MLSALT11: Large Vocabulary Speech Recognition

Riashat Islam
Department of Engineering
University of Cambridge
Trumpington Street, Cambridge, CB2 1PZ, England
ri258@cam.ac.uk

I. Introduction

The objective of our work is to develop an overall large scale speech recognition system. The system is to be built for a Broadcast News transcription task. Most of the current speech recognition systems are built based on the Hidden Markov Model (HMMs). The speech recognition task is broken down into first decomposing the speech into a sequence of acoustic observatons or frames, and the goal of the recogniser is to find the most likely sequence of words given the acoustics. The language model sets prior to the word sequences, while the acoustic model links word sequences to acoustics. Current large-scale speech recognition systems consider the acoustic and language model components separately, and the evaluation of these components are considered separately in this work. Current systems also performs the function of speaker adaptation and adjusts the acoustic models to match the specifics of an individual voice. In acoustic model adaptation, the acoustic featues are adjusted to more closely match the generic acoustic models [1], while some methods also consider adjusting the acoustic models to match the feature vectors [2]. Section II below we present a brief overview to the tasks of language and acoustic modelling.

II. Background

In acoustic modelling, we want to determine the acoustic probabilities conditioned on the words, $P(a|w)$. Acoustic modelling is done by the use of HMMs that can model speech as being produced by a speaker whose vocal trat configuration proceeds through a sequence of states, and produces one or more acoustic vectors in each state. Furthermore, the observation probabilities for each state is modelled with a mixture of Gaussians. Maximum Likelihood estimation is performed for training the HMM based systems such that by estimating the parameters, we can compute the posterior state-occupancy probabilities for the HMM states.

For the language model, finite state grammars for the speech recognition task. These grammars may be expressed as an unweighted regular expression that can recognize a finite state of statements. N-gram language models are often used in large scale systems, where the probability of each word is conditioned on the $n-1$ preceding words. Furthermore, for building N-gram language models, we will consider using interpolated models which are empirically found to give good performance on a variery of tasks. Details of cross language-model adaptation are further included in section IV. In LM interpolation, all the sources are trained together to build the final LM model.

III. Overview

This report is organised as follows. In section IV, we focus on improving the language model using LM interpolation techniques based on minimising the perplexity. We investigate the weight interpolation techniques based on language model interpolation and adaptation, and investigate the different language modelling schemes for the overall system. In section V, we then evaluate the significance of acoustic model adaptation, based on MLLR transforms, and considering cross-adaptation techniques for adaptation. Section VI further evaluates the significance of system combination for improving the overall system performance. We consider combination schemes based on combining multiple sets of hypothesis together to improve system performance. Finally, in section VII we evaluate the performance of our system on the Challenge Dataset, which is based on Youtube shows, for performing large-scale speech recognition.

IV. Language Modelling

We consider improving language models by combining several language models together. In LMs, multiple component models are combined using interpolation weights. Common approaches to language model combination includes linear interpolation, log-linear interpolation, backing off and maximum entropy. In this work, we consider language model interpolation based on minimising the perplexity by estimating the interpolation weights.

Model Interpolation with HTK: We consider using the HTK language modelling tools to provide the capabilities of producing and evaluating interpolated language models. The LMERGE tool can be used to combine a number of existing models to produce a new model. LPLEX can also be used to compute perplexities using linearly interpolated n-gram probabilities from a number of source language models.

A. *Question 1: 1-best Output from Lattices*

First we consider obtaining the 1-best output from the lattices using the 1bestlats.sh script.

```
./scripts/1bestlats.sh dev03_DEV001 20010117 XX2000
    lattices decode plp bg
```

The above scrpt creates the MLF files. HTK supports the logical label MLF files, that can combine multiple transcription files into a single file. Furthermore, we can use MLF files as input and hence merge multiple MLF files. The recognition output MLF can then be compared with the reference MLF to obtain the word error performance. This uses the HResults HTK command, as part of the suppled score.sh script.

```
./scripts/score.sh plp bg dev03sub 1best/LM12.0_IN
    10.0
```

The word error rate (WER) is given as:

$$WER = 1 - Acc = 1 - \frac{N - S - D - I}{N} = \frac{S + D + I}{N} \times 100\% \quad (1)$$

Corr	Sub	Del	Ins	Err
87.4	9.5	3.0	2.7	15.3

Table I
SCORING ON DEVSUB BASED ON 1-BEST OUTPUTS

B. *Question 2: WER of Language Models*

To evaluate performance of the supplied language models, we first need to rescore the lattices. In ASR, a lattice is generated which is a weighted directed acyclic graph that can represent the lattice results. With the lattice input, the language model decoupling step splits the language model scores of the lattice for the lattice rescoring. Having rescored the lattices, we can then extract the N-best list from the lattice. During the rescoring step, the sentences are rescored for the N-best list with the LM rescourses. Now, the n-best list can be ordered according to the new scores.

The performance of the supplied language models can then be evaluated. Using each of the lattices, the langauge models can be rescored using the lmrescore.sh script. The script applies a new language model to each of the lattices.

```
./scripts/lmrescore.sh dev03_DEV001 20010117 XX2000
    lattices decode lms/lm4 plp tglm4 FALSE
```

The WER on each of the language models can then be evaluated. The results can be seen below for each LM.

```
./scripts/score.sh plp tglm1 dev03 rescore
```

LM	Corr	Sub	Del	Ins	Err
1	85.1	11.5	3.4	2.8	17.7
2	83.8	12.6	3.6	2.8	19.1
3	80.8	14.4	4.8	2.4	21.6
4	80.1	14.9	4.9	2.7	22.5
5	83.7	12.2	4.1	2.3	18.6 height

Table II
WER ON EACH OF THE LANGUAGE MODELS

The results show that the WER performance is different for each of the language models. This is expected since the language model data is from different sources, and therefore the N-best or 1-best list for each of the sentence rescoring would be differnet for the LMs.

C. *Question 3: Perplexity of LMs*

Perplexity of a language model is the inverse probability of a test set. A language model with a lower perplexity (i,e higher probability and lower cross-entropy) on the test set is better than a LM with higher perplexity. The perplexity of each of the language models can then be found using the LPlex command. The following command is run for each of LM1 to LM5.

```
base/bin/LPlex C lib/cfgs/hlm.cfg u t lms/lm1 lib
    /texts/dev03.dat
```

Language Model	Perplexity
1	198.6167
2	242.7938
3	283.7183
4	337.4166
5	220.4432

Table III
PERPLEXITY FOR EACH LANGUAGE MODEL

The results show that the perplexity values correlates with the WER. It shows that the LM1 is much better than LM4, since it has a perplexity value lowest of 198.6 and a WER of 17.7, compared to LM4 which performs worst with the highest perplexity and WER values. Additionally, for each of the LMs, we see that the perplexity values have a strong correlation with the WER.

D. *Question 4: Language Model Interpolation Tool*

We consider the task of combining multiple n-gram language models based on using a simple linear interpolation. Given a set of M training texts, we can build a combined LM using multiple techniques. The combined LMs are constructed by building multiple component models that are combined using interpolation weights. By tuning the interpolation weights using perplexity, it is possible to adapt LMs to a particular task [3] We consider tuning the interpolation weights based on minimising the perplexity.

Since sufficient training data is available, and there exists a strong correlation between perplexity and WER, we consider a maximum likelihood or perplexity based estimation scheme used for optimizing the global interpolation weights. **Perplexity/ML Estimation:** The global interpolation weights are found by minimizing the perplexity measure as follows:

$$F_{PP} = \exp(-\frac{\ln P(W)}{L}) \quad (2)$$

This is equivalent to maximizing the log-likelihood of the entire word sequence $\ln P(W)$. Here, W is the held-out data that is used for interpolation tuning or supervision in the case of model adaptation. The optimal interpolation weights for the $m_t h$ component model is therefore given as:

$$\lambda^{ML} = \underset{\lambda_m}{\operatorname{argmax}} \ln P(W) \quad (3)$$

Based on this, the Baum-Welch algorithm may then be used to iteratively re-estiamte the weights:

$$\lambda^{ML} = \frac{C^{ML}(null)}{\sum_m C^{ML}(null)} \quad (4)$$

Note that if the perplexity base adaptation mode was required to be performed in supervised model, we would then need the transcriptions. We have therefore implemented our language model interpolation tool such as to improve the performance of the LMs by combining multiple LMs together. The interpolation weights were estimated using the development set only. Our results show that the perplexity value for the combined language models is much lower than the individual LMs.

```
python3.4 LanguageModelInterpolation.py \
  lm_paths lms/lm{1..5} \
  text_data_path \
lib/texts/dev03.dat \
  output_lm_path \
$lm_dev03_interpolation
```

*1) **Explanation of Code - LM Interpolaton Tool:**:* The code is available at /home/ri258/Documents/MLSALT11/. The code listing is also included in appendix.

The interpolation is done by the main function *lm˙interpolation˙tool*. This function is sued to evaluate the word conditional probabilities. We used an initial random interpolation weight. Using the while loop, the inteprolation weights are updated based on the procedure described above. Finally, the word marginal probabilites can be extracted from the interpolated language models.

E. Question 5: Performance of Interpolated LMs

We have therefore used our LM interpolationt technique based on the Baum-Welch algorithm for iterative re-estimation of the weights to robustly estimate the context dependent language model intrepolation weights on the training data. The commands below allows us to then evaluate the performance on the development and evaluation sets.

```
base/bin/LPlex -C lib/cfgs/hlm.cfg -u -t ${
    lm_dev03_interpolation} \ lib/texts/dev03.dat

base/bin/LPlex -C lib/cfgs/hlm.cfg -u -t ${
    lm_dev03_interpolation} \ lib/texts/eval03.dat
```

Combined Language Model	Perplexity
Development Set	151.5218
Evaluation Set	154.8489

Table IV
PERPLEXITY OF COMBINED LANGUAGE MODELS

```
# WER (dev03)
for show in $(cat ./lib/testlists/dev03.lst); do
  ./scripts/lmrescore.sh $show lattices decode
    lm_dev03_interpolation plp tg${
    lm_dev03_interpolation} FALSE
done
./scripts/score.sh plp tg${lm_dev03_interpolation}
    dev03 rescore
```

```
##WER (eval03)
for show in $(cat ./lib/testlists/eval03.lst); do
  ./scripts/lmrescore.sh $show lattices decode
    lm_dev03_interpolation plp tg${
    lm_dev03_interpolation}_eval03 FALSE
done
./scripts/score.sh plp tg${lm_dev03_interpolation}
    _eval03 eval03 rescore
```

Combined LM	Corr	Sub	Del	Ins	Err
dev03	85.6	10.6	3.8	2.3	16.8
eval03	87.0	9.7	3.3	2.0	15.0

Table V
WER ON THE COMBINED LANGUAGE MODEL

*1) **Discussion of Results: LM Interpolation**:* Our results show that the perplexity value of the combined language model is much lower than each of the language models that was considered previously. The combined LM has a perplexity value of **151.2** on the development set, whereas the previous best individual LM1 had a perplexity of **198.6**. Additionally, the WER for the combined LM is **16.8** on the development set, which is also lower than the WER of individual best LM1 of **17.7**. Our results therefore shows that by combining multiple language models together, we can significantly improve the performance of the speech recognition application. The model interpolation technique based on minimising the perplexity can therefore be used for language model combination.

F. Question 6: Unsupervised Language Model Adaptation

We then consider unsupervised methods of language model adaptation to a specific speaker. We want to evaluate whether the adaptation methods and their combinations can reduce the perplexity and WER in this task. Furthermore, we take the approach of unsupervised method by making direct use of the initial recognition result for generating an enhanced model.

We consider the adaptation method using initial recognition hypothesis since the shows have long speech and their transcriptions. We have the previous interpolation weights obtained using the inteprolation tool, based on which adaptation is performed and the interpolation weights are trained again. Here, we use the test set for the estimation of the weights directly. After computing the interpolation weights based on the test set, we can again combine multiple LMs together and then apply the adapted combined LM to each of the lattices. The following commands are run for the unsupervised LM adaptation.

```
for show in $(cat ./lib/testlists/eval03.lst); do
./scripts/1bestlats.sh $show lattices decode plp tg$
    {lm_dev03_interpolation}_eval03
done
```

```
for show in $(cat ./lib/testlists/eval03.lst); do
one_best_path=plp tg${lm_dev03_global}_eval03/$show
    /1best/LM12.0_IN 10.0
```

```
if [[ -a ${one_best_path}/text.dat ]]; then
    rm ${one_best_path}/text.dat
fi
python3.4 Conversion.py ${one_best_path}/rescore.mlf
    ${one_best_path}/text.dat
python3.4 LanguageModelInterpolation.py lm_paths
    lms/lm{1..5} text_data_path ${one_best_path}/
    text.dat output_lm_path ${one_best_path}/
    lm_int
./scripts/lmrescore.sh $show lattices decode ${
    one_best_path}/lm_int plp tg unsupervised lm
    adaptation FALSE
done
./scripts/score.sh plp tg unsupervised lm adaptation
    eval03 rescore
```

The table below shows the WER performance of the adapted language model based on using the evaluation set only, as development set transcriptions are not used for the adaptation task.

Corr	Sub	Del	Ins	Err
86.9	9.7	3.3	2.0	15.1

Table VI
SCORING BASED ON UNSUPERVISED LANGUAGE MODEL ADAPTATION

```
for show in $(cat ./lib/testlists/eval03.lst); do
  one_best_path=plp tg${lm_dev03_global}_eval03/
    $show/1best/LM12.0_IN 10.0
  base/bin/LPlex C lib/cfgs/hlm.cfg u \
     t ${one_best_path}/lm_int \
    lib/texts/eval03.dat
done
```

Adapted Language Model	Perplexity
DEV011	156.2
DEV012	156.1
DEV013	155.6
DEV014	156.6
DEV015	156.3
DEV016	159.7

Table VII
SHOW-SPECIFIC PERPLEXITY OF UNSUPERVISED ADAPTED LANGUAGE MODELS

*1) **Discussion of Results: Unsupervised LM Adaptation**:* Our results show that the each of the show specific lattice perplexity values are higher than the peplexity achieved by the combined LM. The combined language model had a perplexity value of **154.8**, whereas, the show-specific LM applied to each of the lattices have values of **155** or **156**. Furthermore, the combined language model had a WER of **15.0** whereas the adapted LMs perform slighly worse of **15.1**. The adapted LMs were performed direclty on the test set. In comparison, the combined LMs were initially tested on the development set using the development set transcriptions, and then generalised to the unseen evaluation set.

G. *Summary: Language Modelling*

In this section, we have so far investigated context dependent form of language model interpolation and adaptation. We implemented our LM interpolation technique based on minimising perplexity and Baum-Welch (BM) algorithm to estimate the context dependent language model interpolation weights on the training data. Further from this, using the interpolation weights, we then investigated LM adaptation and evaluted the performance. Our exprimental results showed that by using LM weight interpolation, better WER and lower perplexity can be achieved on the combined interpolated language model, compared to the adapted language models based on the evaluation. Other approaches to LM interpolation and adaptation uses discriminate training techniques, and a range of approaches (not considered here) are available for intergrating weight estimation in LM interpolation and adaptation.

V. ACOUSTIC MODEL ADAPTATION

In this section, we perform acoustic model adaptation in speech recognition task such that the model can take account of mismatches between speakers, environmental noise etc. Acoustic model adaptation can compenstate for the acoustic mismatch between the training and test data, and also to adapt the speaker independent systems to individual speakers [4], []. Acoustic model adaptation is performed to modify the parameters of the acoustic model to fit the actual acoustic characteristics by using a few utterances from the target user. In this work, we consider different approaches to acoustic model adaptation.

In speaker adaptation, techniques such as MAP estimation can be used to estimate the model parameters robustly than the ML estimation when the amount of available training data is small. However, there also exists other approaches such as trnansform based techniques like Maximum Likelihood Linear Regression (MLLR) that uses a linear mapping between the acoustic feature spaces of different speakers as the adaptation model [1], [5], [6]. MLLR is also one of the widely used adaptation methods. MLLR uses a linear transformation of Gaussian model parameters to adapt to a given speaker. MLLR transforms aims to maximise the likelihood of the adaptation data given the correct hypothesis in the case of supervised MLLR.

A. *Question 1: Determinisation and Rescoring of Lattices*

We consider the use of lattice-based information for supervised speaker adaptation. In order to consider the new acoustic models, we need to first determinise the lattices, followed by rescoring. This is because since the MLLR may often incorrectly find a transform that can incrrase the likelihood of the incrrect models and lower the likelihood for the correct hypothesis. Therefore, since the oracle word error rate of a lattice is much lower than the 1-best or N-best hypothesis, we need to perform adaptation against a word lattice, such that the correct models are more likely to be estimated using the transform.

```
./scripts/mergelats.sh dev03_DEV001 20010117 XX2000
    lattices decode plp bg
./scripts/hmmrescore.sh dev03_DEV001 20010117 XX2000
    plp bg merge plp bg plp
```

```
./scripts/lbestlats.sh dev03_DEV001 20010117 XX2000
    plp bg decode plp merge bg
```

The HTK command HLRescore can be applied to expand the lattices for acoustic modelling with a higher order language model to get an improved recognition result and a set of new lattices that have been rescored with the higher order langauge model. The above commands, by using HLRescore, can threfore find the 1-best path through the lattice, and prune the lattices using the forward-backward scores while also expanding the lattices based on the new acoustic models. The HLRescore command can be used to find the best transcription for a given parameter setting. Following this, the lattices can again be scored as follows:

```
./scripts/score.sh plp merge bg dev03sub 1best/LM12
    .0_IN 10.0
```

Corr	Sub	Del	Ins	Err
87.2	9.7	3.1	2.8	15.6

Table VIII
SCORING AFTER DETERMINIZATION AND RESCORING OF LATTICES

Comment on performance compared to original lattices: The original lattices with the 1-best output had a score of **15.3%**. However, after lattice rescoring with the new acoustic models, followed by beam pruning and rescoring with the original models, the performance improves by **0.3%** to **15.6%**.

B. Question 2: CMLLR and MLLR Transforms

MLLR was previously evaluated on the Wall Street Journal task [1]. In MLLR, the mean vectors of the Gaussian distributions in the HMMs are updated according to the following transformaiton:

$$\hat{\mu} = A\mu + b \tag{5}$$

where A is a $n \times n$ matrix and b is a n-dimensional vector. We can further write $\hat{mu} = W\eta$, where $W = [bA]$ and $\eta = [1\mu^T]^T$. In MLLR, W is estimated so as to maximize the likelihood of the adapted data. A single transform W can be shared across a set of Gaussian components. The linear transformation matrix W can be estimated by finding the setting which maximizes the log likelihood.

MLLR can provide a class of adaptation models, where MAP methods can also be used to estimate the paramters. Using MAP estimation within the MLLR framework can furhter provide significant improvements compared to using them separately. In model-based adaptation techniques, MLLR can be used to adapt the parameters of the acoustic model to better match the observed data.

In the constrained MLLR approach, the basic idea is to use the same linear transform for both the mean and the covariance.

$$\hat{\mu} = A\mu + b \tag{6}$$

$$\hat{\Sigma} = A\Sigma A^T \tag{7}$$

The above equations need to be solved iteratively since no closed form solutions exist. Constrained MLLR therefore considers transforming the covariance matrix and transforms the features into a feature space. There also exists unconstrained approaches (not considered in our work) which assumes that the covariance matrix represents speaker characteristics that are different from the mean, and therfore estimates the paramters independently. However, adaptation of the variances considered in constrained and unconstrained MLLR does not bring much improvement to simply using MLLR.

Experimental Results: We therefore considered producing CMLLR and MLLR transforms using the 1-best hypothesis.

```
./scripts/hmmadapt.sh dev03_DEV001 20010117 XX2000
    plp bg decode plp adapt bg plp
```

The transforms can then be used to rescore the lattices.

```
./scripts/hmmrescore.sh ADAPT plp adapt bg adapt
    dev03_DEV001 20010117 XX2000 plp bg merge plp
    adapt bg plp
```

The plp acoustic model can then be scored, to obtain the results below:

```
./scripts/score.sh plp adapt bg dev03sub decode
```

Corr	Sub	Del	Ins	Err
88.4	8.5	3.1	2.2	13.8

Table IX
SCORING ON ADAPTED ACOUSTIC MODEL BASED ON MLLR AND CMLLR TRANSFORMS

*1) **Discussion of Results: MLLR and CMLLR transforms**:* The acoustic model adaptation technique based on MLLR and CMLLR transforms considered above have reduced the WER to 13.8%. By considering forced alignment of the acoustic model and the hypothesis, and by using the transform estimation using the HERest tool in HTK, we have produced MLLR and CMLLR transforms which can then be used to rescore the lattices. The WER based on these transforms (13.8%) are much lower compared to simply determinising and rescoring the lattices which had a WER of 15.6%.

C. Question 3: Graphemic PLP System, Cross Adaptation of Acoustic Models

In context-dependent acoustic models, decisions trees are constructed based on questions on the phone attributes. However, an alternative approach for context-dependent acoustic modelling is to consider using graphemic lexicon, where the prounciation for a word is defined by the letters forming the word. In this section, we consider the use of graphemic system for acoustic model adaptation. We particularly focus on cross-system adaptation where complementary information is dervied from several systems. In cross-system adaptation, the same data is used by different

systems. In other words, one system S1 is adapted by using the output of a different system S2. Here, we evalaute the performance of cross sysem adaptation of the acoustic model, using the graphemic system.

By using graphemes as subword units in ASR, the pronounciation of words can be easily derived from the transcription. This further gives the adavantage of using cross-adaptation systems, or sharing data resources from different languages when training acoustic models. We consider whether using graphemes as subword units, we can improve the acoustic model adaptation technique for the ASR task.

Experimental Results: Cross Adaptation with Graphemic PLP system Cross adaptation is then performed for acoustic model adaptation. The previous lattices are now again scored, but now using the graphemic plp system.

```
./scripts/hmmrescore.sh dev03_DEV001 20010117 XX2000
    plp bg merge grph plp bg grph plp
```

The output of this system is again scored as below. Note that the output of this graphemic PLP system is worse than the unadapted PLP. This is because the plp lattices are now rescored using the graphemic plp.

```
./scripts/1bestlats.sh dev03_DEV001 20010117 XX2000
    grph plp bg decode grph plp bg
./scripts/score.sh grph plp bg dev03sub 1best/LM12.0
    _IN 10.0
```

Corr	Sub	Del	Ins	Err
86.6	10.2	3.1	2.8	16.1

Table X
SCORING ON THE GRAPHEMIC PLP ACOUSTIC MODEL

The 1-best output hypothesis can then be used for adaptation using the hmmadapt.sh script, and the lattices again rescored.

```
./scripts/hmmadapt.sh  OUTPASS adapt grph plp
    dev03_DEV001 20010117 XX2000 grph plp bg decode
    plp adapt bg plp
./scripts/hmmrescore.sh  ADAPT plp adapt bg adapt
    grph plp  OUTPASS decode grph plp dev03_DEV001
    20010117 XX2000 plp bg merge plp adapt bg plp
```

Having performed adaptation, we again generate the 1 best lattice output using the adapted acoustic models using graphemic plp, and score the output.

```
./scripts/1bestlats.sh  OUTPASS 1best grph plp
    dev03_DEV001 20010117 XX2000 plp adapt bg decode
     grph plp plp adapt bg

./scripts/score.sh plp adapt bg dev03sub 1best grph
    plp/LM12.0_IN 10.0
```

Corr	Sub	Del	Ins	Err
88.5	8.5	3.0	1.9	13.4

Table XI
SCORING ON THE GRAPHEMIC PLP ACOUSTIC MODEL

The result shows that using the cross-adaptation scheme based on using the graphemic PLP system, the WER performance can be slightly improved from 13.8% to 13.4% compared to using the PLP system based on MLLR transforms. We can therefore conclude that cross-adaptation schemes for acoustic model adaptation might perform better compared to using MLLR transforms as a model adaptation scheme independently for the overall system.

D. *Summary: Acoustic Model Adaptation*

In this section, we have therefore evaluated two different approaches for acoustic model adaptation for the overall speech recognition system. We evaluated the system using both the phoneme and grapheme subword units. Our results showed that in ASR, using graphemes as subword unit yields a significant improvement in overall performance. Furthermore, since we showed that using cross-adaptation scheme, such that acoustic modelling can be performed by modelling both phoneme and grapheme subword units, the cross-adapted scheme could help in improving the performance of the ASR system, further reducing the WER.

VI. SYSTEM COMBINATION

In this section, we consider cross-system adaptation schemes and system combination methods such as ROVER and confusion networks to further improve the performance of the speech recognition system [7]. Previously, having adapted both the language and acoustic models for better fitting the system to the speakers, we now consider the significance of combining the adapted language and acoustic models. Simply considering adaptation methods does not always lead to an improvement in performance. During adaptation schemes, since one system makes different errors than another system, one system mainly gets complementary information that it could not gain from its own output. Furthermore, it is also possible to utilize this complementary information contained in the hypothesis from different recognition systems by using system output combination methods. We particularly consider the confusion network combination (CNC) [8] and ROVER [9] system combination schemes. There are different approaches to system combination. We can combine the probability estimates of several acoustic models before they are being fed into the decoder. Alternatively, we can also combine several language models estimated on different corpora by interpolating their probability estimates. Cross-adaptation and system combination may also be performed based on PLP via confusion networks.

A. *Question 1: Compare alignments and error rates*

For system combination of different acoustic models, based on PLP and graphemic PLP, the first step in combining the hypothesis together is to align the 1-best hypothesis. This alignment would work towards minimising a WER like measure. The alignment becomes more important as the differene between the hypothesis being aligned becomes large. Script errorMLF.py is the tool that can take two MLF

files and compute the error rate between them.

```
python3.4 errorMLF.py \
plp bg/dev03_DEV001 20010117 XX2000/decode/rescore.mlf \
grph plp bg/dev03_DEV001 20010117 XX2000/decode/rescore.mlf
```

```
./base/bin/HLEd -i plp bg/dev03_DEV001 20010117 XX2000/decode/score.mlf \
-l '*' /dev/null \
plp bg/dev03_DEV001 20010117 XX2000/decode/rescore.mlf
```

Then using this "reference" score the output from the graphemic system:

```
./base/bin/HResults -t -f -I plp bg/dev03_DEV001 20010117 XX2000/decode/score.mlf lib/wlists/train.lst grph plp bg/dev03_DEV001 20010117 XX2000/decode/rescore.mlf
```

Based on using the alignment process, and then using the reference to score the output from the graphemic system, we achieve a correctness of **87.98%** and an accuracy measure of 86.41%.

B. *Question 2: Idealised Combination*

We then perform idealised combination using the script mlfAlignment.py which combines MLF files to generate a network from which the best path can be used for combination.

```
python3.4 mlfAlignment.py \
  plp bg/dev03_DEV001 20010117 XX2000/decode/rescore.mlf \
  grph plp bg/dev03_DEV001 20010117 XX2000/decode/rescore.mlf \
  outfile score oracle.mlf
```

The above is done for each of the lattices from the development set. We use the script mlfAlignment.py to modify the MLF combination scheme to combine several hypothesis and generate a MLF that may allow the oracle error rate to be found. Below, we include a description of the code.

*1) **Explanation of Code: mlfAlignment.py**:* Code for the mlfAlignment process is included in Appendix. It is also available in /home/ri258/Documents/MLSALT11.

The main function takes the two mlf files in consideration. It uses the functions $mlf\_parsing$ function to to parse the reference mlf file. The $mlf\_parsing$ function reads the reference mlf file and maintains a dictionary. Parsing is performed on the reference mlf for entries of transcript, start, end, word and log probabilities. Mlf parsing is also performed on the other mlf file, and the two sets of outputs are then merged together by the mlf_merge function. The mlf_merge further considers the $alignment$ function that takes as arguments the set of transcripts from the two mlf files and using dynamic programming returns the minimum number of insertions and deletions for each entry in the reference transcription with transcription from the other mlf file. The $alignment$ function also uses the tracking function to return the results of the alignment and the required statistics. Finally, the return from the $main$ function is the result of the two merged mlfs. This is then used as argument to the function $conversion$ that returns the word and scoring results as strings for the alignment process.

C. *Question 3: Generating Confusion Networks*

Previously, we only considered the best ASR 1-best output from the lattices. In this section, we consider the task of using word confusion networks obtained from the lattices to provide a compact representation of multiple aligned ASR hypotheses in addtion to using the word confidence scores. Here, instead of using 1-best hypotheses, we exploit word confusion networks and evaluate the system performance.

We consider the approach of generating word confusion scores and confusion networks from the ASR lattice output. Confusion networks are much smaller in size than the word lattices, but still able to preserve the accuracy of the original search space. Confusion networks are generated by using the script cnrescore.sh. In generating confusion networks, one possible approach is to first compute the posterior probability of each transition in the ASR lattice by applying the forward and backward algorithms to the lattice. This posterior probability can also be used as a confidence score by itself. However, performance can be further improved if the confusion network is built by taking account the competing hypothesis in the same time slot. Eventually, we can obtain the posterior probability of a word w. By considering this approach, we can avoid missing the important contribution of the same word, often due to the small differences in the time alignment. A detailed description of the process of generating a confusion network is given in [10].

Experimental Results: The confusion networks can be generated using the cnrescore.sh script used as follows. This is done for each of the lattices in the development set.

```
./scripts/cnrescore.sh dev03_DEV001 20010117 XX2000 plp bg decode plp bg
```

The generated confusion network and MLF can then be scored.

```
./scripts/score.sh plp bg dev03sub decode_cn
```

Corr	Sub	Del	Ins	Err
87.6	9.6	2.8	3.0	15.4

Table XII
SCORING THE GENERATED CONFUSION NETWORKS

Scoring based on the generated confusion network achieves a WER of **15.4%**. This is much better than scoring based on the 1-best output hypothesis considering each of the languge models that we examined initially. We have used word confusion networks that are obtained from word lattices instead of using ASR 1-best output. By evaluating the results, we find that the confusion networks achieve a much larger reduction in the error rate compared to the ASR 1-best output.

Furthermore, using the provided confidence mapping trees, the confidence scores can be mapped to better reflect the probability of the word being correct. Having applied the confidence mapping trees, surprisingly, we found the same evaluation score of a WER of 15.4%. Therefore, based on our results, we did not find any significant impact of mapping the confidence scores, although it was expected that confidence mapping may further lower the WER.

D. Question 4: Combination of two sets of hypothesis - ROVER

Having derived an approach to generate a confusion network, we now further apply the confusion network decoding to the graphemic PLP system. We can then perform combination on the two sets of hypothesis. This can be done as follows:

```
./scripts/cnrescore.sh dev03_DEV001 20010117 XX2000
    grph plp bg decode grph plp bg
```

We can then modify the alignment process to combine two word sequences and assign a confidence score.

```
sysCombPath=grph plp syscomb/dev03_DEV001 20010117
    XX2000/decode/
mkdir -p $sysCombPath
python3.4 mlfAlignment.py \
  plp bg/dev03_DEV001 20010117 XX2000/decode_cn/
    rescore.mlf \
  grph plp bg/dev03_DEV001 20010117 XX2000/decode_cn
    /rescore.mlf \
    outfile $sysCombPath/aligned.mlf \
    NULLSCORE 0.2
```

Details of the ROVER combination scheme are included in the next section. ROVER allows combination by combining the single best output from multiple systems. However, the hypothesis must first be aligned along with considering the time overlap between the words. Furthermore, a confidence score must also be assigned to the NULL hypothesis during the ROVER combination scheme. The peformance of the system can be seen as follows:

```
python3.4 ROVER.py grph plp syscomb/dev03_DEV001
    20010117 XX2000/decode/aligned.mlf \
    outfile $sysCombPath/rescore ROVER.mlf
```

*1) **Explanation of Code: ROVER:**:* This script uses the mlf obtained from the previous alignment process, and can be used for the combination scheme. The main function takes the aligned MLF file and performs parsing similar to before. For each entry in the aligned mlf, we take the max of the entries. The $mlf\_parsing$ function also uses the function $word\_parsing$ to return a list of the splitted word strings.
The following results were obtained, using a NULLScore of 0.2

Corr	Sub	Del	Ins	Err
88.6	9.3	2.2	4.3	15.8

Table XIII
SCORING BASED ON ROVER COMBINATION - COMBINATION OF TWO SETS OF HYPOTHESIS

Compared to the previous score of 15.4% based on the generated confusion networks, the scoring after combining the two hypothesis based on the ROVER combination scheme with a NULL score of 0.2 is larger at 15.4%. However, the performance of the system also depends on the 0.2 score assigned to the NULL hypothesis. We then analyse how the system performance further varies with the as the confidence score assigned to the NULL hypothesis changes.

NULLSCORE	Corr	Sub	Del	Ins	Err
0.0	88.6	9.3	2.2	4.3	15.8
0.2	88.6	9.3	2.2	4.3	15.8
0.5	88.4	9.2	2.3	3.9	15.5
0.7	88.2	8.9	2.9	3.2	15.0
0.75	88.2	8.9	3.0	3.0	14.9
0.8	88.1	8.9	3.1	2.9	14.9
0.9	87.9	8.9	3.2	2.5	14.6
1.0	87.6	8.7	3.7	2.2	14.6

Table XIV
SCORING DEPENDING ON THE CONFIDENCE SCORE ASSIGNED TO THE NULL HYPOTHESIS

Our results above shows that as a higher confidence score is applied to the NULL hypothesis, the WER reduces significantly from 15.8% to 14.6%. This shows that the performance of the system is quite dependent to the confidence score assigned to the NULL hypothesis, and WER can be significantly removed by assigning a high confidence score to the NULL hypotheses.

E. Question 5: Confusion Network Combination

We consider the task of lattice based system combination called confusion network combination (CNC) [11]. In later section, we consider another combination scheme called ROVER. ROVER is a simple combination technique which can combine the single best output from multiple ASR systems. The lattices can be used to provide word confidence scores. In ROVER the hypothesis are first aligned with the time overlap between words as a local cost. Due to the alignment, there exists a series of slots each containing a word per system together with its confidence score. During decoding, each slot is treated separately where the word with the maximum total confidence is chosen.

In confusion network combination, instead of aligning the single best output, we can align confusion networks together which are built from individual lattices. CNC combination provides advantages over ROVER since each slot now contains many hypotheses and hence CNC has a much lower oracle error rate. We get posterior distribution over all words in each slot in the source confusion networks. Compared to ROVER, in confusion network combination, we can compute for each slot and word the combined word posterior probability over all systems. The word posterior probability for the combined confusion network is the weighted average of the individual slot-wise posteriors. During decoding, the word with the highest combined posterior probability is then chosen.

Confusion Network Combination (CNC) therefore works by integrating multiple confusion networks together. We evaluate the recognition performnace of CNC and provide a comparison with previous results. In CNC, multiple networks are combined together based on posterior probabilities attached to each word. The proposed CNC method in our work is based on a combination of confusion networks derived by aligning multiple confusion networks together, for which we also need to consider distance measures. We perform a sub-network-based alignment between confusion networks, and then a new confusion network is composed by combining the two networks based on posterior probabilities.

The following commands were run to evaluate the performance of our confusion network combination scheme.

```
mv $sysCombPath/rescore.mlf $sysCombPath/rescore
    ROVER smooth.mlf
python3.4 CNC Comb.py \
  plp bg/dev03_DEV001 20010117 XX2000/decode_cn/
    lattices/ \
  plp bg/dev03_DEV001 20010117 XX2000/decode_cn/
    lattices/
```

```
python3.4 CNC Comb.py \
  plp bg/dev03_DEV001 20010117 XX2000/decode_cn/
    lattices/ \
  grph plp bg/dev03_DEV001 20010117 XX2000/decode_cn
    /lattices/
```

```
python3.4 CNC2.py \
  plp bg/dev03_DEV001 20010117 XX2000/decode_cn/
    lattices/ \
  grph plp bg/dev03_DEV001 20010117 XX2000/decode_cn
    /lattices/ \
  > $sysCombPath/rescore CNC.mlf
```

```
base/conftools/smoothtree mlf.pl \
  lib/trees/grph plp bg_decode_cn.tree \
  $sysCombPath/rescore CNC.mlf > $sysCombPath/
    rescore.mlf
```

The aligned and combined confusion networks, based on the graphemic PLP and the PLP system can then be scored to get the following result.

```
./scripts/score.sh grph plp syscomb dev03sub decode
```

Corr	Sub	Del	Ins	Err
85.0	10.6	4.4	4.7	19.7

Table XV
SCORING BASED ON CONFUSION NETWORK COMBINATION

The results of our above evaluation, based on system combination of multiple confusion networks achieves a much larger WER compared to simply scoring based on the single confusion network. In other words, the combination of multiple confusion networks obtained by system combination of graphemic and plp system does not necessarily improve the performance of the overall system.

*1) **Explanatio of Code: CNC Combination**:* Code for the CNC combination scheme is included in appendix. The code is also available in /home/ri258/Documents/MLSALT11.

The main function takes the two lattices directories as arguments. These are the two directories based on which CNC combination would be carried out. Confusion network parsing is then performed on both the lattices using the $CN_Parsing$ function. This function opens the lattice files and stores the arcs of the lattice in a dictionary, parses the set of entries using two other functions that returns integers using the python built-in function $readline$. The function $parse$ returns the set of entires for each word, start and end entires and the probability values. It then returns the lattice arcs for performing and constructing the confusion network. The main function also uses the function $confusion\_network\_alignment$ that can compute the cost of the best alignment of each of the nodes. This function returns the two sets of alignment results based on the sorted list of lattice parsing.

F. Question 6: Overall System Performance

Finally, we can evaluate the performance of the combination scheme on the development set and the evaluation set. The performance on the entire development set can be evaluated as follows

```
./scripts/hmmrescore.sh dev03_DEV001 20010117 XX2000
    plp bg merge grph plp bg grph plp
./scripts/cnrescore.sh dev03_DEV001 20010117 XX2000
    grph plp bg decode grph plp bg
./scripts/cnrescore.sh dev03_DEV001 20010117 XX2000
    plp bg decode plp bg
```

```
mkdir -p dev03 eval/dev03_DEV001 20010117 XX2000/
    decode

python3.4 mlfAlignment.py \
plp bg/dev03_DEV001 20010117 XX2000/decode_cn/
    rescore.mlf \
grph plp bg/dev03_DEV001 20010117 XX2000/decode_cn/
    rescore.mlf \
  outfile dev03 eval/dev03_DEV001 20010117 XX2000/
    decode/aligned.mlf \
  NULLSCORE 0.0

python3.4 ROVER.py dev03 eval/dev03_DEV001 20010117
    XX2000/decode/aligned.mlf \
  outfile dev03 eval/dev03_DEV001 20010117 XX2000/
    decode/rescore ROVER.mlf

base/conftools/smoothtree mlf.pl \
lib/trees/grph plp bg_decode_cn.tree \
dev03 eval/dev03_DEV001 20010117 XX2000/decode/
    rescore ROVER.mlf > dev03 eval/dev03_DEV001
    20010117 XX2000/decode/rescore.mlf
```

Having combined the two sets of hypotheses from the acoustic models, performing MLF alignment followed by system combination using ROVER to combine the 1-best hypothesis, we can then evaluate the development performance on the entire development set as follows:

```
./scripts/score.sh dev03 eval dev03 decode
```

We can then evaluate the speech recognition system performance on the entire evaluation (unseen) dataset, to see how

Corr	Sub	Del	Ins	Err
57.5	8.1	34.4	3.0	45.4

Table XVI
DEVELOPMENT SET PERFORMANCE

our system generalises to unseen data for the Broadcast News Transcription System.

```
./scripts/score.sh dev03 eval eval03 decode
```

Corr	Sub	Del	Ins	Err
57.5	9.1	34.8	3.0	46.8

Table XVII
EVALUATION SET PERFORMANCE

The above WER results for evaluation of the entire system is much higher than the pervious WER results we were analysing. Note that this is particularly becuase we are now scoring our output on the entire system, which has much higher number of lattices, compared to using only a subset of the development set for rapid system evaluation. Therefore, the WER of 46.8% on the entire evaluation system makes sense since we are trying to recognise the speech for a much larger show, which is a more difficult task compared to system performance on only a very small show.

G. *Summary: System Combination*

In this section, we approached different acoustic model system combination schemes, based on generating confusion networks from the lattices. In particular, we evaluated the ROVER and Confusion Network Combination (CNC) schemes, to analyse how the results may be improved due to system combination. Our results shows that by performing the ROVER combination, with assigning appropriate confidence score to the NULL values in the confusion network, the WER can be reduced to upto 14.6% which is a significant reduction in WER compared to our results previously. Compared to this, using the CNC method, the WER was 19.7% which showed that the ROVER scheme performs much better than combination of confusion networks. Finally, we evaluated our overall system performance on the entire show based on the development and evaluation set. The WER on the entire development set achieved a score of 45.4% and a score of 46.8% on the evaluation set. Although we expected these WER values to be much lower on the entire set, however, the results are appropriate since we are considering a larger number of shows for evalution. In section VII below, we would consider evaluation on the Challenge Data, which is based on Youtube shows, for evaluating performance of our large-scale speech recognition system.

VII. CHALLENGE DATA EVALUATION

In this section, we consider the different approaches we took for developing our overall system for evaluation on the challenge development set. We consider evaluating our system on large scale Youtube data for speech recognition. We perform experiments on the Youtube development set. To develop the final evaluation system, we again combine the approaches that we have examined together previously.

A. *Language Model*

Considering the lattices for the Yotube development set, by simply taking the 1-best output from the lattices, a score of 49.7% only. However, if we take the language model interpolation weights, using our LM interpolation tool based on the EM algorithm, a WER of 46.2% could be achieved. Furthermore, we considered the use of global interpolation weights to generate 1-best hypothesis. By taking the adapted language models, and considering perplexities of adapted LMs, the WER could be further reduced to 45.4% on the Youtube development subset. The results are summarised in table below. Again, after considering language model interpolation and combination, the overall performance could be significantly improved.

Language Modelling	
Method	DevSub WER
1-best	49.7
LM Interpolation	46.2
Unsupervised Adaptation	45.4

Language Model Interpolation and Adaptation using Youtube development subset

B. *Acoustic Model Adaptation*

For the acoustic model adaptation, we considered using all the approaches such as plp, grph-plp, grph-tandem and the hybrid approach. For each of the adaptation techniques, we evaluated the individual acoustic model adaptation performance based on the Youtube development subset. For each approach, we also include a brief description for the type of adaptation being considered.

PLP: Considering only the PLP system, and initially performing lattice determinization followed by beam pruning, a WER of **50%** was achived. However, when we used the MLLR and CMLLR transforms, the WER could be further reduced to **46.3%** based on the adapted PLP system.

Graphemic PLP: On simply obtaining the 1-best output from the lattices, based on using the graphemic PLP system, a score of **50.6%** was achieved only. However, similar to above, when we consider using the hypothesis for adaptation, the adapted graphemic PLP system could achieve a WER of **46.3%**.

Graphemic Tandem: Similar to the HMM/GMM sysems, in Tandem systems, the acoustic units are also clustered context-dependent subword units and the lexical and acoustic units are also deterministically related. The tandem systems are capable of exploiting cross-domain acoustic and lexical resources by using a multi-layer perceptron (MLP) trained on cross-domain resources. However, by considering Youtube development subset, we found that the performance of the

tandem systems are worse than that of the Graphemic PLP and PLP systems. We achieved a WER of **58%** after cross adaptation with the PLP system. However, we doubt if the evaluation was done properly or not, since there were errors observed in the scoring output. Overall, we considered the use of Tandem/MLP features since it can bridge the gap between phoneme-based and grapheme-based ASR systems. As discussed previously, the correspondence between grapheme and phoneme based systems is weak. Therefore, using a tandem system for cross-adaptation might be useful.

Hybrid - DNN-HMM Systems: We further considered the use of deep neural network (DNN) and HMM hybrid acoustic models for acoustic model adaptation tasks. Recent results have shown that hybrid models can significantly improve the accuracy than conventional systems based on Gaussian mixture models (GMMs). We consider the task of speaker adaptation for the neural network acoustic models by performing a small of additional training using data from the target speaker. In order to exploit the discriminant nature of the neural networks, it is better to train a single model to discriminate both between different phone classes and between the target speaker and the rest of the world. In the hybrid DNN-HMM approach, a neural network classifier is trained to estimate the posterior probability of context independent phone classes, and then use these probilities as inputs in the conventional HMM decoder. Such hybrid systems have been shown to scale to large Broadcast News training sets. In hybrid systems, the backpropagation training algorithm used to train the original network models could be applied at recognition time to shift the model towards a particular speaker based on the labellings from a first-pass recognition. By considering the hybrid system for acoustic model adaptation, an overall score of **41.9%** was achived on the Yotube development set. We could not however perform cross adaptation with other plp, grph-plp or grph-tandem systems when using the hybrid approach. Since the scoring of **41.9%** was only achieved based on determinization followed by rescoring with the hybrid approach.

Acoustic Model Adaptation	
Approaches	DevSub WER (%)
PLP	50
PLP-Adapt	46.3
Graphemic	50.6
Graphemic-Adapt	46.3
Graphemic-Tandem	58
Hybrid	41.9

Acoustic Model adaptation techniques for Youtube challenge development set

The table above summarizes our results for the different approaches for acoustic model adaptation. Overall, our results show that the hybrid approach performs the best for acoustic model adaptation achieving the lowest WER of **41.9%**. However, since we could not experiment with analysing the cross-adaptation approach based on hybrid system, we will eventually consider the graphemic-adapted PLP and the graphemic-tandem system and evaluate the test set performance based on these acoustic model adaptation schemes.

C. System Combination

We consider the task of system combination based on using the ROVER and CNC combination schemes discussed previously.

*1) **PLP and Graphemic-PLP**:* It is possible to combine the PLP and Graphemic-PLP systems based on both the ROVER and CNC combination schemes. To combine the two sets of hypothesis from these two systems, we again need to apply confusion network decoding to the graphemic system output. The ROVER combination scheme can be applied as follows:

```
sysCombPath=grph_plp_syscomb_YTBE/YTBElect_YTB$i_XXXXXXXX_XXXXXX/decode/
python3.4 ROVER.py \
grph_plp_syscomb_YTBE/YTBElect_YTB$i_XXXXXXXX_XXXXXX/decode/aligned.mlf \
  outfile $sysCombPath/rescore_ROVER.mlf
```

Further to this, we can also use the mapped confidence scores. We can also apply the CNC combination scheme based on these two systems. We can combine the two systems to produce the confusion network for the combined system. This can be done as follows:

```
sysCombPath=grph_plp_syscomb_YTBE/YTBElect_YTB$i_XXXXXXXX_XXXXXX/decode/
python3.4 CNC_Comb.py \
plp_bg_YTBE/YTBElect_YTB$i_XXXXXXXX_XXXXXX/decode_cn/lattices/ \
plp_bg_YTBE/YTBElect_YTB$i_XXXXXXXX_XXXXXX/decode_cn/lattices/
```

On evaluating the system on the Youtube development set, based on combination of PLP and Grph-PLP system, we achieved an overall WER of **45.4%**.

*2) **PLP-Adapt and Graphemic-PLP Adapt**:* We again considered evaluating the combination of the adapted PLP and Graphemic-PLP systems, based on using both the ROVER and CNC combination schemes. Using the generated decoding lattices, the generated confusion networks with confidence scores can be scored again. This achieves an overall WER of **44.5%**. Using the ROVER combination scheme, we can again score the combined system, and we notice a slight reduction in the WER performance to **43.8%** based on combining the adapted systems. We submitted the CTM file for this combined system as part of the initial submission for evaluating results on Youtube evaluation set.

*3) **Graphemic-Tandem and PLP Adapt**:* By simply generating the confusion networks based on the lattices, and using the graphemic-tandem approach, a WER of **43.8%** could be achieved without initially performing any combination. However, when we combined the output hypothesis from the graphemic-tandem and the adapted-plp systems, the performance of our system on the development dataset was significantly reduced, achieving the loweest WER so far of **42.9%**. As discussed later, as part of our second submission for Yotube evaluation set, we submitted the CTM file generated using this combination scheme.

*4) **Hybrid, PLP-Adapt and Tandem-Adapt**:* We further considered using the ROVER combination for combining the acoustic model adaptation methods based o the hybrid and the adapted PLP systems. Confusion network decoding was again applied to both the models. We can then combine the two word sequences and again generate an output MLF based on the ROVER combination script. However, due to scripting issues, we could not obtain a score on the development set using the combination of these two systems. Furthermore, we also tried to evaluate the combination model based on CNC using the outputs from the hybrid system and the tandem system. Again, a scoring result for this was not obtained. Our understanding is that using the hybrid systems, we made errors in evaluating the entire system due to which an overall score on the Youtube development set could not be achieved. Hence, no evaluation submission was made based on hybrid acoustic model adaptation techniques.

5) Summary of Results: Finally, we summarize all the WER results we obtained based on Youtube development set, using the different system combination schemes discussed above.

System Combination Approaches	
Approaches	Development Set WER (%)
PLP and Graphemic-PLP	45.4
PLP-Adapt and Graphemic-PLP Adapt	43.8
Graphemic-Tandem and PLP Adapt	42.9

Our results show that using the combination of the acoustic models based on graphemic-tandem and adapted-plp systems, the lowest WER score of **42.9%** can be achieved on the entire Yotube development set. This combination scheme was also used in our final submission for the Yotube evaluation score for the speech recognition task.

VIII. YOUTUBE EVALUATION SET SUBMISSIONS

A. *1st Submission*

For the first submission, we used the language model inteprolation technique, followed by acoustic model system combination based on the adapted-PLP and the adapted graphemic-PLP system. On the development set, we achived a WER of **43.8%** using this sytem combination scheme, having done both language model interpolation and acosutic model adaptation. The following result was obtained based on evaluating the Yotube evaluation set. We received a score of **43.2%** on the Youtube evaluation set, as shown below.

Corr	Sub	Del	Ins	Err
61.2	19.8	19.0	4.5	43.2

Table XVIII
1ST SUBMISSION EVALUATION SET RESULT

B. *2nd Submission*

For the second submission, we again first considered using the language model interpolation tool for the given language models. Following this, during acoustic model adaptation, we first considered using the adapted-PLP system, and then evaluated the performance of the development set using the graphemic-tandem approach. System combination was then performed using the ROVER method for combination of the graphemic-tandem and the adapted-plp systems. Using this approach, we received a WER of **42.9%** on the entire Youtube development set. On submitting our submission for evaluation, the following result was then obtained on the Youtube evaluation set. We recieved a WER score of **36.7%** on the Youtube evaluation set.

Corr	Sub	Del	Ins	Err
68.7	22.1	9.2	5.4	36.7

Table XIX
2ND SUBMISSION EVALUATION SET RESULT

IX. OVERALL BEST SYSTEM

In this section, we summarize the overall based system that we have developed for the speech recognition task. The model is developed using the BBC News transcription training data, where we used a sub-development set for rapid system evaluation. We first considered using the language model combination using our interpolation tool. We then developed our acoustic model speaker adapted system based on combining the graphemic-tandem and adapted-plp systems. Development set results showed that using the combination scheme based on both the ROVER and CNC methods, the overall system performance based on acoustic model adaptation can be improved. The overall best system was therefore evaluated using the combination of the graphemic-tandem and adapted-plp acoustic model adaptation schemes. On the development set, our developed system achieved a WER of **43.2%**. On evaluating our developed speech recognition system on the Youtube shows, we achieved an overall evaluation score of **36.7%** which is our final WER score for our large-scale speech recognition system.

X. CONCLUSION

In this work, we have considered building the different components of a speech recognition system. We considered improving the language models based on using the EM algorithm, through the language interpolation tool to combine multiple language models. Unsupervised langauge model adaptation was also evaluated based on the available training data. Our results showed that by using combined LMs, we can achieve better perplexity and WER scores based on the development set. We then considered acoustic model adaptation techniques based on cross adaptation with another acoustic model. In particular, we evaluated our overall system performance based on several appraoches, such as using plp, grph-tandem and hybrid systems. For each acoustic model adaptation technique, we evaluated the overall system performance based on the development set. Finally, we considered the task of combining several acoustic models together. For each combined system, we evalauted the system performance based on the

development set, compared the different system performances and finally evaluated the overall best system for this speech recognition task. Using our overall speech recognition system, we achieved the overall score of **36.7%** on the challenge evaluation Youtube dataset.

XI. Acknowledgment

The author would like to thank the Cambridge University Engineering Department for providing access to the HTK toolkit. We thank Mark Gales for the enormous support through teaching and for developing initial scripts to run the experiments. Finally, we thank the Cambridge University MPhil Machine Learning, Speech and Langauge Technology program for providing the practical experimentation opportunity.

References

[1] M. J. F. Gales. Maximum likelihood linear transformations for hmm-based speech recognition. *Computer Speech & Language*, 12(2):75–98, 1998.

[2] C.J. Leggetter and P.C. Woodland. Flexible speaker adaptation using maximum likelihood linear regression. In *Proc. ARPA Spoken Language Technology Workshop*, pages 110–115. Morgan Kaufmann, 1995.

[3] X. Liu, M. J. F. Gales, and P. C. Woodl. Context dependent language model adaptation. In *In Proc. InterSpeech-2008*, 2008.

[4] Steve Renals. Overview speaker adaptation, 2013.

[5] Silke Goronzy Ralf and Ralf Kompe. Srf99056 a combined map + mllr approach for speaker adaptation.

[6] Hao Chao and Wenju Liu. Speaker adaptation of stochastic segment models using maximum likelihood linear regression. In *7th International Symposium on Chinese Spoken Language Processing, ISCSLP 2010, November 29 2010-December 3, 2010, Tainan & Sun Moon Lake, Taiwan*, pages 119–122, 2010.

[7] Hugo Meinedo Jo. Combination of acoustic models in continuous speech recognition hybrid systems.

[8] Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding consensus in speech recognition: word error minimization and other applications of confusion networks. *Computer Speech & Language*, 14(4):373–400, 2000.

[9] Jonathan G. Fiscus. A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (rover). pages 347–352, 1997.

[10] Dilek Z. Hakkani-Tür, Frédéric Béchet, Giuseppe Riccardi, and Gökhan Tür. Beyond ASR 1-best: Using word confusion networks in spoken language understanding. *Computer Speech & Language*, 20(4):495–514, 2006.

[11] G. Evermann and P.C. Woodland. Posterior probability decoding, confidence estimation and system combination, 2000.

APPENDIX: Major Speech Practical CODE

April 21, 2016

1 Language Model Interpolation Tool

```
1
2 #!/usr/bin/python3.4
3
4 if __name__ == '__main__':
5     import argparse
6     parser = argparse.ArgumentParser(description = 'Using EM for Language Model
      Interpolation')
7     parser.add_argument('  lm_paths', nargs = '+', type=str)
8     parser.add_argument('  text_data_path', type=str)
9     parser.add_argument('  output_lm_path', type=str)
10    args = parser.parse_args()
11    lm_interpolation_tool(args.path, args.t_path, args.output)
12
13
14 import random
15 import subprocess
16
17
18 def lm_interpolation_tool(path, t_path, output):
19     langauge_probability = {}
20     words = 0
21     for lm in path:
22         subprocess.call("base/bin/LPlex  C lib/cfgs/hlm.cfg  s stream  u  t {0} {1}".format(
       lm, t_path).split(" "))
23
24
25         with open('stream') as f:
26             langauge_probability[lm] = list(map(float, f))
27             words = len(langauge_probability[lm])
28
29     interpolation_weights = [0.2 for _ in path]
30     interpolation_weights = list(map(lambda x: x / sum(interpolation_weights),
       interpolation_weights))
31     interpolation_weights = dict(zip(path, interpolation_weights))
32
33     old_w = None
34     while (old_w is None) or sum([(old_w[lm]   interpolation_weights[lm])**2 for lm in
       interpolation_weights]) > 1e 6:
35         old_w = dict(interpolation_weights)
36
37         D = [sum([interpolation_weights[lm] * langauge_probability[lm][w] for lm in
       interpolation_weights])
38                 for w in range(words)]
39         for lm in interpolation_weights:
40             interpolation_weights[lm] = (1. / words) * sum([(interpolation_weights[lm] *
       langauge_probability[lm][w]) / D[w] for w in range(len(langauge_probability[lm]))])
41
42
43     merged = "base/bin/LMerge  C lib/cfgs/hlm.cfg " + ' '.join(\
44             map(lambda lm: " i " + str(interpolation_weights[lm]) + ' ' + lm, filter(lambda
       x: x != path[0], interpolation_weights))) + " lib/wlists/train.lst {0} {1}".format(path
       [0], output)
```

```
    subprocess.call(merged.split(" "))
```

Listing 1: Script for Language Model Interpolation

2 MLF Alignment

```

#!/usr/bin/python
from collections import defaultdict
from copy import deepcopy
from math import log

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description='Align multiple MLFs for oracle decoding/
    ROVER')
    parser.add_argument('ref_mlf', type=str)
    parser.add_argument('other_mlfs', nargs="+", type=str)
    parser.add_argument('  outfile', type=str)
    parser.add_argument('  NULLSCORE', default=0.2, required=False, type=float)
    args = parser.parse_args()
    with open(args.outfile, 'w') as out:
        out.write(conversion(main(args.NULLSCORE, args.reference, args.other)))



insertion = 5
substitution = 10


def mlf_merge(NULLSCORE, ref, other):
    new_mlf = []
    for f1, f2 in zip(ref, other):
        (path, _) = alignment(NULLSCORE, f1['transcript'], f2['transcript'])
        f1['transcript'] = path
        new_mlf.append(f1)
    return new_mlf


def main(NULLSCORE, reference, other):
    with open(reference) as ref:
        this = mlf_parsing(ref)
        for other_mlf in other:
            with open(other_mlf) as other:
                this_mlf = mlf_merge(NULLSCORE, this, mlf_parsing(other))
    return this_mlf



def DynamicProgramming(ref, other):

    A = [[0 for _ in range(len(other)+1)] for _ in range(len(ref)+1)]
    B = [['' for _ in range(len(other)+1)] for _ in range(len(ref)+1)]
    # base case
    B[0][0] = ('START', '')
    for i in range(1,len(ref)+1):
        A[i][0] = i*insertion
        B[i][0] = ('DEL', ref[i 1])
    for j in range(1,len(other)+1):
        A[0][j] = j*insertion
        B[0][j] = ('INS', other[j 1])
    for i in range(1,len(ref)+1):
        for j in range(1,len(other)+1):
            actions = [
                    (('DEL', ref[i 1]), A[i 1][j]+insertion), # insertion in ref
                    (('INS', other[j 1]), A[i][j 1]+insertion), # insertion in other
                    (('SUB', ref[i 1], other[j 1]), A[i 1][j 1]+substitution) # substitution
```

```
                    ]
            if ref[i-1]['word'] == other[j-1]['word']:
                actions.append((('MATCH', ref[i-1]), A[i-1][j-1]))
            bestAction = min(actions, key=lambda x: x[1])
            B[i][j] = bestAction[0]
            A[i][j] = bestAction[1]
    return (A, B)


def mlf_parsing(mlf):
    mlf.readline()
    index = []
    entry = dict()
    for line in mlf:
        if ".\n" in line:
            index.append(entry)
            entry = dict()
        elif line[0] == '"':
            name = "\"*/" + line.strip().split('/')[-1]
            entry["name"] = name
            entry["transcript"] = []
        else:
            line = line.strip().split(' ')
            entry["transcript"].append({
                'start': int(line[0]),
                'end': int(line[1]),
                'word': [line[2]],
                'logProb': [float(line[3])],
                })
    return index



def tracking(NULLSCORE, A, B):

    ret = defaultdict(int)

    path = []
    i = len(B)-1
    j = len(B[0])-1
    while B[i][j][0] != 'START':
        if B[i][j][0] == 'DEL':
            entry = B[i][j][1]
            entry['word'].append('<DEL>')
            entry['logProb'].append(NULLSCORE)
            path.append(entry)
            ret['N'] += 1
            ret['NUM_DEL'] += 1
            i -= 1
        elif B[i][j][0] == 'INS':
            entry = B[i][j][1]
            entry['word'] = ['!NULL'] + entry['word']
            entry['logProb'] = [NULLSCORE] + entry['logProb']
            path.append(entry)
            ret['NUM_INS'] += 1
            j -= 1
        elif B[i][j][0] == 'SUB':
            entry = B[i][j][1]
            entry['word'].extend(B[i][j][2]['word'])
            entry['logProb'].extend(B[i][j][2]['logProb'])
            path.append(entry)
            ret['N'] += 1
            ret['NUM_SUB'] += 1
            i -= 1
            j -= 1
        elif B[i][j][0] == 'MATCH':
            entry = B[i][j][1]
            path.append(entry)
```

```
            ret['N'] += 1
            ret['H'] += 1
            i  = 1
            j  = 1

    return (list(reversed(path)), ret)




def conversion(mlf):
    output = ''
    def system(entry):
        word = entry['word'][0]
        score = '_'.join(map(lambda x: '{:0.2f}'.format(x), entry['logProb']))
        if len(entry['word']) > 1:
                word += '_<ALTSTART>_' \
                        + '_<ALTEND>_<ALTSTART>_'.join(entry['word'][1:]) \
                        + '_<ALTEND>'
        return '{0} {1} {2} {3}'.format(entry['start'], entry['end'], word, score)
    output += "#!MLF!#\n"
    for entry in mlf:
        output += entry['name'] + '\n'
        output += '\n'.join(map(system, entry['transcript']))
        output += "\n.\n"
    return output

def alignment(NULLSCORE, ref, other):
    (A, B) = DynamicProgramming(ref, other)
    (path, ret) = tracking(NULLSCORE, A, B)
    return (path, ret)
```

Listing 2: MLF Alignment script

3 Rover Combination

```
#!/usr/bin/python

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description='Error computed between two mlfs')
    parser.add_argument('aligned_mlf', type=str, help='Use the Aligned MLF')
    parser.add_argument('  outfile', type=str)
    args = parser.parse_args()
    with open(args.outfile, 'w') as out:
        out.write(conversion(main(args.aligned)))

def conversion(mlf):

    output = ''
    def format_entry(entry):
        return '{0} {1} {2} {3}'.format(entry['start'], entry['end'], entry['word'], entry['
    logProb'])
    output += "#!MLF!#\n"
    for entry in mlf:
        output += entry['name'] + '\n'
        output += '\n'.join(map(format_entry, entry['transcript']))
        output += "\n.\n"
    return output

def mlf_parsing(mlf):
    mlf.readline()
    entries = []
    entry = dict()
    for line in mlf:
        if ".\n" in line:
            entries.append(entry)
```

```
            entry = dict()
        elif line[0] == '"':
            name = "\"*/" + line.strip().split('/')[1]
            entry["name"] = name
            entry["transcript"] = []
        else:
            line = line.strip().split(' ')
            entry["transcript"].append({
                'start': int(line[0]),
                'end': int(line[1]),
                'word': parse_words(line[2]),
                'logProb': list(map(float, line[3].split('_')))
                })
    return entries


def main(aligned):
    with open(aligned) as f:
        mlf = mlf_parsing(f)
        for instance in mlf:
            for entry in instance['transcript']:
                (entry['word'], entry['logProb']) = max(
                        zip(entry['word'], entry['logProb']),
                        key=lambda x: x[1])
            instance['transcript'] = list(filter(
                lambda x: x['word'] not in ['<DEL>', '!NULL'],
                instance['transcript']))
        return mlf


def parse_words(word):
    return list(filter(lambda x: len(x) > 0,
            re.split(r"_<ALTSTART>_|_<ALT>_|_<ALTEND>", word)))
```

Listing 3: Rover combination script

4 Confusion Network Combination

```
#!/usr/bin/python

from collections import defaultdict
import re
import gzip


if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description='Confusion Network Combination Scheme')
    parser.add_argument('dir1', type=str)
    parser.add_argument('dir2', type=str)
    parser.add_argument('  outfile', type=str)
    args = parser.parse_args()
    main(args.directory1, args.directory2)




def main(directory1, directory2):
    lattices = (CN_Parsing(directory1), CN_Parsing(directory2))
    fName = sorted(list(lattices[0].keys()))[0]
    A, B = Confusion_Network_Alignment(lattices[0][fName], lattices[1][fName])
    path = tracking(B)

def tracking(B):
    path = []
    i = len(B) 1
```

```
    j = len(B[0]) 1
    while B[i][j] != 'STAT':
        action = B[i][j]
        path.append(action)
        if action == 'ALIGNMENT':
            i  = 1
            j  = 1
        elif action == 'SKIP_A':
            i  = 1
        elif action == 'SKIP_B':
            j  = 1

    return list(reversed(path))


def Confusion_Network_Alignment(cn1, cn2):
    t1 = time_entries(cn1)
    t2 = time_entries(cn2)

    A = [[float("Inf") for _ in range(len(t2))] for _ in range(len(t1))]
    B = [['' for _ in range(len(t2))] for _ in range(len(t1))]

    A[0][0] = 0.
    B[0][0] = 'START'
    for i in range(1,len(t1)):
        A[i][0] = abs(t1[i]   t2[0])
        B[i][0] = 'SKIP_A'
    for j in range(1,len(t2)):
        A[0][j] = abs(t1[0]   t2[j])
        B[0][j] = 'SKIP_B'
    for i in range(1,len(t1)):
        for j in range(1,len(t2)):
            act = [ ('ALIGNMENT', A[i 1][j 1] + abs(t1[i]   t2[j])) ]
            if i != range(len(t1)):
                act.append(('SKIP_A', A[i 1][j]))
            elif j != range(len(t2)):
                act.append(('SKIP_B', A[i][j 1]))
            best = min(act, key=lambda x: x[1])
            B[i][j] = best[0]
            A[i][j] = best[1]
    return (A, B)



def CN_Parsing(lattDir):
    latts_CN = {}
    for fName in os.listdir(lattDir):
        path = lattDir + fName
        with gzip.open(path, 'rb') as f:
            N = parse_1(f)
            arcs = defaultdict(lambda: defaultdict(list))
            for _ in range(N):
                k = parse_2(f)
                for _ in range(k):
                    arc = parse(f)
                    arcs[arc['start']][arc['end']].append(arc)
            latts_CN[fName] = arcs
    return latts_CN

def parse(f):
    line = re.split(r"\s+", str(f.readline())[: 5])
    return {
            'word': line[0][4:],
            'start': float(line[1][2:]),
            'end': float(line[2][2:]),
            'prob': float(line[3][2:])
            }

```

```
98
99  def time_entries(cn):
100     k = list(cn.keys())
101     e = list(cn[max(k)].keys())
102     return sorted(k + e)
103
104
105 def parse_1(f):
106     return int(f.readline()[2:].strip())
107
108 def parse_2(f):
109     return int(f.readline()[2:])
```

Listing 4: Confusion Network combination scheme