is 499
Saving in txt format to lm_testBi.lm
inpfile: lm_testBi.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 499
OOV code is 499
Saving in txt format to /dev/stdout
inpfile: lm_devUni.ilm.gz
outfile: lm_devUni.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 326
OOV code is 326
Saving in txt format to lm_devUni.lm
inpfile: lm_devUni.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
```

```
dub: 10000000
OOV code is 326
OOV code is 326
Saving in txt format to /dev/stdout
inpfile: lm_devBi.ilm.gz
outfile: lm_devBi.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 326
OOV code is 326
Saving in txt format to lm_devBi.lm
inpfile: lm_devBi.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 326
OOV code is 326
Saving in txt format to /dev/stdout
```

4.2.4. Creating FST Language Dictionary with `prepare_lang.sh`

Η εντολή `prepare_lang.sh` παράγει το λεξικό του FST και τα απαραίτητα αρχεία για το ακουστικό μοντέλο.

Έχουμε το script:

```
source ./path.sh
source ./cmd.sh

export IRSTLM="$KALDI_ROOT/tools/irstlm/"
export PATH="${PATH}:$IRSTLM/bin"

./utils/prepare_lang.sh data/local/dict "<oov>" data/local/lm_tmp data/lang
```

Το οποίο δίνει τα εξής αποτελέσματα:

```
./getLangFST.sh
./utils/prepare_lang.sh data/local/dict <oov> data/local/lm_tmp data/lang
Checking data/local/dict/silence_phones.txt ...
--> reading data/local/dict/silence_phones.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/silence_phones.txt is OK

Checking data/local/dict/optional_silence.txt ...
--> reading data/local/dict/optional_silence.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/optional_silence.txt is OK

Checking data/local/dict/nonsilence_phones.txt ...
--> reading data/local/dict/nonsilence_phones.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/nonsilence_phones.txt is OK
```

```
Checking disjoint: silence_phones.txt, nonsilence_phones.txt
--> disjoint property is OK.

Checking data/local/dict/lexicon.txt
--> reading data/local/dict/lexicon.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/lexicon.txt is OK

Checking data/local/dict/lexiconp.txt
--> reading data/local/dict/lexiconp.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/lexiconp.txt is OK

Checking lexicon pair data/local/dict/lexicon.txt and
data/local/dict/lexiconp.txt
--> lexicon pair data/local/dict/lexicon.txt and data/local/dict/lexiconp.txt
match

Checking data/local/dict/extra_questions.txt ...
--> data/local/dict/extra_questions.txt is empty (this is OK)
--> SUCCESS [validating dictionary directory data/local/dict]

fstaddselfloops data/lang/phones/wdisambig_phones.int
data/lang/phones/wdisambig_words.int
prepare_lang.sh: validating output directory
utils/validate_lang.pl data/lang
Checking existence of separator file
separator file data/lang/subword_separator.txt is empty or does not exist,
deal in word case.
Checking data/lang/phones.txt ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/lang/phones.txt is OK

Checking words.txt: #0 ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/lang/words.txt is OK

Checking disjoint: silence.txt, nonsilence.txt, disambig.txt ...
--> silence.txt and nonsilence.txt are disjoint
--> silence.txt and disambig.txt are disjoint
--> disambig.txt and nonsilence.txt are disjoint
--> disjoint property is OK

Checking sumation: silence.txt, nonsilence.txt, disambig.txt ...
--> found no unexplainable phones in phones.txt

Checking data/lang/phones/context_indep.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 5 entry/entries in data/lang/phones/context_indep.txt
--> data/lang/phones/context_indep.int corresponds to
data/lang/phones/context_indep.txt
```

```
--> data/lang/phones/context_indep.csl corresponds to
data/lang/phones/context_indep.txt
--> data/lang/phones/context_indep.{txt, int, csl} are OK

Checking data/lang/phones/nonsilence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 160 entry/entries in data/lang/phones/nonsilence.txt
--> data/lang/phones/nonsilence.int corresponds to
data/lang/phones/nonsilence.txt
--> data/lang/phones/nonsilence.csl corresponds to
data/lang/phones/nonsilence.txt
--> data/lang/phones/nonsilence.{txt, int, csl} are OK

Checking data/lang/phones/silence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 5 entry/entries in data/lang/phones/silence.txt
--> data/lang/phones/silence.int corresponds to data/lang/phones/silence.txt
--> data/lang/phones/silence.csl corresponds to data/lang/phones/silence.txt
--> data/lang/phones/silence.{txt, int, csl} are OK

Checking data/lang/phones/optional_silence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.int corresponds to
data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.csl corresponds to
data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.{txt, int, csl} are OK

Checking data/lang/phones/disambig.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 2 entry/entries in data/lang/phones/disambig.txt
--> data/lang/phones/disambig.int corresponds to
data/lang/phones/disambig.txt
--> data/lang/phones/disambig.csl corresponds to
data/lang/phones/disambig.txt
--> data/lang/phones/disambig.{txt, int, csl} are OK

Checking data/lang/phones/roots.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 41 entry/entries in data/lang/phones/roots.txt
--> data/lang/phones/roots.int corresponds to data/lang/phones/roots.txt
--> data/lang/phones/roots.{txt, int} are OK

Checking data/lang/phones/sets.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 41 entry/entries in data/lang/phones/sets.txt
--> data/lang/phones/sets.int corresponds to data/lang/phones/sets.txt
--> data/lang/phones/sets.{txt, int} are OK

Checking data/lang/phones/extra_questions.{txt, int} ...
```

```
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 9 entry/entries in data/lang/phones/extra_questions.txt
--> data/lang/phones/extra_questions.int corresponds to
data/lang/phones/extra_questions.txt
--> data/lang/phones/extra_questions.{txt, int} are OK

Checking data/lang/phones/word_boundary.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 165 entry/entries in data/lang/phones/word_boundary.txt
--> data/lang/phones/word_boundary.int corresponds to
data/lang/phones/word_boundary.txt
--> data/lang/phones/word_boundary.{txt, int} are OK

Checking optional_silence.txt ...
--> reading data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.txt is OK

Checking disambiguation symbols: #0 and #1
--> data/lang/phones/disambig.txt has "#0" and "#1"
--> data/lang/phones/disambig.txt is OK

Checking topo ...

Checking word_boundary.txt: silence.txt, nonsilence.txt, disambig.txt ...
--> data/lang/phones/word_boundary.txt doesn't include disambiguation symbols
--> data/lang/phones/word_boundary.txt is the union of nonsilence.txt and
silence.txt
--> data/lang/phones/word_boundary.txt is OK

Checking word-level disambiguation symbols...
--> data/lang/phones/wdisambig.txt exists (newer prepare_lang.sh)
Checking word_boundary.int and disambig.int
--> generating a 65 word/subword sequence
--> resulting phone sequence from L.fst corresponds to the word sequence
--> L.fst is OK
--> generating a 88 word/subword sequence
--> resulting phone sequence from L_disambig.fst corresponds to the word
sequence
--> L_disambig.fst is OK

Checking data/lang/oov.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang/oov.txt
--> data/lang/oov.int corresponds to data/lang/oov.txt
--> data/lang/oov.{txt, int} are OK

--> data/lang/L.fst is olabel sorted
--> data/lang/L_disambig.fst is olabel sorted
--> SUCCESS [validating lang directory data/lang]
```

4.2.5. Sorting Files

Με το ακόλουθο script ταξινομούμε τα αρχεία που φτιάξαμε ώστε να τα εισάγουμε στα μοντέλα:

```
sort data/dev/wav.scp
sort data/dev/text
sort data/dev/utt2spk
sort data/test/wav.scp
sort data/test/text
sort data/test/utt2spk
sort data/train/wav.scp
sort data/train/text
sort data/train/utt2spk
```

Και εκτελούμε ως εξής:

```
./sortFiles.sh
```

4.2.6. Creating the spk2utt File

Τρέχουμε το πρόγραμμα `utt2spk_to_spk2utt.pl` το οποίο μετατρέπει το `utt2spk` στο `spk2utt`, το οποίο αντιστοιχίζει τους ομιλητές στα δείγματα που τους ανήκουν, για κάθε dataset.

Εκτελούμε το ακόλουθο script:

```
source ./path.sh

./utils/utt2spk_to_spk2utt.pl data/train/utt2spk > data/train/spk2utt
./utils/utt2spk_to_spk2utt.pl data/test/utt2spk > data/test/spk2utt
./utils/utt2spk_to_spk2utt.pl data/dev/utt2spk > data/dev/spk2utt
```

Τα αποτελέσματα φαίνονται εδώ:

```
./getspk2utt.sh
```

Για παράδειγμα, στο directory `test` παίρνουμε το αρχείο `spk2utt` το οποίο δηλώνει τα uttids του κάθε ομιλητή για το testing:

```
f1 f1_002 f1_014 f1_027 f1_039 f1_052 f1_064 f1_077 f1_089 f1_102 f1_114
f1_127 f1_139 f1_152 f1_164 f1_177 f1_189 f1_202 f1_214 f1_227 f1_239 f1_252
f1_264 f1_277 f1_289 f1_302 f1_314 f1_327 f1_339 f1_352 f1_364 f1_377 f1_389
f1_402 f1_414 f1_427 f1_439 f1_452

f5 f5_002 f5_014 f5_027 f5_039 f5_052 f5_064 f5_077 f5_089 f5_102 f5_114
f5_127 f5_139 f5_152 f5_164 f5_177 f5_189 f5_202 f5_214 f5_227 f5_239 f5_252
f5_264 f5_277 f5_289 f5_302 f5_314 f5_327 f5_339 f5_352 f5_364 f5_377 f5_389
f5_402 f5_414 f5_427 f5_439 f5_452

m1 m1_002 m1_014 m1_027 m1_039 m1_052 m1_064 m1_077 m1_089 m1_102 m1_114
m1_127 m1_139 m1_152 m1_164 m1_177 m1_189 m1_202 m1_214 m1_227 m1_244 m1_257
m1_269 m1_282 m1_294 m1_307 m1_319 m1_332 m1_344 m1_357 m1_369 m1_382 m1_394
m1_407 m1_419 m1_432 m1_444 m1_457

m3 m3_002 m3_014 m3_027 m3_039 m3_052 m3_064 m3_077 m3_089 m3_102 m3_114
m3_127 m3_139 m3_152 m3_164 m3_177 m3_189 m3_202 m3_214 m3_227 m3_239 m3_252
m3_264 m3_277 m3_289 m3_302 m3_314 m3_327 m3_339 m3_352 m3_364 m3_377 m3_389
m3_402 m3_414 m3_427 m3_439 m3_452
```

Ομοίως και για το training και validation στα directories `train`, `dev`

4.2.7. Creating the FST of the Grammar

Η διαδικασία timit του Kaldi είναι η εξής:

```
#!/bin/bash

# Copyright 2013  (Author: Daniel Povey)
# Apache 2.0

# This script takes data prepared in a corpus-dependent way
# in data/local/, and converts it into the "canonical" form,
# in various subdirectories of data/, e.g. data/lang, data/train, etc.

source ./path.sh || exit 1;

echo "Preparing train, dev and test data"
srcdir=data/local/dict
lmdir=data/local/nist_lm
tmpdir=data/local/lm_tmp
lexicon=data/local/dict/lexicon.txt
mkdir -p $tmpdir



# Next, for each type of language model, create the corresponding FST
# and the corresponding lang_test_* directory.

echo "Preparing language models for test"

for x in train dev test; do
  test=data/lang_test
  mkdir -p $test
  cp -r data/lang/* $test


  gunzip -c $lmdir/lm_${x}Uni.arpa.gz \
   | arpa2fst --disambig-symbol=#0 \
              --read-symbol-table=$test/words.txt -
data/lang_test/G_${x}Uni.fst
 # The output is like:
 # 9.14233e-05 -0.259833
 # we do expect the first of these 2 numbers to be close to zero (the second
is
 # nonzero because the backoff weights make the states sum to >1).
 # Because of the <s> fiasco for these particular LMs, the first number is
not
 # as close to zero as it could be.

done



for x in train dev test; do
  test=data/lang_test

  gunzip -c $lmdir/lm_${x}Bi.arpa.gz \
   | arpa2fst --disambig-symbol=#0 \
```

```
            --read-symbol-table=$test/words.txt -
data/lang_test/G_${x}Bi.fst
 # The output is like:
 # 9.14233e-05 -0.259833
 # we do expect the first of these 2 numbers to be close to zero (the second
is
 # nonzero because the backoff weights make the states sum to >1).
 # Because of the <s> fiasco for these particular LMs, the first number is
not
 # as close to zero as it could be.

done
#utils/validate_lang.pl data/lang_test || exit 1
echo "Succeeded in formatting data."
```

Την εκτελούμε και έχουμε τα κατώθι αποτελέσματα:

```
./timit_format_data.sh
Preparing train, dev and test data
Preparing language models for test
arpa2fst --disambig-symbol=#0 --read-symbol-table=data/lang_test/words.txt -
data/lang_test/G_trainUni.fst
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:94) Reading \data\
section.
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:149) Reading \1-
grams: section.
LOG (arpa2fst[5.5.1071~1-19185]:RemoveRedundantStates():arpa-lm-
compiler.cc:359) Reduced num-states from 1 to 1
arpa2fst --disambig-symbol=#0 --read-symbol-table=data/lang_test/words.txt -
data/lang_test/G_devUni.fst
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:94) Reading \data\
section.
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:149) Reading \1-
grams: section.
LOG (arpa2fst[5.5.1071~1-19185]:RemoveRedundantStates():arpa-lm-
compiler.cc:359) Reduced num-states from 1 to 1
arpa2fst --disambig-symbol=#0 --read-symbol-table=data/lang_test/words.txt -
data/lang_test/G_testUni.fst
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:94) Reading \data\
section.
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:149) Reading \1-
grams: section.
LOG (arpa2fst[5.5.1071~1-19185]:RemoveRedundantStates():arpa-lm-
compiler.cc:359) Reduced num-states from 1 to 1
arpa2fst --disambig-symbol=#0 --read-symbol-table=data/lang_test/words.txt -
data/lang_test/G_trainBi.fst
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:94) Reading \data\
section.
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:149) Reading \1-
grams: section.
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:149) Reading \2-
grams: section.
WARNING (arpa2fst[5.5.1071~1-19185]:ConsumeNGram():arpa-lm-compiler.cc:313)
line 52 [-2.82084   <s> <s>] skipped: n-gram has invalid BOS/EOS placement
LOG (arpa2fst[5.5.1071~1-19185]:RemoveRedundantStates():arpa-lm-
compiler.cc:359) Reduced num-states from 42 to 42
```

```
arpa2fst --disambig-symbol=#0 --read-symbol-table=data/lang_test/words.txt -
data/lang_test/G_devBi.fst
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:94) Reading \data\
section.
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:149) Reading \1-
grams: section.
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:149) Reading \2-
grams: section.
WARNING (arpa2fst[5.5.1071~1-19185]:ConsumeNGram():arpa-lm-compiler.cc:313)
line 52 [-1.88064   <s> <s>] skipped: n-gram has invalid BOS/EOS placement
LOG (arpa2fst[5.5.1071~1-19185]:RemoveRedundantStates():arpa-lm-
compiler.cc:359) Reduced num-states from 42 to 42
arpa2fst --disambig-symbol=#0 --read-symbol-table=data/lang_test/words.txt -
data/lang_test/G_testBi.fst
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:94) Reading \data\
section.
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:149) Reading \1-
grams: section.
LOG (arpa2fst[5.5.1071~1-19185]:Read():arpa-file-parser.cc:149) Reading \2-
grams: section.
WARNING (arpa2fst[5.5.1071~1-19185]:ConsumeNGram():arpa-lm-compiler.cc:313)
line 52 [-2.26828   <s> <s>] skipped: n-gram has invalid BOS/EOS placement
LOG (arpa2fst[5.5.1071~1-19185]:RemoveRedundantStates():arpa-lm-
compiler.cc:359) Reduced num-states from 42 to 42
Succeeded in formatting data.
```

Κώδικας για το ερώτημα 1

```
source ./path.sh || exit 1;
source ./cmd.sh || exit 1;

export IRSTLM=$KALDI_ROOT/tools/irstlm/
export PATH=${PATH}:$IRSTLM/bin

echo "+-+-+-+ Evaluating data unigram model's perplexity +-+-+-+"
prune-lm --threshold=1e-6,1e-6 data/local/lm_tmp/lm_devUni.ilm.gz
data/local/lm_tmp/lm_devUni.plm
compile-lm data/local/lm_tmp/lm_devUni.lm --eval=data/local/dict/lm_dev --
dub=10000000
echo "+-+-+-+-+-+-+-+ Process  Completed +-+-+-+-+-+-+-+-+"

echo "+-+-+-+ Evaluation data bigram model's perplexity +-+-+-+"
prune-lm --threshold=1e-6,1e-6 data/local/lm_tmp/lm_devBi.ilm.gz
data/local/lm_tmp/lm_devBi.plm
compile-lm data/local/lm_tmp/lm_devBi.lm --eval=data/local/dict/lm_dev --
dub=10000000
echo "+-+-+-+-+-+-+-+ Process  Completed +-+-+-+-+-+-+-+-+"

echo "+-+-+-+ Test data unigram model's perplexity +-+-+-+"
prune-lm --threshold=1e-6,1e-6 data/local/lm_tmp/lm_testUni.ilm.gz
data/local/lm_tmp/lm_testUni.plm
compile-lm data/local/lm_tmp/lm_testUni.lm --eval=data/local/dict/lm_test --
dub=10000000
echo "+-+-+-+-+-+-+-+ Process  Completed +-+-+-+-+-+-+-+-+"
```

```
echo "+-+-+-+ Test data bigram model's perplexity +-+-+-+"
prune-lm --threshold=1e-6,1e-6 data/local/lm_tmp/lm_testBi.ilm.gz
data/local/lm_tmp/lm_testBi.plm
compile-lm data/local/lm_tmp/lm_testBi.lm --eval=data/local/dict/lm_test --
dub=10000000
echo "+-+-+-+-+-+-+-+ Process  Completed +-+-+-+-+-+-+-+-+"
```

Εκτελούμε:

```
./calculatePerplexity.sh
+-+-+-+ Evaluating data unigram model's perplexity +-+-+-+
OOV code is 42
OOV code is 42
pruning LM with thresholds:

DEBUG_LEVEL:0/1 Everything OK
inpfile: data/local/lm_tmp/lm_devUni.lm
outfile: lm_devUni.lm.blm
evalfile: data/local/dict/lm_dev
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Start Eval
OOV code: 42
%% Nw=4930 PP=32.23 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
+-+-+-+-+-+-+-+ Process  Completed +-+-+-+-+-+-+-+-+
+-+-+-+ Evaluation data bigram model's perplexity +-+-+-+
OOV code is 42
OOV code is 42
pruning LM with thresholds:
 1e-06
DEBUG_LEVEL:0/1 Everything OK
inpfile: data/local/lm_tmp/lm_devBi.lm
outfile: lm_devBi.lm.blm
evalfile: data/local/dict/lm_dev
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Start Eval
OOV code: 42
%% Nw=4930 PP=12.47 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
+-+-+-+-+-+-+-+ Process  Completed +-+-+-+-+-+-+-+-+
+-+-+-+ Test data unigram model's perplexity +-+-+-+
OOV code is 42
OOV code is 42
pruning LM with thresholds:

DEBUG_LEVEL:0/1 Everything OK
inpfile: data/local/lm_tmp/lm_testUni.lm
outfile: lm_testUni.lm.blm
evalfile: data/local/dict/lm_test
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
```

```
Start Eval
OOV code: 42
%% Nw=12795 PP=31.87 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
+-+-+-+-+-+-+-+ Process  Completed +-+-+-+-+-+-+-+-+
+-+-+-+ Test data bigram model's perplexity +-+-+-+
OOV code is 42
OOV code is 42
pruning LM with thresholds:
 1e-06
DEBUG_LEVEL:0/1 Everything OK
inpfile: data/local/lm_tmp/lm_testBi.lm
outfile: lm_testBi.lm.blm
evalfile: data/local/dict/lm_test
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Start Eval
OOV code: 42
%% Nw=12795 PP=13.26 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
+-+-+-+-+-+-+-+ Process  Completed +-+-+-+-+-+-+-+-+
```

Ερώτημα 1

Οι τιμές Perplexity που υπολογίσαμε για τα γλωσσικά μοντέλα στα evaluation και test sets έχουν ως εξής:

- Μοντέλο Unigram για το validation set: $Perplexity\ (PP) = 32.23$
- Μοντέλο Bigram για το validation set: Perplexity (PP) = 12,47
- Μοντέλο Unigram για το test set: Perplexity (PP) = 31.87
- Μοντέλο Bigram για το test set: Perplexity (PP) = 13.26

Η Perplexity μετρά την αβεβαιότητα ή την "έκπληξη" ("surprisal") ενός γλωσσικού μοντέλου κατά την πρόβλεψη του επόμενου token σε μια ακολουθία. Μια χαμηλότερη τιμή Perplexity υποδηλώνει ότι το γλωσσικό μοντέλο είναι πιο «σίγουρο» για τα δεδομένα στα οποία εκπαιδεύεται. Με άλλα λόγια, οι χαμηλότερες τιμές Perplexity δηλώνουν ότι το μοντέλο μπορεί να προβλέψει το επόμενο token με μεγαλύτερη ακρίβεια.

Συγκρίνοντας τις τιμές Perplexity μεταξύ των μοντέλων, παρατηρούμε ότι το μοντέλο bigram αποδίδει αρκετά καλύτερα από το unigram και στα 2 datasets, αφού έχει χαμηλότερη τιμή Perplexity. Αυτό δείχνει ότι το μοντέλο bigram «κατανοεί» περισσότερες διαδοχικές εξαρτήσεις μεταξύ των tokens και κάνει ακριβέστερς προβλέψεις.

Συνολικά, επιθυμούμε χαμηλότερες τιμές Perplexity, καθώς σχετίζονται με την καλύτερη απόδοση του μοντέλου αναφορικά με την πρόβλεψη του επόμενου συμβόλου σε μια ακολουθία.

Η μετρική, λοιπόν, που χρησιμοποιούμε στα μοντέλα μας δεν είναι η απλή πιθανότητα, αλλά η Perplexity. Ορίζεται ως εξής[5]: $perplexity(W) = P(w_1 w_2 \dots w_N)^{-1/N}$

Για ένα unigram μοντέλο έχουμε:	Για ένα bigram μοντέλο έχουμε:
$$perplexity(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i)}}$$	$$perplexity(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_{i-1})}}$$

Παρατηρούμε ότι η Perplexity είναι ο γεωμετρικός μέσος των αντιστρόφων των πιθανοτήτων εμφάνισης tokens για τα n – gram models.

4.3. Εξαγωγή ακουστικών χαρακτηριστικών

Για κάθε dataset (train, test και dev), εξάγει χαρακτηριστικά MFCC μέσω του steps/make_mfcc.sh

Το σενάριο παράγει χαρακτηριστικά MFCC για το δεδομένο σύνολο με την επεξεργασία των δεδομένων ήχου.

Χρησιμοποιεί το αρχείο ρυθμίσεων `conf/mfcc.conf` για τις παραμέτρους εξαγωγής χαρακτηριστικών MFCC.

Κατόπιν, υπολογίζει τα στατιστικά στοιχεία CMVN (Cepstral Mean Variance Normalization) χρησιμοποιώντας `compute_cmvn_stats.sh`.

Τέλος, υπάρχει το script `feat-to-dim` τυπώνει τις διαστάσεις των χαρακτηριστικών για το σύνολο εκπαίδευσης. Βλέπουμε ότι η διαστατικότητα των χαρακτηριστικών είναι 13.

```
source ./path.sh
source ./cmd.sh

for set in train test dev; do
    echo "Extracting MFCC features from $set set"
        steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj 12 --write-
utt2num-frames true --write-utt2dur true data/$set data/mfcc/$set mfcc ||
exit 1;
        echo "Computing CMVN stats for $set set"
        steps/compute_cmvn_stats.sh data/$set || exit 1;
    echo ""
done
```

Και έχουμε τα αποτελέσματα:

```
./getAcousticCharacteristics.sh
Extracting MFCC features from train set
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj 12 --write-utt2num-
frames true --write-utt2dur true data/train data/mfcc/train mfcc
utils/validate_data_dir.sh: Successfully validated data-directory data/train
steps/make_mfcc.sh: [info]: no segments file exists: assuming wav.scp indexed
by utterance.
steps/make_mfcc.sh: Succeeded creating MFCC features for train
Computing CMVN stats for train set
steps/compute_cmvn_stats.sh data/train
Succeeded creating CMVN stats for train

Extracting MFCC features from test set
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj 12 --write-utt2num-
frames true --write-utt2dur true data/test data/mfcc/test mfcc
utils/validate_data_dir.sh: Successfully validated data-directory data/test
steps/make_mfcc.sh: [info]: no segments file exists: assuming wav.scp indexed
by utterance.
steps/make_mfcc.sh: It seems not all of the feature files were successfully
procesed (366 != 367); consider using utils/fix_data_dir.sh data/test
steps/make_mfcc.sh: Succeeded creating MFCC features for test
Computing CMVN stats for test set
steps/compute_cmvn_stats.sh data/test
```

```
Succeeded creating CMVN stats for test

Extracting MFCC features from dev set
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj 12 --write-utt2num-
frames true --write-utt2dur true data/dev data/mfcc/dev mfcc
utils/validate_data_dir.sh: Successfully validated data-directory data/dev
steps/make_mfcc.sh: [info]: no segments file exists: assuming wav.scp indexed
by utterance.
steps/make_mfcc.sh: Succeeded creating MFCC features for dev
Computing CMVN stats for dev set
steps/compute_cmvn_stats.sh data/dev
Succeeded creating CMVN stats for dev
```

Επιπλέον για τη διαστατικότητα:

```
source ./path.sh

echo "Printing feature dimensions for train set:"
../../src/featbin/feat-to-dim scp:data/train/feats.scp -
```

Με αποτέλεσμα:

```
./getFeaturesDimensions.sh
Printing feature dimensions for train set:
../../src/featbin/feat-to-dim scp:data/train/feats.scp -
13
```

Ερώτημα 2

Ο σκοπός του υπολογισμού των Cepstral Mean and Variance Normalization (CMVN) στην αναγνώριση ομιλίας είναι να κανονικοποιηθούν οι cepstral συντελεστές ώστε να έχουν consistent στατιστικές ιδιότητες σε διαφορετικά δείγματα ή τμήματα ομιλίας. Η κανονικοποίηση βοηθά στη μείωση του θορύβου, των διαταραχών του καναλιού (ατελειών των δειγμάτων) και των μεταβολών στα χαρακτηριστικά του ομιλητή. Έτσι, έχουμε βελτίωση του robustness και της ακρίβειας του συστήματος.

Η CMVN μετασχηματίζει τους Cepstral συντελεστές της segmented ομιλίας ώστε να έχουν μηδενική μέση τιμή και μοναδιαία διακύμανση [8].

Στην αναγνώριση ομιλίας θέλουμε να αποφύγουμε οποιαδήποτε διαταραχή του συστήματος (δωμάτιο καταγραφής, μεταβολές στη χρειά της ομιλίας) από το οποίο διέρχεται το αρχικό σήμα $x[n]$. Αν η κρουστική απόκριση του συστήματος είναι $h[n]$, το σήμα που έχει καταγραφεί θα είναι:

$$y[n] = x[n] * h[n]$$

Το οποίο στο πεδίο της συχνότητας είναι:

$$Y[f] = X[f] \cdot H[f]$$

Αν υπολογίσουμε το cepstrum αυτού, έχουμε:

$$Y[q] = \log Y[f] = \log(X[f] \cdot H[f]) = X[q] + H[q]$$

Η μεταβλητή q είναι γνωστή με το όνομα *quefrency*. Φαίνεται λοιπόν πως το cepstrum του τελικού σήματος είναι το άθροισμα των cepstra. Ας δούμε τι επίπτωση έχουν διάφορες διαταραχές στο cepstrum του τελικού σήματος:
Έστω ότι είναι απλώς στατικές. Τότε για το $i - οστό$ frame θα είναι:

$$Y_i[q] = H[q] + X_i[q]$$

Με mean normalization, όμως, θα έχουμε:

$$\frac{1}{N}\sum_i Y_i[q] = H[q] + \frac{1}{N}\sum_i X_i[q]$$

Η διαφορά είναι εμφανής αλλά με μαθηματικά μπορούμε να δούμε:

$$Diff_i[q] = Y_i[q] - \frac{1}{N}\sum_j Y_j[q] = H[q] + X_i[q] - \left(H[q] + \frac{1}{N}\sum_j X_j[q]\right) =$$

$$= X_i[q] - \frac{1}{N}\sum_j X_j[q]$$

Φαίνεται πως είναι πολύ καλύτερο έτσι.

Ερώτημα 3

Όπως εξηγήσαμε στην περιγραφή του άνω αλγορίθμου, η διάσταση των χαρακτηριστικών είναι 13 και όσο για τα ακουστικά frames (βρίσκονται στο αρχείο uttnum_frames που παράξαμε) των πρώτων 5 προτάσεων του training set έχουμε:

ID	Number of Frames
f1_003	317
f1_004	371
f1_005	399
f1_007	328
f1_008	464

4.4. Εκπαίδευση Ακουστικών Μοντέλων και Αποκωδικοποίηση Προτάσεων

4.4.1. Monophonic GMM-HMM Acoustic Model Training

Σε αυτό το βήμα εκτελούμε εκπαίδευση και ευθυγράμμιση του μονοφωνικού ακουστικού μοντέλου GMM – HMM. Τα βήματα είναι τα εξής:

Το train_mono.sh εεκπαιδεύει το μονοφωνικό ακουστικό μοντέλο GMM – HMM χρησιμοποιώντας το train dataset.

Το align_si.sh, χρησιμοποιώντας το εκπαιδευμένο μονοφωνικό μοντέλο, ευθυγραμμίζει αναγκαστικά (forced alignment) τα train data.

Η αναγκαστική ευθυγράμμιση είναι η διαδικασία με την οποία οι orthographic transcriptions ευθυγραμμίζονται με τις ηχογραφήσεις των ομιλητών για την αυτόματη δημιουργία segmentation σε επίπεδο φωνημάτων[6]. Η αναγκαστική ευθυγράμμιση λειτουργεί καλύτερα σε ηχογραφήσεις που:

- έχουν έναν ομιλητή να μιλάει κάθε φορά
- έχουν ελάχιστο περιβαλλοντικό θόρυβο

Ακόμα, Το analyze_alignments.sh αναλύει τις ευθυγραμμίσεις και υπολογίζει στατιστικά στοιχεία.

Το script είναι το εξής:

```
source ./path.sh
source ./cmd.sh

steps/train_mono.sh --boost-silence 1.25 --nj 4 --cmd "$train_cmd" \
  data/train data/lang_test exp/mono0a || exit 1;
steps/align_si.sh --boost-silence 1.25 --nj 4 --cmd "$train_cmd" \
  data/train data/lang_test exp/mono0a exp/mono0a_ali
```

Και τα αποτελέσματα φαίνονται εδώ:

```
./getMonophoneGMMHMMmodel.sh
steps/train_mono.sh --boost-silence 1.25 --nj 4 --cmd run.pl data/train
data/lang_test exp/mono0a
steps/train_mono.sh: Initializing monophone system.
steps/train_mono.sh: Compiling training graphs
steps/train_mono.sh: Aligning data equally (pass 0)
steps/train_mono.sh: Pass 1
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 2
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 3
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 4
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 5
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 6
```

```
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 7
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 8
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 9
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 10
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 11
steps/train_mono.sh: Pass 12
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 13
steps/train_mono.sh: Pass 14
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 15
steps/train_mono.sh: Pass 16
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 17
steps/train_mono.sh: Pass 18
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 19
steps/train_mono.sh: Pass 20
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 21
steps/train_mono.sh: Pass 22
steps/train_mono.sh: Pass 23
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 24
steps/train_mono.sh: Pass 25
steps/train_mono.sh: Pass 26
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 27
steps/train_mono.sh: Pass 28
steps/train_mono.sh: Pass 29
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 30
steps/train_mono.sh: Pass 31
steps/train_mono.sh: Pass 32
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 33
steps/train_mono.sh: Pass 34
steps/train_mono.sh: Pass 35
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 36
steps/train_mono.sh: Pass 37
steps/train_mono.sh: Pass 38
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 39
steps/diagnostic/analyze_alignments.sh --cmd run.pl data/lang_test exp/mono0a
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
35.28072837632777% of the time at utterance begin.  This may not be optimal.
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
28.90743550834598% of the time at utterance end.  This may not be optimal.
steps/diagnostic/analyze_alignments.sh: see stats in
exp/mono0a/log/analyze_alignments.log
180 warnings in exp/mono0a/log/acc.*.*.log
```

```
2682 warnings in exp/mono0a/log/align.*.*.log
120 warnings in exp/mono0a/log/update.*.log
2 warnings in exp/mono0a/log/analyze_alignments.log
exp/mono0a: nj=4 align prob=-83.51 over 1.67h [retry=1.4%, fail=0.2%]
states=125 gauss=996
steps/train_mono.sh: Done training monophone system in exp/mono0a
steps/align_si.sh --boost-silence 1.25 --nj 4 --cmd run.pl data/train
data/lang_test exp/mono0a exp/mono0a_ali
steps/align_si.sh: feature type is delta
steps/align_si.sh: aligning data in data/train using model from exp/mono0a,
putting alignments in exp/mono0a_ali
steps/diagnostic/analyze_alignments.sh --cmd run.pl data/lang_test
exp/mono0a_ali
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
34.446130500758 73% of the time at utterance begin.  This may not be optimal.
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
29.590288315629742% of the time at utterance end.  This may not be optimal.
steps/diagnostic/analyze_alignments.sh: see stats in
exp/mono0a_ali/log/analyze_alignments.log
steps/align_si.sh: done aligning data.
```

4.4.2. Generating HCLG Graph from Grammarwith Unigrams and Bigrams

Ο γράφος HCLG αναπαριστά το ακουστικό και το γλωσσικό μοντέλο που χρησιμοποιούνται για την αποκωδικοποίηση της ομιλίας.

Για να δημιουργηθεί ο γράφος HCLG πρέπει πρώτα να έχουμε τη γραμματική (G) της γλώσσας. Η τελευταία ορίζει το σύνολο των πιθανών λέξεων και των συνδυασμών τους στη γλώσσα.

Για τη δημιουργία του γράφου HCLG, θα χρησιμοποιήσουμε το πρόγραμμα mkgraph.sh. Αυτό λαμβάνει τη γραμματική ως είσοδο και κατασκευάζει τον γράφο HCLG συνδυάζοντας τις πληροφορίες από τα μοντέλα.

Στην περίπτωσή μας, έχουμε να δημιουργήσουμε το γράφο HCLG χρησιμοποιώντας unigrams και bigrams. Τα unigrams αναφέρονται σε μεμονωμένες λέξεις, ενώ τα bigrams σε ζεύγη διαδοχικών λέξεων στη γλώσσα.

Τέλος, εξετάζουμε τα unigrams και τα bigrams ως προς την ακρίβεια του συστήματος αναγνώρισης ομιλίας.

Το script που γράψαμε είναι:

```
source ./path.sh

cp data/lang_test/G_trainUni.fst data/lang_test/G.fst
utils/mkgraph.sh --mono data/lang_test \
   exp/mono0a exp/mono0a/graph_Uni

cp data/lang_test/G_trainBi.fst data/lang_test/G.fst
utils/mkgraph.sh --mono data/lang_test \
   exp/mono0a exp/mono0a/graph_Bi
```

Και όταν το τρέχουμε, μας δίνει:

```
./getHCLGgraph.sh
WARNING: the --mono, --left-biphone and --quinphone options are now
deprecated and ignored.
tree-info exp/mono0a/tree
tree-info exp/mono0a/tree
fstpushspecial
fstdeterminizestar --use-log=true
fstminimizeencoded
fsttablecompose data/lang_test/L_disambig.fst data/lang_test/G.fst
fstisstochastic data/lang_test/tmp/LG.fst
0.000566171 0.000529703
fstcomposecontext --context-size=1 --central-position=0 --read-disambig-
syms=data/lang_test/phones/disambig.int --write-disambig-
syms=data/lang_test/tmp/disambig_ilabels_1_0.int
data/lang_test/tmp/ilabels_1_0.15512 data/lang_test/tmp/LG.fst

fstisstochastic data/lang_test/tmp/CLG_1_0.fst
0.000566171 0.000529703
make-h-transducer --disambig-syms-out=exp/mono0a/graph_Uni/disambig_tid.int -
-transition-scale=1.0 data/lang_test/tmp/ilabels_1_0 exp/mono0a/tree
exp/mono0a/final.mdl
```

```
fsttablecompose exp/mono0a/graph_Uni/Ha.fst data/lang_test/tmp/CLG_1_0.fst
fstdeterminizestar --use-log=true
fstminimizeencoded
fstrmsymbols exp/mono0a/graph_Uni/disambig_tid.int
fstrmepslocal
fstisstochastic exp/mono0a/graph_Uni/HCLGa.fst
0.000976562 -0.000411634
add-self-loops --self-loop-scale=0.1 --reorder=true exp/mono0a/final.mdl
exp/mono0a/graph_Uni/HCLGa.fst
WARNING: the --mono, --left-biphone and --quinphone options are now
deprecated and ignored.
tree-info exp/mono0a/tree
tree-info exp/mono0a/tree
fsttablecompose data/lang_test/L_disambig.fst data/lang_test/G.fst
fstdeterminizestar --use-log=true
fstminimizeencoded
fstpushspecial
fstisstochastic data/lang_test/tmp/LG.fst
-0.00882101 -0.00953992
fstcomposecontext --context-size=1 --central-position=0 --read-disambig-
syms=data/lang_test/phones/disambig.int --write-disambig-
syms=data/lang_test/tmp/disambig_ilabels_1_0.int
data/lang_test/tmp/ilabels_1_0.15565 data/lang_test/tmp/LG.fst

fstisstochastic data/lang_test/tmp/CLG_1_0.fst
-0.00882101 -0.00953989
make-h-transducer --disambig-syms-out=exp/mono0a/graph_Bi/disambig_tid.int --
transition-scale=1.0 data/lang_test/tmp/ilabels_1_0 exp/mono0a/tree
exp/mono0a/final.mdl
fsttablecompose exp/mono0a/graph_Bi/Ha.fst data/lang_test/tmp/CLG_1_0.fst
fstrmsymbols exp/mono0a/graph_Bi/disambig_tid.int
fstdeterminizestar --use-log=true
fstrmepslocal
fstminimizeencoded
fstisstochastic exp/mono0a/graph_Bi/HCLGa.fst
0.00041081 -0.0104954
HCLGa is not stochastic
add-self-loops --self-loop-scale=0.1 --reorder=true exp/mono0a/final.mdl
exp/mono0a/graph_Bi/HCLGa.fst
```

4.4.3. Decoding Validation and Test Data Sentences with the Viterbi Algorithm

Ο αλγόριθμος Viterbi είναι ένας αλγόριθμος δυναμικού προγραμματισμού που χρησιμοποιείται για την αποκωδικοποίηση κρυφών μοντέλων Markov (HMM). Όσον αφορά στα συστήματα αυτόματης αναγνώρισης ομιλίας (ASR), ο αλγόριθμος Viterbi χρησιμοποιείται για την αποκωδικοποίησή της και τον προσδιορισμό της πιο πιθανής ακολουθίας φωνημάτων (maximum a posteriori probability estimate) που αντιστοιχούν στο ηχητικό σήμα[7].

Το shell script που γράψαμε χρησιμοποιεί το πρόγραμμα decode.sh, το οποίο επιτελεί τη διαδικασία της αποκωδικοποίησης.

Η διαδικασία αποκωδικοποίησης εκτελείται δύο φορές, για τους γράφους HCLG που προέκυψαν από τα μοντέλα unigram και bigram. Ο αλγόριθμος Viterbi εφαρμόζεται τόσο στο validation set, όσο και στο test set.

Το script μας έχει ως εξής:

```
source ./path.sh

decode_cmd="run.pl"

steps/decode.sh --nj 3 --cmd "$decode_cmd" exp/mono0a/graph_Uni \
   data/dev exp/mono0a/decode_dev || exit 1;
steps/decode.sh --nj 3 --cmd "$decode_cmd"  exp/mono0a/graph_Uni \
   data/test exp/mono0a/decode_test || exit 1;

steps/decode.sh --nj 3 --cmd "$decode_cmd" exp/mono0a/graph_Bi \
   data/dev exp/mono0a/decode_dev || exit 1;
steps/decode.sh --nj 3 --cmd "$decode_cmd"  exp/mono0a/graph_Bi \
   data/test exp/mono0a/decode_test || exit 1;
```

Και τρέχοντάς το μας δίνει:

```
./getDecodedSentences.sh
steps/decode.sh --nj 3 --cmd run.pl exp/mono0a/graph_Uni data/dev
exp/mono0a/decodeUni_dev
decode.sh: feature type is delta
steps/diagnostic/analyze_lats.sh --cmd run.pl exp/mono0a/graph_Uni
exp/mono0a/decodeUni_dev
steps/diagnostic/analyze_lats.sh: see stats in
exp/mono0a/decodeUni_dev/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(9,49,669) and mean=331.2
steps/diagnostic/analyze_lats.sh: see stats in
exp/mono0a/decodeUni_dev/log/analyze_lattice_depth_stats.log
local/score.sh --cmd run.pl data/dev exp/mono0a/graph_Uni
exp/mono0a/decodeUni_dev
local/score.sh: scoring with word insertion penalty=0.0,0.5,1.0
steps/decode.sh --nj 3 --cmd run.pl exp/mono0a/graph_Uni data/test
exp/mono0a/decodeUni_test
decode.sh: feature type is delta
steps/diagnostic/analyze_lats.sh --cmd run.pl exp/mono0a/graph_Uni
exp/mono0a/decodeUni_test
steps/diagnostic/analyze_lats.sh: see stats in
exp/mono0a/decodeUni_test/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(9,58,853) and mean=340.5
```

```
steps/diagnostic/analyze_lats.sh: see stats in
exp/mono0a/decodeUni_test/log/analyze_lattice_depth_stats.log
local/score.sh --cmd run.pl data/test exp/mono0a/graph_Uni
exp/mono0a/decodeUni_test
local/score.sh: scoring with word insertion penalty=0.0,0.5,1.0
steps/decode.sh --nj 3 --cmd run.pl exp/mono0a/graph_Bi data/dev
exp/mono0a/decodeBi_dev
decode.sh: feature type is delta
steps/diagnostic/analyze_lats.sh --cmd run.pl exp/mono0a/graph_Bi
exp/mono0a/decodeBi_dev
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
38.5135135135135168 of the time at utterance begin.  This may not be optimal.
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
27.027027027027028% of the time at utterance end.  This may not be optimal.
steps/diagnostic/analyze_lats.sh: see stats in
exp/mono0a/decodeBi_dev/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(3,17,122) and mean=59.6
steps/diagnostic/analyze_lats.sh: see stats in
exp/mono0a/decodeBi_dev/log/analyze_lattice_depth_stats.log
local/score.sh --cmd run.pl data/dev exp/mono0a/graph_Bi
exp/mono0a/decodeBi_dev
local/score.sh: scoring with word insertion penalty=0.0,0.5,1.0
steps/decode.sh --nj 3 --cmd run.pl exp/mono0a/graph_Bi data/test
exp/mono0a/decodeBi_test
decode.sh: feature type is delta
steps/diagnostic/analyze_lats.sh --cmd run.pl exp/mono0a/graph_Bi
exp/mono0a/decodeBi_test
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
21.584699453551913% of the time at utterance begin.  This may not be optimal.
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
34.97267759562842% of the time at utterance end.  This may not be optimal.
steps/diagnostic/analyze_lats.sh: see stats in
exp/mono0a/decodeBi_test/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(4,21,143) and mean=78.2
steps/diagnostic/analyze_lats.sh: see stats in
exp/mono0a/decodeBi_test/log/analyze_lattice_depth_stats.log
local/score.sh --cmd run.pl data/test exp/mono0a/graph_Bi
exp/mono0a/decodeBi_test
local/score.sh: scoring with word insertion penalty=0.0,0.5,1.0
```

4.4.4. Evaluating Decoding Results Using Phone Error Rate (PER) Metric

Σε αυτό το βήμα θα αξιολογήσουμε τα αποτελέσματα της αποκωδικοποίησης χρησιμοποιώντας τη μετρική Phone Error Rate (PER). Η τελευταία ποσοτικοποιεί την ακρίβεια του συστήματος αυτόματης αναγνώρισης ομιλίας (ASR) μετρώντας τα σφάλματα (σε επίπεδο φωνήματος) στην αποκωδικοποιημένη έξοδο ως προς το reference transcription.

Το PER ορίζεται ως:

$$PER = 100\frac{insertions + substitutions + deletions}{\#phonemes}$$

Το script score.sh του Kaldi χρησιμοποιείται για τον υπολογισμό του PER και έχει ως εισόδους την αποκωδικοποιημένη έξοδο και το reference transcription.

Από τη μετρική PER, μπορούμε να αξιολογήσουμε την ακρίβεια του συστήματος ASR όσον αφορά τα σφάλματα σε επίπεδο φωνήματος και να συγκρίνουμε την απόδοση διαφορετικών μοντέλων ή διαμορφώσεων. Όσο χαμηλότερη είναι η τιμή PER, τόσο καλύτερη είναι η ακρίβεια του συστήματος στην αναγνώριση των φωνημάτων στην αποκωδικοποιημένη έξοδο.

Γράψαμε το ακόλουθο script:

```
#Now make subset with the shortest 1k utterances from train data.
source ./path.sh

train_cmd="run.pl"

utils/subset_data_dir.sh --shortest data/train 1000 data/train_1kshort ||
exit 1;
utils/data/fix_data_dir.sh data/train_1kshort


#Computes training alignments using a model with  delta+delta-delta features
features. Specifically
#Training the model of the subset of 1k utterances

steps/train_deltas.sh --boost-silence 1.25 --cmd "$train_cmd" 2000 10000 \
        data/train_1kshort data/lang_test exp/mono0a_ali exp/tri1 || exit 1;

steps/align_si.sh --boost-silence 1.25 --nj 4 --cmd "$train_cmd" \
        data/train_1kshort data/lang_test exp/tri1 exp/tri1_ali || exit 1;
```

Και έχουμε:

```
./getAlignment.sh

utils/subset_data_dir.sh: reducing #utt from 1320 to 1000
fix_data_dir.sh: kept all 1000 utterances.
fix_data_dir.sh: old files are kept in data/train_1k/.backup
steps/train_deltas.sh --boost-silence 1.25 --cmd run.pl 2000 10000
data/train_1k data/lang_test exp/mono0a_ali exp/tri
steps/train_deltas.sh: accumulating tree stats
```

```
steps/train_deltas.sh: getting questions for tree-building, via clustering
steps/train_deltas.sh: building the tree
WARNING (gmm-init-model[5.5.1071~1-19185]:InitAmGmm():gmm-init-model.cc:55)
Tree has pdf-id 1 with no stats; corresponding phone list: 6 7 8 9
** The warnings above about 'no stats' generally mean you have phones **
** (or groups of phones) in your phone set that had no corresponding data. **
** You should probably figure out whether something went wrong, **
** or whether your data just doesn't happen to have examples of those **
** phones. **
steps/train_deltas.sh: converting alignments from exp/mono0a_ali to use
current tree
steps/train_deltas.sh: compiling graphs of transcripts
steps/train_deltas.sh: training pass 1
steps/train_deltas.sh: training pass 2
steps/train_deltas.sh: training pass 3
steps/train_deltas.sh: training pass 4
steps/train_deltas.sh: training pass 5
steps/train_deltas.sh: training pass 6
steps/train_deltas.sh: training pass 7
steps/train_deltas.sh: training pass 8
steps/train_deltas.sh: training pass 9
steps/train_deltas.sh: training pass 10
steps/train_deltas.sh: aligning data
steps/train_deltas.sh: training pass 11
steps/train_deltas.sh: training pass 12
steps/train_deltas.sh: training pass 13
steps/train_deltas.sh: training pass 14
steps/train_deltas.sh: training pass 15
steps/train_deltas.sh: training pass 16
steps/train_deltas.sh: training pass 17
steps/train_deltas.sh: training pass 18
steps/train_deltas.sh: training pass 19
steps/train_deltas.sh: training pass 20
steps/train_deltas.sh: aligning data
steps/train_deltas.sh: training pass 21
steps/train_deltas.sh: training pass 22
steps/train_deltas.sh: training pass 23
steps/train_deltas.sh: training pass 24
steps/train_deltas.sh: training pass 25
steps/train_deltas.sh: training pass 26
steps/train_deltas.sh: training pass 27
steps/train_deltas.sh: training pass 28
steps/train_deltas.sh: training pass 29
steps/train_deltas.sh: training pass 30
steps/train_deltas.sh: aligning data
steps/train_deltas.sh: training pass 31
steps/train_deltas.sh: training pass 32
steps/train_deltas.sh: training pass 33
steps/train_deltas.sh: training pass 34
steps/diagnostic/analyze_alignments.sh --cmd run.pl data/lang_test exp/tri
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
47.99599198396793% of the time at utterance begin.  This may not be optimal.
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
43.28657314629258% of the time at utterance end.  This may not be optimal.
steps/diagnostic/analyze_alignments.sh: see stats in
exp/tri/log/analyze_alignments.log
68 warnings in exp/tri/log/acc.*.*.log
```

```
161 warnings in exp/tri/log/update.*.log
1 warnings in exp/tri/log/questions.log
65 warnings in exp/tri/log/init_model.log
2 warnings in exp/tri/log/analyze_alignments.log
36 warnings in exp/tri/log/align.*.*.log
1 warnings in exp/tri/log/build_tree.log
exp/tri: nj=4 align prob=-78.51 over 1.15h [retry=0.8%, fail=0.2%] states=744
gauss=10059 tree-impr=5.44
steps/train_deltas.sh: Done training system with delta+delta-delta features
in exp/tri
steps/align_si.sh --boost-silence 1.25 --nj 4 --cmd run.pl data/train_1k
data/lang_test exp/tri exp/tri_ali
steps/align_si.sh: feature type is delta
steps/align_si.sh: aligning data in data/train_1k using model from exp/tri,
putting alignments in exp/tri_ali
steps/diagnostic/analyze_alignments.sh --cmd run.pl data/lang_test
exp/tri_ali
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
48.99799599198397% of the time at utterance begin.  This may not be optimal.
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only
44.98997995991984% of the time at utterance end.  This may not be optimal.
steps/diagnostic/analyze_alignments.sh: see stats in
exp/tri_ali/log/analyze_alignments.log
steps/align_si.sh: done aligning data.
```

Και τυπώνουμε τις τιμές PER για τα μοντέλα:

```
source ./path.sh

[ -d exp/mono0a/decodeUni_dev ] && grep WER exp/mono0a/decodeUni_dev/wer_* |
utils/best_wer.sh
[ -d exp/mono0a/decodeUni_test ] && grep WER exp/mono0a/decodeUni_test/wer_*
| utils/best_wer.sh
[ -d exp/mono0a/decodeBi_dev ] && grep WER exp/mono0a/decodeBi_dev/wer_* |
utils/best_wer.sh
[ -d exp/mono0a/decodeBi_test ] && grep WER exp/mono0a/decodeBi_test/wer_* |
utils/best_wer.sh
```

Οπότε βλέπουμε τα αποτελέσματα:

```
source ./path.sh

[ -d exp/mono0a/decodeUni_dev ] && grep WER exp/mono0a/decodeUni_dev/wer_* |
utils/best_wer.sh
[ -d exp/mono0a/decodeUni_test ] && grep WER exp/mono0a/decodeUni_test/wer_*
| utils/best_wer.sh
[ -d exp/mono0a/decodeBi_dev ] && grep WER exp/mono0a/decodeBi_dev/wer_* |
utils/best_wer.sh
[ -d exp/mono0a/decodeBi_test ] && grep WER exp/mono0a/decodeBi_test/wer_* |
utils/best_wer.sh

./getPER.sh

%WER 52.09 [ 2491 / 4782, 80 ins, 1349 del, 1062 sub ]
exp/mono0a/decodeUni_dev/wer_7_0.0
%WER 51.68 [ 6402 / 12388, 119 ins, 3634 del, 2649 sub ] [PARTIAL]
exp/mono0a/decodeUni_test/wer_7_0.0
```

```
%WER 46.03 [ 2201 / 4782, 111 ins, 907 del, 1183 sub ]
exp/mono0a/decodeBi_dev/wer_7_0.0
%WER 44.76 [ 5545 / 12388, 175 ins, 2514 del, 2856 sub ] [PARTIAL]
exp/mono0a/decodeBi_test/wer_7_0.0
```

Οι υπερπαράμετροι της διαδικασίας scoring είναι το ελάχιστο και το μέγιστο LM-weight για το lattice resourcing. Αυτές για το καλύτερο μοντέλο είχαν τιμές 7 και 19. Το LM-weight καθορίζει την επιρροή των βαθμολογιών του γλωσσικού μοντέλου στη συνολική βαθμολογία του lattice.

4.4.5. Alignment, Triphone Training, HCLG Graph Generation, and Re-decoding Results

Επαναλαμβάνουμε την κωδικοποίηση για το triphone model ως εξής:

```
#Decoding again dev, test with the new trained model
#FEATURE TYPE IS DELTA

source ./path.sh
source ./cmd.sh

decode_cmd="run.pl"
#train_cmd=""

utils/mkgraph.sh data/lang_test \
 exp/mono0a/tri exp/mono0a/tri/graph_Tri || exit 1;

#utils/mkgraph.sh data/lang_test \
#  exp/mono0a exp/mono0a/graph_Tri || exit 1;

steps/train_deltas.sh --boost-silence 1.25 --cmd "$train_cmd" 2000 10000 \
    data/train_1k data/lang_test exp/tri_ali exp/mono0a/tri || exit 1;

#steps/train_deltas.sh --boost-silence 1.25 --cmd "$train_cmd" 2000 10000 \
#   data/train_1k data/lang_test exp/mono0a/tri_ali exp/mono0a/tri || exit 1;

steps/decode.sh --nj 3 --cmd "$decode_cmd" exp/mono0a/graph_Tri \
# exp/tri/graph \
# steps/decode.sh --nj 3 --cmd "$decode_cmd" exp/mono0a/graph_Tri \
  data/dev exp/mono0a/decoded_dev_1k  || exit 1;

#echo "HERE 2"
steps/decode.sh --nj 3 --cmd "$decode_cmd" exp/mono0a/graph_Tri \
# steps/decode.sh --nj 3 --cmd "$decode_cmd" exp/mono0a/graph_Tri \
  data/test exp/mono0a/decoded_test_1k  || exit 1;


compute-wer --text --mode=present
ark:exp/mono0a/decoded_test_1k/scoring_kaldi/test_filt.txt ark,p:-
compute-wer --text --mode=present
ark:exp/mono0a/decoded_dev_1k/scoring_kaldi/dev_filt.txt ark,p:-


[ -d exp/tri/decoded_dev_1k ] && grep WER exp/tri/decoded_dev_1k/wer_* |
utils/best_wer.sh
[ -d exp/tri/decoded_test_1k ] && grep WER exp/tri/decoded_test_1k/wer_* |
utils/best_wer.sh
```

Και παίρνουμε τα νέα scores:

```
%WER 37.63 [ 2323 / 6174, 264 ins, 752 del, 1307 sub ]
exp/tri/decodeUni_dev/wer_7_0.0
%WER 36.70 [ 2200 / 5995, 181 ins, 811 del, 1208 sub ]
exp/tri/decodeUni_test/wer_8_0.0
%WER 37.63 [ 2323 / 6174, 264 ins, 752 del, 1307 sub ]
exp/tri/decodeBi_dev/wer_7_0.0
%WER 36.70 [ 2200 / 5995, 181 ins, 811 del, 1208 sub ]
exp/tri/decodeBi_test/wer_8_0.0
```

Για τις τιμές PER έχουμε τον ακόλουθο πίνακα:

Model / Dataset	Monophone Unigram	Monophone Bigram	Triphone Unigram	Triphone Bigram
Validation	52.09%	46.03%	37.63%	37.63%
Test	51.68%	44.76%	36.70%	36.70%

Παρατηρούμε βελτίωση των scores (μείωση PER) όταν χρησιμοποιούμε Bigram σε σχέση με το Unigram και τεράστια βελτίωση όταν χρησιμοποιούμε Triphone σε σχέση με το Monophone!

Ερώτημα 4

Για το σκοπό της αναγνώρισης φωνής, χρειαζόμαστε έναν ταξινομητή, ο οποίος θα αναγνωρίζει ποιο φώνημα «ακούγεται» στο συγκεκριμένο ακουστικό frame.

Αυτό μπορεί να επιτευχθεί με χρήση ενός απλού GMM, δηλαδή σε κάθε frame θα αποφαινόμαστε βάσει ποιου από τα GMMs που έχουμε εκπαιδεύσει, έχει μεγαλύτερη πιθανοφάνεια το διάνυσμα χαρακτηριστικών που παρατηρούμε.

Όμως, με τον τρόπο αυτό αγνοούμε πλήρως το ακουστικό περιβάλλον του frame καθώς κάνουμε ταξινόμηση.

Για να αντιμετωπιστεί το πρόβλημα, χρησιμοποιείται το ακουστικό μοντέλο GMM – HMM (Gaussian Mixture Model-Hidden Markov Model). Σε αυτό το μοντέλο, κάθε τμήμα της λέξης στο recording μοντελοποιείται ως μια κατάσταση ενός HMM, ώστε να αποτυπωθεί η χρονική δυναμική της ομιλίας και η κατανομή των ακουστικών χαρακτηριστικών της παράγεται με τη βοήθεια ενός GMM.

GMM (Gaussian Mixture Model):

Γκαουσιανά μείγματα: Ένα GMM είναι ένα πιθανοτικό μοντέλο που αναπαριστά την κατανομή των ακουστικών χαρακτηριστικών ως σταθμισμένο άθροισμα πολλαπλών γκαουσιανών συνιστωσών. Κάθε γκαουσιανή συνιστώσα αντιπροσωπεύει μια συστάδα στο χώρο των χαρακτηριστικών και τα βάρη καθορίζουν τη σχετική σημασία τους. Ο σκοπός του συστατικού GMM του μοντέλου είναι υπεύθυνο για τη μοντελοποίηση της κατανομής των ακουστικών χαρακτηριστικών σε κάθε κατάσταση του HMM. Αποτυπώνει τις στατιστικές ιδιότητες των παρατηρούμενων δεδομένων ομιλίας.

HMM (Hidden Markov Model):

Ο σκοπός της συνιστώσας HMM είναι να μοντελοποιήσει τη χρονική δυναμική της ομιλίας αναπαριστώντας την ακολουθία φωνητικών μονάδων (π.χ. φωνήματα ή λέξεις) ως ακολουθία κρυφών καταστάσεων.

Κρυφά Μαρκοβιανά μοντέλα: Σε αυτό το στατιστικό μαρκοβιανό μοντέλο, το υπό μελέτη σύστημα αντιμετωπίζεται ως μια μαρκοβιανή αλυσίδα της οποίας οι καταστάσεις δεν μπορούν να παρατηρηθούν. Οι καταστάσεις της «κρυφής» μαρκοβιανής αλυσίδας επηρεάζουν με γνωστό τρόπο μια δεύτερη παρατηρήσιμη μαρκοβιανή αλυσίδα. Έτσι, μπορούμε να αντλήσουμε πληροφορίες για την «κρυφή» μαρκοβιανή αλυσίδα, μέσω της παρατηρήσιμης. Επιπλέον, η κατάσταση της «ορατής» αλυσίδας επηρεάζεται αποκλειστικά από την τρέχουσα κατάσταση της «κρυφής» σε μια δεδομένη χρονική στιγμή, ενώ οι καταστάσεις των δύο αλυσίδων σε οποιαδήποτε προηγούμενη στιγμή είναι στατιστικώς ανεξάρτητες από την τρέχουσα κατάσταση της «ορατής» σε μια συγκεκριμένη στιγμή, δεδομένης της κατάστασης της «κρυφής» αλυσίδας την ίδια στιγμή.

Σε αυτό το πλαίσιο, η μαρκοβιανή ιδιότητα αναφέρεται στην υπόθεση ότι η τρέχουσα κατάσταση στο HMM εξαρτάται μόνο από την προηγούμενη κατάσταση και όχι από ολόκληρο το ιστορικό. Η υπόθεση αυτή απλοποιεί τη μοντελοποίηση και επιτρέπει τον αποτελεσματικό υπολογισμό των πιθανοτήτων[9].

Η δομή ενός ακουστικού μοντέλου GMM – HMM μπορεί να συνοψιστεί ως εξής:

Κάθε τμήμα λέξης στο σήμα ομιλίας αναπαρίσταται ως κατάσταση στο HMM. Οι μεταβάσεις μεταξύ αυτών των καταστάσεων αποτυπώνουν τη χρονική δυναμική της ομιλίας.

Κάθε κατάσταση συνδέεται με ένα GMM, το οποίο μοντελοποιεί τα ακουστικά χαρακτηριστικά που παρατηρούνται εντός της συγκεκριμένης κατάστασης. Το GMM αποτυπώνει τις στατιστικές ιδιότητες των χαρακτηριστικών, επιτρέποντας τη διάκριση μεταξύ διαφορετικών φωνητικών μονάδων.

Η εκπαίδευση ενός μονοφωνικού μοντέλου GMM – HMM περιλαμβάνει συνήθως τα ακόλουθα βήματα:

- Συλλογή δεδομένων: Συλλέγεται ένα μεγάλο σώμα δεδομένων ομιλίας με transcriptions.
- Εξαγωγή χαρακτηριστικών: Από τα δεδομένα ομιλίας εξάγονται ακουστικά χαρακτηριστικά, όπως οι συντελεστές Mel-Frequency Cepstral Coefficients (MFCC). Τα χαρακτηριστικά αυτά αποτυπώνουν τα φασματικά χαρακτηριστικά του σήματος ομιλίας.

- Αρχικοποίηση: Αρχικά, οι παράμετροι HMM και GMM αρχικοποιούνται τυχαία. Αυτό περιλαμβάνει τις αρχικές πιθανότητες κατάστασης, τις πιθανότητες μετάβασης και τις παραμέτρους GMM (mean value, variance και mixture weights).
- Αλγόριθμος Baum – Welch (μεγιστοποίηση likelihood): Το μοντέλο εκπαιδεύεται χρησιμοποιώντας τον αλγόριθμο Baum – Welch. Αυτός εκτιμά επαναληπτικά τις παραμέτρους του μοντέλου μεγιστοποιώντας την πιθανότητα των δεδομένων εκπαίδευσης.
- Επανεκτίμηση: Σε κάθε επανάληψη του αλγορίθμου Baum – Welch, οι παράμετροι του μοντέλου επανεκτιμώνται με βάση τις αναμενόμενες πιθανότητες των κρυφών καταστάσεων και τα παρατηρούμενα διανύσματα χαρακτηριστικών.
- Σύγκλιση: Ο αλγόριθμος Baum – Welch συνεχίζει να ενημερώνει τις παραμέτρους του μοντέλου μέχρι να επιτευχθεί σύγκλιση.
- Αξιολόγηση του μοντέλου: Μετά την εκπαίδευση, το μοντέλο αξιολογείται χρησιμοποιώντας ένα validation και test sets για να εκτιμηθεί η απόδοσή του.

Ερώτημα 5

Στην αναγνώριση φωνής, η a posteriori πιθανότητα υπολογίζεται χρησιμοποιώντας τον τύπο του Bayes για να προσδιοριστεί το πιο πιθανό φώνημα δεδομένης μιας ακολουθίας ακουστικών χαρακτηριστικών. Αυτό που κάνει ο κανόνας του Bayes είναι να συσχετίζει την a posteriori πιθανότητα εμφάνισης των φωνημάτων με την a priori πιθανότητα της εμφάνισης κάποιας προηγούμενης ακολουθίας φωνημάτων χρησιμοποιώντας και την πιθανότητα παρατήρησης του εκάστοτε φωνήματος.

Ο τύπος του Bayes για εφαρμογές αναγνώρισης φωνής έχει ως εξής:

$$P(phonem|acoustic\ features) = \frac{P(phonem) \cdot P(acoustic\ features|phonem)}{P(acoustic\ features)}$$

Σημείωση: Σε επίπεδο λέξης θα είχαμε όπου $phonem$, το $word$.

- $P(phonem)$: Η a priori πιθανότητα που θεωρούμε ότι έχει κάθε $phonem$ να εμφανιστεί στο δείγμα (χωρίς να ληφθούν υπόψη τα γειτονικά φωνήματα). Εκτιμάται από τα στατιστικά του dataset.
- $P(acoustic\ features)$:
- $P(acoustic\ features|phonem)$: Η πιθανότητα παρατήρησης των ακουστικών χαρακτηριστικών δεδομένης της εμφάνισης του φωνήματος $phonem$. Αυτή η πιθανότητα μοντελοποιείται από το Gaussian Mixture Model (GMM).
- $P(phonem|acoustic\ features)$: Η a posteriori πιθανότητα του φωνήματος $phonem$ δεδομένων των ακουστικών χαρακτηριστικών που έχουν έρθει.

Συγκεκριμένα, το πιο πιθανό φώνημα δεδομένης μίας ακολουθίας ακουστικών χαρακτηριστικών θα είναι εκείνο με τη μέγιστη a posteriori πιθανότητα που υπολογίσαμε με τον κανόντα του Bayes.

$$\begin{aligned}\widehat{phonem} &= \arg\max_{phonem} P(phonem|acoustic\ features) \\ &= \arg\max_{phonem} \frac{P(phonem) \cdot P(acoustic\ features|phonem)}{P(acoustic\ features)}\end{aligned}$$

Ερώτημα 6

Περιγράφεται η δομή του γράφου HCLG στο kaldi:

1) Το L.fst αντιστοιχίζει τα φωνήματα σε λέξεις. Δέχεται ως είσοδο ακολουθία φωνημάτων και ακολουθώντας τις μεταβάσεις παίρνουμε ως έξοδο την αντίστοιχη λέξη.
2) Το G.fst είναι το γλωσσικό μοντέλο μας.
3) Το C.fst αντιστοιχίζει τα φωνήματα σε context-dependent φωνήματα. Αντιστοιχίζει tri-phones σε mono-phones.
4) Το H.fst αντιστοιχίζει πολλαπλές HMM σε triphones

Συνολικά, αναπαριστά το γλωσσικό μοντέλο, το «λεξικό» με την αντιστοίχηση λέξεων σε φωνήματα, την πληροφορία των συμφραζομένων και το HMM. Δέχεται word-ids και επιστρέφει pdf-ids (αντιστοιχούν σε μοντέλα GMM)

4.5. Bonus: Μοντέλο DNN-HMM με PyTorch

Κατεβάζουμε τις απαραίτητες βιβλιοθήκες.

```
pip install git+https://github.com/vesis84/kaldi-io-for-python
```

4.5.1. Aligning Triphones and Computing CMVN Statistics

Έχοντας υπολογίσει τα CMVN, πρέπει να φτιάξουμε τα alignments του triphone model. Συγκεκριμένα, με το κατώθι script έχουμε τις εξής διαδικασίες:

1. Εξάγουμε τις ευθυγραμμίσεις τριφώνου για τα 3 datasets χρησιμοποιώντας το script `align_si.sh`. Οι ευθυγραμμίσεις τριφώνων, όπως έχει αναφερθεί, παράγονται από την ευθυγράμμιση των ακουστικών χαρακτηριστικών με το μοντέλο HMM.

2. Υπολογίζουμε τα στατιστικά (CMVN) για κάθε dataset χρησιμοποιώντας την εντολή `compute-cmvn-stats`. Η CMVN, όπως εξηγήσαμε προηγουμένως, είναι μια τεχνική κανονικοποίησης στην αναγνώριση ομιλίας για τη μείωση των μεταβολών στα χαρακτηριστικά που προκαλούνται από αλλοιώσεις της ηχογράφησης και τη διαφορετικότητα των ομιλητών.

3. Τέλος, Τα στατιστικά (CMVN) αποθηκεύονται σε αρχεία `.ark`.

Ο κώδικας φαίνεται εδώ:

```
source ./path.sh
source ./cmd.sh

echo $(pwd)
# ------------------- Data preparation for DNN -------------------- #
# Compute cmvn stats for every set and save them in specific .ark files
# These will be used by the python dataset class that you were given

for set in train dev test; do
  compute-cmvn-stats --spk2utt=ark:data/${set}/spk2utt
scp:data/${set}/feats.scp ark:data/${set}/${set}"_cmvn_speaker.ark"
  compute-cmvn-stats scp:data/${set}/feats.scp
ark:data/${set}/${set}"_cmvn_snt.ark"
done

compute-cmvn-stats --spk2utt=ark:data/train_1kshort/spk2utt
scp:data/train_1kshort/feats.scp
ark:data/train_1kshort/train_1kshort"_cmvn_speaker.ark"
compute-cmvn-stats scp:data/train_1kshort/feats.scp
ark:data/train_1kshort/train_1kshort"_cmvn_snt.ark"

steps/align_si.sh --nj 3 --cmd "$train_cmd" \
  data/train_1kshort data/lang_test  exp/tri1 exp/tri1_ali

steps/align_si.sh --nj 3 --cmd "$train_cmd" \
  data/dev data/lang_test exp/tri1 exp/tri1_dev_ali

steps/align_si.sh --nj 3 --cmd "$train_cmd" \
        data/test data/lang_test exp/tri1 exp/tri1_test_ali
```

```
copy-feats scp:/mnt/e/HMMY/github/NLP-
lab2/kaldi/egs/wsj/s5/data/dev/feats.scp ark:- |
apply-cmvn --utt2spk=ark:/mnt/e/HMMY/github/NLP-
lab2/kaldi/egs/usc/data/dev/utt2spk
ark:/mnt/e/HMMY/github/NLP-
lab2/kaldi/egs/wsj/s5/data/dev/dev_cmvn_speaker.ark ark:- ark:- |
add-deltas --delta-order=2 ark:- ark:- | splice-feats --left-context=3 --
right-context=3 ark:- ark:feats

gunzip -c /mnt/e/HMMY/github/NLP-
lab2/kaldi/egs/wsj/s5/exp/tri1_dev_ali/ali.*.gz | ali-to-pdf

/mnt/e/HMMY/github/NLP-lab2/kaldi/egs/wsj/s5/exp/tri1_dev_ali/final.mdl ark:-
ark:labels
```

4.5.2. DNN Training and Decoding

Χρησιμοποιούμε το script που μας δόθηκε. Αλλάζουμε τις μεταβλητές ώστε να ταιριάζουν στο σύστημά μας. Τα βήματα που ακολουθούνται στο script είναι τα εξής:

1. Ορίζουμε τα paths που δείχνουν στα δεδομένα του συστήματός μας.
2. Υπολογίζουμε τα στατιστικά (CMVN) για κάθε dataset (train, validation και test) μέσω του προγράμματος `compute-cmvn-stats`, όπως και στο βήμα (4.5.1).
3. Εκπαιδεύουμε ένα βαθύ νευρωνικό δίκτυο (DNN) χρησιμοποιώντας το script `timit_dnn.py`. Η εκπαίδευση συμβαίνει χρησιμοποιώντας το αρχείο `best_usc_dnn.pt` ως είσοδο.
4. Με το πρόγραμμα `extract_posteriors` υπολογίζουμε τις επιθυμητές πιθανότητες για τις οποίες έχει γίνει λόγος στην αναφορά.
5. Τέλος, γίνεται η αποκωδικοποίηση του DNN μέσω του script `decode_dnn.sh`. Με εισόδους:
   - Το γράφο HCLG
   - Το dataset Και τυπώνουμε τα αποτελέσματα της αποκωδικοποίησης.

Το script που χρησιμοποιήσαμε είναι το παρακάτω:

```
#!/usr/bin/env bash

source ./path.sh
source ./cmd.sh

DATA_PATH=./data/test


# FIXME: CHANGE THESE PATHS TO MATCH YOUR CONFIG
GRAPH_PATH=./exp/mono0a
TEST_ALI_PATH=./exp/tri1_ali_test
OUT_DECODE_PATH=./exp/tri1/decode_test_dnn

CHECKPOINT_FILE=./best_usc_dnn.pt
DNN_OUT_FOLDER=./dnn_out
```

```
# ------------------- Data preparation for DNN -------------------- #
# Compute cmvn stats for every set and save them in specific .ark files
# These will be used by the python dataset class that you were given
for set in train dev test; do
  compute-cmvn-stats --spk2utt=ark:data/${set}/spk2utt
scp:data/${set}/feats.scp ark:data/${set}/${set}"_cmvn_speaker.ark"
  compute-cmvn-stats scp:data/${set}/feats.scp
ark:data/${set}/${set}"_cmvn_snt.ark"
done


# ------------------ TRAIN DNN ----------------------------------- #
python timit_dnn.py $CHECKPOINT_FILE


# ----------------- EXTRACT DNN POSTERIORS ----------------------- #
python extract_posteriors $CHECKPOINT_FILE $DNN_OUT_FOLDER


# ----------------- RUN DNN DECODING ------------------------------ #
./decode_dnn.sh $GRAPH_PATH $DATA_PATH $TEST_ALI_PATH $OUT_DECODE_PATH "cat
$DNN_OUT_FOLDER/posteriors.ark"
```

4.5.3. Defining `TorchSpeechDataset`

Εξηγούμε συνοπτικά τις κύριες ονομασίες.

1. `recipe_dir`: Το directory που έχει τα data
2. `ali_dir`: Το directory που έχει τα alignments
3. `feature_context`: Το πλήθος των tokens στο context
4. `dset`: Το dataset που θα χρησιμοποιείται κάθε φορά (train, dev, test)

- feats: Εδώ οποίο αποθηκεύονται οι συντελεστές MFCC.
- labels: Εδώ αποθηκεύονται τα φωνήματα τα οποία αντιστοιχούν στα targets.
- uttids: Εδώ αποθηκεύονται τα uttids.
- end_indexes: Ο πίνακας περιέχει πληροφορία για τη θέση στην οποία τελειώνουν τα χαρακτηριστικά μιας πρότασης και αρχίζει η επόμενη.

Ακόμα, παρουσιάζουμε με λίγα λόγια και τις μεθόδους της κλάσης:

1. `read_data`: Κάνει ανάγνωση και επεξεργασία των δειγμάτων ομιλίας. Διαβάζει τα MFCC από το αρχείο `feats.scp` και εκτελεί τις προηγούμενες διαδικασίες (υπολογισμός CMVN, delta features). Ακόμα, οι ετικέτες ευθυγράμμισης και τα χαρακτηριστικά αποθηκεύονται σε λεξικά (feats και lab) όπου τα κλειδιά είναι τα uttids.
2. `unify_data`: Ταξινομεί τα χαρακτηριστικά και τις ετικέτες βάσει των uttids και υπολογίζει τον πίνακα `end_idexes` που αναφέραμε πιο πάνω.
3. `__getitem__`: Επιστρέφει την tuple των χαρακτηριστικών και των ετικετών για τον εκάστοτε δείκτη.
4. `__len__`: Επιστρέφει το συνολικό αριθμό των δειγμάτων του dataset.

```
import os
import numpy as np
import kaldi_io
from torch.utils.data import Dataset

class TorchSpeechDataset(Dataset):
    def __init__(self, recipe_dir, ali_dir, feature_context=2,
dset='train_1kshort'):
        self.recipe_dir = recipe_dir
        self.ali_dir = ali_dir
        self.feature_context = feature_context
        self.dset = 'dev' if dset == 'valid' else dset

        self.feats, self.labels = self.read_data()
        self.feats, self.labels, self.uttids, self.end_indexes =
self.unify_data(self.feats, self.labels)

    def read_data(self):
        feat_path = os.path.join(self.recipe_dir, 'data', self.dset,
'feats.scp')
        if self.dset == 'train':
            label_path = os.path.join(self.recipe_dir, 'exp', self.ali_dir)
        else:
            label_path = os.path.join(self.recipe_dir, 'exp', self.ali_dir +
'_' + self.dset)
        feat_opts = "apply-cmvn --utt2spk=ark:{0} ark:{1} ark:- ark:- |". \
            format(os.path.join(self.recipe_dir, 'data', self.dset,
'utt2spk'),
                   os.path.join(self.recipe_dir, 'data', self.dset,
                                self.dset + '_cmvn_speaker.ark'))
        feat_opts += " add-deltas --delta-order=2 ark:- ark:- |"
        if self.feature_context:
            feat_opts += " splice-feats --left-context={0} --right-
context={0} ark:- ark:- |". \
                format(str(self.feature_context))
        label_opts = 'ali-to-pdf'

        feats = {k: m for k, m in kaldi_io.read_mat_ark(
            'ark:copy-feats scp:{} ark:- | {}'.format(feat_path, feat_opts))}
        lab = {k: v for k, v in kaldi_io.read_vec_int_ark(
            'gunzip -c {0}/ali*.gz | {1} {0}/final.mdl ark:- ark:-
|'.format(label_path, label_opts))
               if k in feats}
        feats = {k: v for k, v in feats.items() if k in lab}

        return feats, lab

    def unify_data(self, feats, lab, optional_array=None):
        fea_conc = np.concatenate([v for k, v in sorted(feats.items())])
        lab_conc = np.concatenate([v for k, v in sorted(lab.items())])
        if optional_array:
            opt_conc = np.concatenate([v for k, v in
sorted(optional_array.items())])
        names = [k for k, v in sorted(lab.items())]
        end_snt = 0
        end_indexes = []
        for k, v in sorted(lab.items()):
```

```
            end_snt += v.shape[0]
            end_indexes.append(end_snt)

        lab = lab_conc.astype('int64')
        if optional_array:
            opt = opt_conc.astype('int64')
            return fea_conc, lab, opt, names, end_indexes
        return fea_conc, lab, names, end_indexes

    def __getitem__(self, idx):
        return self.feats[idx], self.labels[idx]

    def __len__(self):
        return len(self.labels)


import sys
sys.path.append("/mnt/e/HMMY/github/NLP-lab2/kaldi/egs/usc")

trainset = TorchSpeechDataset(recipe_dir='.', ali_dir='tri_ali',
feature_context=3, dset='train_1kshort')


success
usctimit_ema_f1_001

testset = TorchSpeechDataset(recipe_dir='.', ali_dir='tri_test_ali',
feature_context=3, dset='test')
validationset = TorchSpeechDataset(recipe_dir='.', ali_dir='tri_dev_ali',
feature_context=3, dset='dev')


usctimit_ema_f1_004
usctimit_ema_f1_021
```

4.5.4. Deep Neural Network for Phoneme Classification

Ορίζουμε τις απαραίτητες κλάσεις για την εκπαίδευση του DNN.

Εξηγούμε εν συντομία τις παραμέτρους του μεθεπόμενου script:

1. `epochs`: Αριθμός εποχών
2. `batch_size`: Batch size για την εκπαίδευση
3. `transform`: Ορίζει το transformation στα data
4. Διακρίνουμε το train set και το test set
5. DNN: Δημιουργεί το δίκτυο
6. for-loop: Βρόχος εκαίδευσης

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
```

```
import torch.utils.data as Data

class ToTensor(object):
    def __init__(self):
          pass
    def __call__(self, datum):
        x, y = datum[0], datum[1]
        t = torch.from_numpy(x).type(torch.FloatTensor)
        return t, y

class GaussianNoise(object):
    def __init__(self, std=0.1, mean=0.0):
        self.mean = mean
        self.std = std

    def __call__(self, datum):
        x = datum[0]
        y = datum[1]
        return x + (self.std * np.random.randn(len(x)) + self.mean), y


class LinearWActivation(nn.Module):
    def __init__(self, in_features, out_features, activation='sigmoid'):
        super(LinearWActivation, self).__init__()
        self.f = nn.Linear(in_features,out_features)
        if activation == 'sigmoid':
            self.a = nn.Sigmoid()
        else:
            self.a = nn.ReLU()

    def forward(self, x):
        return self.a(self.f(x))

class DNN(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        """
        In the constructor we instantiate two nn.Linear modules and assign
them as
        member variables.
        """
        super(DNN, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        """
        In the forward function we accept a Tensor of input data and we must
return
        a Tensor of output data. We can use Modules defined in the
constructor as
        well as arbitrary operators on Tensors.
        """
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred

class Data(Dataset):
```

```
    def __init__(self, X, y, trans=None):
        import numpy as np
        self.data = list(zip(X, y))
        self.trans = trans

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        if self.trans is not None:
            return self.trans(self.data[idx])
        else:
            return self.data[idx]
# Define constants
epochs = 40
batch_size = 30
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define data transformations
transform = transforms.Compose([GaussianNoise(), ToTensor()])

# Get train and test data
X_train, y_train = trainset.feats, trainset.labels
X_test, y_test = testset.feats, testset.labels

# Create train and test datasets
trainset = Data(X_train, y_train, trans=transform)
testset = Data(X_test, y_test, trans=transform)

# Create train and test data loaders
train_dl = DataLoader(dataset=trainset, batch_size=batch_size, shuffle=False)
test_dl = DataLoader(dataset=testset, batch_size=BATCH_SZ, shuffle=False)

# Define model parameters
batchSize, dimInput, hiddenUnitNum, dimOutput = 1, X_train.shape[1], 10,
len(y_train)

# Create the neural network model
net = DNN(dimInput, hiddenUnitNum, dimOutput)
net.to(device)

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(list(net.parameters()), lr=0.1, momentum=0.9)

# Set the model to training mode
net.train()

# Training loop
for epoch in range(epochs):
    running_average_loss = 0

    # Iterate over batches in the train data loader
    for i, data in enumerate(train_dl):
        X_batch, y_batch = data
        X_batch = X_batch.requires_grad_()
```

```
        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        out = net(X_batch)

        # Compute the loss
        loss = criterion(out, y_batch)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

        # Update the running average loss
        running_average_loss += loss.detach().item()

        if i % 500 == 0:
            print("Epoch: {}\tBatch: {}\tLoss: {}".format(epoch, i,
float(running_average_loss) / (i + 1)))
```

Εξηγούμε τι συμβαίνει στο επόμενο python script:

1. Αξιολογούμε το νευρωνικό δίκτυο χρησιμοποιώντας τη μέθοδο net.eval().
2. Το accuracy του μοντέλου υπολογίζεται σε batches που δημιουργεί ο dataloader.
3. Το accuracy προκύπτει διαιρώντας το συνολικό αριθμό σωστών προβλέψεων με το συνολικό αριθμό samples στο validation set.

```
# Set the model to evaluation mode
net.eval()

# Initialize variables
total_correct = 0
total_samples = 0

# Disable gradient computation
with torch.no_grad():
    for i, data in enumerate(test_dl):
        # Extract batch inputs and labels
        X_batch, y_batch = data

        # Forward pass through the network
        outputs = net(X_batch)

        # Get the predicted labels
        _, predicted_labels = outputs.max(1)

        # Calculate the number of correct predictions in the batch
        total_correct += (y_batch == predicted_labels).sum().item()

        # Update the total number of samples
        total_samples += y_batch.size(0)

        print(total_correct)

# Calculate the accuracy
accuracy = total_correct / total_samples
print("Accuracy: {:.2%}".format(accuracy))
```

4.5.5. Generating A Posteriori Probabilities

Υπολογίζουμε τα posteriors:

1. Τα posteriors το validation set υπολογίζονται με χρήση της μεθόδου `calculate_posteriors()` του DNN.

2. Στο αρχείο `post_file` αποθηκεύονται τα posteriors σε μορφή συμβατή με το Kaldi.

```
# Calculate and save the posteriors
posteriors = net.calculate_posteriors(testset.feats)
post_file = kaldi_io.open_or_fd('/mnt/e/HMMY/github/NLP-lab2/kaldi/egs/usc',
'wb', newline='')

# Iterate over utterances and write posteriors to file
start_index = 0
testset.end_indexes[-1] += 1
for i, uttid in enumerate(testset.uttids):
    # Extract posteriors for the current utterance
    utterance_posteriors = posteriors[start_index:testset.end_indexes[i]]
    start_index = testset.end_indexes[i]

    # Write posteriors to file using Kaldi's write_mat function
    kaldi_io.write_mat(post_file, utterance_posteriors, uttid)
```

4.5.6. Decoding Test Set

Με το δοθέν αρχείο `decode_dnn.sh`, γίνεται η αποκωδικοποίηση χρησιμοποιώντας το DNN. Τα ορίσματά του είναι:

- Graph path
- Data path
- Alignments path
- Decoding path

Παράγει τα HMM states τα οποία χρησιμοποιούνται για τον υπολογισμό των πιθανοτήτων. Ύστερα, κάνει την αποκωδικοποίηση εφαρμόζοντας τον lattice generation algorithm.

```
#!/bin/bash

# Copyright 2013    Yajie Miao    Carnegie Mellon University
# Apache 2.0

# Decode the DNN model. The [srcdir] in this script should be the same as dir
in
# build_nnet_pfile.sh. Also, the DNN model has been trained and put in
srcdir.
# All these steps will be done automatically if you run the recipe file run-
dnn.sh

echo $1
echo $2
echo $3
echo $4
echo $5
echo $6
echo $7


## Begin configuration section
stage=0
nj=1
cmd=utils/run.pl
num_threads=1

min_active=200
max_active=7000 # limit of active tokens
max_mem=50000000 # approx. limit to memory consumption during minimization in
bytes

beam=13.0 # beam used
latbeam=8.0 # beam used in getting lattices
acwt=0.2 # acoustic weight used in getting lattices
max_arcs=-1

skip_scoring=false # whether to skip WER scoring
scoring_opts="--min-lmwt 1 --max-lmwt 10"
```

```
norm_vars=false # when doing cmvn, whether to normalize variance; has to be
consistent with build_nnet_pfile.sh

## End configuration section

echo "$0 $@"  # Print the command line for logging

. parse_options.sh || exit 1;

if [ $# != 5 ]; then
   echo "Wrong #arguments ($#, expected 5)"
   echo "Usage: steps/decode_dnn.sh [options] <graph-dir> <data-dir> <ali-
dir> <decode-dir>"
   echo " e.g.: steps/decode_dnn.sh exp/tri4/graph data/test exp/tri4_ali
exp/tri4_dnn/decode"
   echo "main options (for others, see top of script file)"
   echo "  --stage                                  # starts from which
stage"
   echo "  --nj <nj>                                # number of parallel
jobs"
   echo "  --cmd <cmd>                              # command to run in
parallel with"
   echo "  --acwt <acoustic-weight>                 # default 0.1 ... used to
get posteriors"
   echo "  --num-threads <n>                        # number of threads to
use, default 4."
   echo "  --parallel-opts <opts>                   # e.g. '-pe smp 4' if you
supply --num-threads 4"
   echo "  --scoring-opts <opts>                    # options to
local/score.sh"
   exit 1;
fi

graphdir=$1
data=$2
alidir=$3
dir=`echo $4 | sed 's:/$::g'` # remove any trailing slash.
featstring=$5 # `cat $5`
srcdir=`dirname $dir`; # assume model directory one level up from decoding
directory.
sdata=$data/split$nj;

thread_string=
[ $num_threads -gt 1 ] && thread_string="-parallel --num-
threads=$num_threads"


mkdir -p $dir/log
split_data.sh $data $nj || exit 1;
echo $nj > $dir/num_jobs

# Some checks.  Note: we don't need $srcdir/tree but we expect
# it should exist, given the current structure of the scripts.
for f in $graphdir/HCLG.fst $data/feats.scp $alidir/tree; do
  [ ! -f $f ] && echo "$0: no such file $f" && exit 1;
done
```

```
# Generate state counts; will be used as prior
$cmd $dir/log/class_count.log \
  ali-to-pdf $alidir/final.mdl "ark:gunzip -c $alidir/ali.*.gz |" ark:- \| \
    analyze-counts --binary=false ark:- $dir/class.counts || exit 1;

finalfeats="ark,s,cs:$featstring |"
$cmd JOB=1:$nj $dir/log/decode.JOB.log \
  latgen-faster-mapped --min-active=$min_active --max-active=$max_active --
max-mem=$max_mem --beam=$beam --lattice-beam=$latbeam --acoustic-scale=$acwt
--allow-partial=true --word-symbol-table=$graphdir/words.txt
$alidir/final.mdl $graphdir/HCLG.fst "$finalfeats" "ark:|gzip -c >
$dir/lat.JOB.gz"

# Copy the source model in order for scoring
cp $alidir/final.mdl $srcdir

if ! $skip_scoring ; then
  [ ! -x local/score.sh ] && \
    echo "$0: not scoring because local/score.sh does not exist or not
executable." && exit 1;
  local/score.sh $scoring_opts --cmd "$cmd" $data $graphdir $dir
fi

exit 0;
```

Σημείωση: Δυστυχώς υπήρξαν προβλήματα με τις αντιστοιχίσεις αρχείων στο kaldi τα οποία δεν καταφέραμε να επιλύσουμε εντός χρόνου. Προσπαθήσαμε όμως να επεξεργαστούμε και να εξηγήσουμε τα ερωτήματα κατά το δυνατόν καλύτερα. Οπότε η αναφορά μας για το βήμα 4.5 είναι ελλιπής :(

Ερώτημα 7

Το DNN δέχεται ως είσοδο το διάνυσμα χαρακτηριστικών της context-dependent κατάστασης του κρυφού μαρκοβιανού μοντέλου και υπολογίζει την a posteriori πιθανότητα εμφάνισης αυτής της κατάστασης. Το επίπεδο εξόδου είναι ένα επίπεδο softmax με διάσταση ίση με το πλήθος των πιθανών context-dependent καταστάσεων.

Η είσοδος ενός DNN μπορεί να είναι πολύ μεγαλύτερων διαστάσεων από την είσοδο ενός GMM. Πιθανώς λοιπόν, το νευρωνικό δίκτυο μπορεί να λάβει και να επεξεργαστεί περισσότερη πληροφορία και για το λόγο αυτό να δίνει πιο ακριβή αποτελέσματα ως ταξινομητής.

Η εκπαίδευση ενός νευρωνικού δικτύου με την προσέγγιση της επιβλεπόμενης μάθησης δεν δείχνει να παράγει καλά αποτελέσματα. Συνεπώς, δεν φαίνεται να είναι καλή πρακτική να εκπαιδεύσουμε εξ αρχής ένα μοντέλο DNN-HMM[11][12].

Ερώτημα 8

Batch Normalization είναι μια τεχνική που χρησιμοποιείται στο Deep Learning για την ομαλοποίηση των inputs των λαυερς ενός νευρωνικού δικτύου κατά τη διάρκεια του training.

Η διαδικασία είναι να υπολογίζει ξανά το center των data και να τα κλιμακώνει εκ νέου, για κάθε batch. Με αυτόν τον τρόπο, πετυχαίνοντας δηλαδή μέση τιμή 0 και διασπορά 1 σε κάθε layer του νευρωνικού δικτύου, έχουμε καλύτερη ευστάθεια εσωτερικά, άρα βελτίωση της συνολικής απόδοσης.

Το πρόβλημα που αντιμετωπίζεται ονομάζεται Internal Covariate Shift και αναφέρεται στο φαινόμενο όπου η κατανομή των inputs σε κάποιο layer αλλάζει κατά τη διάρκεια του training λόγω της τυχαιότητας στην αρχικοποίηση των παραμέτρων και των στοιχείων του dataset. Γενικά, το Internal Covariate Shift επιβραδύνει τη διαδικασία εκπαίδευσης και εμποδίζει τη σύγκλιση του δικτύου.

Συμβολίζοντας με B τη mini – batch μεγέθους m του training set, έχουμε για τη μέση τιμή και τη διασπορά της batch:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \, , \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

Εν κατακλείδι, το Batch Normalization οδηγεί συνήθως σε ταχύτερη σύγκλιση, καλύτερη ροή των gradients (βελτίωση στο propagation των gradients κατά την ανανέωση των βαρών) και μεγαλύτερη ανθεκτικότητα σε τυχαία inputs[13].

References

[1] https://en.wikipedia.org/wiki/Mel-frequency_cepstrum

[2] https://en.wikipedia.org/wiki/Mel_scale

[3] https://medium.com/@tanveer9812/mfccs-made-easy-7ef383006040

[4] https://en.wikipedia.org/wiki/Acoustic_model

[5] https://web.stanford.edu/~jurafsky/slp3/

[6] https://linguistics.berkeley.edu/plab/guestwiki/index.php?title=Forced_alignment

[7] https://en.wikipedia.org/wiki/Viterbi_algorithm

[8] https://en.wikipedia.org/wiki/Cepstral_mean_and_variance_normalization

[9] https://en.wikipedia.org/wiki/Hidden_Markov_model

[10] https://jonathan-hui.medium.com/speech-recognition-gmm-hmm-8bb5eff8b196

[11] https://www.journalijdr.com/sites/default/files/issue-pdf/9474.pdf

[12] https://www.researchgate.net/publication/306064220

[13] https://en.wikipedia.org/wiki/Batch_normalization