TIMIT Phonetic Continuous Speech Recognition

MLMI2 - Speech Recognition

January 16, 2022

Candidate no.: J902C

Word count: 4287[1]

UNIVERSITY OF CAMBRIDGE
Department of Engineering

[1] Excluding appendix, tables, footnotes, images and titles

Trumpington Street
Cambridge CB2 1PZ
Tlf.: +44 01223 332600
http://www.eng.cam.ac.uk/

Table of contents

1 Introduction 1

2 HTK Fundamentals 2

3 Data setup 3

3.1 TIMIT Database 3

3.2 Feature types 3

3.3 Front-end parameterisations 3

4 Initialisation, training and prediction procedures 4

4.1 Initialisation 4

4.2 Training 4

4.3 Prediction 4

5 GMM-HMMs 5

5.1 Monophone Experiments 5

5.1.1 Feature type 5

5.1.2 Delta coefficients 6

5.1.3 Initialisation 6

5.1.4 Number of gaussian mixture components 6

5.1.5 Insertion penalty 7

5.2 Triphone Decision Tree State Tying 8

6 ANN-HMMs 10

6.1 Single Hidden Layer Experiments 10

6.2 Adding Multiple Layers 11

6.3 Triphone Target Units 13

6.4 RNN-HMMs 13

7 Extension: Extended RNN and LSTMs 15

7.1 Extended RNNs 15

7.2 LSTMs 16

8 Experimental Findings and Conclusions 18
8.1 Discussion 18
8.2 Conclusion and further work 20
A GMMs triphone experiments 21

1. Introduction

The goal of this project is to analyse the performance of several phone-based continuous speech recognition models based on Hidden Markov Models (HMMs) for the TIMIT Acoustic-Phonetic speech database [2]. First, Gaussian Mixture Models (GMMs-HMMs) are going to be used to form the state output distributions; and then, Artificial Neural Networks (ANNs-HMMs) are employed using different configurations. Deep Neural Networks (DNNs), Recurrent Neural Networks (RNNs) and Long short-term memory (LSTMs) networks are investigated.

The practical involves training and testing HMMs for the TIMIT corpus and examining various configurations, such us different front-end features, several differential coefficients, the use of context independent and context dependent models, and other extensions.

The HTK toolkit [1] is used for training and testing. Scripts, written either in C-shell or Python, are provided to execute all the experiments. Additionally, different Bash or Python scripts are coded to tune different hyperparameters and to get the results exposed in this report.

2. HTK Fundamentals

The HTK Toolkit is a library of useful routines for preprocessing speech files, label files, managing HMM definitions, and it includes tools for defining, training and testing HMM recognisers. Descriptions of all the HTK utilities can be found in the HTK Book [1].

The main HTK tools used *in this practical* are:

- **HCopy** copies speech files and changes the parameter encoding.
- **HLE** transforms label files in order to reduce the number of phone classes present.
- **HInit** creates a single Gaussian HMM, taking as an input a prototype HMM definition and a set of training data files and produces as output a new initialised HMM. It uses Viterbi alignment during parameter re-estimation.
- **HRest** re-estimates the parameters of a initialised HMM using Baum-Welch algorithm in the given model boundaries. This improves the training set likelihood and sometimes improves recognition results.
- **HERest** (similar to HRest) uses embedded training technique in order to avoid using hand-produced label boundaries when re-estimating the parameters of a full set of HMM definitions.
- **HVite** tests the created HMMs by using them to recognise test sentences. It is a Viterbi decoder for word-based speech recognition that uses a word network and dictionary. It supports various HMM model types (context-dependent acoustic models or triphones) and structures (GMM-HMMs or ANN-HMMs).
- **HParse** creates a phone-loop network as specified.
- **HBuild** generates a suitable network for recognition with a bigram language model.
- **HResults** compares predicted label files with the hand-produced labels derived from the TIMIT `.phn` files. It uses a dynamic programming based string alignment procedure to find the minimum error string alignment and then accumulates statistics on substitutions, deletions and insertion errors.

3. Data setup

3.1. TIMIT Database

The TIMIT corpus of read speech is designed to provide data for acoustic-phonetic studies and for the evaluation of automatic speech recognition systems. It consists of recordings of 630 speakers of eight major dialects of American English, each reading ten phonetically-rich sentences. Test and training subsets, balanced for phonetic and dialectal coverage, are specified.

3.2. Feature types

Two feature types are considered for this project:

1. **FBANK**: A filter bank is an array of bandpass filters that separates the input signal into multiple components, each one carrying a single frequency sub-band of the original signal. In this case, a **24 channel filter bank** is used.

2. **MFFCs**: The mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively describe a MFC. They are derived from a type of cepstral representation of the audio clip. In the practical, **12 MFFCs with normalised log energy** are used.

3.3. Front-end parameterisations

The previous feature types are expanded into the following versions, called ”qualifiers” or ”front-end parameterisations”:

- **FBANK_Z** or **MFFC_E_Z**: Sentence-based mean removal.

- **FBANK_D_Z** or **MFFC_E_D_Z**: Sentence-based mean removal and first differentials.

- **FBANK_D_A_Z** or **MFFC_E_D_A_Z**: Sentence-based mean removal, first and second differentials.

4. Initialisation, training and prediction procedures

4.1. Initialisation

Baum-Welch algorithm is used for training and, since it is a EM-algorithm, it requires parameter initialisation.
The standard way of initialising is to execute **Viterbi alignment** followed by phone-based Baum-Welch algorithm which uses fixed phone boundaries. **Flatstart** is an alternative initialisation method, where all gaussian emission probabilities of the models are initialised with the global mean and variance of the training data.

4.2. Training

HMM systems are trained using **Viterbi-training** to iteratively compute the model parameters, and **Baum-Welch** EM algorithm is also used to refine those parameters. The output distributions are modelled employing GMMs (results in chapter 5) or ANNs (chapter 6).

4.3. Prediction

Viterbi algorithm is a dynamic programming procedure to obtain the most likely path for a model with certain parameters. This is used for prediction using the trained model.

Viterbi algorithm is also executed with a pruning procedure in order to reduce the run time and increase the decoding efficiency. To do so, a beam width is specified, whose value will be between 150 and 500 in this practical.

In addition, the preferred **evaluation metric** of a speech recognition system is the ***Word Error Rate Metric (WER)***, since this is what is being minimised. The WER is computed using the formula $WER = \frac{I+S+D}{N}$, where I, S, D and N are the number of insertions, substitutions, deletions and total words used, respectively. However, HTK tools usually report the results using ***Accuracy Metric (ACC)***. Word Error Rate can be directly calculated from the accuracy value by using 4.1:

$$ACC = (1 - WER) \cdot 100\%. \tag{4.1}$$

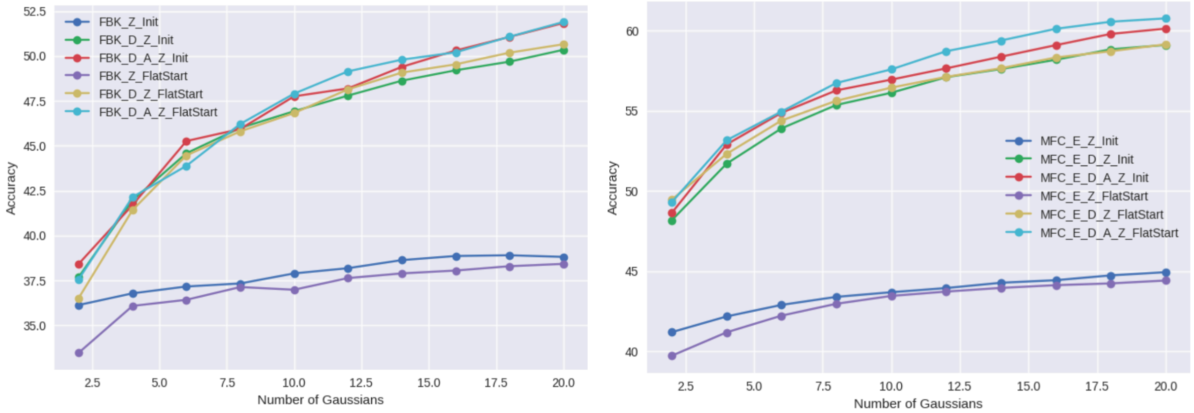
5. GMM-HMMs

In this chapter GMMs are used to model the state-output distribution of the HMM systems. Several models are going to be evaluated, experimenting with different feature types, delta coefficients (differentials), number of gaussian mixture components, initialisation method and context type.

In section 5.1 context-independent models are constructed, whereas context-dependent (triphones) systems are tested in section 5.2.

5.1. Monophone Experiments

Various context-independent systems are considered, and all the results can be seen in figure 5.1. The effect of feature type, delta coefficients, initialisation and the number of mixture components are investigated in subsections 5.1.1, 5.1.2, 5.1.3 and 5.1.4 respectively.



(a) Models using monophone FBANK features (b) Models using monophone MFFCs features

Figure 5.1: Context-independent (monophone) systems results, varying feature type, front-end parameterisation, initialisation and number of gaussian mixture components

5.1.1. Feature type

Figure 5.1 shows that MFCC features clearly outperform Filter Bank features. All FBANK results are lower than 52.5% of accuracy, whereas MFCC results are usually higher than 55%. This is because MFCC feature are more discriminable and nearly uncorrelated, so they allow the diagonal of the covariance matrix to be used by HMMs effectively since cross-correlation matrices are employed to model GMM-HMMs. Additionally, MFCC-based models can more easily distinguish individual subphones since they are able to extract the acoustic filter without the acoustic source (FBANK-based models are not).

5.1.2. Delta coefficients

For both feature types, MFCC and FBANK, the usage of differentials increases considerably the performance of the model. Figure 5.1 suggests that using 1st differentials increases the resulting accuracy compared to just sentence-based mean removal front-end parameterisations. Furthermore, adding 2nd differentials does not have a significant effect on FBANK features but it improves the accuracy slightly for MFCCs. This is because delta coefficients provide information about the subphone context and capture more dependency between each of the feature vectors, which contributes to correct phone identification.

5.1.3. Initialisation

Baum-Welch algorithm is used to train GMM-HMM models. Because Baum-Welch is a EM-algorithm it requires parameter initialisation.
The standard way of initialising, named as **Init** in this project, is to execute Viterbi alignment (`HInit` script) followed by phone-based Baum-Welch algorithm (`HRest` tool) which uses fixed phone boundaries. **Flatstart** is an alternative initialisation method, where all gaussian emission probabilities of the models are initialised with the global mean and variance of the training data.
The experiments (figure 5.1) show that models using **Flatstart initialisation** performed slightly better on the test set for both input features (MFCC and FBANK). Flatstart procedure is less dependent to the existing model since there is no need to fix boundaries manually, so it allows better generalisation.

So far, the best evaluated system is built using **MFCC features**, 1st and 2nd differentials (**EDAZ**) and **Flatstart** initialisation. The following sections' experiments are going to be constructed using this model structure, in order to obtain a good estimation for the number of gaussian mixture components (subsection 5.1.4) and for the insertion penalty value, in 5.1.5.

5.1.4. Number of gaussian mixture components

Results of figure 5.1 suggest that increasing the number of gaussian mixture components rises the system accuracy (so reduces the WER). However, training accuracy has not been taken into account yet. The input parameter `-SUBTRAIN` can be set when decoding to test on a sub-set of the training set.
Figure 5.2 illustrates that increasing the number of mixture components raises the disparity between training and test accuracy. This phenomenon is known as **overfitting**, and it is a generalisation issue. These results suggest that increasing the number of mixture components is not desirable and then the **number of gaussian components is going to be set to 8** for the rest of sections in this chapter.

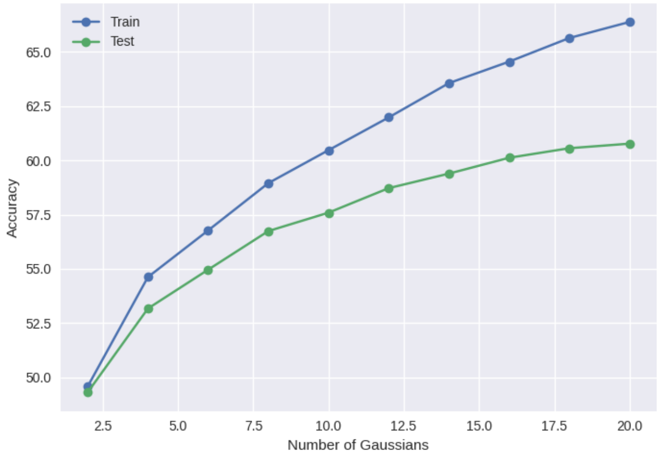


Figure 5.2: Training vs Test accuracy for MFCC_EDAZ model

5.1.5. Insertion penalty

Insertion penalty controls the trade-off between deletions and insertion errors. The input parameter `-INSWORD` can be set when decoding to specify a insertion penalty value for testing.

To tune the insertion penalty, the best performing system is executed, i.e., a model built with MFCC_EDAZ front-end parameterisation, using FlatStart initialisation and 8 gaussian mixture components has been run with different insertion penalty values, resulting in figure 5.3. This image suggests that the **best insertion penalty value is around -10**, where the accuracy is maximum (WER is minimised).

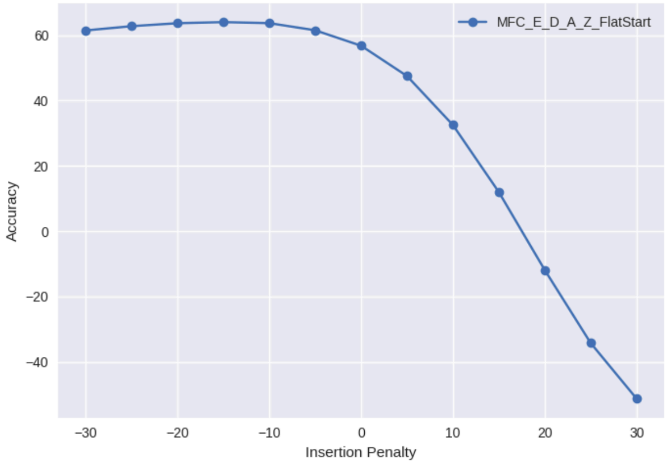


Figure 5.3: Insertion penalty tuning for MFCC_EDAZ with 8 GMM comp. and FlatStart

To sum up **GMM-HMM context-independent systems**, the **best obtained model** uses **MFCC features**, adding 1st and 2nd differentials (**EDAZ**), with an **insertion penalty**

of -10, using **8 gaussian mixture components** and **FlatStart** as initialisation method. This model has an **accuracy of 63.65%**, that is a **WER of 36.35%**.

5.2. Triphone Decision Tree State Tying

Context-independent phones or monophones are insufficient to model speech since phone pronunciation is influenced by neighbouring phones. This is known as *co-articulation* and it can be modelled by using context-dependent phones or triphones (triples of phones). However, the number of possible triphones increases exponentially with the number of monophones, since N phones lead to N^3 potential triphones, and data sparsity becomes an issue. To avoid that and ensure all state distribution are robustly estimated, states that model similar subphones are tied using decision trees [3] that have been provided for this practical. The phonetic decision tree is configured by two values or thresholds, that alter the number of final clustered states:
- **TB**: it sets the threshold for minimum log-likelihood increase in tree growth. This can be specified by `-TBVAL` input parameter when training a triphone-based model.
- **RO**: it sets the threshold for outlier states removal, i.e, it controls the number of leaf nodes in the decision tree. These nodes should be built in order to prevent creating too many clusters with little associated training data. This can be specified by `-ROVAL` option.

In this section, several triphone-based systems are going to be evaluated, using a well-performing initial monophone model, i.e., `MFCC_EDAZ_FlatStart` is going to be used in order to build various clustered ”cross-word” triphone models. RO value, TB value and the number of mixture components are the hyperparameters that are being **tuned** using **grid search** method. RO values in $[100, 200, 300, 400]$ are tested, TB in $[600, 800, 1000, 1200]$ and the tested number of mixtures are in $[4, 8, 10, 12]$. In total $4^3 = 64$ configurations are investigated for triphone GMM-HMM models, and since each configuration is really expensive to test, the **core-test set** is used to decode (activating `-CORETEST` option), which is a sub-set of the test set. All the results can be found in the appendix A in the table A.1.

The top 5 performing configurations are listed in table 5.1. It's clear that the best number of mixture components is 12, and 1000 seems to be the best suitable value for TB threshold, since TB=1000 achieves the two best accuracies.

RO value	TB value	No. Mix. comp.	No. Clustered states	Accuracy
100	600	12	5052	68.45
200	1200	12	1655	68.48
300	**1000**	**12**	**2103**	**68.71**
400	**1000**	**12**	**2039**	**68.66**
400	1200	12	1555	68.60

Table 5.1: Results of various GMM-triphone models

Focusing on RO threshold, it's not obvious what value performs the best, so a more exhaustive search is run over RO, fixing the rest of hyperparameters. This time the **full test set is used** to evaluate the performance using the RO threshold in the interval $[100, 500]$, and the results are visible in figure 5.4. The graphic suggests that RO=300 is the best value for this threshold, and that model reports the lowest WER so far, WER=30.71%.

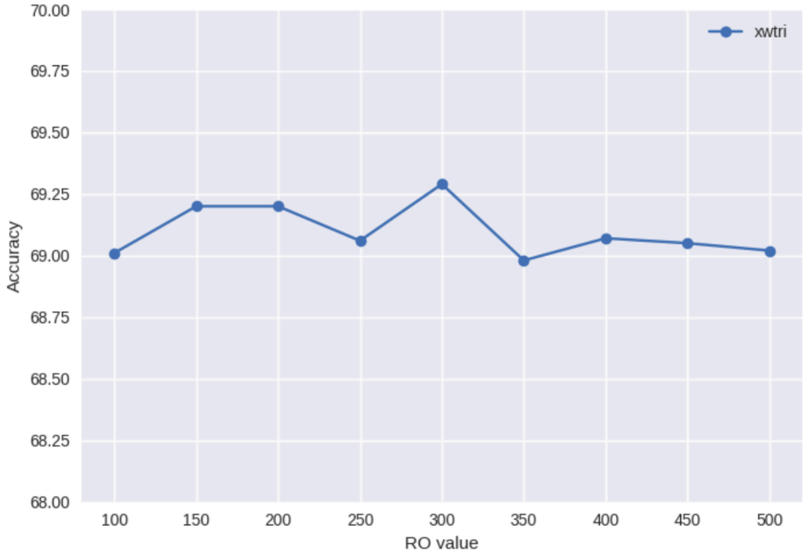


Figure 5.4: Tuning RO threshold for triphone-based GMM-HMM system. The other hyperparameters are fixed to TB=1000, 12 mixture components and IP=-10.

To sum up, the **best triphone-based GMM-HMM model** uses **12 gaussian mixture components**, **RO=300**, **TB=1000**, **-10** as **insertion penalty** value and it has been built on a monophone-based GMM-HMM model with **MFCC_EDAZ** front-end parameterisation, **8 GMM components** and **FlatStart** initialisation. The **resulting accuracy is 69.29%**, i.e., **WER of 30.71%**, improving monophone results.

6. ANN-HMMs

In this chapter, ANN-HMM speech recognition models are investigated. Different configurations are tested as a function of the context window width, feature type (`FBANK` vs `MFCC`) and some front-end parameterisations in section 6.1. Additionally, insertion penalty is tuned for these systems, as well as ANNs with multiple layers (DNNs) in section 6.2, triphone target units (section 6.3) and RNNs, in section 6.4.

First, to train an ANN-HMM system, it's necessary to specify a frame-level alignment of the training data with the target state-level labels, depending on the model type: context independent (monophones) or context dependent (triphones). Then, a subphone classifier is trained and it's tested at modelling the state-output distribution.

6.1. Single Hidden Layer Experiments

Since the best performing GMM-HMM models (chapter 5) use 1st and 2nd differentials, in this chapter the insertion penalty is estimated to keep the the number of insertions low, while maximising accuracy (minimising WER), using both differentials and both feature types (`FBANK` and `MFCC`) for ANN-HMM systems. Figure 6.1 shows that the best insertion penalty is in the [-10,0] interval. The **best performing IP can be set to -8** w.l.g. and this value is going to be used in the following sections.

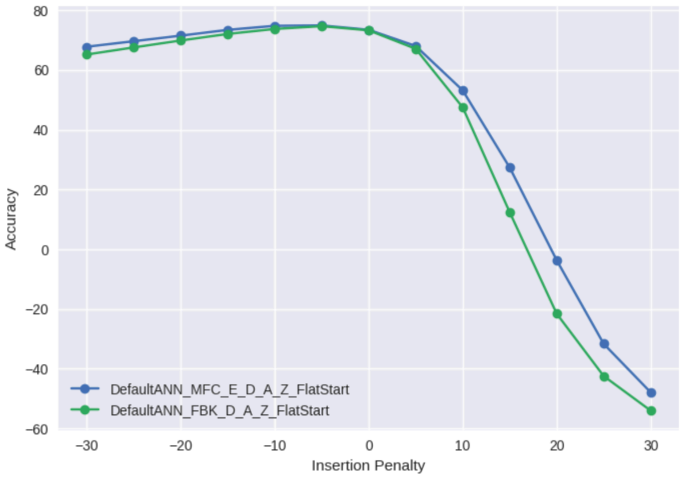


Figure 6.1: Insertion penalty tuning for single layer ANN using both feature types

After tuning the insertion penalty, several experiments are executed to jointly determine the best front-end parameterisation, feature type and context window width. In this case, a **single layer ANN** with a **GMM** (**8 mixture components** per state) is used to model the state-output distribution.

Figure 6.2 shows the results of these experiments. First, **systems using 1st and 2nd differentials** are **clearly better** than the other ones. Both input feature types, `MFCC` and `FBANK`, have similar results. **`FBANK` features are slightly better** than `MFCC` on the test set. On the training set and cross-validation set, `MFCC` features tend to perform better than `FBANK`, but they don't generalise well on the test set. Finally, increasing the context width improves performance on the training set, but generally decreases the resulting accuracy in both CV and test sets. A context width greater than 6 ends up *overfitting* the system, since the difference between training accuracy and the cross-validation accuracy is increasing as a function of the context width. **The best investigated context window width is 5**, and it is going to be used in the rest of the ANN-HMM classifiers. Smaller context widths miss important information, while larger ones might include irrelevant features.

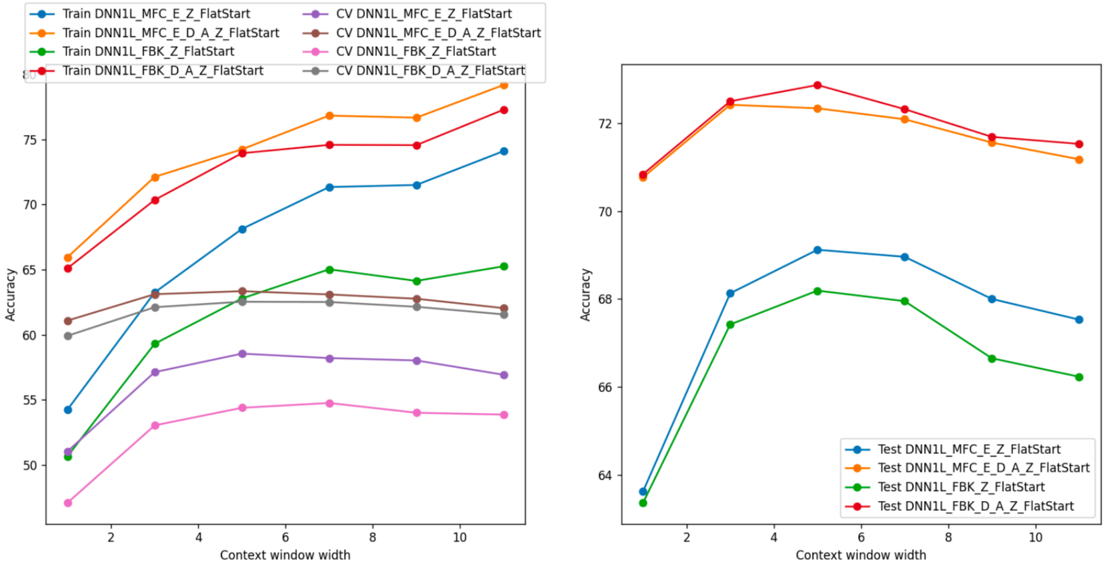


Figure 6.2: Single layer ANN tuning (context width, feature type and front-end param.) Accuracies for train set, cross-validation set and test set are illustrated. Note that a model using a context width of 8 is considering the 8 previous frames, the current and the 8 posterior ones.

A **single hidden layer ANN** (a **GMM** of **8 mixture** components, **FlatStart** initialisation, is employed for the alignments) using **FBANK_DAZ** front-end parameterisation and a **context width of 5** obtains a **accuracy of 73.94%**, i.e., a **WER of 26.06%**.

6.2. Adding Multiple Layers

In this section, models using **Filter Bank** input features, **first and second differentials**, as well as a **context window width of 5** are going to be studied. Basically, ANN models with multiple hidden layers are investigated while keeping 500 nodes per hidden layer in order to keep computation reasonably fast. GMMs of 8 mixture components and FlatStart initialisation are used for the alignments.

Figure 6.3 illustrates the effect of increasing the number of hidden layers. It's easy to see that both training and CV accuracies increase, reaching a peak at 5 hidden layers. The performance difference between sets also raises, what can mean that overfitting is occurring and it could be improved by tuning L2 regularisation parameter or using less layer nodes. For more than 5 hidden layers, training accuracy diminishes, and after adding 8 hidden layers the performance drastically drops to less than 10%. Similarly, the performance on the test set rises until using 5 hidden layers. With 8 and 9 hidden layers, the resulting accuracy dramatically drops to about 7%. This suggests that the network is stuck in a local minimum during SGD procedure, or it has other problems, such as gradient vanishing problem.

This drop could also be explained by the **increasing number of parameters to be estimated** when the number of hidden layers is increased. When the depth of the networks rises, the estimation problem becomes really hard because the **parameter space dimension increases**, causing the SGD algorithm to fall in a **poor local optima** more easily and make it difficult to escape from it. We could try to use batch gradient descent or other activation functions in order to improve these models, but it would be more computationally expensive.

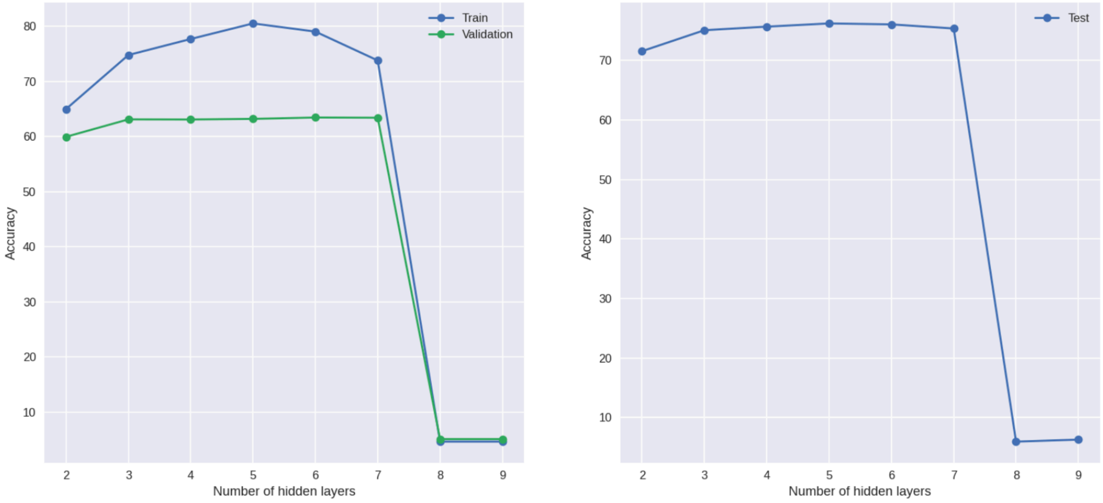


Figure 6.3: Train, cross-validation and test accuracies for multiple hidden layer ANNs. Note that the total number of layers for each ANN model is the number of hidden layers plus two, since there are always an input layer and an output layer.

The **best performing model** on the test set is obtained with **5 hidden layers**, reporting an **accuracy of 76.23%** (**WER of 23.77%**). However, other similar structures could also be studied in the following sections, like ANNs of 4 hidden layers (simpler models and harder to overfit).

6.3. Triphone Target Units

After determining a well-performing ANN configuration (ANN with **5 hidden layers**, **context shift of 5** and **FBANK_DAZ** as input features), triphone target units are examined. Since the targets change, the state-level alignments have to be modified too. The triphone model using RO=300, TB=1000 and 12 mixture components trained in the GMM-HMM part 5.2 is employed for the alignments.

When using a context-dependent output layer the number of parameters to be estimated also increases, since the system takes into account and more frames at training and prediction phases. This can lead to parameter estimation problems, such as the mentioned in section 6.2, where the model can fall into a poor local optima due to the ”curse of dimensionality”. Anyway, using a suitable configuration, in terms of input features, amount of acoustic context and number of hidden layers, the triphone-based models seem to perform even better than the context-independent ones.

In table the results of this section are shown. An 4 hidden layer ANN is also studied since it had a good performance in section 6.2 and the 5 hidden layer ANN seemed to suffer from overfitting.

Model description	No. hidden layers	Train Acc.	CV Acc.	Test Acc.
ANN-HMM, Triphone targets	4	65.51%	46.32%	77.35%
ANN-HMM, Triphone targets	**5**	70.77%	46.00%	**77.49%**

Table 6.1: Performance on training, cross-validation and test sets of ANN-HMM systems using triphone target units

The **5 hidden layer ANN**, despite having more accuracy disparity between training and CV sets, keeps performing better in the test set, as it happened in the context-independent section 6.2, with a resulting **accuracy of 77.49%**, i.e., a **WER of 22.51%**.

6.4. RNN-HMMs

Another ANNs that can be used are Recurrent Neural Networks (RNNs). In this section the ANNs are formed by connecting a RNN as input layer to a fully connected (hidden) layer, that is connected to the output layer. The RNN at each frame takes in input of the current frame t and the output of a hidden vector at time $t{-}1$. Truncated back-propagation through time is used in order to train the models in which the model is unfolded for a fixed number of steps to form a feed-forward network.

In figure 6.4 the amount of unfolding is investigated for context-independent models using FBANK as input features. The difference between training set and cross-validation set is barely constant for all unfoldings (around 15%), and the **greatest accuracy is achieved when 20 steps** are used: **76.56%**, i.e., **23.44% WER**. So this unfolding is going to be used in the extension section 7.

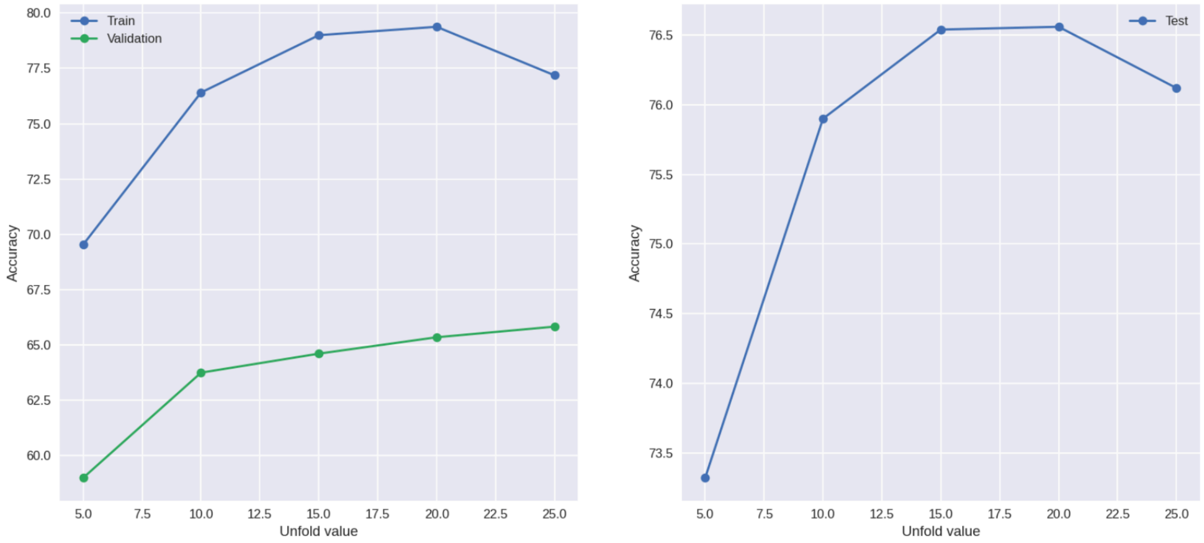


Figure 6.4: Tuning unfolding for a 1 layer RNN (monophones)

Now, **context-dependent models** (**triphone**-based models) can be tested using the mentioned ANN architecture. The results are visible in figure 6.5. Again, the best performance in the test set is obtained when **20 steps** are employed, with an **accuracy** of **78.73%**, i.e., **21.27% WER**. In general, increasing the unfold value benefits the model because it can take into account more context information before and after each phone.

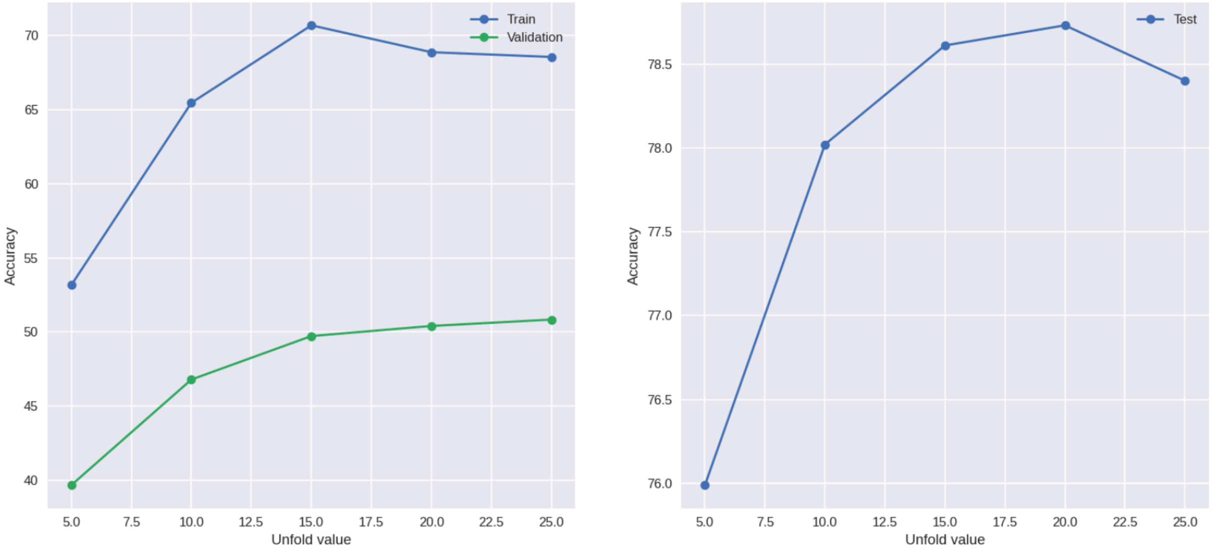


Figure 6.5: Tuning unfolding for a 1 layer RNN (triphones)

7. Extension: Extended RNN and LSTMs

Several extensions can be implemented to the previous experiments. In this section, more fully connected layers are going to be added after the RNN input layer of section 6.4. Both monophone and triphone features are going to be evaluated for these systems in section 7.1. Additionally, LSTM-based models are examined in 7.2, using a LSTM network as input layer followed by two fully connected layers (a hidden layer and the output layer).

7.1. Extended RNNs

More fully connected (FC) layers are placed after the RNN input layer of section 6.4. The RNN is expected to capture the sequential information in the speech stream, whereas the several FC layers are expected to identify more complex features. For the RNN layer, an unfolding of 20 is used since it reported the best performance in section 6.4.

First, monophone-based systems are investigated, and its results can be found in figure 7.1. Analysing train-subset and cross-validation set performances we can see that overfitting does not occur since the difference between both sets is barely constant independently the number of added FC layers. The performance improves in all sets while more FC layers are included, until 5 hidden FC layers, that the accuracy drops below 10% as it happened in section 6.2. This can be explained by the gradient vanishing problem or the network might be stuck in a low-performance local minimum during SGD. The best performing architecture is obtained with **4 hidden FC layers**, with an **accuracy of 77.09%**, i.e., a **WER of 22.91%**. This model is superior than its equivalent version using just 1 hidden layer (section 6.4, figure 6.4).

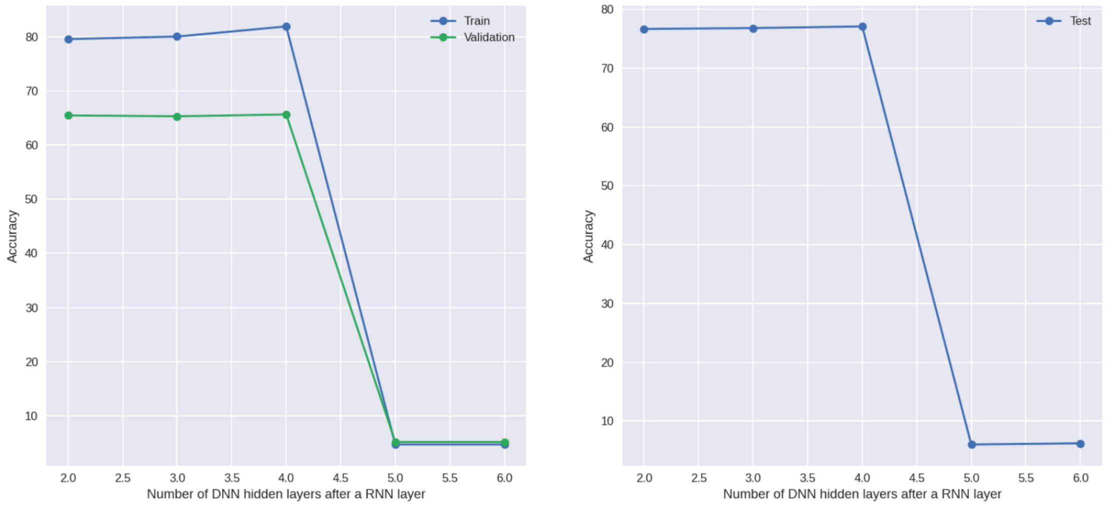


Figure 7.1: a RNN followed by multiple FC layers (monophones)

Now, triphone features are taken into account. The results are visible in figure 7.2. Similarly, when the number of linked hidden layers is 6 the accuracy drops to about 5%, so the network at this point is literally useless. With fewer hidden FC layers, the system seems to perform decently. With 4 hidden layers or less, the resulting accuracy in all sets and the train/CV sets difference look constant. However, when 5 hidden layers are examined, the accuracy in test set decreases a little bit, while the difference between accuracies of train and CV sets increases considerably, suggesting that overfitting occurs.

To wrap up, the best triphone-based model is the one using a **RNN input layer and 3 hidden FC layers**, resulting in an **accuracy of 78.69%**, i.e., a **WER of 21.23%**. This result is practically the same as its analogous version with 1 hidden layer of section 6.4 (figure 6.5).

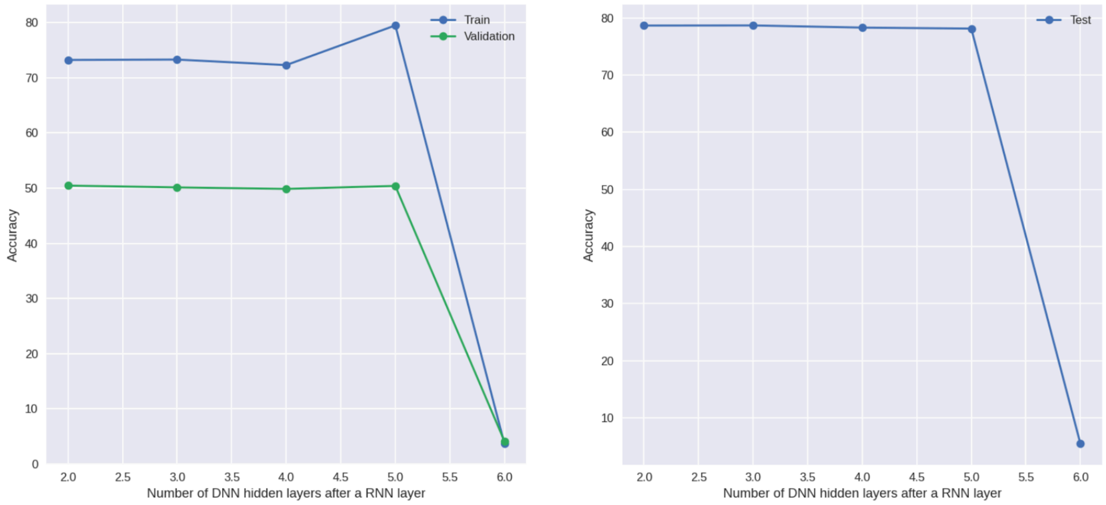


Figure 7.2: a RNN followed by multiple FC layers (triphones)

7.2. LSTMs

In this section another type of RNN is tested, known as Long-Short Term Memory (LSTM) network.

The investigated network is designed using 1 LSTM network as input linked to a fully connected hidden layer, that is also connected to the output layer. Both context-independent and context-dependent models have been tested, and the results are reported in table 7.1.

Model description	Context type	Train Acc.	CV Acc.	Test Acc.
ANN-HMM: 1 LSTM + 1 FC hidden layer. Unfolding=20	Monophones	78.51%	63.42%	74.70%
ANN-HMM: 1 LSTM + 1 FC hidden layer. Unfolding=20	Triphones	83.37%	47.51%	**76.09%**

Table 7.1: Performance on training, cross-validation and test sets of LSTM networks

The performance seems to be slightly worse than systems based on RNNs. Nevertheless, the accuracy reported on the test set is similar to the previous models studied in the ANN-HMM chapter (chapter 6).

Focusing on the results using **LSTMs**, the best performance is obtained using context-dependent features (**triphones**), with an **accuracy of 76.09%**, i.e., a **23.91 WER**. Although there exists a considerable disparity between performances on the training and cross-validation sets (overfitting might be occurring), this could be fixed by tuning the regularisation parameter or other regularising techniques, such as dropout.

In conclusion, LSTM-based models have a similar performance compared to the rest of ANN-HMM models, but it is marginally worse than RNN-based systems.

8. Experimental Findings and Conclusions

8.1. Discussion

The main results of this work are summarised in table 8.1. It includes a description of the trained model, the front-end parameterisation (feature type and/or delta coefficients), the context type (monophones or triphones) and the resulting performance, measured in both accuracy and Word Error Rate metrics.

Model description	Front-end param.	Context type	Accuracy	WER
GMM-HMM, 8 Mix. Comp., FlatStart, IP=-10	MFCC_EDAZ	Monophones	63.65%	36.35%
GMM-HMM, 12 Mix. Comp., FlatStart, IP=-10, RO=300, TB=1000	MFCC_EDAZ	Triphones	69.29%	30.71%
1 hidden layer ANN-HMM, Context width=5, GMM of 8 M.C., FlatStart, IP=-8	FBANK_DAZ	Monophones	73.94%	26.06%
5 hidden layer ANN-HMM, Context width=5, GMM of 8 M.C., FlatStart, IP=-8	FBANK_DAZ	Monophones	76.23%	23.77%
5 hidden layer ANN-HMM, Context width=5, GMM of 12 M.C., FlatStart, IP=-8, RO=300, TB=1000	FBANK_DAZ	Triphones	77.49%	22.51%
ANN-HMM (RNN + 1 hidden FC layer), unfolding=20, GMM of 8 M.C., FlatStart, IP=-8	FBANK_DAZ	Monophones	76.56%	23.44%
ANN-HMM **(RNN + 1 hidden FC layer)**, unfolding=20, GMM of 12 M.C., FlatStart, IP=-8, RO=300, TB=1000	FBANK_DAZ	Triphones	**78.73%**	**21.27%**
ANN-HMM (RNN + 4 hidden FC layers), unfolding=20, GMM of 8 M.C., FlatStart, IP=-8	FBANK_DAZ	Monophones	77.09%	22.91%
ANN-HMM **(RNN + 3 hidden FC layers)**, unfolding=20, GMM of 12 M.C., FlatStart, IP=-8, RO=300, TB=1000	FBANK_DAZ	Triphones	**78.69%**	**21.23%**

ANN-HMM (1 LSTM + 1 hidden FC layer), unfolding=20, GMM of 8 M.C., FlatStart, IP=-8	FBANK_DAZ	Monophones	74.70%	25.30%
ANN-HMM (1 LSTM + 1 hidden FC layer), unfolding=20, GMM of 8 M.C., FlatStart, IP=-8	FBANK_DAZ	Triphones	76.09%	23.91%

Table 8.1: Summarised results of this project - Best performing models
Notation: IP := Insertion penalty, M.C. := Mixture components, FC := fully connected

Several comparisons can be made analysing the best performing models for GMM-HMMs chapter (5), ANN-HMMs chapter (6) and the extension (7):

- **Base model**: In general ANN-HMM systems outperform GMM-HMM models. This is because a deep neural network is capable of capturing more complex information than Gaussian Mixture Models. Since speech recognition is a very difficult task because of co-articulation and phone variability between speakers, considerable complex models have to be used if the training data is sufficient.

- **Initialisation method**: Flatstart was proven to be a slightly better initialisation method in section 5.1.3, because of the fact that this procedure is less dependent to the existing model since there is no need to fix boundaries manually, so it allows better generalisation. This method set all gaussian emission probabilities of the models to the mean and variance of the training data. This initialisation strategy was used in the rest of the sections.

- **Insertion penalty**: the best insertion penalty value to control the trade-off between deletions and insertion errors was -10 for GMM-HMM models and in the interval (-10,0) for ANN-HMMs.

- **Context type**: All speech recognising systems have been tested with context-independent and context-dependent features. For all of them, triphones increases the resulting accuracy (i.e., reduces the WER) compared to monophones. However, we have to keep in mind that triphones use more target units, so it gets harder to estimate the model parameters.

- **Input feature type**: In the first part of the practical (GMM-HMMs, chapter 5), MFCC features clearly outperformed FBANK features, since MFCCs are uncorrelated, which favors Gaussian components that have diagonal covariance matrices. On the other hand, in the second part of the project (ANN-HMMs, chapter 6), models using Filter Bank features had slightly higher accuracy, mainly because ANN systems don't assume uncorrelated variances and they are able to separate the source and the acoustic filter.

- **Delta coefficients**: In all sections adding delta coefficients resulted in a WER reduction. This is because differentials provide information about the phone context and add information about the dependency between each feature vector.
- **Number of mixture components** (GMM-HMM): As it can be seen in chapter 5, increasing the number of mixture components results in a raise in the model performance. However, it also makes the task of estimating the model parameters more difficult, since the parameter space grows considerably. It has been proven that for more 8 mixture components GMM-HMM models (using mnonophones) tend to overfit, so GMM-HMMs with 8 Gaussians are considered. If the target units are triphones, the number of parameters increases too, but we've seen that using 12 mixture components end up with a reduction in the WER, so it's employed when triphones are considered.
- **Context width** (ANN-HMM): In section 6.1, several context window widths were investigated. A shorter window tends to miss important information, whereas a larger window include irrelevant features, increasing noise. The results show that a context window width of 5 returns the best results for networks of 1 hidden layer.
- **Unfolding** (RNN-ANN-HMM): The number of unfolding steps control the amount of context the recurrent neural networks processes at each time step. In section 6.4, various unfolding parameters were tested, and 20 unfolding steps was proven to be the best value among the ones investigated. More unfolding steps ended up confusing the network with irrelevant information, whereas a fewer number of steps resulted in a feature context loss.

8.2. Conclusion and further work

Overall, the **best performing models** were **RNN-based systems**, formed by connecting a RNN (input layer) to several fully connected layers, employing 20 unfolding steps, RO=300, TB=1000 and a GMM of 12 mixture components for the alignments. Using **one** or **three hidden fully connected layers** reported the best results, as well as triphones as target units and FBANK_DAZ as the preferred front-end parameterisation. These models obtained an **accuracy of 79%**, i.e., a **21% WER**.

The results can be considered decent taking into account HTK is an old tool (originally from 1995, revised version 3.5 in 2015) [1] and that other projects dated in 2015 got similar results using more powerful techniques, such as attention-based models. An example of that is [4], where a speech recognition architecture based on a hybrid attention mechanism and RNNs was investigated, resulting in a 17.6% WER.

Our project could be even further improved testing different possibilities, such as trying other initialisation methods in ANN-HMMs systems, tuning the regularisation parameters or using dropout to avoid overfitting or even using other types of context-dependent features, such as bigrams or N-grams (with N>3) in general.

A. GMMs triphone experiments

Various triphone-based systems are evaluated using a well-performing initial monophone model, in this practical `MFCC_EDAZ_FlatStart`. The RO value, TB value and the number of mixture components are the hyperparameters that are being **tuned** using **grid search** method. RO values in $[100, 200, 300, 400]$ are tested, TB in $[600, 800, 1000, 1200]$ and the tested number of mixtures are in $[4, 8, 10, 12]$. In total $4^3 = 64$ configurations are investigated for triphone GMM-HMM models, and since each configuration is really expensive to test, the **core-test set** is used to decode (activating `-CORETEST` option), which is a sub-set of the test set. All the results can be found in the table A.1.

RO value	TB value	No. Mix. comp.	No. Clustered states	Accuracy
100	600	4	5052	66.90
100	600	8	5052	67.98
100	**600**	**12**	**5052**	**68.45**
100	800	4	3165	65.47
100	800	8	3165	67.75
100	800	12	3165	68.04
100	1000	4	2207	66.00
100	1000	8	2207	67.80
100	1000	12	2207	68.00
100	1200	4	1653	65.49
100	1200	8	1653	67.73
100	1200	12	1653	68.23
200	600	4	4797	66.67
200	600	8	4797	67.12
200	600	12	4797	68.40
200	800	4	3119	65.50
200	800	8	3119	67.43
200	800	12	3119	67.90
200	1000	4	2165	66.17
200	1000	8	2165	68.00
200	1000	12	2165	68.36
200	1200	4	1655	65.43
200	1200	8	1655	67.94
200	**1200**	**12**	**1655**	**68.48**
300	600	4	4313	67.07
300	600	8	4313	67.65
300	600	12	4313	67.79
300	800	4	2923	66.20

300	800	8	2923	67.70
300	800	12	2923	68.37
300	1000	4	2103	66.06
300	1000	8	2103	67.66
300	**1000**	**12**	**2103**	**68.71**
300	1200	4	1638	65.38
300	1200	8	1638	67.76
300	1200	12	1638	68.34
400	600	4	3901	66.45
400	600	8	3901	67.31
400	600	12	3901	67.13
400	800	4	2790	65.99
400	800	8	2790	68.10
400	800	12	2790	68.30
400	1000	4	2039	66.05
400	1000	8	2039	68.21
400	**1000**	**12**	**2039**	**68.66**
400	1200	4	1555	65.72
400	1200	8	1555	68.33
400	**1200**	**12**	**1555**	**68.60**

Table A.1: Results of different GMM-triphone models

Bibliography

[1] The HTK Book (2006). S. J. Young, G. Evermann, M. J. F. Gales, P. Woodland, et al. Version 3.5.

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

[3] S. J. Young, J. J. Odell, P. C. Woodland. Tree-based state tying for high accuracy acoustic modelling. *Proceedings of the workshop on Human Language Technology*. Association for Computational Linguistics. pages 307-312. 1994.

[4] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho and Y. Bengio. Attention-Based Models for Speech Recognition. *arXiv e-prints*. 2015.