UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and, Machine Intelligence 2018-2019

Module Coursework Feedback

Module Title: Speech Recognition

Module Code: MLMI2

Candidate Number: F606F

Coursework Number: 1

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.

Marker's Comments:

This piece of work has been completed to the following standard *(Please circle as appropriate)*:

	Distinction			Pass			Fail (C+ - marginal fail)		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

MLMI2 Speech Recognition Practical Report

1. Gaussian Mixture Model – Hidden Markov Models (GMM-HMMs)

1.1 Initial Setup and Testing

After creating the working directory 'pracgmm' which included the important files of 'convert', 'data', 'tools' and 'exp'. In the 'exp' directory, based on the HInit/HRest initialization, the GMM-HMMs monophone model was built by the following code,

- **../tools/steps/step-mono -NUMMIXES 8 ../convert/fbk25d/env/environment_Z FBK_Z_Init/mono**

It corresponds to the form of,

- **(Operation_command) (-NUMMIXES integer) (environment folder) (folder for storage)**

The 1st segment is the operation used to create the GMM-HMMs monophone model. The 2nd segment specifies the number of Gaussian mixtures created, later in the FBK_Z_Init/mono file, there are several hmm sub-directories each with two integers behind it. The first integer specifies the number of Gaussians and the final integer denotes the number of HEREST iterations.

To find out how the model performs, the file above was decode by the following code,

- **../tools/steps/step-decode $PWD/FBK_Z_Init/mono hmm84 FBK_Z_Init/decode-mono-hmm84**

It corresponds to the simplified form of,

- **(Operation_command) (target file path which stores target hmm) (target hmm directory to be decoded) (folder for storage)**

There are several options available to step-decode. The overall step-decode operation can be written in the form of,

- **../tools/steps/step-decode [-CORETEST|-SUBTRAIN] [-DECODEHTE path] [-GRAMMARSCALE float] [-INSWORD float] [-BEAMWIDTH float] [-TOOLSDIR path] (abspath_sourcedir) (hmmdir) (systemdir)**

To be clear, '**abspath_sourcedir**' is the target file path which stores target hmm (**$PWD/FBK_Z_Init/mono**), '**hmmdir**' is the target hmm (**hmm84**) and '**systemdir**' is the folder for storing the result (**FBK_Z_Init/decode-mono-hmm84**).

Then various experiments were done to understand the options of the step-decode.

TABLE 1

HInit/HRest monophone model		Default	-BEAMWIDTH		-INSWORD (-BEAMWIDTH -1000)	
			-8	-1000	1000	-50
Sentence	Number	1341	No Transcriptions Found.	1344	1344	1344
Word	Hit	30708		30757	31601	22855
	Insertion	11428		11450	78522	11115
	%Correctness	59.46		59.47	66.28	44.19
	%Accuracy	37.33		37.33	-98.42	42.84

The percentage of word correctness shown in Table 1 is given by,

$$\%Correctness = \left(\frac{H}{N}\right)100\%$$

, where H is the number of correct phones called hits and N is the total number of phones.

The percentage of word accuracy shown in Table 1 is given by,

$$\%Accuracy = \left(1 - \frac{D + I + S}{N}\right)100\%$$

, where D the number of phone deletions, I the number of insertions and S the number of substitutions.

From Table 1, the total sentence number with the default decode setting was 1341 which was not 1344 (the correct number of sentences). To correct this value, the 'BEAMWIDTH' option was applied. It was found that the negative sign of a large integer, 1000 for instance, was needed to tune the sentence number back to the correct one. If a small integer, 8 for example, with a negative sign was applied, no transcription was found in the decode file.

Then the 'INSWORD' option was also tested. If a positive value was used, it allowed more insertion, which made the word accuracy significantly low and the correctness high. This made the evaluation for the performance of the model unrealistic, since the number of insertions (78522) was greater than the total number of phones (51721). On the other hand, if a negative value was used, less insertion was used, the correctness was reduced but the accuracy increased.

1.2 Flat Start

There is another way to build the GMM-HMMs monophone model via the step-mono script, and it is called 'flat-start'.

To use the flat-start to build the model, the following code was used.

- **../tools/steps/step-mono -FLATSTART -NUMMIXES 8 ../convert/fbk25d/env/environment_Z FBK_Z_FlatStart/mono**

The following flat-start model was then decoded by the code below,

- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/FBK_Z_FlatStart/mono hmm84 FBK_Z_FlatStart/decode-mono-hmm84**

TABLE 2

FlatStart monophone model		-BEAMWIDTH -1000
Sentence	Number	1344
Word	Hit	30922
	Insertion	11716
	%Correctness	59.79
	%Accuracy	37.13

Refer to Tables 1 and 2, the flat-start model showed a similar result from the HInit/HRest model. The flat-start model had a higher correctness, but the HInit/HRest model had a higher accuracy percentage.

The flat-start initialization was chosen for the future experimentation.

1.3 Other Front-End Parameterisations

To compare the performance differences between different differential coefficients (_D,_D_A) and feature types (FBANK and MFCC_E), the following procedure was went through,

Firstly, the following monophone GMM-HMMs were built.

For the FBANK feature type, the models without differential and with 2nd differential were built with 8 and 16 Gaussians.

- **../tools/steps/step-mono -FLATSTART -NUMMIXES {8,16} ../convert/fbk25d/env/environment_Z FBK_Z_FlatStart/mono{,16}**
- **../tools/steps/step-mono -FLATSTART -NUMMIXES {8,16} ../convert/fbk25d/env/environment_D_A_Z FBK_D_A_Z_FlatStart/mono{,16}**

For the MFCC_E feature type, the models with different differential coefficients were built with 8 and 16 Gaussians.

- **../tools/steps/step-mono -FLATSTART -NUMMIXES {8,16} ../convert/mfc13d/env/environment_E_Z MFCC_Z_FlatStart/mono{,16}**
- **../tools/steps/step-mono -FLATSTART -NUMMIXES {8,16} ../convert/mfc13d/env/environment_E_D_Z MFCC_D_Z_FlatStart/mono{,16}**
- **../tools/steps/step-mono -FLATSTART -NUMMIXES {8,16} ../convert/mfc13d/env/environment_E_D_A_Z MFCC_D_A_Z_FlatStart/mono{,16}**

Then all of them were decoded with the beamwidth of 1000 to maximize the sentence number,

- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/FBK_Z_FlatStart/mono{,16} hmm84 FBK_Z_FlatStart/decode-mono-hmm{84, 164b1000}**
- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/FBK_D_A_Z_FlatStart/mono{,16} hmm84 FBK_D_A_Z_FlatStart/decode-mono-hmm{84, 164b1000}**
- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/MFCC_Z_FlatStart/mono{,16} hmm84 MFCC_Z_FlatStart/decode-mono-hmm{84, 164b1000}**
- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/MFCC_D_Z_FlatStart/mono{,16} hmm84 MFCC_D_Z_FlatStart/decode-mono-hmm{84, 164b1000}**
- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/MFCC_D_A_Z_FlatStart/mono{,16} hmm84 MFCC_D_A_Z_FlatStart/decode-mono-hmm{84, 164b1000}**

The LOG file inside the test folder of the decode folder showed the performance of the specific feature type with specific differential coefficient.

Consider the phone error rate (PER) is not shown in the file, it can be calculated by the following equation,

$$PER = 100\% - \%Accuracy$$

The word accuracy can help to compute the phone error rate.

TABLE 3

FBANK	Differential Coefficient	No Differential (_Z)		2nd Differential (_D_A_Z)	
	Number of Gaussians	8	16	8	16
Word	Hit	30922	31927	34850	36371
	Insertion	11716	12220	10940	10409
	%Correctness	59.79	61.73	67.38	70.32
	%Accuracy	37.13	38.05	46.23	50.20
	%Phone Error Rate	62.87	61.95	53.74	49.80

TABLE 4

MFCC_E	Differential Coefficient	No Differential (_Z)		1st Differential (D_Z)		2nd Differential (D_A_Z)	
	Number of Gaussians	8	16	8	16	8	16
Word	Hit	31970	32833	36250	37491	37515	38795
	Insertion	9747	10006	7478	7322	8168	7701
	%Correctness	61.81	63.48	70.09	72.49	72.53	75.01
	%Accuracy	42.97	44.13	55.63	58.33	56.74	60.12
	%Phone Error Rate	57.03	55.87	44.37	41.67	43.26	39.88

1.3.1 Differential Coefficients

Refer to Table 4, with the fixed number of Gaussians, the percentages of correctness and accuracy were increased with the addition of differential coefficients. Contrary to other parameters, the phone error rate decreased with the addition of differential coefficients.

This is because the differential coefficients appended the dynamic information to the static cepstral features [1]. This model can then have greater explanations to the speech dynamic information and gave a better performance.

1.3.2 Feature Types

MFCC_E had a better performance than FBANK. MFCC_E was found to have higher percentages of correctness and accuracy, and lower phone error rate.

This can be explained as follows. Since the filter bank coefficients computed are highly correlated [2], it can be problematic to GMM-HMMs since GMM-HMMs only assume that there are only dependencies between the hidden state and its observation states, and between the consecutive hidden states. The extra correlated information cannot fit into the assumption of the GMM-HMMs, and therefore, FBANK did not have a better performance.

Besides, MFCC_E applies the Discrete Cosine Transform (DCT) to decorrelate filter bank coefficients and make them less correlated. It is then more beneficial to GMM-HMMs training. Thus, MFCC_E feature type had a better performance than FBANK feature type.

1.3.3 Number of Gaussian mixtures

With the same differential coefficient and feature type, the percentages of correctness and accuracy were increased as the number of Gaussians was increased. Meanwhile, the phone error rate decreased. This showed the performance was improved with the increased number of Gaussians. It is reasonable because GMM-HMMs model relies on the Gaussian mixtures to explain the extracted features, the more the Gaussian mixtures are used, the better the model can explain the inputs.

It is interesting to note that, the number of insertions did not decrease but increased when the number of Gaussians was increased in the model without differential.

1.4 Triphone Decision Tree State Tying

The step-xwtri script is a function to build decision tree based cross-word triphone GMM-HMMs. The command in the terminal has the following standard,

- **../tools/steps/step-xwtri [-NUMMIXES n] [-ROVAL r] [-TBVAL t] [-TOOLSDIR path] (abspath_srcdir) (hmmdir) (systemdir)**

When building the model, the clustering thresholds must be set. -ROVAL controls the thresholds for outlier states removal and -TBVAL sets the minimum log-likelihood increase in tree growth.

Since the triphone model should be built on a well-performing front-end parameterization, the model with MFCC_E feature type and 2nd differential coefficient (_D_A) was utilized.

To compare how the clustering thresholds (-ROVAL and -TBVAL) control the final number of clustered states (can be seen from the LOG file in xwtri/hmm10/LOG), the following triphone models were built with different values of RO and TB.

- **../tools/steps/step-xwtri -NUMMIXES 8 -ROVAL 200 -TBVAL 800 $PWD/MFCC_D_A_Z_FlatStart/mono hmm14 MFCC_D_A_Z_FlatStart/xwtri200_800**
- **../tools/steps/step-xwtri -NUMMIXES 8 -ROVAL 100 -TBVAL 800 $PWD/MFCC_D_A_Z_FlatStart/mono hmm14 MFCC_D_A_Z_FlatStart/xwtri100_800**
- **../tools/steps/step-xwtri -NUMMIXES 8 -ROVAL 200 -TBVAL 1000 $PWD/MFCC_D_A_Z_FlatStart/mono hmm14 MFCC_D_A_Z_FlatStart/xwtri200_1000**

They were then decoded with the beamwidth 1000.

- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/MFCC_D_A_Z_FlatStart/xwtri200_800 hmm84 MFCC_D_A_Z_FlatStart/decode-xwtri-hmm84b1000R200T800**
- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/MFCC_D_A_Z_FlatStart/xwtri100_800 hmm84 MFCC_D_A_Z_FlatStart/decode-xwtri-hmm84b1000R100T800**
- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/MFCC_D_A_Z_FlatStart/xwtri200_1000 hmm84 MFCC_D_A_Z_FlatStart/decode-xwtri-hmm84b1000R200T1000**

TABLE 5

MFCC_D_A_Z	Monophone -NUMMIXES	8	8	16	8	8
	-ROVAL	200	200	200	100	200
	-TBVAL	800	800	800	800	1000
	Triphone -NUMMIXES	8	16	8	8	8
Number of cluster states		3119	3119	3119	3165	2165
Word	%Correctness	76.02	79.74	78.94	75.75	75.62
	%Accuracy	51.89	62.68	61.59	51.97	52.16

With the fixed TB value of 800, the decrease in RO value from 200 to 100 resulted in the increase of the number of clustered states. It also decreased the percentage of correctness from 76.02% to 75.75% but increased the percentage of accuracy from 51.89% to 51.97%. As the RO value was decreased, less outliner was classified, less penalty was introduced to the state which included the outliner, the number of cluster states then rose because more were allowed to be the cluster states.

With the fixed RO value of 200, the increase in TB value led to the reduced number of clustered states. It decreased the percentage of correctness from 76.02% to 75.62% but increased the percentage of accuracy from 51.89% to 52.16%. As the TB threshold increased, it became difficult to allow the phones to be separated out and to form a new cluster state. So, increasing TB value reduced the number of cluster states.

Since the triphone training is based on the hmm14 file of the monophone model, it is interesting to investigate how the number of Gaussians set to train the monophone model affects the performance of the triphone model.

Firstly, the monophone model with 8 Gaussians was trained, and then the triphone model was trained with this monophone model and 8 Gaussians.

Secondly, another triphone model was trained with the same monophone model but with 16 Gaussians.

Thirdly, another monophone model with 16 Gaussians was trained and then the triphone model was trained with the 16 Gaussians monophone model using 8 Gaussians.

The result showed the number of cluster states did not depend on the numbers of Gaussians defined in the monophone and triphone models. The number of cluster states remained 3119 even when the triphone model was based on the 16-Gaussians monophone model. But word correctness and accuracy depended on the number of Gaussians used in the monophone model. When the triphone model was now based on the 16-Gaussians monophone model, the word correctness went up to 79.74% and the accuracy rose to 62.68%

Noteworthy is the fact that the number of Gaussians defined in the triphone model had more effect on the performance than that in the monophone model. The word correctness and accuracy of the 16-Gaussians triphone based on the 8-Gaussians monophone were greater than those of the 8-Gaussians triphone based on the 16-Gaussians monophone.

2. Artificial Neural Network – Hidden Markov Models (ANN-HMMs)

2.1 Initial Setup and Testing

To train an ANN-HMM system, it is necessary to supply a frame-level alignment of the training data with the target state-level labels.

The step-align script is the function for alignment. Its command standard is given as follows,

- **../tools/steps/step-align [-TESTSET] [-TOOLSDIR path] (abspath_sourcedir) (hmmdir) (systemdir)**

The 'TESTSSET' option allows the system to align full test set rather than the train set.

In this section, the context -independent system was firstly utilized for the alignment. In the directory of /MLMI2/pracgmm/exp, multiple terminals were opened to align multiple flat-start monophone models with different feature types and differential coefficients by the following codes,

- **../tools/steps/step-align $PWD/FBK_D_A_Z_FlatStart/mono hmm84 FBK_D_A_Z_FlatStart/align-mono-hmm84**
- **../tools/steps/step-align $PWD/MFCC_{,D_,D_A_}Z_FlatStart/mono hmm84 MFCC_{,D_,D_A_}Z_FlatStart/align-mono-hmm84**

Then the shell variable pracgmmexp was set differently in different terminals,

- **pracgmmexp=~/MLMI2/pracgmm/exp/FBK_D_A_Z_FlatStart**
- **pracgmmexp=~/MLMI2/pracgmm/exp/MFCC_{,D_,D_A_}Z_FlatStart**

The step-dnntrain script is the function for building ANN-HMMs with different kinds of target. Its command has the following input standard,

- **../tools/steps/step-dnntrain [-h] [-VERBOSE] [-TOOLS TOOL] [-GPUID GPUID] [-MODELINI MODELINI] (SYS) (MLF) (HMM) (LST) (TGT)**

Specifying the GPUID allows the model training quicker and specifying the MODELINI allows the model to be trained with another network architecture.

Then moved to the directory /MLMI2/pracann/exp, trained the hybrid Deep Neural Network Model – Hidden Markov Models (DNN-HMM) by the following codes in multiple terminals,

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../tools/htefiles/DNN-7L.ReLU.FBANK_D_A_Z.ini -vvv ../convert/fbk25d/env/environment_D_A_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/dnntrainFBFB**
- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../tools/htefiles/DNN-7L.ReLU.MFCC_E_Z.ini -vvv ../convert/mfc13d/env/environment_E_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/dnntrainMFZMFZ**
- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../tools/htefiles/DNN-7L.ReLU.MFCC_E_D_Z.ini -vvv ../convert/mfc13d/env/environment_E_D_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/dnntrainMFDZMFDZ**
- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../tools/htefiles/DNN-7L.ReLU.MFCC_E_D_A_Z.ini -vvv ../convert/mfc13d/env/environment_E_D_A_Z**

$pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/dnntrainMFMF

And they were decoded. The following code showed how one of the DNN model was decoded.

- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/MHO/dnntrainMFMF hmm0 MHO/decode-dnntrain_sing_MFDAZ44**

This decoding standard is universal to all other ANN-HMMs, such as the RNN model, the TDNN model, etc. So, it will not be listed again.

TABLE 6

Activation		ReLU			
Feature type	Differential Coefficient	FBANK_D_A_Z	MFCC_E_Z	MFCC_E_D_Z	MFCC_E_D_A_Z
Validation Accuracy		63.26	56.24	62.30	63.50
Training Accuracy		74.41	72.80	78.88	81.38
Without insertion penalty					
Sentence	%Correctness	0.15	0.07	0.15	0.15
Word	%Correctness	80.50	78.50	80.52	81.38
	%Accuracy	74.26	69.03	73.44	74.33
With insertion penalty -INSWORD -8					
Sentence	%Correctness	0.15	0.15	0.22	0.37
Word	%Correctness	77.42	74.51	77.56	78.47
	%Accuracy	75.19	71.56	75.10	75.85

Similar observations from section 1.2 can be obtained in this section 2.1. With the same feature type, the addition of differential coefficient can lead to higher percentages of correctness and accuracy. It also improved the validation and training accuracies.

With the insertion penalty, the word correctness decreased but the word accuracy increased. However, it is interesting to note that the sentence correctness increased slightly when the penalty was introduced.

The accuracy difference between the validation set and the train set tended to increase with the addition of the differential coefficients. The one with more coefficients tended to have a higher training accuracy. The FBANK feature type model was also more likely to have a smaller accuracy gap between the validation and train sets than the MFCC_E feature type model.

As the training accuracy was always greater than the validation accuracy, some can claim that the model was overfitted. It meant all the models trained in this report were overfitting. The addition of differential coefficients and the use of MFCC_E feature type made the model more overfit.

It is important to note that the front-end parameterization of the flat-start file, environment file and the DNN train model should be matched. For example, the DNN model for 2nd differential FBANK feature type should be trained with the 2nd differential FBANK feature type, as shown in follows,

- **pracgmmexp=~/MLMI2/pracgmm/exp/FBK_D_A_Z_FlatStart**

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../tools/htefiles/DNN-7L.ReLU.FBANK_D_A_Z.ini -vvv ../convert/fbk25d/env/environment_D_A_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/dnntrainMFMF**

The underlined parts of the codes above showed the feature type and the differential coefficient must be matched. If not, when the DNN model for 2nd differential MFCC_E feature type was trained with the 2nd differential FBANK feature type training set as follows,

- **pracgmmexp=~/MLMI2/pracgmm/exp/FBK_D_A_Z_FlatStart**
- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../tools/htefiles/DNN-7L.ReLU.MFCC_E_D_A_Z.ini -vvv ../convert/mfc13d/env/environment_E_D_A_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/dnntrainMFMF**

It gave much lower correctness and accuracy as shown in Table 7. Both the word correctness and accuracy dropped to 6.06% which were significantly lower than those stated in Table 6.

TABLE 7

Alignment		FBANK_D_A_Z
Environment and DNN model		MFCC_E_D_A_Z
%Validation Accuracy		56.96
%Training Accuracy		64.35
Without insertion penalty		
Sentence	%Correctness	0.00
Word	%Correctness	6.06
	%Accuracy	6.06

2.2 Single Hidden Layers Experiments

To conduct the single hidden layer experiments, all the network files from pracann/tools/hetfiles folder were copied to pracann/network folder.

```
ycl60@mlsalt-cpu1:~/MLMI2/pracann/tools/htefiles$ ls
DNN-7L.ReLU.FBANK_D_A_Z.ini                DNN-7L.sigmoid.pretrain4.MFCC_E_D_A_Z.ini
DNN-7L.ReLU.FBANK_D_Z.ini                  DNN-7L.sigmoid.pretrain4.MFCC_E_D_Z.ini
DNN-7L.ReLU.FBANK_Z.ini                    DNN-7L.sigmoid.pretrain4.MFCC_E_Z.ini
DNN-7L.ReLU.MFCC_E_D_A_Z.ini               DNN-7L.sigmoid.pretrain5.FBANK_D_A_Z.ini
DNN-7L.ReLU.MFCC_E_D_Z.ini                 DNN-7L.sigmoid.pretrain5.FBANK_D_Z.ini
DNN-7L.ReLU.MFCC_E_Z.ini                   DNN-7L.sigmoid.pretrain5.FBANK_Z.ini
DNN-7L.sigmoid.FBANK_D_A_Z.ini             DNN-7L.sigmoid.pretrain5.MFCC_E_D_A_Z.ini
DNN-7L.sigmoid.FBANK_D_Z.ini               DNN-7L.sigmoid.pretrain5.MFCC_E_D_Z.ini
DNN-7L.sigmoid.FBANK_Z.ini                 DNN-7L.sigmoid.pretrain5.MFCC_E_Z.ini
DNN-7L.sigmoid.MFCC_E_D_A_Z.ini            DNN-7L.sigmoid.pretrain6.FBANK_D_A_Z.ini
DNN-7L.sigmoid.MFCC_E_D_Z.ini              DNN-7L.sigmoid.pretrain6.FBANK_D_Z.ini
DNN-7L.sigmoid.MFCC_E_Z.ini                DNN-7L.sigmoid.pretrain6.FBANK_Z.ini
DNN-7L.sigmoid.pretrain3.FBANK_D_A_Z.ini   DNN-7L.sigmoid.pretrain6.MFCC_E_D_A_Z.ini
DNN-7L.sigmoid.pretrain3.FBANK_D_Z.ini     DNN-7L.sigmoid.pretrain6.MFCC_E_D_Z.ini
DNN-7L.sigmoid.pretrain3.FBANK_Z.ini       DNN-7L.sigmoid.pretrain6.MFCC_E_Z.ini
DNN-7L.sigmoid.pretrain3.MFCC_E_D_A_Z.ini  RNN-1L.ReLU.FBANK_D_A_Z.ini
DNN-7L.sigmoid.pretrain3.MFCC_E_D_Z.ini    RNN-1L.ReLU.FBANK_D_Z.ini
DNN-7L.sigmoid.pretrain3.MFCC_E_Z.ini      RNN-1L.ReLU.FBANK_Z.ini
DNN-7L.sigmoid.pretrain4.FBANK_D_A_Z.ini   RNN-1L.ReLU.MFCC_E_D_A_Z.ini
DNN-7L.sigmoid.pretrain4.FBANK_D_Z.ini     RNN-1L.ReLU.MFCC_E_D_Z.ini
DNN-7L.sigmoid.pretrain4.FBANK_Z.ini       RNN-1L.ReLU.MFCC_E_Z.ini
```

Then using the 'DNN-7L.ReLU.FBANK_D_A_Z.ini' and 'DNN-7L.ReLU.MFCC_E_D_A_Z.ini' files, they were modified to single hidden layer. The examples were shown below. The left one was named as 'DNNReLU_sing_MFDAZ_44.ini' and the right one was named as 'DNNReLU_sing_FBDAZ_44.ini'.

```
# model definition part
[ModelSet]
@ParmKind = <MFCC_E_D_A_Z>
@ParmDim = 39
@HiddenActFunc = ReLU
@HiddenLayerDim = 500
InputObservation.Type = @ParmKind
InputObservation.Dim = @ParmDim
[NeuralNet:DNN]
Layer2.Name = layerin
Layer3.Name = layerout
[Layer:layerin]
Kind = FC
FeatureElement.Num = 1
FeatureElement1.Dim = @ParmDim
FeatureElement1.ContextShiftSet = {-4,-3,-2,-1,0,+1,+2,+3,+4}
FeatureElement1.Source = @ParmKind
HasBias = True
OutputDim = @HiddenLayerDim
ActivationFunction = @HiddenActFunc
[Layer:layerout]
Kind = FC
FeatureMixture.Num = 1
FeatureElement1.Dim = @HiddenLayerDim
FeatureElement1.ContextShiftSet = {0}
FeatureElement1.Source = layerin
HasBias = True
OutputDim = @auto
ActivationFunction = Softmax
```

```
# model definition part
[ModelSet]
@ParmKind = <FBANK_D_A_Z>
@ParmDim = 72
@HiddenActFunc = ReLU
@HiddenLayerDim = 500
InputObservation.Type = @ParmKind
InputObservation.Dim = @ParmDim
[NeuralNet:DNN]
Layer2.Name = layerin
Layer3.Name = layerout
[Layer:layerin]
Kind = FC
FeatureElement.Num = 1
FeatureElement1.Dim = @ParmDim
FeatureElement1.ContextShiftSet = {-4,-3,-2,-1,0,+1,+2,+3,+4}
FeatureElement1.Source = @ParmKind
HasBias = True
OutputDim = @HiddenLayerDim
ActivationFunction = @HiddenActFunc
[Layer:layerout]
Kind = FC
FeatureMixture.Num = 1
FeatureElement1.Dim = @HiddenLayerDim
FeatureElement1.ContextShiftSet = {0}
FeatureElement1.Source = layerin
HasBias = True
OutputDim = @auto
ActivationFunction = Softmax
```

2.2.1 Feature types

To compare the performance difference between the feature types, the flat-start monophone models of MFCC and FBK with 2nd differential coefficient were aligned.

Then the DNN models were trained using the single-layer scripts above and decoded.

TABLE 8

Feature types		FBANK_D_A_Z	MFCC_E_D_A_Z
Validation Accuracy		59.84	60.74
Training Accuracy		65.60	67.24
Sentence	%Correctness	0.15	0.22
Word	%Correctness	77.54	77.47
	%Accuracy	70.82	70.97

Models built on FBANK and MFCC feature types performed quite similarly. In general, the MFCC model performed slightly better than the FBANK one in terms of validation accuracy, training accuracy, sentence correctness and word accuracy. However, the FBANK model had a slightly higher word correctness.

Compared to the GMM-HMMs, the DNN-HMMs seemed less susceptible to high correlated FBANK coefficients [2]. Therefore, the DNN-HMMs were relatively independent of the feature types.

2.2.2 Differential coefficients

To compare the performance difference between the differential coefficients, the flat-start monophone models of MFCC without and with 2nd differential coefficients were aligned.

TABLE 9

Differential Coefficients		MFCC_E_Z	MFCC_E_D_A_Z
Validation Accuracy		54.42	60.74
Training Accuracy		58.88	67.24
Word	%Correctness	74.43	77.47
	%Accuracy	65.23	70.97

Table 9 showed the addition of differential coefficient leads to better performance. The one with 2nd differential had higher percentages of validation accuracy, training accuracy, word correctness and word accuracy than that without differential. It is reasonable because the additional dynamic speech information helps the model to explain the phones sequencing better.

The addition of differential coefficients induced the greater difference between the train accuracy and the validation accuracy.

2.2.3 Context Width

To investigate how the context width affects the performance, the following .ini files were created.

```
DNNReLU_sing_FBDAZ_11.ini
DNNReLU_sing_FBDAZ_34.ini
DNNReLU_sing_FBDAZ_43.ini
DNNReLU_sing_FBDAZ_44.ini
DNNReLU_sing_FBZ_44.ini
DNNReLU_sing_MFDAZ_11.ini
DNNReLU_sing_MFDAZ_14.ini
DNNReLU_sing_MFDAZ_41.ini
DNNReLU_sing_MFDAZ_44.ini
DNNReLU_sing_MFZ_44.ini
```

These files were the copies of 'DNNReLU_sing_FBDAZ_44.ini' and 'DNNReLU_sing_MFDAZ_44.ini' files, but they were all modified by changing the components of the context shift set in the first input layer to different values.

```
[Layer:layerin]
Kind = FC
FeatureElement.Num = 1
FeatureElement1.Dim = @ParmDim
FeatureElement1.ContextShiftSet = {-4,-3,-2,-1,0,+1,+2,+3,+4}
FeatureElement1.Source = @ParmKind
```

The last integer of the filename defined the context shift set in the layerin of the .ini. file. For example, _44 denotes {-4,-3,-2,-1,0,1,2,3,4}, _14 denotes {-1,0,1,2,3,4} and _0 defines {0} no additional acoustic context.

For the one with FBANK feature type and 2nd differential, different DNN models were trained with different context windows, and their performances were listed in Table 10.

TABLE 10

Front-end parameterization		FBANK_D_A_Z				
Context window		11	34	43	44	0
Validation Accuracy		58.09	59.23	59.32	59.84	53.67
Training Accuracy		62.59	65.37	64.92	65.60	57.01
Word	%Correctness	76.71	77.42	77.35	77.54	75.19
	%Accuracy	68.55	70.60	70.48	70.82	64.65
Sentence	%Correctness	0.00	0.00	0.07	0.15	0.00

For the one with MFCC feature type and 2nd differential, different DNN models were trained with different context windows. The results were stated in Table 11.

TABLE 11

Front-end parameterization		MFCC_E_D_A_Z				
Context window		11	14	41	44	0
Validation Accuracy		59.77	60.21	60.21	60.74	55.35
Training Accuracy		64.54	65.66	66.19	67.24	58.39
Word	%Correctness	76.97	76.81	77.52	77.47	75.09
	%Accuracy	69.29	69.83	70.35	70.97	65.09
Sentence	%Correctness	0.00	0.15	0.07	0.22	0.00

Tables 10 and 11 both showed extending context window helped to improve the performance in terms of validation accuracy, training accuracy, word accuracy and correctness. But the accuracy difference between validation and train sets increased with the context width. When there was no additional acoustic context, the accuracy difference was around 3-4%; when the context width increased to 8, the difference became 6%. This may indicate the model with wider context window tended to overfit.

2.3 Adding Multiple Layers

Model with MFCC_E feature type and 2nd differential coefficient was chosen for the experiment in this section. To begin with, 'DNNReLU_sing_MFDAZ_44.ini' in the directory of /MLMI2/pracann/network was utilized as the standard 1-layer model. Based on this model, multiple layers were added between the input and output layers to form the n-layers models. Each model was then trained and decoded. The overall result was listed in Table 12.

TABLE 12

Front-end parameterization		MFCC_E_D_A_Z						
Layer		1	2	3	4	5	6	7
Validation Accuracy		60.74	62.52	62.96	64.30	63.50	63.50	5.17
Training Accuracy		67.24	73.70	79.34	77.68	81.38	76.17	6.04
Sentence	%Correctness	0.22	0.15	0.07	0.30	0.15	0.15	0.00
Word	%Correctness	77.47	79.35	80.60	80.87	81.38	80.36	7.24
	%Accuracy	70.97	73.00	74.18	74.70	74.33	74.03	7.24

The performance was improved with the number of hidden layers until the number of layers became 5. Then the performance begun to decline with the number of layers. As the number of layers was increased to 7, the increase in parameters also led to parameter estimation issues. From Table 12, the performance became bad when the model had 7 hidden layers.

2.4 Triphone Target Units

In this section, the 5 layers model of MFCC_E feature type with 2nd differential coefficient was chosen as the start. The triphone GMM-HMM was aligned, the triphone DNN-HMMs with different number of layers were then trained and decoded. The result was listed in Table 13.

- **../tools/steps/step-align $PWD/MFCC_D_A_Z_FlatStart/xwtri200_800 hmm84 MFCC_D_A_Z_FlatStart/align-xwtri-hmm84**
- **pracgmmexp=~/MLMI2/pracgmm/exp/MFCC_D_A_Z_FlatStart**
- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/DNNReLU_{5,6,7,8}_MFDAZ_44.ini -vvv ../convert/mfc13d/env/environment_E_D_A_Z $pracgmmexp/align-xwtri-hmm84/align/timit_train.mlf $pracgmmexp/xwtri200_800/hmm84/MMF $pracgmmexp/xwtri200_800/hmms.mlist MHO/dnntrainX{5,6,7,8}**
- **../tools/steps/step-decode -BEAMWIDTH 1000 $PWD/MHO/dnntrainX{5,6,7,8} hmm0 MHO/decode-dnntrainX{5,6,7,8}**

TABLE 13

Front-end parameterization		MFCC_D_A_Z			
Layer		5	6	7	8
Validation Accuracy		50.70	50.47	49.75	5.47
Training Accuracy		71.32	69.61	69.06	4.62
Sentence	%Correctness	0.37	0.37	0.30	0.00
Word	%Correctness	82.92	83.04	82.86	6.18
	%Accuracy	75.61	75.63	74.76	6.18

In comparison to the model with 5 layers from Table 12, the triphone model had better performances in terms of sentence correctness, word correctness and word accuracy. But the monophone model performed better in validation accuracy and training accuracy.

Similar to Section 2.3, 1-3 extra hidden layers were added to the 5-layers model. The validation and training accuracies started to decrease after the 5-layers model. The sentence correctness, word correctness and word accuracy begun to decrease after the 6-layers model. However, the parameter estimation problem did not occur in the 7-layers model, which was different from the 7-layers monophone model.

When the number of hidden layers became 8, the performance became bad because the number of parameters became difficult to estimate. It only had 6.18% word correctness and accuracy. Therefore, the parameter estimation problem occured earlier in the context-independent model than in the context-dependent model. It maybe due to the fact that the context-independent target had less information for the weights to represent, if the number of layers exceeded a certain limit, there were extra weights that cannot be trained properly, causing the decline in the performance.

2.5 Recurrent Neural Network – Hidden Markov Models (RNN-HMMs)

This section continued with the MFCC_E feature type and 2nd differential coefficient front-end parameterization.

- **pracgmmexp=~/MLMI2/pracgmm/exp/MFCC_D_A_Z_FlatStart**

To test how the unfolding affects the performance, the value of 'UnfoldValue' in the script of 'RNN-1L.ReLU.MFCC_E_D_A_Z.ini' was changed to 10.

The RNN model was firstly trained with the context independent target,

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/RNNReLU_{20,10}.ini -vvv ../convert/mfc13d/env/environment_E_D_A_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/rnn{20,10}X**

They were then decoded to give the following result,

TABLE 14

Monophone RNN model			
Number of folding		10	20
Validation Accuracy		64.40	65.46
Training Accuracy		77.60	79.88
Sentence	%Correctness	0.15	0.15
Word	%Correctness	80.83	81.23
	%Accuracy	74.64	75.90

The monophone RNN model with more folding showed better performance.

Then the model was trained with the triphone target,

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/RNNReLU_{20,10}.ini -vvv ../convert/mfc13d/env/environment_E_D_A_Z $pracgmmexp/align-xwtri-hmm84/align/timit_train.mlf $pracgmmexp/xwtri200_800/hmm84/MMF $pracgmmexp/xwtri200_800/hmms.mlist MHO/rnn{20,10}D**

After decoding, the result was listed in Table 15.

TABLE 15

Triphone RNN model			
Number of folding		10	20
Validation Accuracy		51.63	54.95
Training Accuracy		71.10	75.38
Sentence	%Correctness	0.22	0.30
Word	%Correctness	83.35	83.18
	%Accuracy	76.17	77.08

RNN model with less unfolding was decoded with less time. With more unfolding, the RNN model performed better in all aspects.

The triphone state RNN model had the higher percentages of sentence correctness, word correctness and word accuracy than the monophone RNN model. But the context-independent RNN model had the higher validation and training accuracies than the context-dependent model.

3. Extended RNN models

3.1 Addition of fully connected layers

Since the supplied LSTM and TDNN scripts were only for the front-end parameterization of FBANK feature type and 1st differential coefficient, the 1st differential FBANK monophone RNN model was built for performance comparison to remain fairness in this section. Thus, the 1st differential FBANK GMM-HMM was firstly aligned.

- **../tools/steps/step-align $PWD/FBK_D_Z_FlatStart/mono hmm84 FBK_D_Z_FlatStart/align-mono-hmm84**
- **pracgmmexp=~/MLMI2/pracgmm/exp/FBK_D_Z_FlatStart**

To investigate how the number of hidden layers added before the recurrent layer affects the performance, using the script of 'RNN-1L.ReLU.FBANK_D_Z.ini', different number of fully connected layers were added before the recurrent layer as shown in follows,

```
Layer5.Name = layer5
Layer6.Name = layerout
[NVector:ZeroVec]
Length = @RNNLayerOutputDim
[Layer:layerin]
Kind = FC
FeatureElement.Num = 1
FeatureElement1.Dim = @FBKInputDim
FeatureElement1.ContextShiftSet = {-4,-3,-2,-1,0,1,2,3,4}
FeatureElement1.Source = <FBANK_D_Z>
HasBias = True
OutputDim = @HiddenLayerDim
ActivationFunction = ReLU
[Layer:layer3]
Kind = FC
FeatureElement.Num = 1
FeatureElement1.Dim = @HiddenLayerDim
FeatureElement1.ContextShiftSet = {0}
FeatureElement1.Source = layerin
HasBias = True
OutputDim = @HiddenLayerDim
ActivationFunction = ReLU
[Layer:layerrnn4]
Kind = RNN
FeatureMixture.Num = 2
FeatureElement1.Dim = @HiddenLayerDim
FeatureElement1.ContextShiftSet = {+5}
FeatureElement1.Source = layer3
FeatureElement2.Dim = @RNNLayerOutputDim
```

Then the modified scripts were run, trained and decoded.

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/RNNReLU5_FBDZ_FCR{0,1,2,3}.ini -vvv ../convert/fbk25d/env/environment_D_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/rnnFBDZ_FCR{0,1,2,3}**

The result was illustrated in Table 16.

TABLE 16

Monophone FBANK_D_Z RNN model (unfolding: 5)					
Layer before recurrent layer		0	1	2	3
Validation Accuracy		54.97	62.59	62.48	62.37
Training Accuracy		64.94	78.23	79.08	80.84
Sentence	%Correctness	0.07	0.15	0.07	0.00
Word	%Correctness	78.36	80.85	80.81	80.63
	%Accuracy	68.69	74.10	73.63	73.04

The performance did not generally increase with the number of layers added before the recurrent layer. When the number of layers increased from 0 to 1, the performance was improved. When the number further increased to 2 or 3, the validation accuracy, sentence correctness, word accuracy and word correctness conversely declined. The models with more layers also tended to overfit. The accuracy difference between train and validation sets increased from around 10% (0 layer) to 18% (3 layers).

Similarly, with the 'RNN-1L.ReLU.FBANK_D_Z.ini', different number of layers were added before the output layer to investigate how it affects the performance. The following codes showed how it was done.

```
FeatureElement2.ContextShiftSet = {0}
FeatureElement2.Source = ~V ZeroVec
UnfoldValue = 5
RecurrencyPeriod = 1
OutputDim = @RNNLayerOutputDim
ActivationFunction = ReLU
[Layer:layer4]
Kind = FC
FeatureMixture.Num = 1
FeatureElement1.Dim = @RNNLayerOutputDim
FeatureElement1.ContextShiftSet = {0}
FeatureElement1.Source = layerrnn3
HasBias = True
OutputDim = @HiddenLayerDim
ActivationFunction = ReLU
```

```
[Layer:layer5]
Kind = FC
FeatureMixture.Num = 1
FeatureElement1.Dim = @HiddenLayerDim
FeatureElement1.ContextShiftSet = {0}
FeatureElement1.Source = layer4
HasBias = True
OutputDim = @HiddenLayerDim
ActivationFunction = ReLU
[Layer:layer6]
Kind = FC
FeatureMixture.Num = 1
FeatureElement1.Dim = @HiddenLayerDim
FeatureElement1.ContextShiftSet = {0}
```

The scripts were run with step-dnntrain and tested with step-decode.

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/RNNReLU5_FBDZ_FCO{1,2,3}.ini -vvv ../convert/fbk25d/env/environment_D_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/rnnFBDZ_FCO{1,2,3}**

The result was shown in Table 17.

TABLE 17

Monophone FBANK_D_Z RNN model (unfolding: 5)					
Layer before output layer		0	1	2	3
Validation Accuracy		54.97	62.06	62.26	5.40
Training Accuracy		64.94	78.86	79.18	4.49
Sentence	%Correctness	0.07	0.07	0.07	0.00
Word	%Correctness	78.36	80.69	81.35	5.43
	%Accuracy	68.69	73.74	74.41	5.43

The performance was improved with the number of fully-connected layers added before the output layer until the number reached 2. The 3-layer model was performing very badly, this may be due to the parameter estimation issue.

In summary, deeper hidden layers did not guarantee the performance improvement. There was always a limit to the number of hidden layers can be added to the neural networks without worsening the performance. One of the causes to this observation was the fact that the information propagation via many hidden layers can introduce information loss which worsened the performance.

3.2 Addition of recurrent layers

The 'RNN-1L.ReLU.FBANK_D_Z.ini' was modified with more recurrent layers. The following codes showed how it was modified.

```
FeatureElement2.Source = ~V ZeroVec
UnfoldValue = 5
RecurrencyPeriod = 1
OutputDim = @RNNLayerOutputDim
ActivationFunction = ReLU
[Layer:layerrnn5]
Kind = RNN
FeatureMixture.Num = 2
FeatureElement1.Dim = @RNNLayerOutputDim
FeatureElement1.ContextShiftSet = {+5}
FeatureElement1.Source = layerrnn4
FeatureElement2.Dim = @RNNLayerOutputDim
FeatureElement2.ContextShiftSet = {0}
FeatureElement2.Source = ~V ZeroVec
UnfoldValue = 5
```

```
UnfoldValue = 5
RecurrencyPeriod = 1
OutputDim = @RNNLayerOutputDim
ActivationFunction = ReLU
[Layer:layer6]
Kind = FC
FeatureMixture.Num = 1
FeatureElement1.Dim = @RNNLayerOutputDim
FeatureElement1.ContextShiftSet = {0}
FeatureElement1.Source = layerrnn5
HasBias = True
OutputDim = @HiddenLayerDim
ActivationFunction = ReLU
[Layer:layerout]
Kind = FC
```

Models with different number of recurrent layers were trained and tested.

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/RNNReLU5_FBDZ_Rec{2,3}.ini -vvv ../convert/fbk25d/env/environment_D_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/rnnFBDZ_Rec{2,3}**

The result was listed in Table 18.

TABLE 18

Monophone FBANK_D_Z RNN model (unfolding: 5)				
Recurrent layer		1	2	3
Validation Accuracy		54.97	51.57	26.29
Training Accuracy		64.94	61.34	34.46
Sentence	%Correctness	0.07	0.00	0.00
Word	%Correctness	78.36	77.02	63.62
	%Accuracy	68.69	66.76	38.87

Adding more recurrent layers led to worse performance as all the accuracies and correctnesses dropped with the increase of recurrent layers. This may be because of vanishing or exploding gradient.

One of the advantages of using RNN layers is the fact that they can link the previous information to the present recognition task. But sometimes the previous information may not be useful, the assumption of dependency just introduces the poor training efficiency and result, which can be shown by the poor performance from the models with more recurrent layers.

3.3 Investigation of a Long Short Term Memory (LSTM) based model

To solve the problem of vanishing or exploding gradient and long-term dependencies, the LSTM based model was developed [3].

To investigate how it performed, the following model was trained and tested. The performance was listed in Table 19.

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/LSTM-{1,2}L.FBANK_D_Z.ini -vvv ../convert/fbk25d/env/environment_D_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/FBDZLSTM{,2}**

TABLE 19

Monophone LSTM based model			
layer		1	2
Validation Accuracy		62.33	61.70
Training Accuracy		73.17	82.28
Sentence	%Correctness	0.22	0.15
Word	%Correctness	79.45	80.10
	%Accuracy	73.68	73.68

In comparison to the model with 1 recurrent layer (the best performing one) from Table 18, LSTM based model performed better. And it did outperform the RNN model with 3 or more recurrent layers and solved the problem of long-term dependencies since the single-layer LSTM based model was approximate to the RNN model with multiple recurrent layers.

When the number of layers in LSTM was increased to 2, the training and decoding process became very slow. 'LSTM-2L.FBANK_D_Z.ini' model had the longest training time (around 3500s per epoch) over all other models.

Unlike adding recurrent layers to the RNN model, adding more recurrent-alike layers to the LSTM based model did result in little performance improvement with regards to the word correctness.

3.4 Time Delay Neural Network (TDNN)

There was another network architecture which attempted to model the long term temporal dependencies in short-term speech features by the time delay method, it is called the feedforward TDNN model [4]. It has the advantage of using less training time to achieve the performance of RNN or LSTM based models without using any recurrent layer.

To investigate the performance of the TDNN, the following 'TDNN.ReLU.FBANK_D_Z.ini' file was utilized for the DNN training,

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/TDNN.ReLU.FBANK_D_Z.ini -vvv ../convert/fbk25d/env/environment_D_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/TDNN**

It was then decoded, and the result was listed in Table 20.

TABLE 20

Monophone TDNN model		
Validation Accuracy		66.42
Training Accuracy		78.67
Sentence	%Correctness	0.22
Word	%Correctness	82.09
	%Accuracy	77.95

In comparison to the LSTM based model, TDNN performed better in all aspects. TDNN is the best performing model ever tested in this report. This might be due to the fact that TDNN does not contain any recurrent layer, so it can be trained easier and gives better performances.

3.5 Residual Network (ResNet) with TDNN

The shallow depth of TDNN models limits the modelling capability. To solve this problem, some researchers [5] suggested that TDNN with a ResNet connection can reduce the WER by 6-9% relatively. To investigate the performance of the TDNN, the following 'ResNetTDNN.ReLU.FBANK_D_Z.ini' file was utilized to train the DNN model,

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/ResNetTDNN.ReLU.FBANK_D_Z.ini -vvv ../convert/fbk25d/env/environment_D_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/ResNet**

It was decoded, and the result was listed in Table 21.

TABLE 21

Monophone ResNet-TDNN model		
Validation Accuracy		66.83
Training Accuracy		79.03
Sentence	%Correctness	0.52
Word	%Correctness	82.13
	%Accuracy	77.59

The ResNet-TDNN model did show a little performance relative to the TDNN model. But the TDNN model did still have the higher word accuracy. This did not show as much improvement as mentioned in the research paper.

3.6 Investigation of a 'mysterious' HORNNP model

In the htefile, there is a file named 'HORNNP-2L.ReLU.FBANK_D_Z.ini' which has not been mentioned in the document 'MLMI2_2018_practical_v0'.

To investigate how this model performs, the following code was run.

- **../tools/steps/step-dnntrain -GPUID 0 -MODELINI ../network/HORNNP-2L.ReLU.FBANK_D_Z.ini -vvv ../convert/fbk25d/env/environment_D_Z $pracgmmexp/align-mono-hmm84/align/timit_train.mlf $pracgmmexp/mono/hmm84/MMF $pracgmmexp/mono/hmms.mlist MHO/HORNNP**

The model was then decoded, the result was listed in Table 22.

TABLE 22

Monophone HORNNP model		
Validation Accuracy		64.66
Training Accuracy		72.24
Sentence	%Correctness	0.22
Word	%Correctness	80.69
	%Accuracy	75.56

This model did not outperform the TDNN model but did have better performances than the single layer RNN model.

4. Conclusions

The monophone and triphone models with MFCC_E feature type and 2nd differential coefficient performed better. In the context of GMM-HMMs, the performance was improved with the number of Gaussians. The triphone model did not necessarily outperform the monophone model, but only when number of Gaussians was increased, the triphone model can perform better.

In the ANN-HMMs, the performances of DNN models built upon the FBANK and the MFCC_E feature types were similar. This might be because the DNN is more independent of feature types than the GMM. But the higher order of differential coefficient still played a role of improving the performance for the DNN. The DNN model with wider context window [6] was also tested to have a greater performance. The performance was improved with more hidden layers, but when too much was added, it became difficult to estimate the parameters and the model performed worse. The parameter estimation problem occured earlier in the context-independent model than the context-dependent model. Wider context window and deeper layers made the model easier to overfit. DNN based on triphone target outperformed that based on monophone in terms of word accuracy and correctness, but DNN based on monophone had greater validation and train accuracies. The RNN-HMMs had a better performance in all aspects than the DNN-HMMs.

During the investigation of RNN models, the addition of hidden layers before and after the recurrent layer was shown to improve the performance. But there was a limit for the number of hidden layers can be added. Once the limit was exceeded, the performance of the model begun to decline. When more recurrent layers were added, the model also performed worse probably because of the vanishing/exploding gradient. Other neural network architectures, such as LSTM based model, TDNN and HORNNP, were also tested. They both had better performances than most of the RNN models tested.

ResNet-TDNN was the best performing model tested in this report. This is because it uses the previous information to help the present recognition task without using recurrent layers which may induce the training complexity. It also uses the ResNet to further deepen its depth and improve the performance.

5. References

[1] The International Computer Science Institute (ICSI), Berkeley. Delta and Acceleration Coefficients. http://www1.icsi.berkeley.edu/Speech/docs/HTKBook/node65_mn.html. [16/12/2018]

[2] Haytham Fayek. Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between. Last update: 21/04/2016. https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html. [16/12/2018]

[3] Christopher Olah. Understanding LSTM Networks. Last update: 27/08/2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [15/12/2018]

[4] Vijayaditya Peddinti, Daniel Povey, Sanjeev Khudanpur, Johns Hopkins University. A time delay neural network architecture for efficient modelling of long temporal contexts. http://speak.clsp.jhu.edu/uploads/publications/papers/1048_pdf.pdf. [16/12/2018]

[5] F.L. Kreyssig, C.Zhang, P.C. Woodland, University of Cambridge. Improved TDNNs Using Kernels and Frequency Dependent Grid RNNs. http://mi.eng.cam.ac.uk/~flk24/doc/icassp_2018.pdf. [16/12/2018]

[6] Deividas Eringis, Gintautas Tamulevicius, Vilnius University Institute of Mathematics and Informatics. Improving Speech Recognition Rate through Analysis Parameters. https://content.sciendo.com/downloadpdf/journals/ecce/5/1/article-p61.xml [16/12/2018]