Εθνικο Μετσοβιο Πολυτεχνειο

Επεξεργασία Φωνής & Φυσικής Γλώσσας

2ο Εργαστηρίο

Γεωργίου Δημήτριος (03115106)

<el15106@central.ntua.gr>

Μπαζώτης Νικόλαος (03115739)

<el15739@central.ntua.gr>

12 Δεκεμβρίου 2018

1 Περιγραφή

2 Θεωρητικό Υπόβαθρο

- **MFCCs** Η διαδικασία της εκπαίδευσης ενός συστήματος αναγνώρισης φωνής ,περνάει μέσα από την εξαγωγή δεδομένων χαρακτηριστικών. Η πιο διαδεδομένη μορφή απεικόνισης αυτών είναι ένα MFCC Vector . Η ανάπτυξη της ιδέας αυτής χρησιμοποιεί και ιδέες από cepstrum , καθώς και ψυχοακουστικές μελέτες για την μετάφραση του ήχου στην κατάλληλη μορφή . Η εξαγωγή των χαρακτηριστικών τμηματοποιείται σε 6 στάδια:

  1. **Προέμφαση** του ήχου μέσα από την μετατόπιση κάθε φωνήματος σε υψηλότερες συχνότητες που βοηθούν την αναγνώριση.
  2. **Παραθυροποίηση** των ηχητικών δεδομένων για να διατηρήσουμε την στατικότητα των στατιστικών χαρακτηριστικών του ήχου(μέση τιμή, διακύμανση) και κατ' επέκταση την εργοδικότητα (αφού έχουμε πλήθος δεδομένων στο χρόνο αλλά δε μπορούμε να επαναλάβουμε διαδοχικά τις ίδιες εκτελέσεις των ίδιων δεδομένων).Συχνά χρησιμοποιούμε Kaiser παράθυρα με δεδομένο overlap για να ομαλοποιηθούν φαινόμενα των άκρων.
  3. **Συχνοτική ανάλυση** σήματος ,μέσω της χρήσης DFT για τον υπολογισμό της συγκέντρωσης της ενέργειας σε κάθε συχνότητα.
  4. **Mel filter bank διαμόρφωση**,που αιτιολογείται επειδή ο ανθρώπινος εγκέφαλος παρουσιάζει δυσκολία στον διαχωρισμό συχνοτήτων σε υψηλοσυχνοτικά σήματα.Γενικά η ανθρώπινη απόκριση είναι γραμμική κάτω από τα 1000Hz και λογαριθμική παραπάνω,οπότε έτσι σχεδιάονται και τα φίλτρα μας.
  5. **Χρήση cepstrum**, για την μετάθεση του σήματος σε πεδίο quefrency και το διαχωρισμό πηγής και φίλτρου. Αποδεικνύεται ότι ο διαχωρισμός αυτός βελτιώνει το μοντέλο καθώς ο εντοπισμός π.χ. του φίλτρου: Θέση της γλώσσας μπορεί να προσφέρει στην ανάλυση παραπάνω χαρακτηριστικά για την αναγνώριση των φωνημάτων.
  6. **Delta-Training**: Τα χαρακτηριστικά αυτά είναι τιμές που δείχνουν πόσο μεταβάλλονται κάποια στατιστικά χαρακτηριστικά του cepstrum μέσα σε ένα παράθυρο. Συμπληρώνουν το διάνυσμα που δημιουργούμε.Για ακρίβεια χρησιμοποιούμε και τα double-deltas που δείχνουν την μεταβολή των deltas . Αυτός ο χειρισμός του σήματος καταλήγει σε MFC Coeficients και περιέχει το μεγαλύτερο μέρος της πληροφορίας για κάθε παραθυροποιημένο σήμα.

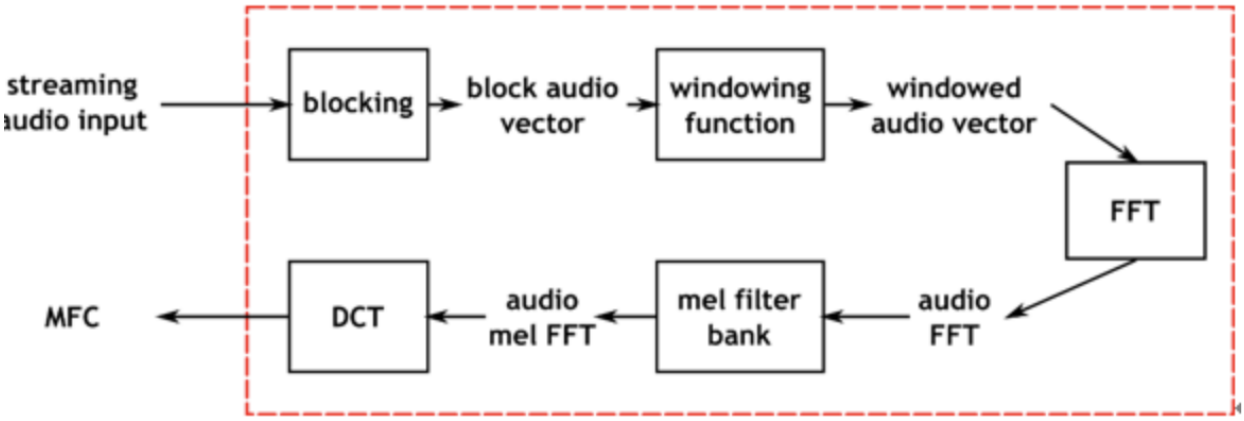


Figure 2 MFCC Computation [9]

- **Γλωσσικό Μοντέλο (Language Model LM)** Για το μοντέλο αυτό συνήθως χρησιμοποιούμε unigrams και bigrams. Ωστόσο, κατά την χρησιμοποίηση του Kaldi για την δημιουργία του μοντέλου αναγνώρισης φωνής βλέπουμε ότι δημιουργεί unigram, bigram συνδυάζοντάς τα και προφανώς αυξάνοντας την υπολογιστική πολυπλοκότητα αλλά και την ακρίβεια όσο αυξάνεται το μέγεθος του μοντέλου.

- **Ακουστικό Μοντέλο (Acoustic Model AM)** Στο στάδιο της δημιουργίας του ακουστικού μοντέλου ουσιαστικά υπολογίζουμε την πιο πιθανή ακολουθία από παρατηρήσεις όταν μας έχουν δοθεί άλλα γλωσσικά χαρακτηριστικά όπως (λέξεις, φωνήματα ή κάποια μέρη φωνημάτων). Πιο συγκεκριμένα δοθέντος μίας πιθανότητας $P(O\|W)$ μπορούμε να κατατάξουμε για κάθε HMM μίας κατάστασης q του αυτομάτου μας, χρησιμοποιώντας Gaussian Mixture Models (GMM's) και τα υπόλοιπα γλωσσικά χαρακτηριστικά, την πιθανοφάνεια.

3 Βήματα Προπαρασσκευής

ΒΗΜΑ 1

1. Κάνουμε import τις κατάλληλες βιβλιοθήκες *os, shutil, re,string,defaultdict*, συναρτήσεις των οποίων θα χρειαστούν για τα παρακάτω βήματα.

2. Χρησιμοποιώντας κατάλληλα την συνάρτηση *copy()*, γίνεται αντιγραφή των αρχείων `"i_utterances.txt"` εσωτερικά του φακέλου **"/kaldi/egs/usc/data/i** ∀ directory i = {train,test,dev} με όνομα **uttids**, αρχεία τα οποία περιέχουν συμβολικά ονομάτα για κάθε πρόταση του συνόλου δεδομένων train data, test data και validation data αντίστοιχα.

3. Σε ∀ directory i = {train,test,dev} δημιουργούμε επιπλέον το αρχείο **"utt2spk"**. Συγκεκριμένα, γίνεται *open()* με 'r' mode του αρχείου **"uttids"** που μόλις δημιουργήσαμε και του **"utt2spk"** με 'w' mode , προσπελαύνουμε το πρώτο γραμμή-γραμμή ενώ ταυτόχρονα γράφουμε στο δεύτερο το συμβολικό όνομα κάθε πρότασης και με κενό ακριβώς δίπλα το id του ομιλητή, εξαγώμενο από την το string της πρότασης με χρήση της `split('_')`.

4. Παρόμοια λογική με το 2ο βήμα, με την μόνη διαφορά ότι εδώ για το αρχείο **"wav.scp"** που θέλουμε να δημιουργήσουμε για κάθε ένα από τα 3 directories, ανοίγουμε με 'r' mode το αρχείο **"utt2spk"** το οποίο προσπελαύνουμε γραμμή-γραμμή, ενώ το speaker_id κάθε γραμμής-πρότασης (πάλι χρήση `split('_')`) χρησιμοποείται για την δημιουργία του path που θέλουμε να γράψουμε στο **"wav.scp"** μαζί με το συμβολικό όνομα.

5. Παρόμοια λογική με το 3ο βήμα. Προσπέλαση γραμμή-γραμμή του **"utt2spk"** και χρήση του τελευταίου element κάθε utterance_id, δηλαδή ενός αριθμού index που προσδιορίζει σε ποια γραμμή βρίσκεται η συγκεκριμένη πρόταση στο αρχείο **"transcription.txt"**. Με χρήση του index προσδιορίζουμε το text της αντίστοιχης πρότασης, το οποίο μαζί με utterance_id το γράφουμε στο αρχείο **"text"** για κάθε ένα από τα directory i = {train,test,dev}

ΒΗΜΑ 2

Έχοντας στην διάθεσή μας το αρχείο **"text"** που ουσιαστικά συσχετίζει utterance_id με το αντίστοιχο string της συγκεκριμένης πρότασης και το **"lexicon.txt"** μπορούμε να μετατρέψουμε την πρόταση αυτή στην αντίστοιχα σειρά από φωνήματα.

Πιο συγκεκριμένα: Ορίζουμε την συνάρτηση *tokenize()*, η οποία δέχεται ως όρισμα ένα string και κάνει process αυτού, μετατρέποντας Κεφαλαία σε πεζά, διατηρώντας μόνο αλφαριθμητικούς χαρακτήρες και τέλος, διασπώντας το σε μια λίστα από λέξεις. Επίσης, χρησιμοποιήσαμε την *" ".join(string.split())* για να εξαλείψουμε τα διπλότυπα κενά, και να μείνουμε μόνο με αυτά μεταξύ των λέξεων ώστε η *split()* να επιστρέψει μόνο αυτές σε μορφή λίστας. (και όχι extra κενά ως λέξεις)

Στην συνέχεια καλούμε την συνάρτηση *create˙lexicon()*, που επίσης ορίσαμε, η οποία προσπελαύνει το αρχείο **"lexicon.txt"** δημιουργώντας ένα python dictionary το οποίο έχει ως κλειδιά τις αντίστοιχες lowercased λέξεις και ως values τα φωνήματα αυτών.

Για ∀ directory i = {train,test,dev}, ανοίγουμε με 'r' mode το αρχείο **"text"** το οποίο προσπελαύνουμε γραμμή-γραμμή, ενώ καλούμε την συνάρτηση *tokenize()* με όρισμα το string της πρότασης (κλήση *split()* για διάσπαση της γραμμής σε λέξεις, αφαίρεση του utterance_id και επιπρόσθετο join των εναπομείναντων λέξεων που ορίζουν την πρόταση μας), στο απότελεσμα της οποίας loopάρουμε επαναληπτικά ώστε να βρούμε τα φωνήματα της κάθε λέξης και εν τέλει να τα κάνουμε join back σε πρόταση. Αυτήν την οποία ενισχύουμε με το φώνημα **"sil"** στην αρχή και στο πέρας, την γράφουμε μαζί με το utterance_id στο αρχείο **"text"**.

Παρατήρηση

Για να πραγματοποιηθεί ορθά το replace των γραμμών στο αρχείο **"text"**, πολύ απλά αφού διαβάσουμε τις γραμμές του με χρήση της *readlines()* - opened with 'r' mode - , το κλείνουμε (*close()*), και γράφουμε πάνω σε αυτό εκ νέου - opened with 'w' mode -

4 Βήματα Κυρίως Μέρος

1. Σε κάθε βήμα σημειώνονται προβλήματα και errors που αντιμετωπίστηκαν είτε μέσω αναζήτησης θεωρητικής πτυχής αυτών είτε μέσω τεχνικών απαντήσεων στο google group του kaldi (σε μορφή footnotes)

2. Για κάθε βήμα του εργαστηρίου συμπεριλαμβάνουμε το αντίστοιχο **".sh"** αρχείο το οποίο ομαδοποιεί τις εντολές συστήματος & εντολές kaldi σε ένα αρχείο για μεγαλύτερη ευελιξία και καλύτερο συντακτικό έλεγχο & διαχείριση error. Τα αρχεία βασίστηκαν σε μεγάλο βαθμό στο αρχείο **"run.sh"** ως προς την ακολουθία των βημάτων, βέβαια υπήρξαν αρκετές τροποποίησεις ώστε να γίνει προσαρμογή στα δοθέντα ακουστικά δεδομένα του εργαστηρίου.

3. Τα αρχικά βήματα έχουν εκτελεστεί τοπικά με χρήση κατάλληλων soft links για να έχουμε πρόσβαση στις εντολές και τα αρχεία στον φάκελο kaldi, ενώ κάποια επόμενα απευθείας στον φάκελο **"/kaldi/egs/wsj/s5"** για καλύτερη διαχείριση.

4. Τα αποτελέσματα εκτέλεσης των **".sh"** αρχείων παρουσιάζονται παρακάτω στο appendix section, ως output του jupyter notebook στο οποίο έγινε run των python scripts. (η χρήση του "!" συμβόλου πριν από την εκτέλεση των **".sh"**)

4.1 Prepareration operation of speech recognition for USC-TIMIT

Μεταφέρουμε τον φάκελο **"/kaldi/egs/usc/data"** που περιέχει τους υποφακέλους i, ∀ directory i = {train,test,dev}, στο εσωτερικό των οποίων υπάρχουν δεδομένα & πληροφορίες δηλαδή για τις εκφράσεις μας (utterances) και τους ομιλητές μας (speakers). Τόσο το training dataset όσο και το testing και validation dataset να παράγουν το συνολικό dataset των ακουσμάτων, όπως και αποδεικνύεται από τα utterances id. Θα αποδειχθεί η αποκωδικοποιητική ικανότητα του εργαλείου Kaldi για δημιουργία speaker-independent system.

1. Τροποποιούμε τα αρχεία **"path.sh"** ώστε να περιέχει το path για το kaldi source directory και το **"cmd.sh"**, στο οποίο κάνουμε τα κατάλληλα export στο αρχείο **"run.pl"** (to run tasks locally), εφόσον δεν χρησιμοποιούμε το GridEngine. [1].

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

Πριν από την εκτέλεση οποιασδήποτε εντολής kaldi κάνουμε source του **"path.sh"**, για να τις έχουμε διαθέσιμες.

2. Δημιουργούμε soft links με την εντολή *ln -s source destination* για έχουμε πρόσβαση στους φακέλους **steps** και **utils** της Wall Street Journal (wsj) στα scripts δηλαδή του kaldi απαραίτητα για την δημιουργία του ASR μοντέλου και για την τροποποίηση αρχείων Kaldi με πολλούς τρόπους, αντίστοιχα.

3. Αντίστοιχα για το αρχείο **"score_kaldi.sh"** του φακέλου **steps**, το οποίο θα χρησιμοποιηθεί παρακάτω για τον υπολογισμό του PER (Phone Error Rate).

4. Περνάμε το modified **mfcc.conf** αρχείο που δόθηκε στις διευκρινίσεις στον φάκελο **local**, το οποίο θα χρησιμοποιηθεί για τον υπολογισμό των MFCC features, με συχνότητα δειγματοληψίας $f = 22050Hz$. Γενικά εντολή *steps/make_fcc.sh*, αναζητά για το αρχείο **conf/mfcc.conf** στον **"conf"** για οποιαδήποτε μη default παράμετρο, οι οποίες προωθούνται ως ορίσματα στα αντίστοιχα binary αρχεία.

5. Δημιουργήθηκαν οι απαιτούμενοι φάκελοι. [2]

4.2 Prepareration of Language Model

- Τόσο το **"lexicon.txt"** όσο και το **"transcriptions.txt"** απαιτούνται εμμέσως μέσω των αρχείων που δημιουργήθηκαν στην προπαρασκευή για την κατασκευή του γλωσσικού μοντέλου, καθώς χρειάζονται για την a priori πιθανότητα εμφάνισης λέξεων/φωνημάτων. Συγκεκριμένα, διαθέτουμε το αρχείο **text** για κάθε ένα από τρία set, το οποίο για κάθε utterance id περιέχει την αντίστοιχη ορθή πρόταση σε μορφή φωνημάτων για κατασκευή του γλωσσικού μοντέλου απευθείας με χρήση του kaldi.

[1] Αποτίμηση τιμών στις μεταβλητές κολλητά χωρίς κενό

[2] Να σημειωθεί στο σημείο αυτό ότι οποιαδήποτε εναλλακτική ονοματοδότηση φακέλου, ιδίως όσον αφορά την κατασκευή του γλωσσικού μοντέλου, θα οδηγήσει σε error. Η δομή πρέπει να τηρηθεί αυστηρά.

- Πρόκειται για 460 προτάσεις, οι οποίες για να γίνει καλή χρήση αυτών, έγινε run των scripts των βημάτων της προπαρασκευής και του συγκεκριμένου βήματος, με την έννοια της οριοθέτησης κάθε segment από **sil** και **ṡ'** με σκοπό την τήρηση των απαιτήσεων του SRILM.

1. Όπως και στην προπαρασκευή, έτσι και εδώ, το παρόν υποβήμα καλύφθηλε εξ'ολοκλήρου με καθαρό κώδικα python, όπως και στις περισσότερες περιπτώσεις διαχείρισης αρχείων. Λόγω ευκολίας, η περαιτέρω επεξήγηση παραλείπεται. Απλώς να σημειωθεί ότι για την δημιουργία του αρχείου **"lm_train.text"**, έγινε *read()* και *write()* από και προς το ίδιο αρχείο.

2. - Γίνεται χρήση της εντολής *build-lm.sh* του πακέτου IRSTLM. Το script κανει split την διαδικασία της εκτίμησης σε 1 και 2 jobs αντίστοιχα για τους φακέλους test, train και dev αντίστοιχα, με σκοπό την δημιουργία των unigram και bigram μοντέλων αντίστοιχα [3].
   - Το script παράγει ένα αρχείο LM στην μορφή *.ilm.gz* που **ΔEN** είναι η τελική ARPA μορφή, αλλά μία ενδιάμεση που ονομάζεται iARPA και η οποία αναγνωρίζεται και απαιτείται από την *compile-lm.sh* του επόμενου υποβήματος καθώς και από τον αποκωδικοποιητή Moses SMT που τρέχει με το πακέτο IRSTLM.[4]

3. Γίνεται χρήση της εντολής *build-lm.sh*, η οποία μετατρέπει τα γλωσσικά μοντέλα iARPA μορφής στα τελικό lm-final, σε ARPA μορφή. Παρακάτω, θα γίνει αναφορά σε τρόπους μεταγλώττισης του γλωσσικού μοντέλο σε πιο συμπαγή και αποδοτική binary μορφή.

4. Γίνεται εκτέλεση της εντολής *prepare_lang.sh* με ορίσματα LM arpa αρχεία πλέον και των 6 μοντέλων για παραγωγή των αρχείων **"words.txt"**,**"oov.txt** και **"phones.txt"**, που περιέχουν αντίστοιχα:

   - Μία λίστα από όλες τις λέξεις στο vocabulary, επιπρόσθετα του sil και #0 συμβόλων (χρήση ως μετάβαση ε στο input του G.fst). Κάθε λέξη έχει μοναδικό index
   - Μία λίστα από όλα τα φωνήματα στο vocabulary,
   - Έχει μία μόνο γραμμή με την λέξη (όχι το φώνημα) για τα εκτός vocabulary αντικείμενα. Εδώ χρησιμοποιείται το **"o̊ov'** γιατί αυτό παίρνουμε από το πακέτο IRSTLM στα γλωσσσικά μοντέλα μας,

   Ακόμα δημιουργούνται δύο fst. Το πρώτο είναι το **"L.fst"**, ένας πεπερασμένων καταστάσεων μετατροπέας από το λεξικό, με σύμβολα-φωνημάτα στην είσοδο και σύμβολα-λέξεις στην έξοδο. Το L κάνει mapping monophone ακολουθίες σε λέξεις. Το δεύτερο είναι το **"L_disambig.fst"**, το φωνητικό λεξικό για αποσαφήνιση των ασαφειών-διφορούμενο συμβόλων-φωνημάτων, τα οποία τα χρειαζόμαστε όταν έχουμε μία λέξη που είναι prefix μιας άλλης (πχ. cat και cats στο ίδιο λεξικό). Εάν δεν τα έχεις, τότε τα μοντέλα γίνονται μη ντετερμινιστικά

5. Τα G.fst, μπορούν να δομηθούν αποκλειστικά πάνω στα unigram και bigram μοντέλα μας. Για την διαδικασσία χρησιμοποιείται η τροποποιημένη *timit_format_data.sh* του kaldi. Πιο συγκεκριμένα: Δημιουργούμε το copy αντίγραφο **"lang_test"**, που περιέχει τα αρχεία και τα αυτόματα του προηγούμενου βήματος [5]. Μέσα στον φάκελο αυτό, το script καλεί την εντολή *gunzip -c* και την *arpa2fst*, επαναληπτικά και για 6 γλωσσικά μοντέλα, με σκοπό την δημιουργία των αντίστοιχων γραμμτικών σε μορφή **"G_i1** και **"G_i2** ∀ directory i = {train,test,dev}. Η *arpa2fst* μάλιστα δέχεται όρισμα την λίστα **"words.txt"**, και λογικό αφού θέλουμε πληροφορία για όλες τις δυνατές λέξεις και φωνημάτα που έχουν προκύψει από όλες τις προτάσεις (άρα για αυτό έγινε εξέταση όλων train,test,dev, αφού συνολικά καλύπτουν 100% της πληροφορίας)

ΕΡΩΤΗΜΑ 1

- Η αξιολόγηση γλωσσωικών μοντέλων περιλαμβάνει και τον υπολογισμό perplexity, εντροπίας και out-of-vocabulary rate τόσο του test set όσο και του validation set.

- Μεγάλα LM αρχεία μπορούν να "κλαδευτούν" με έξυπνο τρόπο με την εντολή που αναφέραμε και νωρίτερα, *prune*, η οποία αφαιρεί n-gram μοντέλα τα οποία για την προσφυγή σε back-off αποτελέσματα οδηγούν σε μικρές-μηδαμινές απώλειες. Η εντολή αυτή δέχεται ως όρισμα ένα threshold,τα οποία έχουν νόημα γα τα bigram μοντέλα του test και του validation set και τα οποία βέβαια έχουν αξία για εμπειρικά δεδομένα. (threshold = 0 δεν οδηγεί σε κλάδεμα).

[3] Βέβαια, ο σχηματισμός του unigram μπορεί να επέλθει αποκλειστικά μέσω την δημιουργία του αντίστοιχου bigram, καθώς είναι αναγκαίο βήμα για αυτό

[4] Για τον υπολογισμό του perplexity στο ερώτημα του συγκεκριμένου βήματος θα γίνει χρήση της *prune* για παραγωγή μικρότερων LM

[5] για backup λόγους πιο πολύ

- Η εφαμοργή του pruning οδηγεί σε LM αρχεία μορφής **".plm"** που περιέχει λιγότερα bigrams, για τις προδιαγραφές του παρόντος εργαστηρίου. Το output είναι ένα ARPA LM αρχείο. Με σκοπό να μετρήσουμε την **απώλεια της ακρίβειας** που εισήχθη με το pruning, το perplexity του output LM μπορεί να υπολογιστεί (δεν θα διαφέρει πολύ από το αντίστοιχο LM χωρίς εφαρμογή pruning).

- Τα LM είναι όντως χρήσιμα και έχουν αξία όταν έχουμε αρκετά δεδομένα στην διάθεση μας, και έτσι μπορούμε να απευθύνουμε queries σε αυτά για υπολογισμό peplexity που έχει ουσία. Συγκεκριμένα, οι προδιαγραφές του εργαστήριου προβλέπουν αρκετά δεδομένα, και για αυτόν ακριβώς τον λόγο **ΠΡΕΠΕΙ** να παρατηρηθεί ότι το perplexity των bigram μοντέλων είναι χαμηλότερο από το αντίστοιχο των unigram μοντέλων τόσο για το test set όσο και για το validation set.

- Υπολογίζεται με την εντολή *compile-lm*, αλλά αυτή την φορά με όρισμα –**eval=test**. Παρακάτω αναδεικνύονται τα αποτελέσματα.

```
###################################################
Calculating perplexity for evaluation data unigram model
###################################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:

DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_dev1.lm
outfile: lm_dev1.lm.blm
evalfile: ../dict/lm_dev.text
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=6332 PP=32.52 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
###################################################
Calculating perplexity for evaluation data bigram model
###################################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:
 1e-06
DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_dev2.lm
outfile: lm_dev2.lm.blm
evalfile: ../dict/lm_dev.text
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=6332 PP=15.26 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
###################################################
```

```
###############################################
Calculating perplexity for test data unigram model
###############################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:

DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_test1.lm
outfile: lm_test1.lm.blm
evalfile: ../dict/lm_test.text
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=6148 PP=32.00 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
###############################################
Calculating perplexity for test data bigram model
###############################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:
 1e-06
DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_test2.lm
outfile: lm_test2.lm.blm
evalfile: ../dict/lm_test.text
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=6148 PP=14.58 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
```

Επομένως παρατηρείτει πτώση από 32.52 σε 15.26 και από 32.00 σε 14.58 για το validation και το test set αντίστοιχα, πράγμα απόλυτα δικαιολογημένο, αφού όσο χαμηλότερο είναι τόσο πιο αποτελεσματική κατανομή πιθανοτήτων έχουμε για πρόβλεψη δειγμάτων.

4.3 Extraction of acoustic features

- Η διαδικασσία εξαγωγής features αναφέρεται και ως παραμετροποίηση ομιλίας (speech parameterization) [6], δηλαδή η ανάλυση των σημάτων ήχου σε μικρή κλίματα για στατιστικούς σκοπούς και για κατηγοριοποίηση των φασματικών features των σημάτων με σκοπό την προετοιμασία τους για την μετέπεια αποκωδικοποίηση (αναφορά παρακάτω). Ουσιαστικά η διαδικασία αυτή "γεφυρώνει" πληροφορίες ήχου και στατιστικών μοντέλων ASR.

- Οι δύο πιο γνωστές μορφές είναι MFCC(Mel Frequency Cepstral Coefficients) και PLP (Perceptual Linear Prediction) που λαμβάνουν υπόψιν την φύση της φωνής ενώ με LPC, γίνεται πρόβλεψη μελλοντικών features βάσει προηγούμενων. Εφόσον η ανθρώπινη φωνή και το ακουστικό σύστημα του ανθρώπου είναι μη γραμμικά, το LPC **ΔΕΝ** είναι καλή επιλογή για εκτίηση ακουστικών features, ενώ τα άλλα βασίζονται στην λογική λογαριθμικά διαχωρισμένων φίλτρων και για αυτό θεωρούνται αποτελεσματικότερα.

- Τώρα θα παράγουμε τα features και τα αντίστοιχα **"feats.scp"** που κάνει mapping τα utterance ids στις θέσεις ενός αρχείου, εδώ **"feats.ark."**. Για GMM-HMM συστήματα, τυπικά χρησιμοποιούμε MFCC ή PLP features και μετά εφαρμόζουμε cepstral mean και variance normalisation

- Εδώ θα παράξουμε MFCCs -για τα οποία δίνουμε μια σχετική θεωρητική ανάλυση και αρχή της παρούσας αναφοράς- για τα δεδομένα μας. Συγκεκριμένα, για ναμ αντισταθμίσουμε την μεταβολή speech και speaker, η εφαρμογή της παραπάνω μεθόδου στα MFCC κρίνεται αναγκαία.

[6] Για ξένους όρους που η ελληνική αντίστοιχη μετάφραση δεν είναι σαφώς ορισμένη, παρατίθεται και ο αντίστοιχος ξένος όρος

- Εκτελούμε τις εντολές kaldi *make_mfcc.sh* [7] και *compute_cmvn_stats.sh*, για την παραγωγή των όσων αναφέρθηκαν, για κάθε ένα από τα τρία directory ξεχωριστά. [8]. Τα τελικά αποτελέσματα περιέχονται στους φακέλους **"mfcc_i** ,i = {train,test,dev} σε μορφή **".ark"**, ενώ εκείνα των στατιστικών αναλύσεων στον φάκελο **"exp/make_mfcc** σε μορφή **".log"**

ΕΡΩΤΗΜΑ 2

- Στην αναγνώριση φωνής χρειάζεται να αφαιρέσουμε οποιοδήποτε σήμα περιπλέκεται με το σήμα του ομιλητή καθώς δεν μπορούμε να εξάγουμε καθαρά τις παραμέτρους και έγκυρα. Τέτοια σήματα μπορεί να προέρχονται από το περιβάλλον του ομιλητή όπως θόρυβος, άλλες ομιλίες και εν γένη άλλες ηχητικές πηγές . Έχοντας το σήμα εισόδου x[n] το οποίο περνάει από φίλτρο απόκρισης h[n] παίρνουμε ένα σήμα εξόδου y[n]=x[n] $\times$ h[n] το οποίο είναι η συνέλιξη των x[n] και h[n].Χρησιμοποιώντας την ιδιότητα του M/Σ Fourier ισχύει ότι: $Y[f] = X[f]H[f]$

- Σαν επόμενο βήμα παίρνουμε cepstrum μέσω του λογαρίθμου του φάσματος:$Y[q] = \log(X[f]H[F]) = log(X[f]) + log(Y[f]) = X[q] + H[q]$

- Παρατήρηση: Από την συνέλιξη στο πεδίο του χρόνου μεταβήκαμε στον πολλαπλασιασμό στο πεδίο της συχνότητας και στην πρόσθεση στο πεδίο του cepsrtum. Τα παραπάνω βήματα θα χρειαστούν για τον υπολογισμό του Cepstral Mean Normalization.

- Πλέον με τις μετατροπές αυτές γνωρίζουμε ότι οποιεσδήποτε αλλοιώσεις έχουμε στο πεδίο του χρόνου από ανεπιθύμητους παράγοντες στο πεδίο του cepstrum τους έχουμε σε μορφή αθροίσματος Κάνοντας λοιπόν την υπόθεση ότι όλα αυτά είναι στατικά(stationary) μπορούμε να θωρήσουμε ότι το i-οστό πλαίσιο έχει την μορφή: $Y_i[q] = H[q] + X_i[q]$.

- Παίρνοντας τον μέσο όρο των πλαισίων του συνόλου των πλαισίων έχουμε : $\frac{1}{N}\sum_i Y_i[q] = H[q] + \frac{1}{N}\sum_i X_i[q]$

- Τώρα ορίζουμε την διαφορά "Ri"του i-οστού πλαισίου ως αυτήν του μέσου όρου ,που υπολογίσαμε πριν ,από την έξοδο: $R_i[q] = Y_i[q] - \frac{1}{N}\sum_i Y_i[q] = H[q] + X_i[q] - (H[q] + \frac{1}{N}\sum_i X_i[q])$. Επομένως: $R_i[q] = X_i[q] - \frac{1}{N}\sum_i X_i[q]$ που εν τέλη καταλήγει να είναι η διαφορά του X_i από τον μέσο όρο όλων των παραθύρων

- Το Cepstral Mean Normalization δεν είναι υποχρεωτικό ,ειδικά όταν πρόκειται για έναν ομιλητή σε ένα φυσιολογικό περιβάλλον χωρίς εξαιρετικό θόρυβο .Ωστόσο κρίνεται εξαιρετικό εργαλείο για την εξαγωγή καλύτερων καλύτερων ποσοστών με τον συνδυασμό ενός κέρδους(gain) πάνω στο σήμα καθώς τότε τα ποσοστά επιτυχούς αναγνώρισης είναι σημαντικά καλύτερα . Παρατήρηση: Έχει αποδειχθεί ότι η μέθοδος CMVN είναι ιδιαίτερα αποτελεσματική για μικρές εκφράσεις(utterances).

ΕΡΩΤΗΜΑ 3

- Γίνεται εκτέλεση του bash script **"4_3_appendix.sh"** το οποίο εκτελεί την εντολή *feat-to-dim* η οποία επιστρέφει την διάσταση των MFCC για το πρώτο από τα 4 sections που έχουμε κάνει split[9] Dimension = 13

- Γίνεται εκτέλεση της *feat-to-len* η οποία επιστρέφει τον αριθμό των frames για κάθε ένα από τα 46 utterrances id's στο πρώτο από τα 4 section, εμείς θέλουμε τα 5 πρώτα. Number of Features = [371,336,519,400,397] για τα utterance ids = [4,16,43,48,53]

Τα αποτελέσματα φαίνονται παρακάτω:

```
feat-to-dim ark:mfcc_test/raw_mfcc_test.1.ark -
13
feat-to-len scp:data/test/feats.scp ark,t:data/test/feats.lengths
usctimit_ema_f1_004 371
usctimit_ema_f1_016 336
usctimit_ema_f1_043 519
usctimit_ema_f1_048 400
usctimit_ema_f1_053 397
```

[7] το script αναζητά το αρχείο text **"conf/mfcc.conf"**,

[8] Σημειώνεται ότι το όρισμα –nj είναι ο αριθμός των jobs που θα σταλθούν, ο αριθμός των sections που θα γίνουν split τα δεδομένα, το οποίο είναι καλό να οριστεί από 1 έως 4, αλλά όχι περισσότερο καθώς το Kaldi αποθηκεύειτα δεδομένα των ομιλητών μαζί, άρα δεν θέλουμε περισσότερα splits απο τον αριθμό των ομιλητών

[9] Το dimension αυτό θα είναι ίδιο για όλα τα features του set

4.4 Training of acoustic models and sentences decoding

- Η αποκωδικοποιητική ικανότητα ενός συστήματος ASR εξαρτάται σχεδόν εξ'ολοκλήρου από τις επιδράσεις του **γλωσσικού** και του **ακουστικού** μοντέλου που κατασκευάσαμε. Το αντικείμενο του πρώτου είναι ο υπολογισμός της πιθανότητας μιας πρότασης στο τωρινό task αναγνώρισμης ανάμεσα σε ένα μεγάλο αριθμό προτάσεων, πράγμα που είναι κρίσιμο για την μείωση της πολυπλοκότητας της αναζήτησης των λέξεων στον αποκωδικοποιητή που θα κατασκευάσουμε. Επομένως, αναμένουμε το bigram μοντέλο που είναι πιο αποτελεσματικό από το unigram να οδηγήσει σε χαμηλότερο WER.

- Το αντικείμενο του δεύτερου είναι να περιγράψει στατιστικώς τις λέξεις σε μορφή φωνημάτων. Η πιο γνωστή μέθοδος - αυτή και που χρησιμοποιείται - είναι η GMM-HMM (Gaussian Mixture Model-Hidden Markov model). Το HMM βασίζεται σε πεπερασμένα αυτόματα για την απότιμηση των πιθανοτήτων μετάβασης, έτσι ώστε να μπορούμε να υπολογίσουμε τις πιθανότητες των αντίστοιχων λέξεων που προκύπτουν απο τις ακολουθίες φωνημάτων. Οι HMM καταστάσεις τυπικώς συγκροτούν είτε ένα **monophone** είτε ένα **triphone** μοντέλο,όπου άλλοι μέθοδοι επίσης εφαρμόζονται για περαιτέρω βελτίωση του μοντέλου. Τα GMM μοντελοποιούν το output-παρατήρηση των HMM καταστάσεων με Gaussian μίγματα. Επομένως οι διάφορες HMM καταστάσεις θα μοντελοποιηθούν με Gaussian κατανομή μέσω ενός clustering δέντρου απόφασης στο βήμα της κατασκευής του **monophone** μοντέλου[10]

1. - Το **monophone** μοντέλο είναι το πρώτο κομμάτι της διαδικασίας training. Αποτελεσματικά **monophone** μοντέλα μπορούν να σχηματιστούν και με λίγα δεδομένα, όπως στην συγκεκριμένη περίπτωση τα 1448 του trainset, κυρίως με σκοπό να δώσουν boost σε μετέπειτα μοντέλα, όπως το **triphone** που θα παράξουμε παρακάτω. Τα απαιτούμενα ορίσματα είναι διαρκώς κατά το training τα ακόλουθα:

```
- Location of the acoustic data: `data/train`
- Location of the lexicon: `data/lang_test`
- Destination directory for the model: `exp/mono0a`
```

   - Εκτελούμε την εντολή *train_mono.sh* με ορίσματα –nj = 4 (έχουμε εξηγήσει το γιατί), –cmd "$train_cmd" ώστε να ορίσουμε ποιο machine θα διαχειριστεί την διαδικασία. (Η μεταβλητή έχει αποτιμηθεί στο 4_1 βήμα), ενώ το θέτουμε και boost silence ως standard πρωτόκολλο για αυτό το training. Το αποτέλεσμα είναι η παραγωγή του **"mono0a"** μοντέλου.
   - Ακολουθούμε ακριβώς την ίδια διαδικασία με προηγούμενως για την κατασκευή των αντίστοιχων alignments του μοντέλου με την μόνη διαφορά ότι:
     Destination directory for the alignment: "exp/monoa0_ali"

2. Παρόμοιο FST framework, που διαδραμάτισε ρόλο κλειδί στην kaldi γραμματική, χρησιμοποίηθηκε τόσο για την training όσο για την testing διαδικασία. Το appendix C αποτελεί απεικόνιση του πως το Kaldi θα μπορούσε να παράξει transcriptions από ακουστικά features όσον αφορά την συγκεκριμένη γραμματική. Δημιουργήσαμε 6 τέτοιους γράφους έναν για κάθε μία από τις 6 γραμματικές (unigram και bigram των test, train και evaluation set) με εκτέλεση της εντολής *utils/mkgraph.sh –mono*. Βέβαια επειδή το kaldi πάει και βρίσκει συγκεκριμένα το **"G.fst"** αν υπάρχει για να εκτελέσει την παραπάνω σύνθεση, έγινε προσωρινή μετονομασία των 6 γραμματικών για παραγωγή του αντίστοιχου HCLG. Τα αποθήκευσαμε σε έναν φάκελο **"HCLG"** που βρίσκεται μέσα στον φάκελο του γράφου **graph_nosp_tgpr** για το **mono0a** μοντέλο, με σκοπό την ανάκτηση τους σε περίπτωση που χρειαστεί.

3. Τελικά, αφού έχουμε ολοκληρώσει το μοντέλο μας, ήρθε η ώρα να το εφαρμόσουμε στο test και evaluation set που μας δίνεται. Το μοντέλο μας, στα νέα πλέον δεδομένα για αυτό προσπαθεί να κάνει την κατάλληλη αποκωδικοποίηση και τα αποτελέσματα βρίσκονται στα αρχεία wer που βρίσκονται στο path **"exp/monoa0/decode_i"**.

   - Εκτελούμε επαναληπτικά τον αλγόριθμο 6 φορές, μία για κάθε HCLG γράφου που δημιουργήσαμε στο προηγούμενο βήμα. Πιο συγκεκριμένα, εκτελούμε την εντολή *steps/decode* με –nj = 4 (αριθμός pipelines αναθετημένα σε 4 CPU πυρήνες), και μηχανή –cmd "$decode_cmd" (runtime parallelisation), με ορίσματα τον γράφο-αποκωδικοποιητή και τον φάκελο **data/i** ∀ directory i = {train,test,dev}, δηλαδή τα utterance συσχετιζόμενα αρχεία.
   - Το βάρος του γλωσσικού μοντέλου για το οποίο ο αποκωδιποιητής θα ψάξει, επιβεβαιώνετε από την "κρυφή" και ταυτόχρονη εκτέλεση του *score_kaldi.sh* (με –scoring-opts από 1 έως 20) για τον υπολογισμό του PER (Phone Error Rate) [11].
   - Η αποκωδικοποίηση περιλαμβάνει λοιπόν wer και log αποτελέσματα στους φακέλους **"decode_dev "** και **"decode_test "** του φακέλου **"mono0a"** [12]

[10] όλα αυτά δημιουργούνται αυτόματα από το kaldi αλλά αναλύονται για ορθή κατανόηση των βημάτων
[11] Εδώ ταυτίζεται με το WER(Word Error Rate)
[12] η εντολή αποκωδικοποίησης ουσιαστικά είναι η εκτέλεση του αλγορίθμου viterbi

4. • Με το που τελειώνει η αποκωδικοποίηση για την δυάδα test και validation set μετά την εφαρμογή κάποιου απου τους 6 HCLG γράφους, εκτελείται η εντολή *utils/best_wer.sh.* για εύρεση του καλύτερου δυνατού PER (δηλαδή του μικρότερου δυνατού) [13].

   • Παρακάτω φαίνονται γραφικά τα αποτελέσματα του καλύτερο WER που πήραμε από την εκτέλεση για όλους τους γράφους HCLG, και άρα για όλα τα unigram και bigram γλωσσικά μοντέλα που κατασκευάσαμε:

```
%WER 50.12 [ 3082 / 6149, 163 ins, 1276 del, 1643 sub ] exp/mono0a/decode_dev/wer_4_0.0
%WER 47.57 [ 2837 / 5964, 136 ins, 1167 del, 1534 sub ] exp/mono0a/decode_test/wer_4_0.0
```

   • **ΕΡΩΤΗΜΑ:** Οι υπερπαράμετροι του scoring script για το συγκεκριμένο βήμα είναι το min και max LM-weight για lattice rescoring που πήραν τιμή 1 και 20 αντίστοιχα (το default-συνηθισμένο είναι 9 και 20).

5. • Το να κάνουμε train ένα **triphone** μοντέλο μεταφράζεται σε παραπάνω arguments για τον αριθμό των φύλλων, ή αριθμό HMM καταστάσσεων στο δέντρο απόφασης και στον αριθμό των Gaussianss. Πριν εκτελέσουμε την εντολή για training έχει προηγηθεί η δημιουργία train subset 1000 δεδομένων αντί για 1448 που ήταν το αρχικό που το ονομάζουμε **"train_1kshort"** με κατάλληλη κλήση της *utils/subset_data_dir.sh* και κατάλληλο fixαρισμα τυχόν απωλειών με κλήση της *utils/fix_data_dir.sh"* για λόγους συμβατότητας

   • Γίνεται εκτέλεση της *steps/train_deltas.sh*, με σκοπό να κάνουμε train τα aligments του **monophone** μοντέλου σε **triphone**, το directory του οποίου ονομάζουμε **"tri1"**. Θα μπορούσαμε να είχαμ μία HMM κατάσταση για κάθε φώνημα, αλλά γνωρίζουμε ότι τα φωνήματα ποικίλλουν σε σημαντικό βαθμό ανάλογα με το αν βρίσκονται στην αρχή, στην μέση ή στο τέλος μίας πρότασης. Αυτό μας οδηγεί σε τουλάχιστον 168 HMM καταστάσεις μόνο για το variation. Με 2000 HMM καταστάσεις, το μοντέλο μπορεί να αποφασίσει αν είναι καλύτερο να κατανείμει μία μοναδική HMM κατάσταση σε ποιο refined αλλόφωνα του αρχικού φωνήματος. Αυτό το split των φωνημάτων αποφασίζεται από φωνητικές ερωτήσεις στο **"questions.txt"** και στο **"extra_questions.txt"**

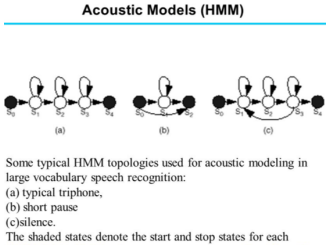
   • Ο ακριβής αιρθμός φύλλων και Gaussian αποφασίζεται συχνά από ευρυστικές, ο οποίος εξαρτάται από την ποσότητα των δεδομένων, τον αριθμό των ερωτήσεων περί φωνημάτων και τον στόχο του μοντέλου. Επίσης υπάρχει ο περιορισμός ότι ο αριθμός των Gaussians πρέπει **ΠΑΝΤΑ** να ξεπερνάει τον αριθμό των φύλλων. Όπως βλέπουμε στο script **"4_4.sh"**, εμπειρικά ορίσαμε #Leaves = 2000 και #Gaussians = 10000.

   • Μετα ακολουθείται πιστά ακριβώς η ίδια διαδικασία που περιγράφηκε στο προηγούμενο βήμα για αποκωδικοποίηση του test και evaluation set βάσει του **triphone** πλέον μοντέλου και εύρεση όλων των σχετικών PER που παρουσιάζονται παρακάτω:

```
%WER 41.62 [ 2559 / 6149, 309 ins, 670 del, 1580 sub ] exp/tri1/decode_dev_1kshort/wer_6_0.5
%WER 39.76 [ 2371 / 5964, 271 ins, 705 del, 1395 sub ] exp/tri1/decode_test_1kshort/wer_6_1.0
```

ΕΡΩΤΗΜΑ 4[14]

• Ένα ακουστικό μοντέλο σκοπό του έχει να περιγράψει στατιστικά τις λέξεις ενός λεξιλογίου μέσω ακολουθιών φωνημάτων. Το μοντέλο GMM-HMM είναι ένα γνωστό μοντέλο αρκετά αποτελεσματικό. Το κομμάτι των HMM(Hidden Markov Model) είναι βασισμένο σε ένα αυτόματο πεπερασμένων καταστάσεων το οποίο αναπαριστά τις πιθανότητες μετάβασης φωνημάτων σε λέξεις .Το μοντέλο αυτό καταλήγει να αναπαριστά την πιθανότητα εμφάνισης λέξεων μέσω των ακολουθιών από διακριτά φωνήματα. Η τυπική μορφή τους φαίνεται στην εικόνα παρακάτω:

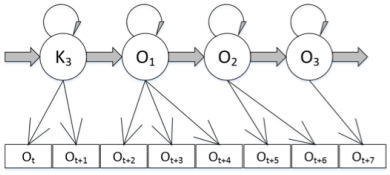


Some typical HMM topologies used for acoustic modeling in large vocabulary speech recognition:
(a) typical triphone,
(b) short pause
(c)silence.
The shaded states denote the start and stop states for each

[13] Εκ νέου εκτέλεση για κάθε γράφο HCLG, γιατί παρατηρήσαμε ότι δεν γίνονται overide τα wer δεδομένα από προηγούμενη εκτέλεση
[14] Έχει περίπου απαντηθεί ήδη από την αρχή του βήματος

- Οι πιο γνωστές αρχιτεκτονικές HMMs για αναγνώριση φωνής συνιστώνται από 3 καταστάσεις όπως παρουσιάζεται δίπλα στις λευκές καταστάσεις . Η λειτουργία του GMM έχει σκοπό να προσδιορίσει πάνω στο αυτόματο και να ομαδοποιήσει καταστάσεις σε συγκεκριμένες υποκατηγορίες το οποίο λειτουργεί κάτι σαν χαρτογράφηση περιοχών πάνω στο αυτόματο του HMM. Για παράδειγμα θα μπορεί να προσδιορίσει ένα φώνημα που εμφανίζεται σε πολλές λέξεις .Η εκπαίδευση αυτού του μοντέλου γίνεται επαναληπτικά πάνω σε ένα train set το οποίο βοηθά το GMM με βάσει τα αποτελέσματα του HMM για τις νέες εισόδους που δέχεται να κάνει ταξινόμηση σε κλάσεις.

- Στην εικόνα παρακάτω βλέπουμε ένα παράδειγμα από μια ακολουθία από καταστάσεις όπου η K3 είναι η τελευταία ενός φωνήματος ονόματος "K" και οι καταστάσεις (O1,O2,O3) είναι μια σειρά καταστάσεων που ανήκουν σε ένα φώνημα ονόματος "O". Τα τετράγωνα από κάτω συμβολίζουν μια σειρά από συμπεράσματα και παρατηρήσεις που προκύπτουν ,όπου t συμβολίζεται η χρονική στιγμή . Μπορούμε να δούμε ότι υπάρχει μια αντιστοίχιση (χαρτογράφηση) μιας κατάστασης σε πολλές παρατηρήσεις και συγκεκριμένα οι παρατηρήσεις για τις χρονικές στιγμές (t+2 , t+7) αφορούν το φώνημα "O".



ΕΡΩΤΗΜΑ 5

- Η πιθανότητα posterior υπολογίζεται με τον ακόλουθο τύπο ,όπου X είναι ένα διάνυσμα παρατηρήσεων(features) που έχει προκύψει από το ακουστικό μοντέλο βάσει των οποίων υπολογίζεται ποια είναι η πιθανότητα της λέξης W δεδομένου του X.

$$P(W|X) = \frac{P(W)P(X|W)}{P(X)}$$

- Για να βρούμε την πιο πιθανή λέξη (ή φώνημα) στην δική μας περίπτωση θα χρησιμοποιήσουμε την συνάρτηση argmax πάνω στο σύνολο των λέξεων (ή φωνημάτων) και ο τύπος θα έχει την μορφή

$$\widetilde{W} = \underset{W \in \omega}{\operatorname{argmax}} P(W|X)$$

ΕΡΩΤΗΜΑ 6

Γενικά υπάρχουν τέσσερα επίπεδα στον γράφο-αποκωδικοποιητή και μια σύνθεση τεσσάτων γράφων-συστατικών ήταν αναμενόμενη για την κατασκευή του τελικού γράφου HCLG:

- H - mapping από HMM μεταβάσεις σε context-dependent labels
- C - mapping από context-dependent labels σε φωνήματα
- L - mapping από φωνήματα σε λέξεις
- G - το γλωσσσικό μοντέλο (input και output είναι λέξεις)

Η γραμματική ουσιαστικά χρησιμοποιείται και στην φάση του training και στην φάση του testing. Το παράρτημα C με το περιεχόμενο των λέξεων αναπαριστά ένα παράδειγμα για το πως το KALDI μοντελοποιεί την μετάβαση από το ακουστικό μοντέλο σύμφωνα με την γραμματική του.

4.5 DNN-HMM by using Pytorch

Εναλλακτική μέθοδος για train του γλωσσικού και του ακουστικού μοντέλου είναι η δημιουργία νευρωνικών δικτύων μοντέλων που αποσκοπούν στην μεγάλη performance για πολύ μεγάλου μεγέθους άγνωστα inputs. Όσον αφορά το γλωσσικό, αποδεικνύεται ότι η εφαρμογή recurrent neural network based language models (RNNLM) μειώνει σημαντικά το WER/PER. Παράλληλα, μέθοδοι για training ακουστικών μοντέλων με την βοήθεια deep neural networks (DNN), όπως και στην περίπτωση του βήματος αυτού, μπορούν να έχουν πολύ καλύτερη αποτελεσματικότητα από τις παρούσες συμβατικές μεθόδους, ειδικά εάν εμπρόκειτο για πολύ μεγάλα datasets και vocabularies.

1. Δημιουργούμε τα aligments για το **triphone** με τον γνωστό πλέον τρόπο εκτελώντας το bash script **"4_5.sh"**. Τονίζεται ότι ακριβώς πριν από αυτό έγινε υπολογισμός των CMVN στατιστικών στοιχείων για κάθε set και το αποθηκεύσαμε σε **".ark"** αρχεία, και τα οποία θα χρησιμοποιηθούν παρακάτω από την python Dataset κλάση που μας δόθηκε στις διευκρινήσεις.

2. - Κάναμε append στο PYTHONPATH τον φάκελο με τα βοηθητικά python dnn scripts ώστε να έχουμε στην διάθεση μας την κλάση **TorchSpeechDataset**,την οποία καλέσαμε 3 φορές για να δημιουργήσουμε τα αντίστοιχα train, test και dev set για τα triphone aligments του προηγούμενο βήματος.[15], καθώς παρά την κατάλληλη εισαγωγή του kaldi_io στο PYTHOPATH,προέκυπτε threading error

3. Ο κώδικας building του μοντέλου περιέχει αναλυτικά σχόλια που είναι ικανοποιητικά για την επεξήγηση του.

4.6 Model Optimizations

Σαν βελτίωση της αλγοριθμικής διαδικασίας μπορούμε να επισημάνουμε το εξής:

- **Όσον αφορά στην εξαγωγή MFCC** Έχει υποτεθεί στην ανάλυση η ανεξαρτησία των παραθύρων των σημάτων πέρα της επικάλυψης και η ανάλυση καθενός ως έχει. Αυτό είναι κάτι που ισχύει σπάνια στην διαδοχή των φωνημάτων. Θα προτείναμε όπως και στα γλωσσικά μοντέλα την εισαγωγή Hidden Markov Models και στην εξαγωγή των χαρακτηριστικών για την δημιουργία κάποιας τύπου "συντακτικής" ανάλυσης του λόγου ή έστω κάποιας συσχέτισης στην ανάλυση διαδοχικών φωνημάτων με χρήση bigrams.

- **Όσον αφορά στην εκπαίδευση του μοντέλου** Σε γενικές γραμμές ,οι ανεξαρτησίες φωνημάτων είναι σπάνιες στην διατύπωση μιας πρότασης. Αυτός είναι και ο λόγος που προσπαθούμε να συγκεντρώνουμε τις συσχετίσεις διαδοχικών φωνημάτων με την χρήση N-grams. Παρατηρούμε ότι το μοντέλο μας σταματάει στην δημιουργία tri-grams. Είναι λοιπόν πιθανόν η χρήση four-grams να βελτιώσει ακόμη περισσότερο τα αποτελέσματα.

- Επιπλέον, συχνά στον τομέα αναγνώρισης φωνής χρησιμοποιούνται και άλλες μέθοδοι για την εξαγωγή χαρακτηριστικών, όπως η οπτική αναγνώριση της θέσης και του σχηματισμού των χειλιών και της γλώσσας του ομιλητή (με τοποθέτηση οπτικού αισθητήρα).Συχνά αυτό οδηγεί σε καλύτερη αναγνώριση των φωνημάτων.

5 Appendix

[15]Κάνουμε include το αρχείο torch_dataset.py γιατί έγιναν τροποποιήσεις με την έννοια της εκτέλεσης των kaldi εντολών στο terminal και αποθήκευση των αποτελεσμάτων feats και labels στον φάκελο **data/test/extra** και **data/dev/extra**

Step_3

December 25, 2018

In [1]:
```
#-----------------------------------------STEP 3 - Preparation-----------------------------------------#
#-------------------(1)-------------------#
import shutil
import re
import string
import os
from collections import defaultdict
destination_directory = !mdfind kind:folder "kaldi"
destination_directory = destination_directory[0].split('/')[:4]
destination_directory = '/'.join(destination_directory)
destination_directory = destination_directory + "/egs/usc/data"
shutil.copy("slp_lab2_data/filesets/test_utterances.txt", destination_directory + "/test/uttids")
shutil.copy("slp_lab2_data/filesets/train_utterances.txt",destination_directory + "/train/uttids")
shutil.copy("slp_lab2_data/filesets/validation_utterances.txt",destination_directory + "/dev/uttids")
```

Out[1]:
```
'/Users/jimmyg1997/kaldi/egs/usc/data/dev/uttids'
```

In [2]:
```
#-------------------(2)-------------------#
import os
for file in os.listdir(destination_directory):
    if(file == ".DS_Store"): continue
    filename_source = open(destination_directory + '/'+file + "/uttids", 'r')
    lines_source = filename_source.readlines()
    filename_destination = open(destination_directory + '/'+ file + "/utt2spk",'w')

    for line in lines_source:
        filename_destination.write(line.strip('\n') + ' ' +line.split('_')[2] + '\n')

filename_source.close()
filename_destination.close()
```

In [3]:
```
#-------------------(3)-------------------#
import os
slp_lab2_data = !mdfind kind:folder "slp_lab2_data"
slp_lab2_data = slp_lab2_data[0] + "/wav/"

for file in os.listdir(destination_directory):
    if(file == ".DS_Store"): continue
    filename_source = open(destination_directory + '/'+file + "/utt2spk", 'r')
    lines_source = filename_source.readlines()
    filename_destination = open(destination_directory + '/'+ file + "/wav.scp",'w')

    for line in lines_source:
        path = slp_lab2_data + line.split()[1] +'/' +line.split()[0] +".wav"
        filename_destination.write(line.split()[0] + ' ' +path + '\n')

filename_source.close()
filename_destination.close()
```

In [4]:
```
#-------------------(4)-------------------#
for file in os.listdir(destination_directory):
    if(file == ".DS_Store"): continue
    filename_source = open(destination_directory + '/'+file + "/utt2spk", 'r')
    lines_source = filename_source.readlines()
    filename_destination = open(destination_directory + '/'+ file + "/text",'w')
    sentences = "slp_lab2_data/transcription.txt"
    sentences = open(sentences, 'r')
    lines_sentences = sentences.readlines()

    for line in lines_source:
        chat = int(line.split()[0].split("_")[3])
        filename_destination.write(line.split()[0]+ ' '+lines_sentences[chat-1].strip('\n') + '\n')



filename_source.close()
filename_destination.close()
```

Step_4

December 24, 2018

In [1]:
```
#-----------------------------------------STEP 4 ----------------------------------------------------#
#-------------------(4.1) - 1--------------------#
#Try to source path.sh#
#This recipe requires to set up $KALDI_ROOT variable so it can use Kaldi
#binaries and scripts from $KALDI_ROOT/egs/wsj/s5/.
import os
kaldi = !mdfind kind:folder "kaldi"
kaldi = kaldi[len(kaldi) - 1]

kaldi_ = kaldi + "/egs/wsj/s5/path.sh"

bashCommand = ". " + kaldi_
os.system(bashCommand)
```

Out[1]: 0

In [2]:
```
#-------------------(4.1) - 2,3--------------------#
#Create softlinks#

kaldi_softlink1 = kaldi[0] + "/egs/wsj/s5/steps"
bashCommand = "ln -s " + kaldi_softlink1 + " steps"
os.system(bashCommand)


kaldi_softlink2 = kaldi[0] + "/egs/wsj/s5/utils"
bashCommand = "ln -s " + kaldi_softlink2 + " utils"
os.system(bashCommand)



!mkdir local

kaldi_softlink3 = "../steps/score_kaldi.sh"
bashCommand = "ln -s " + kaldi_softlink3 + " score_kaldi.sh"
os.system(bashCommand)

!mv score_kaldi.sh local
```

```
mkdir: local: File exists
```

In [3]:
```
#-------------------(4.2) - 1 - a--------------------#

filename = open("data/local/dict/silence_phones.txt",'w')
filename.write('sil\n')
filename.close()

filename = open("data/local/dict/optional_silence.txt",'w')
filename.write('sil\n')
filename.close()
```

In [4]:
```
#-------------------(4.2) - 1 - b--------------------#
lexicon = "slp_lab2_data/lexicon.txt"
lexicon = open(lexicon,'r')
lines_lexicon = lexicon.readlines()
lexicon.close()

phones = "data/local/dict/nonsilence_phones.txt"
phones = open(phones,'w')
phonemes = []
for line in lines_lexicon:
    sentence_phonemes = line.split()[1:]
    phonemes += sentence_phonemes

phonemes = sorted(list(set(phonemes)))
for phonem in phonemes:
```

```
        if(phonem == 'sil'):continue
        phones.write(phonem + '\n')
    phones.close()
```

In [ ]:
```
#-------------------(4.2) - 1 - c-------------------#
lexicon = "data/local/dict/lexicon.txt"
lexicon = open(lexicon,'w')

for phonem in phonemes:
    lexicon.write(phonem + ' '+ phonem + '\n')
lexicon.close()
```

In [ ]:
```
#-------------------(4.2) - 1 - d-------------------
import shutil
import os
from collections import defaultdict

destination_directory = kaldi + "/egs/usc/data"

for file in os.listdir(destination_directory):
    if(file == ".DS_Store"): continue

    filename_source_destination = open(destination_directory + '/'+file + "/text", 'r')
    lines_source_destination = filename_source_destination.readlines()
    filename_source_destination.close()
    line_pointer = 0

    for line in lines_source_destination:
        line_splitted = line.split()
        utterance_id = line_splitted[0]
        sentence = "<s> " + ' '.join(line_splitted[1:]) + " </s>\n"
        lines_source_destination[line_pointer] = sentence
        line_pointer+=1

    with open("data/local/dict/" +"/lm_"+file+".text", 'w') as filename_source_destination:
        filename_source_destination.writelines( lines_source_destination )

    filename_source_destination.close()
```

In [ ]:
```
#-------------------(4.2) - 1 - e-------------------#
for file in os.listdir(destination_directory):
    if(file == ".DS_Store"): continue

    filename_source_destination = open("data/local/dict/"+"/extra_questions.txt", 'w')
    filename_source_destination.close()
```

In [ ]:
```
### -------------------(4.2) - 2 ----------------------#
##Notice that build-lm.sh produces a LM file train.ilm.gz that is NOT in the
##final ARPA format, but in an intermediate format called iARPA, that is recognized by the compile-lm
##command and by the Moses SMT decoder running with IRSTLM. To convert the file into the standard ARPA
##format you can use the command:  compile-lm train.ilm.gz --text yes train.lm
##ENVIRONMENT /Users/jimmyg1997/kaldi/tools/irstlm/bin
os.chdir("/Users/jimmyg1997/Desktop/SLP_lab2/")

!./4_2_2.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 
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing "$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c $tmpdir/n
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c > $tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary $additional_parameters 
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing "$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c $tmpdir/n
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c > $tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary $additional_parameters 
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c > $tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary $additional_parameters 
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing "$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c $tmpdir/n
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c > $tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary $additional_parameters 
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing "$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c $tmpdir/n
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c > $tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary $additional_parameters 
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing "$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c $tmpdir/n
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c > $tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dictionary $additional_parameters 
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smoothing "$additional_smoothing_parameters" --size $order --ngrams "$gunzip -c $tmpdir/n
```

```
In [ ]: ### -------------------(4.2) - 3 ----------------------#
        !./4_2_3.sh

inpfile: lm_train1.ilm.gz
outfile: lm_train1.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to lm_train1.lm
inpfile: lm_train1.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to /dev/stdout
inpfile: lm_train2.ilm.gz
outfile: lm_train2.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to lm_train2.lm
inpfile: lm_train2.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to /dev/stdout
inpfile: lm_test1.ilm.gz
outfile: lm_test1.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to lm_test1.lm
inpfile: lm_test1.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to /dev/stdout
inpfile: lm_test2.ilm.gz
outfile: lm_test2.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to lm_test2.lm
inpfile: lm_test2.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to /dev/stdout
inpfile: lm_dev1.ilm.gz
outfile: lm_dev1.lm
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to lm_dev1.lm
inpfile: lm_dev1.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to /dev/stdout
inpfile: lm_dev2.ilm.gz
outfile: lm_dev2.lm
loading up to the LM level 1000 (if any)
```

```
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to lm_dev2.lm
inpfile: lm_dev2.ilm.gz
outfile: /dev/stdout
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 42
OOV code is 42
Saving in txt format to /dev/stdout
```

In [ ]:
```
### --------------------(4.2) - 4 -----------------------#
# This script prepares a directory such as data/lang/, in the standard format,
# given a source directory containing a dictionary lexicon.txt in a form like:
# word phone1 phone2 ... phoneN
# per line (alternate prons would be separate lines), or a dictionary with probabilities
# called lexiconp.txt in a form:
# word pron-prob phone1 phone2 ... phoneN
#
#This step is considered as the second part of the language-data preparation.
#After creating the "local/dict" directory, the user needs to create another required directory,
#called "lang". This directory can be generated by using "prepare_lang.sh" script that is provided by
#Kaldi toolkit. This directory can be prepared as follows:
#https://blogs.cornell.edu/finitestatecompling/2016/09/26/utilsprepare_lang-sh/


#L_disambiguate.fst:In general, you need to have disambiguation symbols when you have one word that is
#a prefix of another (cat and cats in the same lexicon would need to have cat being pronounced "k ae t #1")
#or a homophone of another word (red: "r eh d #1", read: "r eh d #2"). If you don't have these then the models
#become nondeterministic.


#Note 1: utils/validate_dict_dir.pl data/local/dict check (like substep of prepare_lang)


!./4_2_4.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 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 74 word sequence
--> resulting phone sequence from L.fst corresponds to the word sequence
--> L.fst is OK
--> generating a 74 word 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]
```

In [ ]:
```
### --------------------(4.2) - 5 -----------------------#
        !./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_train1.fst
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:94) Reading \data\ section.
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:149) Reading \1-grams: section.
LOG (arpa2fst[5.5.139~1-0e5d75]: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_dev1.fst
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:94) Reading \data\ section.
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:149) Reading \1-grams: section.
LOG (arpa2fst[5.5.139~1-0e5d75]: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_test1.fst
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:94) Reading \data\ section.
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:149) Reading \1-grams: section.
LOG (arpa2fst[5.5.139~1-0e5d75]: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_train2.fst
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:94) Reading \data\ section.
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:149) Reading \1-grams: section.
LOG (arpa2fst[5.5.139~1-0e5d75]: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_dev2.fst
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:94) Reading \data\ section.
```

```
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:149) Reading \1-grams: section.
LOG (arpa2fst[5.5.139~1-0e5d75]: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_test2.fst
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:94) Reading \data\ section.
LOG (arpa2fst[5.5.139~1-0e5d75]:Read():arpa-file-parser.cc:149) Reading \1-grams: section.
LOG (arpa2fst[5.5.139~1-0e5d75]:RemoveRedundantStates():arpa-lm-compiler.cc:359) Reduced num-states from 1 to 1
Succeeded in formatting data.
```

In [ ]:
```
#Computation of the Perplexity
#With the estimated LM, we can now compute the perplexity of any text contained in "test.txt"
#by running the commands below.
#To be compliant with the training data actually used to estimate the LM, start and end symbols
#are added to the text as well.

!./4_2_appendix.sh
```

```
##################################################
Calculating perplexity for train data unigram model
##################################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:

DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_train1.lm
outfile: lm_train1.lm.blm
evalfile: ../dict/lm_train.text
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=48443 PP=31.88 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
##################################################
Calculating perplexity for train data bigram model
##################################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:
 1e-06
DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_train2.lm
outfile: lm_train2.lm.blm
evalfile: ../dict/lm_train.text
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=48443 PP=15.44 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
##################################################
Calculating perplexity for evaluation data unigram model
##################################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:

DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_dev1.lm
outfile: lm_dev1.lm.blm
evalfile: ../dict/lm_dev.text
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=6332 PP=32.52 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
##################################################
Calculating perplexity for evaluation data bigram model
##################################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:
```

```
 1e-06
DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_dev2.lm
outfile: lm_dev2.lm.blm
evalfile: ../dict/lm_dev.text
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=6332 PP=15.26 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
###################################################
Calculating perplexity for test data unigram model
###################################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:

DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_test1.lm
outfile: lm_test1.lm.blm
evalfile: ../dict/lm_test.text
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=6148 PP=32.00 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
###################################################
Calculating perplexity for test data bigram model
###################################################
OOV code is 42
OOV code is 42
pruning LM with thresholds:
 1e-06
DEBUG_LEVEL:0/1 Everything OK
inpfile: lm_test2.lm
outfile: lm_test2.lm.blm
evalfile: ../dict/lm_test.text
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=6148 PP=14.58 PPwp=0.00 Nbo=0 Noov=0 OOV=0.00%
```

In [ ]:
```
#-------------------(4.3) - 1--------------------#

        !./4_3_1.sh
```

```
steps/make_mfcc.sh --cmd run.pl --nj 20 data/train exp/make_mfcc/train mfcc_train
steps/make_mfcc.sh: moving data/train/feats.scp to data/train/.backup
steps/make_mfcc.sh: [info]: no segments file exists: assuming wav.scp indexed by utterance.
Succeeded creating MFCC features for train
steps/compute_cmvn_stats.sh data/train exp/make_mfcc/train
Succeeded creating CMVN stats for train
steps/make_mfcc.sh --cmd run.pl --nj 20 data/test exp/make_mfcc/test mfcc_test
steps/make_mfcc.sh: moving data/test/feats.scp to data/test/.backup
steps/make_mfcc.sh: [info]: no segments file exists: assuming wav.scp indexed by utterance.
Succeeded creating MFCC features for test
steps/compute_cmvn_stats.sh data/test exp/make_mfcc/test
Succeeded creating CMVN stats for test
steps/make_mfcc.sh --cmd run.pl --nj 20 data/dev exp/make_mfcc/dev mfcc_dev
steps/make_mfcc.sh: moving data/dev/feats.scp to data/dev/.backup
steps/make_mfcc.sh: [info]: no segments file exists: assuming wav.scp indexed by utterance.
Succeeded creating MFCC features for dev
steps/compute_cmvn_stats.sh data/dev exp/make_mfcc/dev
Succeeded creating CMVN stats for dev
```

In [ ]:
```
### -------------------(4.4) - 1--------------------#
        !./4_4.sh
```

```
steps/train_mono.sh --boost-silence 1.25 --nj 4 --cmd run.pl data/train data/lang_test exp/mono0a
** split_data.sh: warning, #lines is (utt2spk,text) is (    1468,    1418); you can
** use utils/fix_data_dir.sh to fix this.
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 51.0969568294% of the time at utterance begin.  This may not be opt
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only 16.4189667374% of the time at utterance end.  This may not be optim
steps/diagnostic/analyze_alignments.sh: see stats in exp/mono0a/log/analyze_alignments.log
3716 warnings in exp/mono0a/log/align.*.*.log
2 warnings in exp/mono0a/log/analyze_alignments.log
2466 warnings in exp/mono0a/log/acc.*.*.log
120 warnings in exp/mono0a/log/update.*.log
6 warnings in exp/mono0a/log/init.log
exp/mono0a: nj=4 align prob=-83.98 over 1.74h [retry=1.7%, fail=0.4%] states=125 gauss=998
```

```
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
** split_data.sh: warning, #lines is (utt2spk,text) is (    1468,    1418); you can
** use utils/fix_data_dir.sh to fix this.
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 50.7430997877% of the time at utterance begin.  This may not be opt
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only 18.1174805379% of the time at utterance end.  This may not be optim
steps/diagnostic/analyze_alignments.sh: see stats in exp/mono0a_ali/log/analyze_alignments.log
steps/align_si.sh: done aligning data.
WARNING: the --mono, --left-biphone and --quinphone options are now deprecated and ignored.
tree-info exp/mono0a/tree
tree-info exp/mono0a/tree
make-h-transducer --disambig-syms-out=exp/mono0a/graph_nosp_tgpr/disambig_tid.int --transition-scale=1.0 data/lang_test/tmp/ilabels_1_0 exp/m
fstrmepslocal
fstminimizeencoded
fstrmsymbols exp/mono0a/graph_nosp_tgpr/disambig_tid.int
fstdeterminizestar --use-log=true
fsttablecompose exp/mono0a/graph_nosp_tgpr/Ha.fst data/lang_test/tmp/CLG_1_0.fst
fstisstochastic exp/mono0a/graph_nosp_tgpr/HCLGa.fst
0.000976562 -0.000313932
add-self-loops --self-loop-scale=0.1 --reorder=true exp/mono0a/final.mdl exp/mono0a/graph_nosp_tgpr/HCLGa.fst
steps/decode.sh --nj 3 --cmd run.pl exp/mono0a/graph_nosp_tgpr data/dev exp/mono0a/decode_dev
decode.sh: feature type is delta
steps/diagnostic/analyze_lats.sh --cmd run.pl exp/mono0a/graph_nosp_tgpr exp/mono0a/decode_dev
steps/diagnostic/analyze_lats.sh: see stats in exp/mono0a/decode_dev/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(8,42,502) and mean=264.9
steps/diagnostic/analyze_lats.sh: see stats in exp/mono0a/decode_dev/log/analyze_lattice_depth_stats.log
local/score_kaldi.sh --cmd run.pl data/dev exp/mono0a/graph_nosp_tgpr exp/mono0a/decode_dev
local/score_kaldi.sh: scoring with word insertion penalty=0.0,0.5,1.0
steps/decode.sh --nj 3 --cmd run.pl exp/mono0a/graph_nosp_tgpr data/test exp/mono0a/decode_test
decode.sh: feature type is delta
steps/diagnostic/analyze_lats.sh --cmd run.pl exp/mono0a/graph_nosp_tgpr exp/mono0a/decode_test
analyze_phone_length_stats.py: WARNING: optional-silence sil is seen only 79.347826087% of the time at utterance end.  This may not be optima
steps/diagnostic/analyze_lats.sh: see stats in exp/mono0a/decode_test/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(9,42,519) and mean=235.7
steps/diagnostic/analyze_lats.sh: see stats in exp/mono0a/decode_test/log/analyze_lattice_depth_stats.log
local/score_kaldi.sh --cmd run.pl data/test exp/mono0a/graph_nosp_tgpr exp/mono0a/decode_test
local/score_kaldi.sh: scoring with word insertion penalty=0.0,0.5,1.0
compute-wer --text --mode=all ark:exp/mono0a/decode_test/scoring_kaldi/test_filt.txt ark,p:-
```

In [ ]:

In [ ]:

Step_5_temp

December 24, 2018

In [1]:
```
#----------------------------------------STEP BONUS----------------------------------------#
#-------------------(4.5) - 1--------------------#
!./4_5.sh
```

In [2]:
```
#-------------------(4.5) - 2--------------------#
import os
import sys

sys.path.append("/Users/jimmyg1997/Desktop/SLP_lab2/lab2_help_scripts/")
from torch_dataset import *

trainset = TorchSpeechDataset(recipe_dir='/Users/jimmyg1997/kaldi/egs/wsj/s5', ali_dir='tri1_ali', feature_context=3, dset='train_1ks
```

```
success
usctimit_ema_f1_001
```

In [3]:
```
#-------------------(4.5) - 3--------------------#
testset = TorchSpeechDataset(recipe_dir='/Users/jimmyg1997/kaldi/egs/wsj/s5', ali_dir='tri1_test_ali', feature_context=3, dset='test'
validationset = TorchSpeechDataset(recipe_dir='/Users/jimmyg1997/kaldi/egs/wsj/s5', ali_dir='tri1_dev_ali', feature_context=3, dset='
```

```
usctimit_ema_f1_004
usctimit_ema_f1_021
```

In [4]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import Dataset
from torch.utils.data import 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__()
        #in_features = #input features
        #out_features = #output features
        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(nn.Module):
            def __init__(self, layers, n_features, n_classes, activation='sigmoid'):
                super(DNN, self).__init__()
                layers_in = [n_features] + layers
                layers_out = layers + [n_classes]
                self.f = nn.Sequential(*[
                    LinearWActivation(in_feats, out_feats, activation=activation)
                    for in_feats, out_feats in zip(layers_in, layers_out)
                ])

            def forward(self, x):
                y = self.f(x)
                return y

        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]


In [5]: EPOCHS = 10
        BATCH_SZ = 10
        #Let's check if a CUDA GPU is available and select our device. Running the network on a GPU will greatly decrease the training/testin
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        ##################STEP1#################
        X_train = torch.Tensor(trainset.feats).to(device)
        y_train = torch.Tensor(trainset.labels).to(device)

        X_test = torch.Tensor(testset.feats).to(device)
        y_test = torch.Tensor(testset.labels).to(device)

        ##################STEP2#################
        #Prepare the data loader
        mytransform = transforms.Compose([GaussianNoise(),ToTensor()])
        #mytransform = transforms.Compose( (ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))))
        trainset = Data(X_train,y_train, trans = mytransform)
        testset = Data(X_test,y_test, trans = mytransform)

        ##################STEP3#################
        train_dl = DataLoader(dataset=trainset, batch_size=BATCH_SZ, shuffle=False)
        test_dl = DataLoader(dataset=testset, batch_size=BATCH_SZ, shuffle=False)

        net = DNN([1],X_train.shape[1],len(y_train))
        net.to(device)
        #One important thing about torch.nn.CrossEntropyLoss is that input has to be a 2D tensor
        #of size (minibatch, n) and target expects a class index (0 to nClasses-1) as the target
        #for each value of a 1D tensor of size minibatch
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(list(net.parameters()),lr=0.001,momentum=0.99)
        net.train()
        #one epoch equals one forward pass (getting the output values), and one backward pass
        #(updating the weights) equals all the training examples.
        for epoch in range(1):
            running_average_loss = 0
            #Total Size = 11895
            print("BEGIN")
            for i, data in enumerate(train_dl):
                print(i)
                #And now, for the dessert, we run our Gradient Descent for 50 epochs.
                #This does the forward propagation, loss computation, backward
                #propagation and parameter updation in that sequence.
                X_batch = X_batch.to(device)
                y_batch = y_batch.to(device)
```

```
                X_batch = X_batch.requires_grad_() #set requires_grad to True for training
                X_batch, y_batch = data
                optimizer.zero_grad()
                # (1) Forward
                out = net(X_batch)
                # (2) Compute diff
                loss = criterion(out, y_batch)
                # (3) Compute gradients
                loss.backward()
                # (4) update weights: The way the neural network "learns"
                optimizer.step()

                running_average_loss += loss.detach().item()
                if i % 1000 == 0:  # print every 1000 mini-batches
                    print("Epoch: {} \t Batch: {} \t Loss {}".format(epoch, i, float(running_average_loss) / (i + 1)))
            #Validation
            #with torch.set_grad_enabled(False):
            #    for local_batch, local_labels in validation_generator:
            #        # Transfer to GPU
            #        local_batch, local_labels = local_batch.to(device), local_labels.to(device)

                    # Model computations
            #        [...]
```

```
BEGIN
0
Epoch: 0         Batch: 0         Loss 12.875078201293945
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
```

12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12796
12797
12797
12797
12797
12797
12797
12797
12797
12797
12797
12797
12797
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12798
12799
12799
12799
12799
12799
12800
12800
12800
12800
12800
12800
12800
12800
12800
12800
12800
12800
12800
12800

12800
12800
12800
12800
12800
12800
12800
12800
12800
12800
12800
12801
12801
12801
12802
12802
12802
12802
12802
12802
12802
12802
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12803
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
12804
0.05009076114171257

In [7]:
```
#-------------------(4.5) - 4-------------------#
posts = net.calculate_posteriors(testset.feats)
#Open the .ark file to write the posteriors:
post_file = kaldi_io.open_or_fd('/Users/jimmyg1997/kaldi/egs/wsj/s5/exp/dnn/posteriors.ark', 'wb')

#Write the posteriors per utterance:
start_index = 0
testset.end_indexes[-1] += 1
for i, name in enumerate(testset.uttids):
    out = posts[start_index:testset.end_indexes[i]]
    start_index = testset.end_indexes[i]
    kaldi_io.write_mat(post_file, out, testset.uttids[i])
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-7-a12bcd4731ba> in <module>
  1 #-------------------(4.5) - 4-------------------#
----> 2 posts = net.calculate_posteriors(testset.feats)
  3 #Open the .ark file to write the posteriors:
  4 post_file = kaldi_io.open_or_fd('/Users/jimmyg1997/kaldi/egs/wsj/s5/exp/dnn/posteriors.ark', 'wb')
  5


/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/torch/nn/modules/module.py in __getattr__(self, name)
533                 return modules[name]
534         raise AttributeError("'{}' object has no attribute '{}'".format(
--> 535             type(self).__name__, name))
536
537     def __setattr__(self, name, value):


AttributeError: 'DNN' object has no attribute 'calculate_posteriors'
```

In [ ]:
```
#-------------------(4.5) - 5-------------------#
# ------------------ Decoding ----------------- #
# The posteriors.ark file is produced by your PyTorch code (you can choose to save it in any directory,
# the example considers it is saved in the output directory exp/dnn)
os.chdir("/Users/jimmyg1997/Desktop/SLP_lab2/")
workdir=tri1
graph_dir=exp/${workdir}/graph_nosp_tgpr
data_dir=data/test
ali_dir=exp/${workdir}_test_ali
out_folder=exp/dnn

./decode_dnn.sh $graph_dir $data_dir $ali_dir $out_folder/decode_test "cat $out_folder"/posteriors.ark""
```

In [ ]: