TIMIT Speech Recognition

Author: Tatiana Matejovicova
University of Cambridge

17-12-2018

Contents

1 Introduction **2**

2 Methods **2**
2.1 Feature types . . . 2
2.2 TIMIT database . . . 2
2.3 Phones and subphones . . . 2
2.4 Training and Prediction . . . 2
2.5 Word Error Rate Metric . . . 3

3 GMM-HMMs **4**
3.1 Insertion Penalty . . . 4
3.2 Training Initialisation . . . 4
3.3 Front-End Parametrisation . . . 5
3.4 Triphone Decision Tree State Tying . . . 7

4 ANN-HMMs **8**
4.1 Insertion Penalty . . . 9
4.2 Single Hidden Layer ANNs . . . 9
4.3 Multiple Hidden Layer DNNs . . . 11
4.4 Triphone Target Units . . . 12
4.5 RNN-HMMs . . . 13

5 Discussion **13**

6 Conclusion **14**

References **15**

7 Appendix **16**

1 Introduction

The goal of this project is to compare the performance of phone-based continuous speech recognition systems based on Hidden Markov Models (HMMs) in various settings. In the first part of the project Gaussian Mixture Models (GMMs) are used to model the state output distributions and in the second part Deep Neural Networks (DNNs) are used in a DNN-HMM hybrid model. The TIMIT Acoustic-Phonetic speech database [3] is used for training to construct a speaker independent phone recognition system.

2 Methods

2.1 Feature types

Two feature types for training are considered.

- FBANK: 24 mel channels with normalised log energy
- $\mathrm{MFCC}_{\mathrm{E}}$: 12 MFCCs with normalised log energy

MFCC features extend FBANK by extracting the first 12 cepstral values. Consequently the variance of different features is uncorrelated and the vocal filter is separated from the vocal source.

Both types of features are provided in the following versions.

- Sentence-based mean removal ($\mathrm{FBANK}_{\mathrm{Z}}$, $\mathrm{MFCC}_{\mathrm{EZ}}$)
- Sentence-based mean removal and first differentials ($\mathrm{FBANK}_{\mathrm{DZ}}$, $\mathrm{MFCC}_{\mathrm{EDZ}}$)
- Sentence-based mean removal, first and second differentials ($\mathrm{FBANK}_{\mathrm{DAZ}}$, $\mathrm{MFCC}_{\mathrm{EDAZ}}$)

2.2 TIMIT database

TIMIT is a corpus of 6300 read sentences from 630 speakers with 10 sentences each that were designed specifically for the corpus. All sentences are phonetically hand-labelled in 100ns units. In this project 1344 randomly selected sentences are used solely for testing.

2.3 Phones and subphones

Original 61 TIMIT phones are mapped to the CMU 48 phones. In the model each phone is further divided into three subphones.

2.4 Training and Prediction

Training of HMM speech systems is performed using the Baum-Welch EM algorithm with the emission distributions modelled either by GMMs (section 3) or DNNs (section 4). Baum-Welch is run 4 times for each desired setting.

Prediction is performed with the Viterbi algorithm using the trained model with the following objective.

$$W^* = \arg\max_W P(X|W)P(W)$$

Objective can modified to include the language model and word insertion penalties.

$$W^* = \arg\max_W P(X|W)P(W)^{LMS}N(W)^{IP}$$
$$W^* = \arg\max_W \log P(X|W) + LMS \log P(W) + IP \log N(W)$$

Where W is a sequence of phones, X is the observed sequence of feature frames, LMS is the language model penalty, $N(W)$ is the number of phones and IP is the insertion penalty. The language model used is phone-loop under which each phone has the same probability. Therefore the objective becomes the following.

$$W^* = \arg\max_W \log P(X|W) + IP \log N(W)$$

In our setting word insertion penalty is a fixed value added everytime the decoder transits from the end of one phone to the start of the next. Its goal is to make sure that the number of deletion and insertion errors are in balance and when it is tuned the recognition accuracy can be improved.

During decoding (prediction) the Viterbi algorithm is accompanied by pruning for greater time-efficiency. It is recommended that a beam width of 5-10% is used [1] but this was shown not be sufficient in the experiments. Therefore for context independent phones a beam width of 200-1000 is used and it is adjusted accordingly to make sure that all the sentences are decoded.

2.5 Word Error Rate Metric

To compute the word error rate metric two sequences of phones are matched to have minimum edit distance. Then the metric is computed in the following way.

$$WER = \frac{I + S + D}{N}$$

Where I, S, D and N are the number of insertions substitutions, deletions and words overall respectively.

The metric used the most in all experiments is accuracy calculated from the word error rate.

$$ACC = (1 - WER)100\%$$

By default accuracy of HMM based model phone classification in the training set is reported.

3 GMM-HMMs

In this section of the project GMM-HMM speech recognition systems are constructed. Experiments are performed to explore the effects of training initialisation, front-end parametrisation and the use of triphones instead of monophones. All the used scripts are provided in the appendix.

3.1 Insertion Penalty

Before different settings can be explored a suitable insertion penalty is estimated. This is done by comparing accuracy and deletion to insertion ratio for different insertion penalties in Figure 1, using the $\mathrm{FBANK_Z}$ front-end parametrisation, 8 GMM components and initialisation as described in section 3.2.

Word insertion penalty controls the trade-off between insertion and deletion errors which should be approximately the same and so the deletions to insertions ratio should be close to one. Furthermore, the achieved accuracy should be as high as possible. Therefore by examining Figure 1 insertion penalty of -8 is selected for the GMM-HMMs section.

$$IP = -8$$

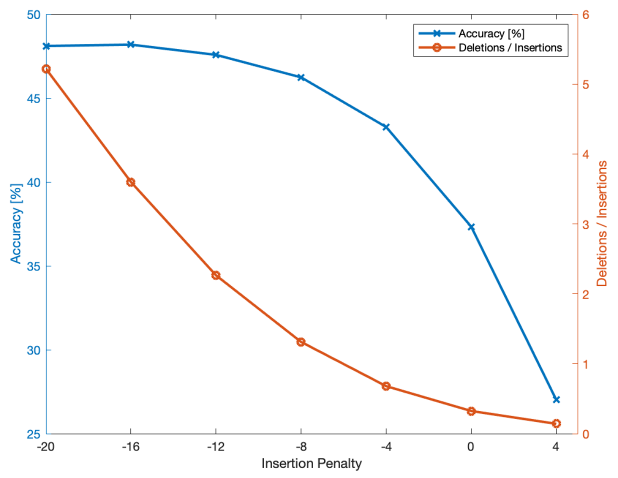


Figure 1: The effect of insertion penalty on accuracy and deletion to insertion ratio ($\mathrm{FBANK_Z}$, 8 mixture components)

3.2 Training Initialisation

GMM-HMMs are trained using the Baum-Welch algorithm which only uses the sequence training information and does not utilise the time-alignment information provided in the training data (HERest). Because Baum-Welch is an EM algorithm it requires parameter initialisation. Two initialisation options are considered.

- Flat Start - The non-zero transition probabilities are initialised evenly. The Gaussian emission probabilities are initialised to the corpus mean and variance.
- Initialise - Viterbi alignment (HInit) followed by phone-level Baum-Welch (HRest) is performed with the use of the fixed phone boundaries.

	Accuracy
Initialised	46.23%
Flat Start	46.19%

Figure 2: Accuracy using initialisation and flat start ($\mathrm{FBANK_Z}$, 8 GMM components)

Both options are tested using the $\mathrm{FBANK_Z}$ front-end parametrisation and 8 GMM components (Figure 2). The accuracy achieved with initialisation is higher only by 0.04%. This suggests that using time-aligned Viterbi and Baum-Welch does not make the resulting system significantly more accurate. Therefore for simplicity, flat start will be used for the remainder of this work.

3.3 Front-End Parametrisation

In this section the effects of various front-end parametrisations are explored. All the versions of FBANK and MFCC feature are tested for a range of 1 to 20 GMM components. The number of mixtures is increased iteratively by 2 with 4 Baum-Welch runs for each setting.

	Number of Mixture Components - Accuracy [%]					
Front-End Parametrisation	1	4	8	12	16	20
$\mathrm{FBANK_Z}$	37.19	44.14	46.19	47.41	48.37	48.95
$\mathrm{FBANK_{DZ}}$	31.9	45.14	49.97	52.38	53.96	55.03
$\mathrm{FBANK_{DAZ}}$	32.03	45.38	49.83	52.85	53.99	55.56
$\mathrm{MFCC_{EZ}}$	46.27	50.05	51.44	52.48	53.15	53.59
$\mathrm{MFCC_{EDZ}}$	51.54	57.9	61.22	63.12	63.98	64.61
$\mathrm{MFCC_{EDAZ}}$	51.73	59.59	63.09	64.94	66.15	66.82

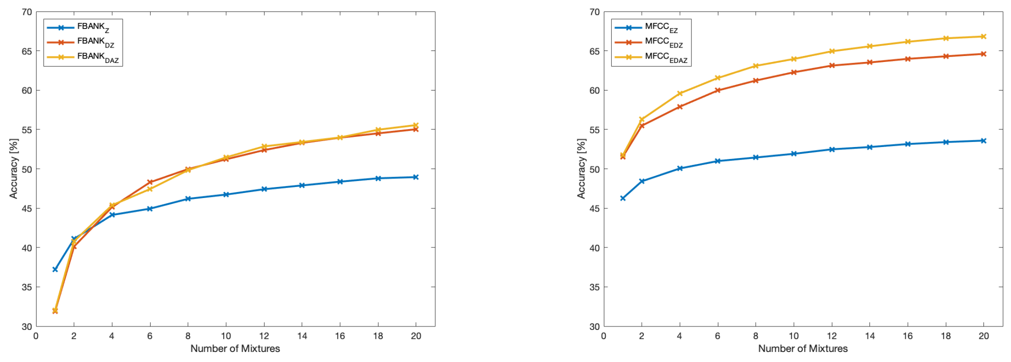
Figure 3: Accuracy with different front-end parametrisations and number of mixture components

Firstly we look at the general trends occurring for both feature types in Figure 4. With increasing number of mixture components, we observe that accuracy increases for all versions of both feature types. This is because by adding mixture components, the model can account for the variability in observations for each subphone. For FBANK features the improvement is more significant than for MFCC features because they do not separate the acoustic filter and which leads to even more phone variability. For example $\mathrm{FBANK_{DAZ}}$ improves by 13.35% in accuracy when 4 component GMM is used instead of a single Gaussian whereas $\mathrm{MFCC_{EDAZ}}$ only improves by 7.86%.

Even though adding more GMM components is appealing Figure 5 reveals that with increasing number of mixture components the disparity between training and testing accuracy increases. This suggests that overfitting occurs when the number of model parameters is increased. This means that the trends observed in training data cannot be easily generalised to unseen data. Therefore for the further sections 8 mixture components are used.

For both features, adding first differentials improves accuracy significantly. This is because differentials provide information about the subphone context which contributes to its correct identification. It appears that adding the second differential does not have a significant effect on FBANK features but it improves the accuracy slightly for MFCC.

Finally the effect of using FBANK or MFCC is compared. Figure 4 shows that MFCC leads to significantly better accuracy. For example for 8 mixture components we get accuracies of 49.83% and 63.09% for $\mathrm{FBANK_{DAZ}}$ and $\mathrm{MFCC_{EDAZ}}$ respectively. This disparity occurs because of two reasons. Firstly MFCC features are able to extract the acoustic filter without the acoustic source whereas FBANK features are not. Therefore model trained with MFCC features can more easily distinguish individual subphones without the added variety of the speaker. Furthermore, the MFCC features have uncorrelated feature variances whereas FBANK features do not. This is favourable because in GMM paramerisation diagonal cross-correlation matrices are used. Therefore for all further sections, $\mathrm{MFCC_{EDAZ}}$ features are used.



(a) FBANK features (b) MFCC features

Figure 4: The effect of front-end parametrisation and number of GMM components on accuracy

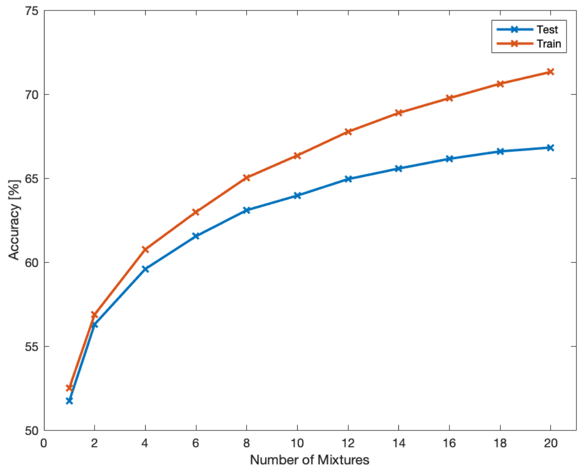


Figure 5: Accuracy using $\text{MFCC}_{\text{EDAZ}}$ features for test (1344 sentences) and subtrain (1168 sentences) with different numbers of mixtures

Considering the analysis above, in all further sections $\text{MFCC}_{\text{EDAZ}}$ with 8 mixture components are used.

3.4 Triphone Decision Tree State Tying

Phone pronunciation is almost always influenced by their neighbouring phones since different sets of phones have different coarticulation. Therefore it is practical to add this information to the model and construct triphones (triples of phones). However because the number of triphones increases exponentially data sparsity becomes an issue. The number of triphone parameters can be reduced by tying subphones whose contexts fall into the same cluster [2]. This is achieved by automatically constructing a phonetic tree to maximise the training data likelihood. This process is parametrised by two thresholds.

- Threshold TB to determine whether the best new tree question and node increase the data likelihood enough
- Threshold RO to determine whether the decrease of data likelihood by merging two nodes (that result in lowest decrease in likelihood possible) is low enough

By altering these two thresholds the resulting number of clustered states changes. In this section we will explore the effect of the use of triphones compared to only using monophones. Systems with various number of clustered states will be tested. To run the triphone clustering system the following steps are taken:

1. Initial monophone model is specified
2. Unclustered triphones are generated from training data

3. Clustering (parameter tying) is performed using the specified RO and TB thresholds
4. Increase the number of mixtures iteratively.

	RO Threshold	TB Threshold	Clustered States
1	100	500	1295
2	200	800	851
3	300	1100	643

Figure 6: The clustered number of states out of 60201 and accuracy for various setups of thresholds ($\text{MFCC}_{\text{EDAZ}}$)

Figure 6 shows the resulting number of clustered states with different threshold setups. To compare the performance of equivalent triphone and monophone models their parameter sets should be similar in size. The $\text{MFCC}_{\text{EDAZ}}$ model with 20 mixture components defined in section 3.3 has $48 \times 3 \times 20 = 2880$ sets of Gaussian parameters. Therefore the triphone models should have similar number of parameters.

Model	GMM Mixtures	Gaussian Parameters	Accuracy [%]
Monophone	20	2880	66.82
Triphone-100-500	2	2590	63.17
Triphone-200-800	4	3404	65.47
Triphone-300-1100	4	2572	65.48

Figure 7: Accuracy for monophone and triphone models ($\text{MFCC}_{\text{EDAZ}}$)

Figure 7 revealed that when the number of Gaussian parameter sets was kept approximately the same, the triphone models did not outperform monophone model and actually lead to a slightly lower accuracy.

4 ANN-HMMs

In this section of the project ANN-HMM speech recognition systems are constructed. Experiments are performed to explore the effects of training front-end parametrisation, number of hidden layers, the use of triphones instead of monophones and recurrent neural networks (RNNs). All the used scripts are provided in the appendix.

To train ANN-HMMs the following steps are taken:

1. A frame-level alignment with the target state-level labels is specified
2. A subphone classifier is trained with the use of a defined ANN model
3. The trained ANN is used to model the state-output distribution

Hidden layers with 500 nodes are used for all the sections. Training is performed using Stochastic Gradient Descent with L_2 regularisation.

4.1 Insertion Penalty

Similarly as in section 3 the insertion penalty is estimated to keep the number of insertions and deletions approximately the same and thus maximise the accuracy.

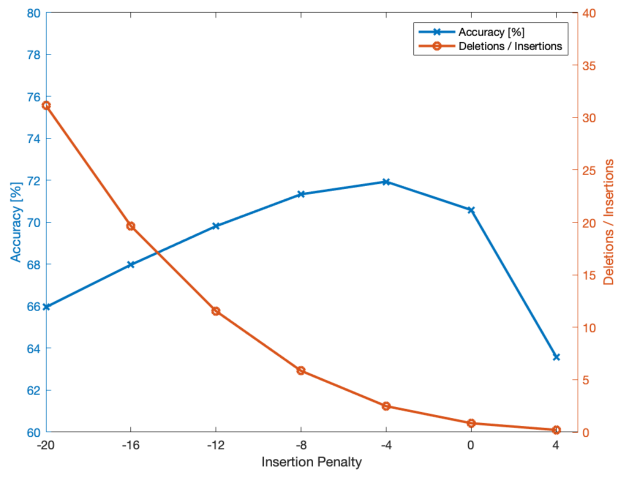


Figure 8: The effect of insertion penalty on accuracy and deletion to insertion ratio ($\mathrm{FBANK}_{\mathrm{DAZ}}$, 1 hidden layer)

Figure 8 shows that insertion penalty $IP = -8$ is plausible in this setup as well. Using the same insertion penalty means that results will be comparable between the GMM-HMM and ANN-HMM sections.

4.2 Single Hidden Layer ANNs

In this section a single hidden layer network is constructed and tested for different front-end parametrisations and input context window widths. The input context window width defines the number of frames that are used to classify each subphone. We test a range of context window widths from 1 to 13 on both $\mathrm{FBANK}_{\mathrm{Z}}$ and $\mathrm{MFCC}_{\mathrm{EZ}}$ feature with and without differentials.

	Context Window Width - Accuracy [%]			
Front-End Parametrisation	1	5	9	13
$\mathrm{FBANK}_{\mathrm{Z}}$	51.82	67.34	69.32	69.64
$\mathrm{FBANK}_{\mathrm{DAZ}}$	68.43	71.34	71.94	71.90
$\mathrm{MFCC}_{\mathrm{EZ}}$	50.37	67.39	69.69	70.28
$\mathrm{MFCC}_{\mathrm{EDAZ}}$	67.63	70.61	71.42	70.88

Figure 9: Accuracy with different front-end parametrisations and context window width.

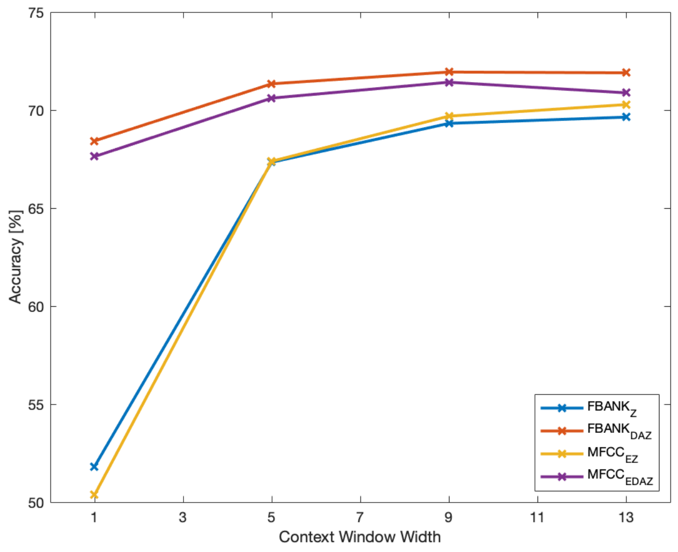


Figure 10: The effect of context window width and front-end parametrisation (single hidden layer ANN)

Firstly we compare this setup to the previous GMM feature selection in section 3.3. With GMMs we concluded that models trained with MFCC features achieve greater accuracy than those trained with FBANK features. However with ANNs they are not longer superior and for some context window widths they perform worse. This is because ANN does not require uncorrelated feature variations and the acoustic and filter separation. The GMM model performed worse when only mel channels were used (FBANK) and thus required cepstral decomposition. Furthermore because only first 12 cepstral values are used some information is lost and the use of MFCC features is inferior.

Secondly we look at the effect of increasing the context window width. Figure 10 shows that the greater the context window width the closer the features with added differentials are to their original versions i.e. FBANK_{Z} compared to $\text{FBANK}_{\text{DAZ}}$ and MFCC_{EZ} compared to $\text{MFCC}_{\text{EDAZ}}$. This is because similar information is contained in the context window width and so adding the differential coeffiecients does not improve the model as much. The highest performing model is one trained on $\text{FBANK}_{\text{DAZ}}$ features with the context window width of 5, 9 or 13.

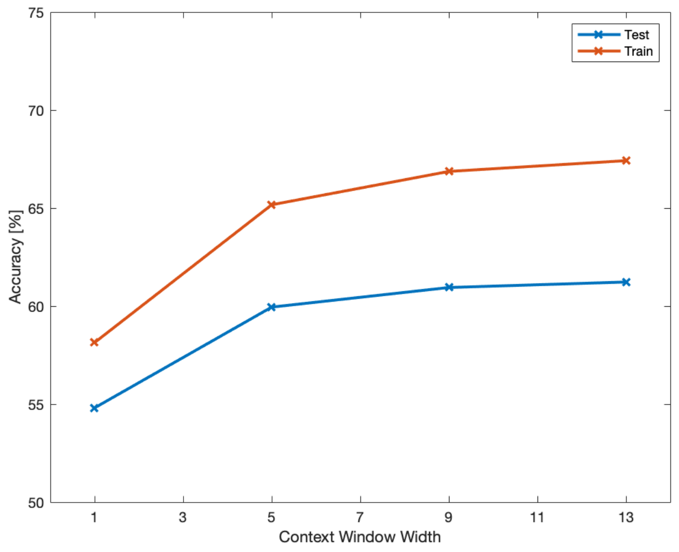


Figure 11: ANN subphone classification for the training set and cross-validation set with varying context window width (FBANK$_{\mathrm{EDAZ}}$)

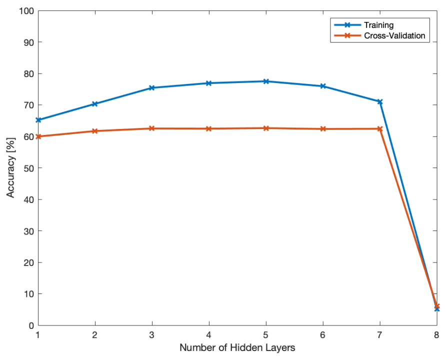
Finally Figure 11 shows that overfitting occurs during the ANN subphone classification and this could be mitigated by tuning the L_2 regularisation parameter. Overfitting occurs especially with bigger input contexts and so for the further work FBANK$_{\mathrm{DAZ}}$ features with context window width 5 are used.

4.3 Multiple Hidden Layer DNNs

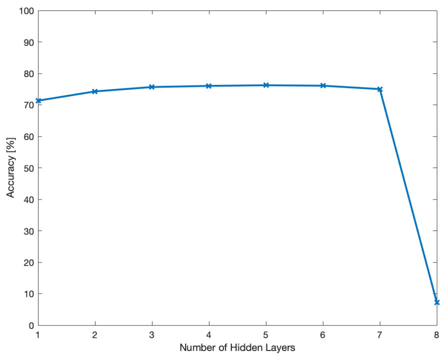
In this section the selected setup from section 4.2 (FBANK$_{\mathrm{DAZ}}$, context window width 5) is extended by adding more hidden layers to the ANN.

	Number of Hidden Layers - Accuracy [%]							
Model	1	2	3	4	5	6	7	8
DNN (Training Set)	65.17	70.33	75.42	76.92	77.51	75.94	71.05	5.17
DNN (CV Set)	59.95	61.69	62.51	62.46	62.62	62.35	62.41	6.04
DNN-HMM Hybrid (CV Set)	71.34	74.28	75.67	76.03	76.24	76.09	75.01	7.24

Figure 12: Accuracy of subphone classification and different number of hidden layers (FBANK$_{\mathrm{DAZ}}$, context window width 5)



(a) DNN subphone classification for the training set and cross-validation set



(b) DNN-HMM hybrid subphone classification for the training set

Figure 13: Accuracy with increasing the number of hidden layers (FBANK$_{\text{EDAZ}}$, context window width 5)

First we examine the DNN subphone classification in Figure 13 (a). With the increasing number of hidden layers training and cross-validation set accuracies increase. The training accuracy increases more significantly with a peak at 5 hidden layers. This disparity suggests that overfitting occurs that could be improved by tuning the L_2 regularisation parameter or adding further regularisation techniques. Both accuracies drop rapidly when 8 hidden layers are used. This could be because of weight vanishing problem that could be improved by using a more sophisticated weight intialisation and batch normalisation. Alternatively this could be because the network gets stuck in a local minimum during SGD which could be improved by using a more complex training procedure with techniques like learning decay and momentum.

The DNN-HMM Hybrid subphone classification in Figure 13 (b) appears to be very closely related to the DNN subphone classification performance as expected. The model with 5 hidden layers achieved the highest accuracy of 76.24% and so it will be used in the further sections.

4.4 Triphone Target Units

In this section the DNN classification is trained to predict the triphone target units. These are provided by the Triphone-200-800 model trained in section 3.4. We use DNNs with 5 hidden layers trained on FBANK$_{\text{EDAZ}}$ features with context window 5.

Model	Accuracy [%]
Monophone DNN	76.24
Triphone DNN	77.65

Figure 14: Accuracy of monophone and triphone system with 5 hidden layers (FBANK$_{\text{EDAZ}}$)

The accuracy achieved with Triphone DNN is higher than that of Monophone DNN but only by 1.4%. The two compared subphone classification networks differ only in the size of input and output layer and so the number of parameters in the Triphone model is bigger.

4.5 RNN-HMMs

In this section the DNN for subphone classification is replaced by a RNN with different levels of unfolding.

	Model - Accuracy [%]	
Unfolding	Monopohone RNN	Triphone RNN
5	74.02	-
10	76.02	-
15	76.93	-
20	76.98	79.19

Figure 15: Accuracy of RNN-HMM Hybrid models with different levels of unfolding ($\text{FBANK}_{\text{EDAZ}}$)

Figure 15 shows that increasing the level of unfolding leads to an increase in accuracy. This is as expected because the model can benefit from more context before and after each subphone.

5 Discussion

The best models from both sections are summarised.

Model	Target Units	Details	Features	Target Units	Accuracy [%]
GMM	Monophone	8 mixtures	MFCC_{DAZ}	144	63.09
GMM	Triphone	8 mixtures	MFCC_{DAZ}	1295	65.47
DNN	Monophone	5 hidden layers, input context 5	$\text{FBANK}_{\text{EDAZ}}$	144	76.24
DNN	Triphone	5 hidden layers, input context 5	$\text{FBANK}_{\text{EDAZ}}$	1295	77.65
RNN	Monophone	20 unfolding steps	$\text{FBANK}_{\text{EDAZ}}$	144	76.98
RNN	Triphone	20 unfolding steps	$\text{FBANK}_{\text{EDAZ}}$	1295	79.19

Figure 16: Accuracy of the best performing systems.

For both the Monophone and Triphone speech system, the ANN systems (DNN and RNN) outperform GMM by more than 10%. For example using the RNN Triphone model results in 79.19% accuracy whereas the GMM Triphone system only achieves 67.59% accuracy. This is likely because they are more flexible in approximating a classifying relationship and make fewer assumptions about the underlying distributions such as that the features having uncorrelated variances.

Using Triphones compared to Monophones increases the accuracy slightly in all systems although we need to keep in mind that because the Triphone systems use almost three times more target units they are more difficult to estimate without overfitting.

Using RNNs compared to DNNs leads to an increase in accuracy for both Monophone and Triphone systems. Furthermore because of parameter tying, RNNs have fewer parameters to estimate. For these reasons RNN based models are superior.

The best performing model that was trained in this work is a Triphone with 1295 target units, RNN-HMM with 20 unfolding steps, using the $\mathrm{FBANK}_{\mathrm{EDAZ}}$ features.

6 Conclusion

In this GMM-HMM and ANN-HMM speech recognition models were constructed.

For the GMM-HMM models the Baum-Welch training initialisation using flat start was shown to be equally good to initialising with frame-aligned Viterbi and Baum-Welch and so it was used for all experiments. GMM-HMM models trained on MFCC front-end parameters performed significantly better than those trained on FBANK parameters because of the effect of the cepstrum. Triphone decision tree state tying improved accuracy when more parameters were used because it provides more context for recognition of each subphone. However when Monophone and Triphone models with approximately equivalent sizes of parameter sets were compared the accuracy was not improved.

In contrast to the GMM-HMM models, the ANN-HMM models trained on FBANK features resulted in a slightly greater accuracy than those trained on MFCC features. This is because ANN classifiers do not assume uncorrelated feature variances and are able to separate the acoustic filter and source. Furthermore the information about the dataset is not reduced by cepstral transforms. The context window width was shown to provide subphone context together with the differential coefficients. Adding more hidden layers resulted in increase in accuracy but overfitting became an issue. RNN-HMM models achieved greater accuracy than DNN-HMM models with fewer parameters to fit because of parameter tying in RNNs. Therefore RNNs are superior to DNNs in this context. Similarly as for GMMs the use of Triphone target units increased accuracy for both DNN and RNN based models but also increased the number of parameters to fit.

For the corresponding systems ANN models outperform GMM models by more than 10% because of its better suitability to model the emission distribution.

The best performing model that was trained in this work is a Triphone with 1295 target units, RNN-HMM with 20 unfolding steps, using the $\mathrm{FBANK}_{\mathrm{EDAZ}}$ features.

References

[1] Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics, 2005.

[2] Steve J Young, Julian J Odell, and Philip C Woodland. Tree-based state tying for high accuracy acoustic modelling. In *Proceedings of the workshop on Human Language Technology*, pages 307–312. Association for Computational Linguistics, 1994.

[3] Victor Zue, Stephanie Seneff, and James Glass. Speech database development at mit: Timit and beyond. *Speech communication*, 9(4):351–356, 1990.

7 Appendix

All the used scripts are provided here grouped by sections. Scripts were written using Python and the subprocess library.

3 GMM-HMMs

3.1 Insertion Penalty

```
import os
import subprocess
import commands

dir = '../results/gmm/initial/'
output_file = 'stdout_1.txt'

model_dir, command = commands.step_mono(flat_start = False, num_mixes = 8, feature_type = 'fbk25d', env_type = 'Z', dir = dir, output_file = 'stdout_1.txt', print_mode = 'w')
subprocess.call(command, shell = True)

model_dir = commands.get_model_dir(dir, feature_type = 'fbk25d', env_type = 'Z', num_mixes = 8, flat_start = False, phones = 'mono')

results_file = 'results_1.txt'
# Test various insertion penalties
for i, ins_penalty in enumerate([4, 0, -4, -8, -12, -16, -20]):
    result_dir, command = commands.step_decode(beam_width = 200, ins_penalty = ins_penalty, model_dir = model_dir, model = 'hmm84', dir = dir, output_file = output_file, print_mode = 'a')
    subprocess.call(command, shell = True)
    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'
    commands.extract_result(result_dir, dir, results_file, print_mode)
```

3.2 Training Initialisation

```
import os
import subprocess
import commands

dir = '../results/gmm/flat_start/'
output_file = 'stdout_1.txt'
results_file = 'results_1.txt'

# Init
model_dir, command = commands.step_mono(flat_start = False, num_mixes = 8, feature_type = 'fbk25d', env_type = 'Z', dir = dir, output_file = 'stdout_1.txt', print_mode = 'w')
subprocess.call(command, shell = True)

result_dir, command = commands.step_decode(beam_width = 200, ins_penalty = -8, model_dir = model_dir, model = 'hmm84', dir = dir, output_file = output_file, print_mode = 'a')
subprocess.call(command, shell = True)

commands.extract_result(result_dir, dir, results_file, print_mode = 'w')

# Flat Start
model_dir, command = commands.step_mono(flat_start = True, num_mixes = 8, feature_type = 'fbk25d', env_type = 'Z', dir = dir, output_file = 'stdout_1.txt', print_mode = 'a')
subprocess.call(command, shell = True)

result_dir, command = commands.step_decode(beam_width = 200, ins_penalty = -8, model_dir = model_dir, model = 'hmm84', dir = dir, output_file = output_file, print_mode = 'a')
subprocess.call(command, shell = True)

commands.extract_result(result_dir, dir, results_file, print_mode = 'a')
```

3.3 Front-End Parametrisation - FBANK

```
import os
import subprocess
import commands

dir = '../results/gmm/params_fbk/'
output_file = 'stdout.txt'
results_file = 'results.txt'

feature_type = 'fbk25d'
env_types = ['Z', 'D_Z', 'D_A_Z']
models = ['hmm14', 'hmm24', 'hmm44', 'hmm64', 'hmm84', 'hmm104', 'hmm124', 'hmm144', 'hmm164', 'hmm184', 'hmm204']
num_mixes = 20

for i, env_type in enumerate(env_types):
    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'
    model_dir, command = commands.step_mono(flat_start = True, num_mixes = num_mixes, feature_type = feature_type, env_type = env_type, dir = dir, output_file = output_file, print_mode = print_
    subprocess.call(command, shell = True)

    for j, model in enumerate(models):
        result_dir, command = commands.step_decode(beam_width = 200, ins_penalty = -8, model_dir = model_dir, model = model, dir = dir, output_file = output_file, print_mode = 'a')
        subprocess.call(command, shell = True)

        if i == 0 and j==0:
            print_mode = 'w'
        else:
            print_mode = 'a'
        commands.extract_result(result_dir, dir, results_file, print_mode = print_mode)
```

3.3 Front-End Parametrisation - MFCC

```
import os
import subprocess
import commands

dir = '../results/gmm/params_mfc/'
output_file = 'stdout.txt'
results_file = 'results.txt'

feature_type = 'mfc13d'
env_types = ['E_Z', 'E_D_Z', 'E_D_A_Z']
models = ['hmm14', 'hmm24', 'hmm44', 'hmm64', 'hmm84', 'hmm104', 'hmm124', 'hmm144', 'hmm164', 'hmm184', 'hmm204']
num_mixes = 20

for i, env_type in enumerate(env_types):
    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'
    model_dir, command = commands.step_mono(flat_start = True, num_mixes = num_mixes, feature_type = feature_type, env_type = env_type, dir = dir, output_file = output_file, print_mode = print_
    subprocess.call(command, shell = True)

    for j, model in enumerate(models):
        result_dir, command = commands.step_decode(beam_width = 200, ins_penalty = -8, model_dir = model_dir, model = model, dir = dir, output_file = output_file, print_mode = 'a')
        subprocess.call(command, shell = True)

        if i == 0 and j==0:
            print_mode = 'w'
        else:
            print_mode = 'a'
        commands.extract_result(result_dir, dir, results_file, print_mode = print_mode)

env_type = 'E_D_A_Z'
model_dir = commands.get_model_dir(dir, feature_type, env_type, num_mixes = num_mixes, flat_start = True, phones = 'mono')
for j, model in enumerate(models):
    result_dir, command = commands.step_decode(beam_width = 200, ins_penalty = -8, model_dir = model_dir, model = model, dir = dir, output_file = output_file, print_mode = 'a')
    subprocess.call(command, shell = True)
    commands.extract_result(result_dir, dir, results_file, print_mode = 'a')

model = 'hmm164'
result_dir, command = commands.step_decode(beam_width = 10000, ins_penalty = -8, model_dir = model_dir, model = model, dir = dir, output_file = output_file, print_mode = 'a')
subprocess.call(command, shell = True)

commands.extract_result(result_dir, dir, results_file, print_mode = 'a')
```

3.4 Triphone Decision Tree State Tying

```
import os
import subprocess
import commands

dir = ’../results/gmm/triphones/’
output_file = ’stdout_3.txt’
num_mixes = 8
feature_type = ’mfc13d’
env_type = ’E_D_A_Z’

mono_dir = ’../results/gmm/overfitting/mfc13d_E_D_A_Z_20_True’
MONO_DIR_ABS = os.path.abspath(mono_dir) + ’/’

rovals = [100, 200, 300]
tbvals = [500, 800, 1100]

models = [’hmm14’, ’hmm24’, ’hmm44’, ’hmm64’]

num_mixes = 8

roval = 200
tbval = 800

model_dir, command = commands.step_xwtri(roval, tbval, MONO_DIR_ABS, ’True’, num_mixes = 8, feature_type = feature_type, env_type = env_type, dir = dir, output_file = output_file, print_mode =
subprocess.call(command, shell = True)

result_dir, command = commands.step_decode(beam_width = 1000, ins_penalty = -8, model_dir = model_dir, model = ’hmm84’, dir = dir, output_file = output_file, print_mode = ’a’)
subprocess.call(command, shell = True)

commands.extract_result(result_dir, dir, results_file, print_mode = ’w’)


for i, roval in enumerate(rovals):
    for j, tbval in enumerate(tbvals):
        if i == 0 and j == 0:
            print_mode = ’w’
        else:
            print_mode = ’a’

        model_dir, command = commands.step_xwtri(roval, tbval, MONO_DIR_ABS, True, num_mixes, feature_type = feature_type, env_type = env_type, dir = dir, output_file = output_file, print_mode
        subprocess.call(command, shell = True)

for i, roval in enumerate(rovals):
    for j, tbval in enumerate(tbvals):

        model_dir = commands.get_model_dir_xwtri(roval, tbval, dir, feature_type, env_type, num_mixes = 8, flat_start = True, phones = ’xwtri’)
        result_dir, command = commands.step_decode(beam_width = 1000, ins_penalty = -8, model_dir = model_dir, model = ’hmm84’, dir = dir, output_file = output_file, print_mode = ’a’)
        subprocess.Popen(command, shell = True)

for i in range(0,3):
    for j in range(0,4):
        if i == 0 and j == 0:
            print_mode = ’w’
        else:
            print_mode = ’a’

        roval = rovals[i]
        tbval = tbvals[i]

        model = models[j]

        model_dir = commands.get_model_dir_xwtri(roval, tbval, dir, feature_type, env_type, num_mixes = 8, flat_start = True, phones = ’xwtri’)
        result_dir, command = commands.step_decode(beam_width = 10000, ins_penalty = -8, model_dir = model_dir, model = model, dir = dir, output_file = output_file, print_mode = print_mode)
        subprocess.Popen(command, shell = True)
```

GMM-HMMs Helper Functions (commands.py)

```
import shutil
import os
```

```
FIRST_LINE = '------------------------ Overall Results --------------------------\n'
LAST_LINE = '===================================================================\n'

def step_xwtri(roval, tbval, mono_dir_abs, flat_start, num_mixes, feature_type, env_type, dir, output_file, print_mode):
    command = '../tools/steps/step-xwtri'

    command += ' -NUMMIXES ' + str(num_mixes)
    command += ' -ROVAL ' + str(roval)
    command += ' -TBVAL ' + str(tbval)

    command += ' ' + mono_dir_abs + 'mono'
    command += ' hmm14'

    if not os.path.exists(dir):
        os.makedirs(dir)
    model_dir = get_model_dir_xwtri(roval, tbval, dir, feature_type, env_type, num_mixes, flat_start, 'xwtri')
    if os.path.exists(model_dir):
        shutil.rmtree(model_dir)

    command += ' ' + model_dir
    command += ' 2>&1 | tee -a ' + dir + output_file

    with open(dir + output_file, print_mode) as f:
        f.write(command + '\n')
    print(command)

    return model_dir, command

def step_mono(flat_start, num_mixes, feature_type, env_type, dir, output_file, print_mode):
    command = '../tools/steps/step-mono'

    if flat_start:
        command += ' -FLATSTART'

    command += ' -NUMMIXES ' + str(num_mixes)

    command += ' ../convert/' + feature_type + '/env/environment_' + env_type

    if not os.path.exists(dir):
        os.makedirs(dir)
    model_dir = get_model_dir(dir, feature_type, env_type, num_mixes, flat_start, 'mono')
    if os.path.exists(model_dir):
        shutil.rmtree(model_dir)

    command += ' ' + model_dir
    command += ' 2>&1 | tee -a ' + dir + output_file

    with open(dir + output_file, print_mode) as f:
        f.write(command + '\n')
    print(command)

    return model_dir, command

def step_decode(beam_width, ins_penalty, model_dir, model, dir, output_file, print_mode, subtrain = False):
    command = '../tools/steps/step-decode'
    command += ' -BEAMWIDTH ' + str(beam_width)
    command += ' -INSWORD ' + str(ins_penalty)

    if subtrain:
        command += ' -SUBTRAIN'

    model_dir_abs_path = os.path.abspath(model_dir)
    command += ' ' + model_dir_abs_path
    command += ' ' + model
    result_dir = model_dir + 'decode-' + model + '-' + str(ins_penalty) + '-' + str(beam_width) + '-' + str(subtrain) + '/'
    if os.path.exists(result_dir):
        shutil.rmtree(result_dir)

    command += ' ' + result_dir
    command += ' 2>&1 | tee -a ' + dir + output_file

    with open(dir + output_file, print_mode) as f:
        f.write(command + '\n')
    print(command)

def get_model_dir(dir, feature_type, env_type, num_mixes, flat_start, phones):
    model_dir = dir + feature_type + '_' + env_type + '_' + str(num_mixes) + '_' + str(flat_start) + '/' + phones + '/'
    return model_dir

def get_model_dir_xwtri(roval, tbval, dir, feature_type, env_type, num_mixes, flat_start, phones):
    model_dir = dir + feature_type + '_' + env_type + '_' + str(num_mixes) + '_' + str(flat_start) + '/' + phones
    model_dir += '_' + str(roval) + '_' + str(tbval) + '/'
    return model_dir

def extract_result(result_dir, dir, result_file, print_mode):
    result_log = result_dir + 'test/LOG'

    with open(dir + result_file, print_mode) as output:
        output.write(result_dir + '\n')
        with open(result_log, 'r') as input:
            printing = False
            for line in input:
                if printing:
                    output.write(line)
                elif line == FIRST_LINE:
                    printing = True
                    output.write(line)
                if line == LAST_LINE:
                    break
        output.write('\n')
```

4 ANN-HMMs

4.1 Insertion Penalty

```
import os
import subprocess
import commands
import commands_ann

dir = '../results/single_fbank_d_a_z/'
output_file = 'stdout_2.txt'
results_file = 'results_2.txt'

align_dir = '../alignments/mono/'
ALIGN_DIR_ABS = os.path.abspath(align_dir) + '/'

feature_type = 'FBANK'
feature_type_2 = 'fbk25d'
env_type = 'D_A_Z'
penalties = [4, 0, -4, -8, -12, -16, -20]
context = 2

for i, context in enumerate(contexts):
    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)

    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'

    model_dir, command = commands_ann.step_dnntrain(model_name, feature_type_2, env_type, ALIGN_DIR_ABS, output_file, print_mode = print_mode, dir = dir)
    subprocess.call(command, shell = True)

for i, penalty in enumerate(penalties):
    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'

    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)
    model_dir = commands_ann.get_model_dir(dir, model_name)
    result_dir, command = commands.step_decode(beam_width = 10000, ins_penalty = penalty, model_dir = model_dir, model = 'hmm0', dir = dir, output_file = output_file, print_mode = print_mode)
    subprocess.call(command, shell = True)
    commands.extract_result(result_dir, dir, results_file, print_mode = print_mode)
```

4.2 Single Hidden Layer ANNs - $\mathbf{FBANK_Z}$

```
import os
import subprocess
import commands
import commands_ann

dir = '../results/single_fbank_z/'
output_file = 'stdout_1.txt'
results_file = 'results_1.txt'

align_dir = '../alignments/mono/'
ALIGN_DIR_ABS = os.path.abspath(align_dir) + '/'
```

```
feature_type = 'FBANK'
feature_type_2 = 'fbk25d'
env_type = 'Z'
contexts = [0,2,4,6]

for i, context in enumerate(contexts):
    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)

    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'

    model_dir, command = commands_ann.step_dnntrain(model_name, feature_type_2, env_type, ALIGN_DIR_ABS, output_file, print_mode = print_mode, dir = dir)
    subprocess.call(command, shell = True)

for i, context in enumerate(contexts):
    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)
    model_dir = commands_ann.get_model_dir(dir, model_name)
    result_dir, command = commands.step_decode(beam_width = 1000, ins_penalty = -8, model_dir = model_dir, model = 'hmm0', dir = dir, output_file = output_file, print_mode = 'a')
    subprocess.call(command, shell = True)

    commands.extract_result(result_dir, dir, results_file, print_mode = 'a')
```

4.2 Single Hidden Layer ANNs - FBANK$_{\mathbf{DAZ}}$

```
import os
import subprocess
import commands
import commands_ann

dir = '../results/single_fbank_d_a_z/'
output_file = 'stdout_1.txt'
results_file = 'results_1.txt'

align_dir = '../alignments/mono/'
ALIGN_DIR_ABS = os.path.abspath(align_dir) + '/'

feature_type = 'FBANK'
feature_type_2 = 'fbk25d'
env_type = 'D_A_Z'
contexts = [0,2,4,6]

for i, context in enumerate(contexts):
    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)

    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'

    model_dir, command = commands_ann.step_dnntrain(model_name, feature_type_2, env_type, ALIGN_DIR_ABS, output_file, print_mode = print_mode, dir = dir)
    subprocess.call(command, shell = True)

for i, context in enumerate(contexts):
    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)
    model_dir = commands_ann.get_model_dir(dir, model_name)
    result_dir, command = commands.step_decode(beam_width = 1000, ins_penalty = -8, model_dir = model_dir, model = 'hmm0', dir = dir, output_file = output_file, print_mode = 'a')
    subprocess.call(command, shell = True)

    commands.extract_result(result_dir, dir, results_file, print_mode = 'a')
```

4.2 Single Hidden Layer ANNs - MFCC$_{\mathbf{EDZ}}$

```
import os
import subprocess
import commands
import commands_ann
```

```
dir = '../results/single_mfcc_e_z/'
output_file = 'stdout_1.txt'
results_file = 'results_1.txt'

align_dir = '../alignments/mono/'
ALIGN_DIR_ABS = os.path.abspath(align_dir) + '/'

feature_type = 'MFCC'
feature_type_2 = 'mfc13d'
env_type = 'E_Z'
contexts = [0,2,4,6]

for i, context in enumerate(contexts):
    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)

    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'

    model_dir, command = commands_ann.step_dnntrain(model_name, feature_type_2, env_type, ALIGN_DIR_ABS, output_file, print_mode = print_mode, dir = dir)
    subprocess.call(command, shell = True)

for i, context in enumerate(contexts):
    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)
    model_dir = commands_ann.get_model_dir(dir, model_name)
    result_dir, command = commands.step_decode(beam_width = 1000, ins_penalty = -8, model_dir = model_dir, model = 'hmm0', dir = dir, output_file = output_file, print_mode = 'a')
    subprocess.call(command, shell = True)

    commands.extract_result(result_dir, dir, results_file, print_mode = 'a')
```

4.2 Single Hidden Layer ANNs - $\mathbf{MFCC_{EDAZ}}$

```
import os
import subprocess
import commands
import commands_ann

dir = '../results/single_mfcc_e_d_a_z/'
output_file = 'stdout_1.txt'
results_file = 'results_1.txt'

align_dir = '../alignments/mono/'
ALIGN_DIR_ABS = os.path.abspath(align_dir) + '/'

feature_type = 'MFCC'
feature_type_2 = 'mfc13d'
env_type = 'E_D_A_Z'
contexts = [0,2,4,6]

for i, context in enumerate(contexts):
    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)

    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'

    model_dir, command = commands_ann.step_dnntrain(model_name, feature_type_2, env_type, ALIGN_DIR_ABS, output_file, print_mode = print_mode, dir = dir)
    subprocess.call(command, shell = True)

for i, context in enumerate(contexts):
    model_name = commands_ann.get_model_name(num_hidden = 1, feature_type = feature_type, env_type = env_type, context = context)
    model_dir = commands_ann.get_model_dir(dir, model_name)
    result_dir, command = commands.step_decode(beam_width = 1000, ins_penalty = -8, model_dir = model_dir, model = 'hmm0', dir = dir, output_file = output_file, print_mode = 'a')
    subprocess.call(command, shell = True)

    commands.extract_result(result_dir, dir, results_file, print_mode = 'a')
```

4.3 Multiple Hidden Layer DNNs

```
import os
import subprocess
import commands
import commands_ann

dir = '../results/multiple_15/'
output_file = 'stdout_1.txt'
results_file = 'results_1.txt'
class_file = 'class_1.txt'

align_dir = '../alignments/mono/'
ALIGN_DIR_ABS = os.path.abspath(align_dir) + '/'

feature_type = 'FBANK'
feature_type_2 = 'fbk25d'
env_type = 'D_A_Z'
context = 2

for num_hidden in range(1, 9):
    model_name = commands_ann.get_model_name(num_hidden = num_hidden, feature_type = feature_type, env_type = env_type, context = context)
    if num_hidden == 1:
        print_mode = 'w'
    else:
        print_mode = 'a'
    model_dir, command = commands_ann.step_dnntrain(model_name, feature_type_2, env_type, ALIGN_DIR_ABS, output_file, print_mode = print_mode, dir = dir)
    subprocess.call(command, shell = True)

for num_hidden in range(1, 9):
    model_name = commands_ann.get_model_name(num_hidden = num_hidden, feature_type = feature_type, env_type = env_type, context = context)
    model_dir = commands_ann.get_model_dir(dir, model_name)
    result_dir, command = commands.step_decode(beam_width = 1000, ins_penalty = -8, model_dir = model_dir, model = 'hmm0', dir = dir, output_file = output_file, print_mode = 'a')
    subprocess.call(command, shell = True)

    if num_hidden == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'

    commands.extract_result(result_dir, dir, results_file, print_mode = print_mode)
    commands_ann.extract_result(result_dir, dir, class_file, print_mode = print_mode)
```

4.4 Triphone Target Units

```
import os
import subprocess
import commands
import commands_ann

dir = '../results/ann_tri/'
output_file = 'stdout_1.txt'
results_file = 'results_1.txt'

align_dir = '../alignments/xwtri/'
ALIGN_DIR_ABS = os.path.abspath(align_dir) + '/'

Alignment
command = '../tools/steps/step-align ' + ALIGN_DIR_ABS + ' hmm84 ' + align_dir + 'align-hmm84'
print(command)
subprocess.call(command, shell = True)

feature_type = 'FBANK'
feature_type_2 = 'fbk25d'
env_type = 'D_A_Z'
context = 2
num_hidden = 3

model_name = commands_ann.get_model_name(num_hidden = num_hidden, feature_type = feature_type, env_type = env_type, context = context)
model_dir, command = commands_ann.step_dnntrain(model_name, feature_type_2, env_type, ALIGN_DIR_ABS, output_file, print_mode = 'w', dir = dir)
subprocess.call(command, shell = True)

model_name = commands_ann.get_model_name(num_hidden = num_hidden, feature_type = feature_type, env_type = env_type, context = context)
model_dir = commands_ann.get_model_dir(dir, model_name)
result_dir, command = commands.step_decode(beam_width = 10000, ins_penalty = -8, model_dir = model_dir, model = 'hmm0', dir = dir, output_file = output_file, print_mode = 'a')
subprocess.call(command, shell = True)

commands.extract_result(result_dir, dir, results_file, print_mode = 'a')
```

4.5 RNN-HMMs - Monophones

```
import os
import subprocess
import commands
import commands_ann

dir = '../results/rnn/'
output_file = 'stdout_1.txt'
results_file = 'results_1.txt'

align_dir = '../alignments/mono/'
ALIGN_DIR_ABS = os.path.abspath(align_dir) + '/'

feature_type = 'FBANK'
feature_type_2 = 'fbk25d'
env_type = 'D_A_Z'
context = 2
num_hidden = 1


unfoldings = [5, 10, 15, 20]
context_str1 = '05'

for i, unfolding in enumerate(unfoldings):
    model_name = commands_ann.get_rnn_model_name(num_hidden = num_hidden, feature_type = feature_type, env_type = env_type, context_str = context_str1, unfolding = unfolding)
    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'
    model_dir, command = commands_ann.step_dnntrain(model_name, feature_type_2, env_type, ALIGN_DIR_ABS, output_file, print_mode = print_mode, dir = dir)
    subprocess.Popen(command, shell = True)

for i, unfolding in enumerate(unfoldings):
    model_name = commands_ann.get_rnn_model_name(num_hidden = num_hidden, feature_type = feature_type, env_type = env_type, context_str = context_str1, unfolding = unfolding)
    model_dir = commands_ann.get_model_dir(dir, model_name)

    result_dir, command = commands.step_decode(beam_width = 10000, ins_penalty = -8, model_dir = model_dir, model = 'hmm0', dir = dir, output_file = output_file, print_mode = 'a')
    subprocess.call(command, shell = True)

    if i == 0:
        print_mode = 'w'
    else:
        print_mode = 'a'

    commands.extract_result(result_dir, dir, results_file, print_mode = print_mode)
```

4.5 RNN-HMMs - Triphones

```
import os
import subprocess
import commands
import commands_ann

dir = '../results/rnn_tri/'
output_file = 'stdout_1.txt'
results_file = 'results_1.txt'

align_dir = '../alignments/xwtri/'
ALIGN_DIR_ABS = os.path.abspath(align_dir) + '/'

feature_type = 'FBANK'
feature_type_2 = 'fbk25d'
env_type = 'D_A_Z'
context = 2
num_hidden = 1

unfolding = 20
context_str = '05'

model_name = commands_ann.get_rnn_model_name(num_hidden = num_hidden, feature_type = feature_type, env_type = env_type, context_str = context_str, unfolding = unfolding)
model_dir, command = commands_ann.step_dnntrain(model_name, feature_type_2, env_type, ALIGN_DIR_ABS, output_file, print_mode = 'w', dir = dir)
subprocess.call(command, shell = True)

model_name = commands_ann.get_model_name(num_hidden = num_hidden, feature_type = feature_type, env_type = env_type, context = context)
model_dir = commands_ann.get_model_dir(dir, model_name)
result_dir, command = commands.step_decode(beam_width = 10000, ins_penalty = -8, model_dir = model_dir, model = 'hmm0', dir = dir, output_file = output_file, print_mode = 'a')
```

```
subprocess.call(command, shell = True)

commands.extract_result(result_dir, dir, results_file, print_mode = 'w')
```

ANN-HMMs Helper Functions (commands_ann.py)

```
import shutil
import os

def step_dnntrain(model_ini, feature_type, env_type, align_dir_abs, output_file, print_mode, dir, gpu = True):
    command = '../tools/steps/step-dnntrain -vv'

    if gpu:
        command += ' -GPUID 0'

    print(align_dir_abs)
    command += ' -MODELINI models/' + model_ini + '.ini'
    command += ' ../convert/' + feature_type + '/env/environment_' + env_type
    command += ' ' + align_dir_abs + 'align-hmm84/align/timit_train.mlf'
    command += ' ' + align_dir_abs + 'hmm84/MMF'
    command += ' ' + align_dir_abs + 'align-hmm84/hmms.mlist'

    if not os.path.exists(dir):
        os.makedirs(dir)
    model_dir = get_model_dir(dir, model_ini)
    if os.path.exists(model_dir):
        shutil.rmtree(model_dir)

    command += ' ' + model_dir
    command += ' 2>&1 | tee -a ' + dir + output_file

    with open(dir + output_file, print_mode) as f:
        f.write(command + '\n')
    print(command)

    return model_dir, command

def get_model_dir(dir, modelini):
    model_dir = dir + modelini + '/'
    return model_dir

def get_model_name(num_hidden, feature_type, env_type, context):
    model_name = 'DNN-' + str(num_hidden)
    model_name += 'L.ReLU.'
    model_name += feature_type
    model_name += '_' + env_type
    model_name += '_' + str(context)
    return model_name

def get_rnn_model_name(num_hidden, feature_type, env_type, context_str, unfolding):
    model_name = 'RNN-' + str(num_hidden)
    model_name += 'L.ReLU.'
    model_name += feature_type
    model_name += '_' + env_type
    model_name += '_' + context_str
    model_name += '_' + str(unfolding)
    return model_name
```