Major Speech Practical

Maximilian Chamberlin

April 29, 2016

Introduction

The practical examines 3 parts of state-of-the-art large vocabulary speech recognition. These are language modelling; acoustic model speaker adaptation; and system combination. The initial language models given in this practical were of the order of many megabytes, and had been trained on hundreds of MWs (million words). This can be a daunting task for speech recognition because of the computational expense: Trigram and higher-order language models can result in a highly complex search-spaces, for which procedures such as decoding and rescoring can take far too much time. As potential solutions, in this practical various methods are discussed to reduce the burden of these procedures, such as trigram rescoring on smaller (bigram) models; combining systems using the ROVER and CNC methods; adapting models on the fly, as is performed in show specific adaptation. Ultimately, the aim of the practical is to explore the effects of improved language and acoustic models on large vocabulary speech recognition.

Language Model Improvements

Provide a description of your implementation of the LM interpolation weight estimation scheme.

Background

The probability of a word as given by a trigram language model is $P(\mathbf{w}) \approx \prod_{i=1}^{K} P(w_i|w_{i-2}, w_{i-1})$. Combining the 5 language language models with different weights λ_a, the probability of a word sequence becomes: $P(\mathbf{w}) \approx \prod_{i=1}^{K} (\sum_{a=1}^{5} P_a(w_i|w_{i-2}, w_{i-1}))$, where P_a denotes a probability from the 5th trigram language model.

As has been explained in the practical handout, these interpolation weights can be optimised using expectation maximisation updates designed to maximise perplexity, which may be expressed as $PP = (P(\mathbf{w}))^{-\frac{1}{K}}$, where K is the number of words . EM requires updating the language model weights according to the following formula:

- $\lambda_a^{(\tau+1)} = \frac{1}{K} \sum_{i=1}^{k} P(a|\mathbf{w},i,\tau)$

where $P(a|\mathbf{w},i,\tau)$ gives the posterior probability that the language model a produced the ith word in $\mathbf{w}$ at time step τ. To perform Language model interpolation, it is necessary to:

1. Generate the trigram stream of probabilities $<P_a(\mathrm{w}_1),...,P_a(\mathrm{w}_N)>$, for each language model a=lm1..lm5.
2. Initialise the language model interpolation weights,λ_a.
3. During each step of EM, to calculate the posterior probabilities $P(a|\mathbf{w},i,\tau)$ to find updates for the weights .
4. Continue until convergence has been reached.

I shall now explain how I implemented this in my code:

1. I generated the stream of trigram probabilities using the Lplex command for each language model on $\mathbf{w}$, the development text. Once these streams have been generated they are output to text-files within a folder called streamed. [1]. Having done this, I read in the streams $<P_a(\mathrm{w}_1),...,P_a(\mathrm{w}_N)>$ to separate list for each language model a=lm1:5. [2]

[1] Code in listing 1 of Appendix 1

[2] using Part 1 of the code in listing 2 (This is the second chunk of text in the listing).

2. The next stage of the process was to initialise the 5 language model interpolation weights $\lambda_{1:5}$. I have set them to be randomly initialised, with the additional constraint that interpolation weights must sum to one. [3]

3. The main body of the expectation maximisation updates are performed. The EM update at the t+1$_{th}$ iteration is $\lambda_a^{(\tau+1)} = \frac{1}{\mathrm{K}}\sum_{\mathrm{i}=1}^{\mathrm{K}} \mathrm{P}(\mathrm{a}|\mathrm{w}, \mathrm{i}, \tau)$. The posterior probability $\mathrm{P}(\mathrm{a}|\mathrm{w}, \mathrm{i}, \tau) = \frac{\lambda a \cdot P_a(w_i|w_{i-2},w_{i-1})}{\sum_{a'=1}^{5} \lambda_{a'} P_{a'}(w_i|w_{i-2},w_{i-1})}$. These calculations are performed using vectorised computations in the EM section of the code in listing 2. The posterior probabilities are computed within the variable prob, and are summed at the last step to yield the EM updates desired.

4. We continue updating until convergence is reached. This is decided by the criterion that $||\lambda_{\mathbf{1:5}}^{(\tau)} - \lambda_{\mathbf{1:5}}^{(\tau-1)}|| \leq$threshold, where the threshold is set to be 0.0001. [4]

5. In addition, I had at first written a short script that performed many random initilasations. However, it seems as though performance was not dependent on initialisation, so did not make further attempts.

6.

Appendix 1

1. Code to generate Streams of Probabilities using LPlex:

(a)
```
# Show names is in  plp
def streams(dir):
        for i in range(1, 6):
            subprocess.call(["base/bin/LPlex",
                             "-C",
                             "lib/cfgs/hlm.cfg",
                             "-s",
                             "{dir}/stream{id}".format(dir=dir, id=i),
                             "-u",
                             "-t",
                             "lms/lm{id}".format(id=i),
                             "{dir}/rescore/rescore.dat".format(dir=dir)])
```

2. EM updates:

(a)
```
def interp(streams, out_lm):
    # question 3 - initialise
    perplexity_old =1000000
    lam_prev=np.matrix([1,1,1,1,1])
    best =None
    init=[np.random.rand()*np.random.rand() for i in range(5)];
    init =[x/sum(init) for x in init ]
    lam=np.matrix(init)


    prob=np.matrix([0,0,0,0,0])
    stream=[]
# PART 1: read in the streams
    for lm in streams:
        # (3A) Get the perplexities from the streamed files
        file =open(lm,'r')
        streamed=file.readlines()
        stream.append( map(lambda s: float(s.strip()), streamed))
        file.close()
        end=len(streamed)

# EM: Untile convergence criterion met perform Em updates
    while np.linalg.norm(lam_prev-lam)>0.0001:
        stream=np.matrix(stream)
        lam_plus= np.matlib.repmat(lam.T,1,end)
        sums=lam*stream
        sums_plus=np.matlib.repmat(sums,5,1)
        probs= np.multiply(lam_plus,stream)
        probs=np.matlib.divide(probs,sums_plus)
        lam_prev=lam
        lam = np.sum(probs,axis=1).T/end

    print(lam)
```

[3]Performed in listing 2, under the headining initilaisaton.
[4]Implemented as a stopping condition of the while loop.

Provide a description of the experiments performed to examine the impact of the language model on perplexity and WER

1. The interpolated language model had a reduced perplexity compared to the best given language models. The best perplexity before interpolation was 198.6167 on the dev03 set, but after interpolation had reduced to 151.5169: which implies a reduction of 50. The results obtained are given in the listing below.

2. However, the reductions in WERs were much more modest. The best WER reduced from **18.4%** to **17.5%** after interpolation, which is a reduction in WER of just **0.9%**.

Results Listing 1

(dev03)	LM_1	LM_2	LM_3	LM_4	LM_5	LM_{int}	Improvement[5]
WER %	18.4	19.3	22.2	22.4	18.8	17.5	0.9
Perplexity	198.6167	242.7938	283.7183	337.4166	220.4432	151.5169	43

Show-Specific Language Model Adaptation

Provide a description of how you ran the unsupervised adaptation of the language model and how it impacted perplexity and word error rate. You should discuss in detail the advantages and limitations of this form of language model adaptation

Producing an interpolated language model LM from a large corpus of data, is a sensible approach if the test-set texts are similar in nature to one another. However, for shows that are very different in nature, like the recordings for news shows, sports commentaries or stand-up comedies, this may not be the best approach. It would be better to find interpolation weights for each show. In unsupervised language model adaptation, we do this is by finding the language model weights $\lambda^*_{\mathbf{1:5}}$ that would minimise the perplexities for the text $\mathbf{w_s}$ for a given show .

Theory

In an ideal world, we would like to find the weights $\lambda^*_{\mathbf{1:5}}$ that minimise the perplexities for the text $\mathbf{w_s}$ for a given show $\mathbf{s}$. However, we do not have $\mathbf{w_s}$; indeed the very essence of our task is to find $\mathbf{w_s}$, given the lattices available. So, we must use an estimation $\hat{\mathbf{w}}_\mathbf{s}$, in our case generated from an interpolated language model, $\mathrm{LM_{int}}$, that predicts word sequences $\hat{\mathbf{w}}_\mathbf{s}$ usign the lattices for the show s.

Advantages and Disadvantages

There are a number of advantages and disadvantages for this scheme, the main ones being outlined below.

Advantages

- (Handles different text): The chief advantage of this method is that we can bias the weights of the language models towards language models that better represent the language used in a particular show. For instance, if we had a 'comedy' show, then the estimates $\mathbf{w_s}$ would still preserve some of this information, and while performing EM on $\hat{\mathbf{w}}_\mathbf{s}$, we would find weights that were biased towards the language model for 'comedy'.

- These advantages extend beyond the different genres of news show, to anything that may effect language. For instance, if we had language models for men, for women, for different dialects etc., we could then bias the weights of our language model towards any of these different LMs/characterisations as appropriate

Disadvantages

- (Slower): Rather than fixing the language model interpolation weights in advance, we must optimise them as and when we receive the lattices for the evaluation data, producing estimates of $\hat{\mathbf{w}}_\mathbf{s}$. This may take some time, and so it may not be possible to use on the fly speech transcription that is used in applications like 'Siri'.

- (Less Data): Since we are performing language model interpolation on each show, there is far less data that can be used to evaluate language model weights. If there are 30 shows, the on average there will be 1/30th of the available data to estimate language model weights.

- (Reliability of $\hat{\mathbf{w}}_\mathbf{s}$): In general, the reliability of this scheme is very muchy dependent on the reliability of the language model that was used to generate the estimate of the text for each show, $\hat{\mathbf{w}}_\mathbf{s}$. If $\hat{\mathbf{w}}_\mathbf{s}$ has been estimated poorly, then using the text as a basis for language model interpolation will yield very poor results.
- (Dependent on language models/overfitting): Ultimately, the performance of this method is dependent on the degree of hetereogenitiy of the language models used. We also may end up overfitting to a particular language model, ultimately ignoring what language models have in common.
- (Space): The Language models will take up >N=#shows times as much space, which can be intractibly large for large data sets.

Ultimately, whether the advantages outweigh the disadvantages will be seen though the performance of the two methods on the development sets.

Appendix 2

1. Converts and MLF to a DAT file.

   (a)
   ```python
   def mlf2dat(dir):
           sentences = []
           f = open(dir+"/rescore/rescore.mlf", "r")
           output=''
           for lines in f:
               lines=lines.replace("\n", "")
               if lines.endswith('"'):
                   output=output+"<s> "
               elif lines[-1].isdigit():
                   split=lines.split()
                   output=output+split[2]+" "
               elif lines.endswith("."):
                   output=output+"</s>\n"
           f.close()
           f = open(dir+"/rescore/rescore.dat", "w")
           f.write(output)
           f.close()
   ```

2. Generates word probabilities for

   (a)
   ```python
   # Show names is in  plp
   def streams(dir):
           for i in range(1, 6):
               subprocess.call(["base/bin/LPlex",
                                "-C",
                                "lib/cfgs/hlm.cfg",
                                "-s",
                                "{dir}/stream{id}".format(dir=dir, id=i),
                                "-u",
                                "-t",
                                "lms/lm{id}".format(id=i),
                                "{dir}/rescore/rescore.dat".format(dir=dir)])
   ```

3. Merges language models using the weights supplied.

   (a)
   ```python

   subprocess.call(['base/bin/LMerge', '-C', 'lib/cfgs/hlm.cfg', '-i', str(lam[0,0]), 'lms/lm1','-i',str(lam[0,1]), 'lms/lm2','-i', str(lam[0,2]), 'lms/lm3', '-i', str(lam[0,3]), 'lms/lm4','lib/wlists/train.lst', 'lms/lm5', out_lm])
   ```

4. Rescores the lattices using the language models given, and then scores these lattices using score.sh.

   (a)
   ```python
   for show in SHOWSET:
       testLM(devset,'lm_own/{show}')

   #  Function definition, after rescoring lattices we also test, hence the name. scores lattices based on set with the llanguage model given, puts the folders in
   def testLM(set,lm):
       for show in FILELIST[set]:
               os.system('rm -rf plp-test/{show}'.format(show=show, folder=folder))
               os.system('./scripts/lmrescore.sh -INSPEN -4.0 -LMSCALE 12.0 {show} lattices decode {lm} plp-test FALSE'.format(lm=lm,show=show,folder=folder).format(show=show,folder=folder))
       h.qwait()
       subprocess.call(['./scripts/score.sh','plp-test', set, 'rescore'])
   ```

5. Show specific Language Model Interpolation

(a)
```


#5 :Rescore the lattices according to the new language model and then generate the 1 best list
#  Generate an interpolated model to minimise perplexity over the streams
# ############################################################################################################
    for show in FILELIST[evalset]:
        # Remove directories if they already exist
        os.system('rm -rf plp-tglm_int/{show}'.format(show=show)) # remove show if it exists
        #(1) Rescore lattices with the interpolated language model, producing a 1 best list
        os.system('./scripts/lmrescore.sh {show} lattices decode lm_int plp-tglm_int FALSE'.format(show=show))
    h.qwait()
    h.mkdir('lm_own')
    for show in FILELIST[evalset]:
        dir='/home/mec68/MLSALT11/{folder}/plp-tglm_int/{show}'.format(folder=folder,show=show)
        #(2) convert the mlfs to dat files.
        h.mlf2dat(dir)
        #(3) Generate probability streams for the show and merge the language models for each show based on
          these weights
        h.streams(dir)
        streams=["plp-tglm_int/{show}/stream{id}".format(show=show, id=i) for i in range(1,6)]
        #(4 and 5) Interpolate using Expectation Maximisation
        h.interp(streams,'lm_own/{show}'.format(show=show))
```

Implementation

To perform unsupervised adaption for the langauge model shows, it was necessary to do the following:

1. Using the interpolated language model lm_int [6], I rescore the lattices for each show, in the process producing a 1-best list mlf-file for each show.
2. The program mlf2 dat [7], compiles MLFs into .dat files. We do this for each show. Once done, the .dat file represents the best estimate $\hat{\mathbf{w}}_{\mathbf{s}}$ of the text for each show s.

   Having done this, the process is exactly the same as the Expectation Maximisation procedure described in the preceeding section, the only differences being that we use the DAT files for each show as the texts over which we will maximise perplexities.
3. The script in the 5th section of appendix 2 carries out the main ancilliary commands that enables show-specific interpolation.

Results

1. I tested the WER performance of the show-specific language models (SSLMs) using the following configurations: [development set=dev03, evaluation set=eval03], [development set=YTBEdev, evaluation set=YTBEdev]. I also tested the perplexity performance for a subset of the shows in YTBEdev. For the YTBE data-sets WER tests could only be performed on the dev-set, since scoring is not available for the unseen test-sets. The results obtained are displayed in 'results listing 2'.
2. The reductions in WERs were very small for both configuations. For both the YTBE and dev03 sets, a reduction in WER of **0.1%** was obtained.
3. Perplexities also decreased from ordinary interpolation, but only very slightly. On average only a **3 point** decrease was observed.

Given the considerable time and space requirements, (SSLMs for all YTBEshows take up 20GB), I concluded that SSLM adaptations would not be worth th considerable effort spent. The time taken to develop SS adaptations could be better spent on various other optimisations for the ASR systems.

[6]lm_int produced using EM interpolation of the language model weights as has been discussed in the first part of the practical

[7]part 1 of the appendix

Results Listing 2

WERs before (LM_INT) and after adaptation (SSLM).

eval03	SSLM	LM_INT	YTBEdev	SSLM	LM_INT
WER %	(14.9)	(15.0)	WER %	(42.5)	(42.6)

Perplexities before and after adaptation for shows in YTBEdevsub:

Show: YTBEdev	Perplexity		Show: YTBEdev	Perplexity	
	Before SSA	After SSA		Before SSA	After SSA
1	121.4396	119.9189	7	252.0078	245.9719
2	135.3942	134.3889	8	150.9248	145.7462
4	141.3958	139.3961	9	89.2615	88.4447
5	405.7591	403.9531	10	184.1107	181.9760
6	125.1491	115.0895	-	-	-

Further Explorations

When performing language model adaptations, I undertook two further investigations not described in the practical.

- Rather than use an interpolated language model to find the estimate of the transcription of a show $\hat{\mathbf{w}}_\mathbf{s}$, I decided to use more sophisticated models, incorporating acoustic information. These models would be developed in later sections. As has been previously discussed, a more accurate $\hat{\mathbf{w}}_\mathbf{s}$, should improve show-specific adaptation.
  - This resulted in a 0.1% percent improvement from the previous show-specific language model interpolation, having obtained WERs on YTBEdev of 42.4%.
- A much more significant improvement in WERs can be found by optimising for the insertion penalty and grammar scale of the language model. These can be given as arguments when rescoring lattices. Changing the grammar scale seemed to have little impact on WERs in the range []. However, for a given grammar scale, it seems WERs decreased with decreasing insertion penalties. as such, I set the insertion penalty and LM scale to -4 and 12 respectively. This gave an improvement of 0.4% from the next best settings.
  -

Insertion Penalty (Horizontal) / Grammar Scale (vertical)	**-4**	**-8**	**-12**
4	48.2	50.4	54.1
8	42.8	46.7	52.5
12	42.4	45.9	51.8

Acoustic Model Adaptation

Provide a description of the experiments performed to examine the impact of acoustic model adaptation and configurations investigated; a discussion of the cross-adaptation experiments run.

Background

An important stage in ASR is adaptation to a particular speaker, or acoustic environment. The first sep of this process is to segment the data for the audio streams into homogeneous blocks, i.e. blocks that contain data from a single speaker or environmental condition. These blocks are then clustered together so that mean transformations can be applied to fairly homogenous groupings of speech. This clustering was performed in advance of our work on the practical, and the 'f_names' in the file-lists for shows contain the relevant cluster labels.

Once the data has been clustered together, linear transforms need to be estimated for each cluster. Two of the three forms of linear adaptation scheme are given below, both of which are standard transformations for Gaussian Mixture Model systems.

- Linear transformations, which take the general form:

$$\hat{\mu} = \mathbf{A}\mu + \mathbf{b}$$

- Constrained linear mean transforms, which take the form:

$$\hat{\mu} = \mathbf{H}\mu + \mathbf{b};\ \hat{\mathbf{\Sigma}} = \mathbf{H}\mathbf{\Sigma}\mathbf{H}^{'}$$

Implementation

There are essentially two steps to the implementation. They are given below.

1. Before performing acoustic model adaptation, it is necessary to first determinise the lattices for the different models. These are then rescored with the original acoustic models.
2. Using the 1-best hypothesis generated from the determinised bigram lattice, we then produce "cascaded" CMLLR and MLLR transforms. The adaptation script, 'hmmadapt.sh ' estimates transfomations using HERest. These transformations can also be generated using supervision from a different system, under cross-adaptation. where, a different accoustic model estimates the transforms during HERest.

There are five models supplied: plp, grph-plp, tandem, grph-tandem, hybrid.

In this practical, I trained models for every kind of acoustic system possible, cross-adapted or otherwise. But, I will first discuss the results obtained without using cross-adapation, before I move on to describe the cross-adaptated systems.

The code that implements the steps described above is contained within appendix 2, where a fuller step-by-step description of what the scripts do, accompanies the script.

Results Without Cross Adaptation

The WERs for the non-cross adapted systems are given below. The hybrid system performed best with a WER of **(37.9%)**, which was an improvement of **(4.6 %)** on the language models without any special acoustic information (only used plp-features).

	plp	grph-plp	tandem	grph-tandem	hybrid
WER %	42.9	42.6	41.4	40.8	37.9

Results With Cross Adaptation

To test all possible adaptations over all viable models, the following settings are used in my program:

- ADAPTEES =['plp', 'tandem', 'grph-tandem'] # only these models can be supervised.
- ADAPTORS = ['grph-tandem', 'hybrid','tandem']# only these models can supervise.

The best models are displayed below:

	hybrid adapts tandem	tandem adapts plp	grph-tandem adapts plp	hybrid adapts plp
WER %	37.8	38.2	38.0	37.5

The hybrid system uses DNN features, and tended to out-perform the other GMM-based systems. In modern times, it has often been found that ASR systems with well-trained DNN features have outperformed more conventional GMM systems. However, the tandem system which used GMM features was almost as good an adaptor as the hybrid system.

The grph-tandem system was only featured as a system in one of the four best models, and this system performed the worst of the four. Perhaps this is because grph-tandem uses graphemic features, which might in general be worse features for speech-recognition tasks than phonemes , which are a much more natural unit of speech.

I decided to use the four best systems above as the base systems for combination in the next section, although with hind-sight it may have been a good idea to also include the hybrid model as well, since it performs just as well. Ultimately, the best system found was the hybrid-adapts-plp model, which achieved a WER of **37.5%**, and improved on hybrid's score by **0.4%.**

Results Listing 3

Displays the WERs for all forms of adaptation, WER bracketed

```
plpadaptedplp
| Sum/Avg            |    9    4333 | 60.8    22.6    16.6     1.4    (40.6)   100.0 |  -0.135 |
grph-plpadaptedplp
| Sum/Avg            |    9    4333 | 62.0    21.4    16.6     1.4    (39.3)   100.0 |  -0.164 |
tandemadaptedplp
| Sum/Avg            |    9    4333 | 63.4    22.3    14.4     1.5    (38.2)   100.0 |  -0.156 |
grph-tandemadaptedplp
| Sum/Avg            |    9    4333 | 63.4    22.0    14.7     1.4    (38.0)   100.0 |  -0.163 |
hybridadaptedplp
| Sum/Avg            |    9    4333 | 63.9    20.3    15.8     1.4    (37.5)   100.0 |  -0.194 |
plpadaptedtandem
| Sum/Avg            |    9    4333 | 62.9    22.6    14.5     1.7    (38.8)   100.0 |  -0.145 |
grph-plpadaptedtandem
| Sum/Avg            |    9    4333 | 62.6    22.4    15.0     1.8    (39.2)   100.0 |  -0.144 |
tandemadaptedtandem
| Sum/Avg            |    9    4333 | 61.3    26.2    12.5     2.1    (40.8)   100.0 |  -0.088 |
grph-tandemadaptedtandem
| Sum/Avg            |    9    4333 | 62.8    23.7    13.4     2.0    (39.2)   100.0 |  -0.124 |
hybridadaptedtandem
| Sum/Avg            |    9    4333 | 63.9    21.3    14.8     1.7    (37.8)   100.0 |  -0.173 |
plpadaptedgrph-tandem
| Sum/Avg            |    9    4333 | 62.0    22.5    15.5     1.7    (39.7)   100.0 |  -0.141 |
grph-plpadaptedgrph-tandem
| Sum/Avg            |    9    4333 | 61.5    22.3    16.2     1.9    (40.4)   100.0 |  -0.138 |
tandemadaptedgrph-tandem
| Sum/Avg            |    9    4333 | 62.0    24.7    13.3     2.2    (40.2)   100.0 |  -0.107 |
grph-tandemadaptedgrph-tandem
| Sum/Avg            |    9    4333 | 61.5    24.4    14.1     2.1    (40.6)   100.0 |  -0.107 |
hybridadaptedgrph-tandem
| Sum/Avg            |    9    4333 | 63.3    20.8    15.9     1.8    (38.6)   100.0 |  -0.173 |
```

Appendix 2

Determinising Lattices For Acoustic Moel Adaptation

1. I determinised using the mergelats command. This enables HMMrescore to be run on the lattices.

   (a)
   ```
   #m outputs a merge next to the decode
   def MERGELATS(show,dir):
       os.system('./scripts/mergelats.sh {show} plp-int rescore {dir}'.format(show=show,dir=dir))

   if two:
    print('MERGE')
    for show in FILELIST[showset]:
        MERGELATS(show,'plp-int')
   ```

   Up until this pont we only have one set of langauge-model lattices stored in a directory called plp-int. The next step is to rescore the lattices using the 5 different acoustic models. This will create 5 different folders, one for each model: hybrid, plp-int, graph-tandem, grph, etc. Code is displayed below:

(b)

```
# rescores merge with the different acoustic models, and puts them in their own directory...takes a while
def HMMRESCORE(show,dir,model):
    os.system('./scripts/hmmrescore.sh {show} {dir} merge {model} {model}'.format(show=show,dir=dir,model=model)
      )

if three:
 print('HMMRESCORE')
 for model in MODELS:
    for show in FILELIST[showset]:
        HMMRESCORE(show,'plp-int',model)
```

Cross-adaptation Step

1. The hypotheses from the previously scored lattices are used for adaptation. The adapring model is named adaptor in the code below, whereas the model being adapted is named 'adaptee'.

    (a)

```
#next 2 involve cross adaptation
def HMMADAPT(show, innerF,model,adaptee,outerF):
    os.system('./scripts/hmmadapt.sh -OUTPASS {innerF} {show} {model} decode {outerF} {adaptee}'.format(show=
      show,outerF=outerF,adaptee=adaptee,innerF=innerF,model=model))

if five:
 for adaptee in ADAPTEES:
    for model in ADAPTORS:
        outerF = 'systems'
        innerF = '{model}-adapts-{adaptee}'.format(adaptee=adaptee,model=model)
        for show in FILELIST[showset]:
            ADAPTRESCORE(show,outerF,adaptee,innerF)
```

2. Once this has ben done, the hypothesis transforms from the adapting systems can be used to rescore the lattices for the adaptees.

    (a)

```
if six:
    for adaptee in ADAPTEES:
        for model in ADAPTORS:
            print(model+'adapted'+adaptee)
            outerF = 'systems'
            innerF = 'decode-{model}-adapts-{adaptee}'.format(adaptee=adaptee,model=model)
            TEST(showset,outerF,innerF)
```

System Combination

1. Provide a description of your implementation of ROVER and CNC combination. This should include a description of the alignment process and cost functions used. A listing of your code should be included as an appendix in the write-up (but does not contribute to the word-count)

Rover Combination

Overview

ROVER combination aligns two word sequences, using a dynamic programming approach that aims to maximise a similarity measure (or equivalently minimse a distance measure) between the words in the word sequences. The approach I take to perform this is as give below. Please note my implementation for ROVER combination is listed in final the appendix.

- Given a list of MLFs, I designate one as the reference and combine one of the other MLFs in the list with it, using a dynamic programming procedure to maximise a similarity measure. This recombination procedure returns another combined MLF file, which is designated as the new reference. The procedure is then repeated until there is only a single reference file remaining.
- Before combining MLF, I first read in the MLF to be combined using the function readmlf(), listed in the final appendix. This function will use the class mlfEntry, described below, to create lists of these objects as the internal representation for a show.

Internal Representation

Central to my implementation of ROVER combination is a special class called 'mlfEntry', whose instantiations store the relevant information in the rows of an MLF file. The rows in an MLF represent a hypothesis for a word or groups of words, occuring at a particular time slot. As an example, if an MLF contained the row given below, it would be represented as the mlfentry below :

```
6000000 11200000 STARTED_<ALTSTART>_<DEL>_<ALTEND> 0.99_0.20
mlfentry('6000000 11200000 STARTED_<ALTSTART>_<DEL>_<ALTEND> 0.99_0.20')
```

The reason, I created such a class was to simplify the implementation of Rover recombination. Using objects offers a way to neatly separate complex procedures such as recombining MLFs using dynamic programming, reading in mlf-row information etc.

The class also contains the fields given as input and also various methods:

1. MakeAlt(S,O): This function corresponds to combining MLF rows for S and O, where S can be thought of as the reference since its start and end times will remain unchanged. In more detail, if the mlfentry O contains a word,w, that is not in S, we update the word-score mapping in S, so that it contains the mapping (w→O.score(w)) . If O contains a word,w, that already belongs to S, we add the score in O to the corresponding score in S.score(w)+=O.score(w) .
2. DPmetric(). This function finds the **similarity** between two mlf entries. The role of this function is to be understood from the greater context of combining two sequences, M,N, of MLF rows through a dynamic programming procedure. To find the reward from aliging two sequences, they must first be aligned so that we know what row in N each row in M has been paried off with. Assuming the best alignment has been found, the similarity between M and N may be found by summing the rewards, given by DPmetric(S,O) over each pairing (m,o) in the alignment[8]. Within my code, there are various similarity metrics that I implemented to align sequences.
    (a) The temporal overlap between the entries entries of two MLF files.
    (b) The total score of the 1-best list produced by the alignment of the two MLFs
    (c) A Similarity measure based on minimising the edit distances between alined wordsa show/
    (d) A weighted combination of the above metrics
3. Tostring(): This method is simply used to print an mlfentry to a row in an MLF file.

[8]Note that $m \in M \cup \{del\}$, $n \in N \cup \{del\}$, and the entries in the alignment are monotonically ordered

Dynamic Programming Alignment

Recall that before combining two MLFs, we will have generated two lists of MLFentries, say a and b, which represent the rows in the MLF for a show. The dynamic programing procedure finds the alignment between a and b that maximise the reward for alignment, and is defined recursivelty.

$$\mathrm{sim}_{a,b}(i,j) = \begin{cases} 0 & \text{if min(i,j)=0} \\ \max \left\{ \begin{array}{l} \mathrm{sim_{a,b}(i-1,j) + DPMet(a(i),\{DEL\})} \\ \mathrm{sim_{a,b}(i,j-i) + DPMet(\{DEL\},a(i))} \\ \mathrm{sim_{a,b}(i-i,j-i) + DPMet(b(j),a(i))} \end{array} \right\} & \text{otherwise} \end{cases}$$

I then programmed this formulation efficiently using dynamic programming in the code in the listing. The different similarity metrics are as given in the preceeding subsection of this report.

A further point to mention is that the dynamic programming algorithm may align a word in the reference text, a, to no word in b. When this occurs, we say that the word in a has been aligned to a 'deletion', which we give a score γ. If γ exceeds the score of the various alternatives in an MLF entry, then this row will not be printed at all, when we finally output the mlf.

Combination

Combination simply involves taking each mlf-entry from the 'combined' mlf-list in turn- and, for each, printing the word with the highest score contained within the mlf-entry. An MLF-entry may contain many alternative words, and so the highest scorign word must be output. This produces a one-best list of words ordered by their start and end times. This 1-best list MLF can then be scored.

When performing ROVER combination, a further parameter that controls for the weighting assigned to each of the MLFs can optionally be supplied . So if the paramaters were $\alpha = [0.1, 0.4, 2]$ and we wanted to combine 3 MLFs, then the scores of rows in each of the three MLFs would be multiplied by 0.1, 0.4 and 2 respectively before being combined. [9]

CNC Combination:

CNC recombination follows almost exactly the same scheme as ROVER. The only difference is that some extra preprocessing is required. It is necessary to first compile the CNC lattices into a single MLF file with alternatives. Given the format of CNCs , this is not a demaniding task. I have written a script that perfoms this task, which is listed in the final appendix.

There are CNC files for each show, which take the form as given below:

```
N=9
k=1
W=</s> s=2.45 e=3.51 p=0.00000
k=2
W=NEWS s=1.45 e=2.45 p=-0.00000
W=!NULL s=1.45 e=2.45 p=-29.80533
k=2
W=C. s=1.27 e=1.45 p=-0.00000
W=!NULL s=1.27 e=1.45 p=-18.54744
k=3
```

The number k, denotes the arc number, with each edge of an arc containing start-time, end-time and score information.

To generate an MLF file, the script reads in all of the CNC-files for a show, and then converts each arc to a row in the MLF that is being generated. CNC-file represents a recording for a show, and so care must be taken to group the rows of an MLF together according to the requisite recording.

The mapping from an arc to a row is a fairly simple one, the only contentious issue is the assigning start and end times for an arc before it gets mapped to a row in th MLF. This is because most words belonging to an arc will have start and end times that are slightly different from one another. To resolve this issue, I simply averaged over the start and end times for an arc.

Once the conversion has taken place, the newly generated MLFs can be combined just as is performed in ROVER combination.

Other Implementations- Confidence Score Remapping

I performed confidence mappings on CNCs because the default confidence scores are typically too high and not amenable to combination. To address this problem the confidence scores can be mapped to better reflect the probability of a word being correct. MLFs could be rescored using commands like:

[9]**The code for this implementation is given in the appendix at the end of this report.**

```
./scripts/score.sh -CONFTREE lib/trees/plp-bg_decode_cn.tree plp-bg dev03sub decode_cn
```

However, the command would not rescore MLFs with multiple entries and to do this it was necessary to amend **the Perl script that would apply** confidence score remappings to lattices, smoothtree-mlf.pl, which is stored in the base/conftools directory. The script to do this is listed in the appendix. Once this had been done, it would be possible to score MLFs generated from CNCs.

Results- ROVER and CNC combination

Provide a description of the experiments performed to examine the impact system combination.

Given my implementations of CNC and ROVER combination described above, it was possible to tune the following parameters/ configurations of the recombination scheme:

1. CNC vs Rover recombination
2. Whether confidence tree mapping improves CNC combination
3. Tuning the weights for the MLFs to be combined
4. Tuning the paramaters of the dynamic programming metric, which controls for the weights of the different similarity metrics used
5. Optimising the 'deletion' scores, which tells us how likely it is a misaligned MLF entry ought to be deleted.

The results are discussed below, and all formal results are given in the results listing. I chose to combine the 4 highest scoring cross-adapted systems. These systems contained models with both graphemic and phonetic features and so hopefully offered a good diversity

1. ROVER combination outperformed CNC combination by precisely **0.7%** reducing WERs to **37.3%** from the best adapted acoustic system, whereas CNC also improved overall WERs by **0.2%** from the next best hybrid-adapted system.
2. Confidence tree mapping did not substantially improve scores after CNC combination. I tried three variations of confidence score mapping. The first used the hybrid-tree scores, since this was the best simple acoustic system. The other variations used confidence tree mappings using the trees for 'adapatee' systems and the trees for 'adaptor' systems of the models to be combined.
   (a) Neither of the remappings improved performance. Using the trees for the underlying models X resulted in a 0.1% increase in WERs, whereas using the supervising models Y and also the hybrid trees did not change WERs from the baseline CNC system, which performed worse than ROVER combination.
3. Having decided that ROVER was the best system for recombination, the next step was to tune the weights for system recombination. To do this, I used a greedy optimisation procedure, which finds the optimal weights for combining 2 systems, and then the optimal weights for the next 2 systems given that the reference for the next combination is the newly combined system, and so on. [10]. The progress the algorithm makes during recombination is given in the results listing. After the greedy optimisation, the WER for the combined systems, for this special kind of sequential optimisation had dropped to 36.6%, which represented a further **0.7%** reduciton in WERs
4. Next, I tuned the weights for the dynamic programming metric. The best dynamic programming metric to use metric was the one that maximised the score of the MLF output. In this case: DPmetric(s,o)= highest score of a word in the combined mlf makealt(s,o). As these settings were no different from those used in the preceeding sections, no further WER reductions were found.
5. Next, I optimised for 'deletion' scores. I discovered that an insertion penalty/ deletion score of 1 was favoured. This is not to disimilar to that suggested in the lecture notes, which advised for a delection score of 0.2 for a single combined system. As I had comed 4 systems, this leads to an effective insertion penalty of **0.25%** (1/4).
6. There was a final additional source of improvement in WERs, that was found by aligning all four of the systems at once, rather than sequentially aligning two MLFs at a time. The reason that the sequential alignment performs worse than an all-at-once alignment is that a sequential alignment discards alternative words early on when combining two MLFs and outputting to a single MLF. Combining MLFs all at once resulted in a **0.3%** improvement to a WER of 36.3%.

[10] **The code that does this is listed in the appendix**

Finally, I tried the various scoring options that were available. The results are tabulated below:

(WERs %)	Without -NOFILT	-NOFILT
YTBEdev		35.2
YTBEdev-v2	33.9	**32.4**

The bst scoring result obtained was 32.4%

Results-Listing-WERs bracketed

1. The results for CNC and ROVER combination are given below. ROVER obtained an imoprovement of **0.7**% on the best previous system, whereas CNC had a **0.2**% improvement.

   (a)
```
CNC combination

| Sum/Avg             |    78   26336 | 65.8    16.1    18.1     3.8    (37.9)   100.0  |  -5.887  |

ROVER combination
| Sum/Avg             |    78   26336 | 66.6    17.8    15.6     3.9    (37.3)   100.0  |  -5.946  |
```

2. Confidence Tree Mapping After CNC combination.

   (a)
```
hybrid treee confidence remapping on the weighted alphas
| Sum/Avg             |    78   26336 | 65.6    15.5    18.9     3.5    (38.0)   100.0  |  -5.786  |
adaptee confusion rescorings
| Sum/Avg             |    78   26336 | 65.4    15.5    19.1     3.5    (38.1)   100.0  |  -5.750  |

adapted confusion rescorings
| Sum/Avg             |    78   26336 | 65.6    15.6    18.8     3.6    (38.0)   100.0  |  -5.811  |
```

3. Greedy optimisation procedure to tune the weights of recombined systems. After optimisation, the WER for sequenctial combination had dropped to (**36.6**)%. The alphas in the code below are the combination weights.

```
Alpha_1=1

Combining first two systems:
ALPHA_2=2.5
| Sum/Avg             |    78   26336 | 66.6    17.8    15.6     3.9    (37.3)   100.0  |  -5.946  |
ALPHA_2=2
| Sum/Avg             |    78   26336 | 66.6    17.7    15.6     3.9    (37.3)   100.0  |  -5.745  |
ALPHA_2=1.8
| Sum/Avg             |    78   26336 | 66.7    17.7    15.6     3.9    (37.2)   100.0  |  -5.586  |
ALPHA_2=0.9
| Sum/Avg             |    78   26336 | 66.9    17.5    15.5     3.9    (36.9)   100.0  |  -4.424  |

BEST ALPHAs=[1,0.9]
Combining with third system:

ALPHA_3=2.5
| Sum/Avg             |    78   26336 | 66.5    17.6    15.9     3.8    (37.4)   100.0  |  -5.963  |
ALPHA_3=0.9
| Sum/Avg             |    78   26336 | 66.6    17.5    15.9     3.9    (37.3)   100.0  |  -5.814  |
ALPHA_3=1.8
| Sum/Avg             |    78   26336 | 66.7    17.5    15.9     3.8    (37.2)   100.0  |  -5.738  |
ALPHA_3=0.9
| Sum/Avg             |    78   26336 | 66.9    17.5    15.6     3.9    (36.9)   100.0  |  -5.232  |


BEST ALPHAs=[1,0.9, 0.9]
Combinign with third system:

ALPHA_4=2.5
| Sum/Avg             |    78   26336 | 66.8    16.5    16.7     3.6    (36.8)   100.0  |  -5.881  |
ALPHA_4=2
| Sum/Avg             |    78   26336 | 67.0    16.5    16.5     3.6    (36.6)   100.0  |  -5.854  |
ALPHA_4=1.8
| Sum/Avg             |    78   26336 | 67.1    16.6    16.4     3.7    (36.6)   100.0  |  -5.853  |
ALPHA_4=0.9
| Sum/Avg             |    78   26336 | 67.0    17.2    15.8     3.9    (36.9)   100.0  |  -5.557  |

Final paramaters found
BEST ALPHAs=[1,0.9, 0.9, 2]
```

4. I discovered that an insertion penalty of 1, the default setting, was favoured and so omit further results.

5. I similarly discovered the best DP metric was to use the DPmetric which output the alignment with highest score, again the default setting. As such results are omitted.

6. Final score, after aligning all at once rather than sequentially.

   (a)
   ```
   | Sum/Avg              |    78   26336 | 67.6    16.4    16.1     3.7    (36.3)   100.0 |  -5.752 |
   ```

7. Further scoring results for: YTBEdev without the -NOFILT option; YTBEdev-v2; and YTBEdev-v2 with the -NOFILT option:

```
h -NOFILT testground2 YTBEdev decode
| Sum/Avg              |    78   26219 | 66.7    16.8    16.5     1.9    (35.2)   100.0 |  -5.690 |
mec68@mlsalt-cpu1:/remote/mlsalt-2015/mec68/MLSALT11/attempt3$ ./scripts/score.sh testground2 YTBEdev-v2 decode
| Sum/Avg              |    75   22363 | 70.0    16.7    13.3     3.8    (33.9)   100.0 |  -5.807 |
mec68@mlsalt-cpu1:/remote/mlsalt-2015/mec68/MLSALT11/attempt3$ ./scripts/score.sh -NOFILT testground2 YTBEdev-v2 decode
| Sum/Avg              |    75   22281 | 69.5    16.7    13.8     1.9    (32.4)   100.0 |  -5.574 |
```

8. The WER of 32.4% was the best of any system found in this practical.

Evaluation System Development

Discuss the process that you have used to develop the evaluation system. Don't forget to include "negative" results that guided your design choice. You should also discuss any changes made to your evaluation system design for the Challenge data.

In this section I will give a summary of the important decisions taken and also the various limitations imposed by this practical. When performing language model adaptation for the evaluation set:

- I interpolated language model weights based on the simple EM scheme. I did not use show-specific LMs. The main reason for this was due to time and compute resources. Producing the language models for each show in the evaluation set would have required upwards of 10GB's worth of data, and taken an hour in compute time. However, the space constraints were by far the greater of the two challenges, since our personal disk quota usage is 20 GBs for all courses. In addition, doing this would have only resulted in a (0.1%) decrease in WERs.
- While experimenting on the development set, I investigated a slightly different method for producing SSAs. Recall from the first section of the practical that we need a first approximation to the texts in the shows for s, $\hat{\mathbf{w}}_\mathbf{s}$, before finding the LM interpolation weights for that estimate. The practical instructions suggested generating $\hat{\mathbf{w}}_\mathbf{s}$ from an interpolated language model. As an optimisation, I considered generating $\hat{\mathbf{w}}_\mathbf{s}$, using lattices with acoustic information from the 'hybrid' system, developed in the second stage of the practical, which had improved WERs and performed better than the interpolated language model. When testing on the development set this optimisation only resulted in modest (0.1%) decreases in WERs from standard SSA. As such, this optimisation was neglected in the final system.
- The best optimisation on the dev set came from tuning the insertion penalties and grammar scales, which resulted in a (0.3%) reduction in WER. I decided to use the same parameter found on the DEV set on the evaluation set (INS=-4, GS=12).This optimisation required no further compute time or run-time space, when developing the evaluation system, since the optimisations had been done beforehand.

Acoustic Model Adaptation:

- I performed accoustic model adaptations, doing exactly what has been described in the previous sections. This was the most straight-forward part of the practical, since we had already determined the 4 best cross-adpated systems on the development set: [tandem adapts plp, grph-tandem adapts plp, hybrid adapts plp]. Instead of training every permutation of adapting-models and adaptee-models, it was only necessary to train these 4 cross-adapted systems, and this was relatively fast (around 1 hour).
- It would have been possible to tune the LMscale and INS penalty for the HMMrescore command, which is called twice for each show, when developing cross-adapted systems. It would have been possible to optimise for insertion penalty for HMMrescore, however this would need to be optimised for each of acoustic model used, and as HMMrescore took much longer to optimise for than the LMrescore for a given lattice, I decided to omit this optimisation both for the dev and eval set data .

System Combination:

- When first training on the development set, I first combined the 5-best models during system combination. However, the 5th best system had a WER (0.6%) worse than the 4th worst model, which was almost as bad as the difference in WERs between the best and the worst model. As such, I decided to omit the corresponding system from recombination when producing the final evaluation system.
- I then combined the various systems, in a weighted manner, using the greedy optimisation procedure described in the ROVER section of the report. This method of combination did not impose a significant deal of time constraints, being polynomial rather than exponential in the #weights for a system at each stage of the procedure. If we had undertaken a grid-search optimisation procedure this would have been exponential and taken significantly more time. I found that a greedy optimisation procedure was also better than a grid-search at finding the optimal parameter settings for the development sets, given a certain amount of allotted time. Other parameter settings were kept, as had been informed by the testing on the development set. The insertion penalty (deletion score) was set to 1 and the weightings for the DPMetric were left unadjusted, with all weight on the 'maximum-score' metric. These introduced no further computational overheads.

This section should finish with a discussion of the impact of the limitation that the practical infrastructure has imposed:

1. In general lattice rescoring and decoding are slow procedures. This is mainly due to the sheer size of trigram and higher order lattices, which can exceed many GBs of data. The language models below comprise part of the infrastructure provided for this ASR task.

	LM1	LM2	LM3	LM4	LM5
Size (Mb)	22	17	9.3	8.2	39
Trained on (Mwords)	275	66	2	2	674

1. In a sense, having both trigram and bigram language models frees us from some of limitations that would be imposed by only having access to either of the two models. Trigram and higher-order language models result in a highly complex search-space, which makes rescoring or finding 1-best lists a difficult task. In contrast, bigram language models may not have a sufficient representational capacity to model complex language. A compromise can be made by generating lattices with a simpler system, and then apply more complex models to these lattices. Despite this compromise, even rescoring smaller lattices proved to be a computationally expensive and time-consuming task, which imposed limitations on testing, development in this practical.

2. Typically, pruned lattices are used to reduce the decoding speed, but here there is a down-side: since only a subset of paths are contained within a lattice, the correct path may not be contained within the lattice. In this case, however good the new language (or acoustic) model is, the correct path can never be recovered. Throughout this practical, we made extensive use of pruning when rescoring, and also during acoustic model adaptation under LPMerge- which will tend to place limitations on accuracy.

3. LM Interpolation led to reductions in WERs, but at the same time increased LM sizes by a factor much larger than the reduction in error rate. The average interpolated LM had a size of 70M, despite the fact the systems it was combined from had a median size of 17M. These costs were additive, when performing show-specific LM interpolation, which led to very large lattices being generated. This was despite the fact that many of the parameters values in these interpolated LMs were very similar. Perhaps a distributed scheme for LM interpolation could be developed, where parameters are to be shared in the network. Alternatively, shows could be clustered together, and LM interpolations could have been performed for a cluster of shows rather than every show.

4. Having said this, in order to keep the disk-usage associated with storing the lattices to a minimum the HTK does support the filters to compress, and decompress, lattices on the fly. The filters were specified in the configuration files, and were used to minimise the size of associated Lattices for language models, which had the special ending .gz ; Despite this, storage requirements were still rather large.

5. In the practical, we only had access to 5 language models, none of which had been trained on data from Youtube. Using language models trained on data from a website like youtube, would possibly lead to improvements in performance.

6. I found that when performing CNC recombination, the approximation of a lattice to a CNC was not as good as it could have been. Often, it seemed words, with very different start and end times were aligned to the same arc, which should not be the case.

7. There were various HTK tools used in this practical, which supported various operations on lattices. For instance, HLRescore reads lattices in standard lattice format and can: find 1-best path through lattice, which allows language model scale factors and insertion penalties to be optimised. Implementing this as an option within LMrescore or HMM-rescore, may have facilitated the optimising of insertion penalities and grammar scales during the practical. It would possibly be good to be able to control for how aggressive the pruning is for a given lattice, so there would be more freedom to select for time or accuracy.

Final Appendix:

1) MLF combination script, used for CNC and ROVER. Contains MLFentry class.

```
import os
import collections as col
import numpy as np
import sys
limit = 1;
DPmetWts=[1,0,0];
DELVAL=1;
thresh=0;

#########################################################################
class mlfEntry():
    t1,t2=None,None; woSc=col.OrderedDict(); scWt=1;

    def __init__(s,line,scWt):
        s.scWt=scWt;
        entries=line.split(' ');
        s.t1=int(entries[0]); s.t2=int(entries[1]);words=entries[2]; scores=entries[3];
        Wo=[x for x in words.replace('<ALTSTART>','').replace('<ALT>','').replace('<ALTEND>','').split('_') if x]
        Sc=[float(x)*s.scWt for x in scores.split('_')]
        s.woSc=col.OrderedDict( zip(Wo,Sc))

    def toSTR(s):
        #s.woSc['DEL']=-1000; s.woSc['<DEL>']=-1000
        words=s.woSc.keys();
        scores=s.woSc.values();

        # If there are more than a certain number of words
        word_x = [x for (y,x) in sorted(zip(scores,words))  ]
        score_x = [y for (y,x) in sorted(zip(scores,words))]
        word_x.reverse();score_x.reverse()
        words = word_x[:limit]
        scores = score_x[:limit]

        # Merge all together
        if len(words)>=2:
            # If there is more than one word, merge them with the given way
            wordOut=words[0]+'_<ALTSTART>_'+'_<ALT>_'.join(words[1:])+'_<ALTEND>'
            scoreOut=str(scores[0])+'_'+'_'.join(str(x) for x in scores[1:])
        else:
            # If there is only one word, then just take that
            wordOut=words[0]
            scoreOut=str(scores[0])
        if words[0]=='<DEL>'or words[0]=='DEL' or scores[0]<thresh:
            return ''

        return ' '.join([str(s.t1),str(s.t2),wordOut,scoreOut])


    def DPmetric(s,o):
        return DPmetWts[0]*s.maxProb(o)+DPmetWts[1]*s.EDMETRIC(o)+DPmetWts[2]*s.oldmetric(o)

    def EDMETRIC(s,o):
        # ovrlap betwen the start and end times
        [a,b,c,d]=sorted([(s.t1,0),(s.t2,0),(o.t1,1),(o.t2,1)])
        overlap= abs(a[1]-b[1])*(c[0]-b[0])
        # print(overlap)
        s_word=max(s.woSc, key=s.woSc.get); o_word=max(o.woSc, key=o.woSc.get)
        if s_word=='<DEL>' or s_word=='DEL':
            ed_score= len(o_word)
        else:
            ed_score=h.edDistRecursiveMemo(s_word,o_word,memo=None)
        return -overlap*ed_score/(max((s.t2-s.t1),(o.t2-o.t1))*max(len(s.woSc),len(o.woSc)))


    def intersections(s,o):
        z= (len(set(s.woSc.keys()).intersection(o.woSc.keys())))
        return z/(max(len(s.woSc.keys()),len(o.woSc.keys())))

    def time(s,o):
        # ovrlap betwen the start and end times
        [a,b,c,d]=sorted([(s.t1,0),(s.t2,0),(o.t1,1),(o.t2,1)])
        overlap= abs(a[1]-b[1])*(c[0]-b[0])
        return overlap/max((s.t2-s.t1),(o.t2-o.t1))

    def maxProb(s,o):
        stuff=[s.woSc[x]+o.woSc[x] for x in set(s.woSc.keys()).intersection(o.woSc.keys())]+s.woSc.values()+o.woSc.values
          ()
        return max(stuff)/2

    def makeAlt(s,o):
        for word in o.woSc.keys():
            if s.woSc.has_key(word):
```

```
                s.woSc[word]+=o.woSc[word]
            else:
                s.woSc[word]=o.woSc[word]
    def oldmetric(s,o):
        # ovrlap betwen the start and end times
        [a,b,c,d]=sorted([(s.t1,0),(s.t2,0),(o.t1,1),(o.t2,1)])
        overlap= abs(a[1]-b[1])*(c[0]-b[0])/10000000
        #print(overlap)
        s_word=max(s.woSc, key=s.woSc.get); o_word=max(o.woSc, key=o.woSc.get)
        '''if s_word=='<DEL>' or s_word=='DEL':
            ed_score= len(o_word);    z=0
        else: '''
        ed_score=edDistRecursiveMemo(s_word,o_word,memo=None)
        stuff=sorted([s.woSc[x]+o.woSc[x] for x in set(s.woSc.keys()).intersection(o.woSc.keys())])
        stuff=2*sum(stuff)+len(stuff)
        z= (len(set(s.woSc.keys()).intersection(o.woSc.keys())))
        z=stuff
        #print(o.woSc.keys())

        return z
###############################################################################################



######################################## Functions ##############################################
# creates MLF mapping
def readmlf(inputFile,scWt):
    mlf = {}
    f= open(inputFile, 'r')
    for line_ in f:
        line=line_.strip('\"/\n*')
        if line[0].isalpha():
            lattice = line.split('/')[-1]
            mlf[lattice] = [None]
        elif line[0].isdigit():
            Entry=mlfEntry(line,scWt)
            mlf[lattice].append(Entry)
    f.close()
    return mlf
###############################################################################################


# alphas are the weightings for the scores, a1,b1,c1 are the metric weights, delval is the delval
def main(outputFile, systems, scoreWeights,DPmet=[1,0,0],delVal=1,threshold=0):
    ############################################ MERGING ##########################################
    # Opening the reference files and initialising the newMLF file
    global DPmetWts
    DPmetWts=DPmet
    global thresh
    thresh=threshold

    refString=systems[0]; othString=systems[1:]; othSCwe=scoreWeights[1:]
    ref_mlf=readmlf(refString,scoreWeights[0]);newMLF={}

    # Looping over the new files
    for oth,scWe in zip(othString,othSCwe):
        # Opening the file paths
        oth_mlf=readmlf(oth,scWe)

        #  Taking each lattice from the reference mlf file
        for lattice in ref_mlf.keys():
            A=None;B=None;
            #  Taking the same lattice from the other file
            if oth_mlf.has_key(lattice):
                A=ref_mlf[lattice]; B=oth_mlf[lattice];
                #   swap so longest is ref
                #if (len(B)>len(A)):
                #    dummy=B; B=A; A=dummy;
                lenA = len(A); lenB = len(B);
                rwdMatrix = np.zeros(shape=[lenA,lenB])
                align = [[None for q in range(lenB)]for m in range(lenA) ]

                # Find Edit Distances
                for a in range(1,lenA):
                    for b in range(1,lenB):
                        Rwd=A[a].DPmetric(B[b])
                        possible =[rwdMatrix[a,b-1],rwdMatrix[a-1,b],rwdMatrix[a-1,b-1]+Rwd]
                        rwdMatrix[a,b]=max(possible);
                        DELMLFEnt= mlfEntry('1 1 <DEL> '+str(delVal),B[b].scWt)
                        instruct={0:['donothing',a,b-1],1:[DELMLFEnt,a-1,b],2:[B[b],a-1,b-1]}
                        align[a][b]=instruct[np.argmax(possible)]

                # Initialising the new lattice
                newMLF[lattice]=[]

                # Recombine to Ref
                a=lenA-1; b=lenB-1;
                while align[a][b]!=None:
```

```
                    [z,a1,b1]=align[a][b]
                    if z!='donothing':
                        A[a].makeAlt(z)
                    a=a1; b=b1 ;
                newMLF[lattice]=A
        ref_mlf = newMLF
    #################################################################################################

    ########################################### output ##############################################
    output = '#!MLF!#\n'
    for lattice in newMLF.keys():
        output+='"'+lattice+'"\n'
        for ent in newMLF[lattice]:
            if ent!=None and not ent.toSTR()=='':
                output+=ent.toSTR()+'\n'
        output+='.\n'
    #print(output[1:])
    ######################################   ##### Printing ################################
    if not os.path.exists(os.path.dirname(outputFile)):
        os.makedirs(os.path.dirname(outputFile))
    #os.rmdir(outputFile)
    f = open(outputFile, "w")
    f.write(output)
    f.close()
    ######################################################################################
```

2) MLF to CNC script

```
import gzip
import operator
import numpy as np
import sys
import os

# All commands packaged toghether
def main(cnc_path,output_mlf):
     lattices=getLat(cnc_path)
     lat_Dict=latDict(lattices)
     out=generate_MLF(lat_Dict)
     printTofile(out,output_mlf)

# Retrieve Lattices
def getLat(cnc_path):
    gz_fil = [f for f in os.listdir(cnc_path) if f.endswith('.gz')]
    for _ in gz_fil:
        print _

    lattices=[]
    for gz in gz_fil:

        lattices.append(["F="+gz])
        f = gzip.open(cnc_path+gz, 'r')
        for a_line in f:
            line = a_line.split()
            lattices.append(line)

    return lattices


# Main body of cnc to MLF
def latDict(lattices):

    lat_dict = {}
    k = -1
    first = False
    # Loop over lattices and print each
    for line in lattices:

        if line[0][0] == 'F':
            f_name = line[0].strip("F=")[:-6] + 'rec'
            lat_dict[f_name] = {}

        if line[0][0] == 'k':

            k = int(line[0].split('=')[1])
            vertext_set = []
            first = True

        if k > 0 and line[0][0] == 'W':
            entries = []
            for entry in line:
                entries.append(entry.split('=')[1])
            vertext_set.append(entries)
            k -= 1

```

```
        if k == 0:
            logProbs = [float(e[3]) for e in vertext_set]
            max_index, _ = max(enumerate(logProbs), key=operator.itemgetter(1))
            startTime = float(vertext_set[max_index][1])

            end_time = float(vertext_set[max_index][2])

            lat_dict[f_name][startTime] = {'end_time': end_time, 'entries': []}
            for entries in vertext_set:
                token = entries[0]

                logProb = float(entries[3])


                lat_dict[f_name][startTime]['entries'].append({'token': token,
                                                                      'logProb': logProb})
    return lat_dict

# create a dict for the entries in a given lattice with a given name -lots of looping
def generate_MLF(lat_dict):
    out = '#!MLF!#\n'
    for f_name in lat_dict.keys():

        out += '"*/{f_name}"\n'.format(f_name=f_name)




        # Loop by start time
        for startTime in sorted(lat_dict[f_name].keys()):

            entries = lat_dict[f_name][startTime]['entries']

            entries = sorted(entries, key=lambda x: x['token'], reverse=True)
             # output start of a sentence
            if entries[0]['token'] != '<s>' and entries[0]['token'] != '</s>':
                startTimeString = '{0:.7f}'.format(startTime).replace('.', '')
                if startTimeString[0] == '0':

                    startTimeString = startTimeString[1:]
                end_timeString = '{0:.7f}'.format(lat_dict[f_name][startTime]['end_time']).replace('.', '')


                if end_timeString[0] == '0':

                    end_timeString = end_timeString[1:]

                out += '{startTime} {end_time} '.format(startTime=startTimeString,
                                                       end_time=end_timeString)
                l = len(entries)
                # Insert Null if no entry
                if entries[0]['token'] == '!NULL':

                    entries[0]['token'] = 'DEL'
                if l == 1:
                    out += '{token} '.format(token=entries[0]['token'])
                    out += '{0:.6f}'.format(np.exp(entries[0]['logProb']))
                else:
                    out += '{token1}_<ALTSTART>_'.format(token1=entries[0]['token'])
                    for i in range(1, l - 1):



                        if entries[i]['token'] == '!NULL':

                            out += 'DEL_'
                        else:
                            out += entries[i]['token'] + '_<ALT>_'

                    if entries[l-1]['token'] != '!NULL':

                        out += entries[l-1]['token']
                    else:
                        out += 'DEL_'
                    out += '<ALTEND> '
                    for i in range(l):
                        out += '{0:.6f}'.format(np.exp(entries[i]['logProb']))
                        if i != l - 1:
                            out += '_'
                out += '\n'
        out += '.\n'
    return out

# cnc to mlf main printing command
def printTofile(out,output_mlf):
    print(os.path.dirname(output_mlf))
    if not os.path.exists(os.path.dirname(output_mlf)):
        os.makedirs(os.path.dirname(output_mlf))
```

```
    fout = open(output_mlf, 'w')
    fout.write(out)
    fout.close()
```

3) Confidence Tree Mapping script for multi-Entry MLFs- Modified from Original

```
#!/usr/local/bin/perl5 -w

$tree_name=$ARGV[0];
$mlf_name=$ARGV[1];

&read_tree;

open (MLF_FILE, $mlf_name) || die "cannot open mlf file $mlf_name";
while (<MLF_FILE>) {
  $line = $_;

  if ($line =~ /^#/ ) {
    print $line;
  } elsif ($line =~ /^#/ ) {
    print $line;
  } elsif ($line =~ /^\"/ ) {
    print $line;
  } elsif ($line =~ /^\./ ) {
    print $line;
  } else {
    @f=split(/\s+/, $line);

    printf "%s %s %s", $f[0], $f[1], $f[2];

    @fs=split('_', $f[3]);
    $orig_x = $fs[0];

    $i=1;
    until ($orig_x <= $x[$i]) {
      $i=$i+1;
    }
    $dx=($orig_x - $x[$i-1]);
    $dy=$dx * $slope{$x[$i]};
    $new_y=$y_ave{$x[$i-1]}+ $dy;

    printf " %f", $new_y;

    foreach my $temp (@fs[1 .. $#fs]){
        $orig_x = $temp;

        $i=1;
        until ($orig_x <= $x[$i]) {
          $i=$i+1;
        }
        $dx=($orig_x - $x[$i-1]);
        $dy=$dx * $slope{$x[$i]};
        $new_y=$y_ave{$x[$i-1]}+ $dy;
         printf "_%f", $new_y;
    }


     printf "\n";
  }
}

exit;

sub read_tree {

  open (TREE_FILE, $tree_name) || die "cannot open tree file $tree_name";
  while (<TREE_FILE>) {
    $line = $_;
    chomp $line;
    # in the R1.2.1 the lines look like:
    # 4) confidence < 0.660969 5511  7383 F ( 0.39249 0.60751 )
    # in the R0.61 the lines look like:
    # 4) confidence<0.654153 8386 10950 F ( 0.35953 0.64047 )

    # normalise to old format
    $line =~ s/confidence\s*([<>])\s*/confidence$1/;

    @fields = split (/ +/, $line);
    #  printf "%s\n", $fields[2];
    if ( $fields[2] =~ /^confidence
     ([<>])
     ([\d.]+)
     $/x ) {

      if ($1 eq "<") {
```

```
    push(@x, $2);
    if ($fields[$#fields] eq "*") {
      $y{$2}=$fields[7];
    }
      }
      elsif ($1 eq ">") {  #  ">"
    if ($fields[$#fields] eq "*") {
      $next_y{$2}=$fields[7];
    }
      }
      else {
        die "parse error! in line $line";
      }
    }
    else {  # root node
      #    print "skipped: |$line|\n";
    }
  }

  push (@x, 0.0);
  $y{0.0}=0.0;
  push (@x, 1.0);

  @x=sort (@x);

  $prev_i=0.0;
  foreach $i (@x) {
    if (!defined ($y{$i})) {
      $y{$i}=$next_y{$prev_i};
    }

    $prev_i=$i;
  }

  $x1=0.0;
  foreach $x2 (@x) {
    if ($x2>0.0) {

      $y_ave{$x1}= ($y{$x2}+$y{$x1})/2;
      $x1=$x2;
    }
  }
  $y_ave{1.0}=($y{1.0}+1.0)/2;

  # calc grad for each piece and store in at right border x-coord
  $x1=0.0;
  foreach $x2 (@x) {
    if ($x2>0.0) {
      $slope{$x2}= ($y_ave{$x2} - $y_ave{$x1}) /($x2 - $x1);
      $x1=$x2;
    }
  }

  close (TREE_FILE) || die "cannot close tree file";
}
```

4) Paramater Tuning Script for DPmetric weights, MLF weights and Insertion Penalty/Delection Scores

```
mport time, shutil


########################################################################
# setup directories
folder='attempt3'
dir='/home/mec68/MLSALT11/{folder}'.format(folder=folder)
os.chdir(dir)
FILELIST = h.readFiles(); showset = 'YTBEdev'
MODELS=['decode-hybrid-adapts-plp','decode-grph-tandem-adapts-plp','decode-tandem-adapts-plp','decode-hybrid-adapts-tandem']
#confMap=['plp-bg_decode_cn.tree','plp-bg_decode_cn.tree','plp-bg_decode_cn.tree','plp-bg_decode_cn.tree','grph-tandem-bg_decode_cn.tree']
confMap=['hybrid-bg_decode_cn.tree','grph-tandem-bg_decode_cn.tree','tandem-bg_decode_cn.tree','hybrid-bg_decode_cn.tree']
#confMap=['hybrid-bg_decode_cn.tree' for x in [1,2,3,4,5]]

########################################################################

# rescores to the folder starting with cnr in systems
def CNRESCORE(show,system,subfolder):
    os.system('./scripts/cnrescore.sh -OUTPASS {subfolder}_cn {show} {system} {subfolder} {system}'.format(show=show,system=system,subfolder=subfolder))

# need some way to map confidence scores


def CNC2MLF(show,sys,subf):
```

```python
   cnc2mlf.main(dir+'/{sys}/{show}/{subf}_cn/lattices/'.format(show=show,sys=sys,subf=subf),dir+'/{sys}/{show}/{subf}_cn/lattices/cnc.mlf'.format(show=show,sys=sys,subf=subf))


# model constants
cnrescore,cnc, =False,False

files=h.matchfile('systems','cnr')
print(files)

print('cnrescore')
if cnrescore:
    for show in FILELIST[showset]:
        for model in MODELS:
                CNRESCORE(show,'systems',model)
##h.qwait()

print('cnc2mlf')
if cnc:
    for model in MODELS:
        for show in FILELIST[showset]:
                CNC2MLF(show,'systems',model)
##h.qwait()

for model,confmap in zip( MODELS,confMap):
    for show in FILELIST[showset]:
        input= '/home/mec68/MLSALT11/{folder}/systems/{show}/{model}_cn/rescore.mlf'.format(show=show,folder=folder,model=model)
        output='/home/mec68/MLSALT11/{folder}/systems/{show}/{model}_cn/rescore_mapped.mlf'.format(show=show,folder=folder,model=model)
        tree='/home/mec68/MLSALT11/{folder}/lib/trees/{confmap}'.format(folder=folder,confmap=confmap)
        os.system('../../smoothtree-mlf.pl '+tree+' '+input+' > '+output)
#######################################################################
findAlphas,FinalCombine=True,False



for model,confmap in zip( MODELS,confMap):
    for show in FILELIST[showset]:

        #input= '/home/mec68/MLSALT11/{folder}/systems/{show}/{model}_cn/rescore.mlf'.format(show=show,folder=folder,model=model)
        input= '/home/mec68/MLSALT11/{folder}/systems/{show}/{model}_cn/lattices/cnc.mlf'.format(show=show,folder=folder,model=model)

        output='/home/mec68/MLSALT11/{folder}/systems/{show}/{model}_cn/rescore_mapped.mlf'.format(show=show,folder=folder,model=model)
        tree='/home/mec68/MLSALT11/{folder}/lib/trees/{confmap}'.format(folder=folder,confmap=confmap)
        os.system('../../smoothtree-mlf.pl '+tree+' '+input+' > '+output)
#######################################################################


findAlphas,FinalCombine=False,True


#######################################################################


def getAlpha2(outdir,list):
    num=0
    h.mkdir('ref1'); h.mkdir('temp1');h.mkdir(outdir)
    max, alphabest=100, None;
    for alpha in [2.5,2,1.8,0.9]:
        print(alpha)
        for show in FILELIST[showset]:
            out='/home/mec68/MLSALT11/{folder}/temp1/{show}/decode/rescore.mlf'.format(show=show,folder=folder)
            #print(show)
            list2=[x.format(show=show) for x in list]
            merge3.main(out,list2,[1,alpha])
        # move directory if better
        val=h.testDecode('temp1',showset); print(val)
        if val<max:
            max=val; alphabest=alpha;
            h.copyfold('temp1','ref1')
        if val>max:
            num+=1

        if num>=2:
            break
    h.copyfold('ref1',outdir)
    return alphabest


def getAlphas(out,list):
    head,tail=list[0],list[2:]
    alphas=[]
    ref='/home/mec68/MLSALT11/'+folder+'/'+out
    refiles='/home/mec68/MLSALT11/'+folder+'/'+out+'/{show}/decode/rescore.mlf'
```

```
    alphas.append(getAlpha2(ref,[head,list[1]]))
    for item in tail:
                alphas.append(getAlpha2(ref,[refiles,item]))
                print(alphas)
    print(alphas)
    return alphas


if findAlphas:
    COMBINEARGS=[dir+'/systems/{show}/'+'{model}_cn/rescore.mlf'.format(model=model) for model in MODELS]
    ##COMBINEARGS=[dir+'/systems/{show}/'+'{model}_cn/rescore_mapped.mlf'.format(model=model) for model in MODELS]
    #COMBINEARGS=[dir+'/systems/{show}/'+'{model}_cn/lattices/cnc.mlf'.format(model=model) for model in MODELS]
    h.mkdir('combined2')
    out='combined2'
    alphas = getAlphas(out,COMBINEARGS)


######################################################################

def combByAlpha(alphas,out,list,DPMetWts,delVal,thresh=0):
        print(list)
        for show in FILELIST[showset]:
            print(show)
            ref='/home/mec68/MLSALT11/'+folder+'/'+out+'/{show}/decode/rescore.mlf'.format(show=show)
            #print(ref)
            basemlf=[x.format(show=show) for x in list]
            merge3.main(ref,basemlf,alphas,DPMetWts,delVal,thresh)


def Test(alphas,out,list):
    max=100.1
    for dpmet in [[1,0,0]]:
        for delval in [1]:
            for thresh in [0]:
                #for delval in [0.1,0.2,0.4,0.8]:
                    combByAlpha(alphas,out,list,dpmet,delval,thresh)
                    val=h.testDecode(out,showset)
                    #print('score: {val} for testing a={a}, b={b}, c={c}, delval={delval}'.format(a=dpmet[0],b=dpmet[1],c=dpmet[2],delval=delval,val=str(val)))
                    if val<max:
                        max=val;# h.copyfold(out,'best')

alphas=[1,3, 2, 2, 1.2]
alphas=[1,1.3, 1, 1, 1.5]
alphas=[1,0.9,0.9,2]
if FinalCombine:
    COMBINEARGS=[dir+'/systems/{show}/'+'{model}_cn/rescore_mapped.mlf'.format(model=model) for model in MODELS]
    print(COMBINEARGS)
    listed=Test(alphas,'final-bad',COMBINEARGS)
    print(listed)


######################################################################
```