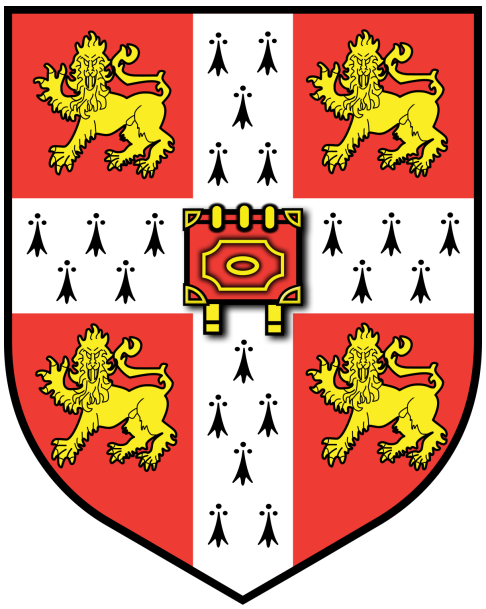

A Comparison of Models used for Automatic Speech Recognition

Student ID: J902G
Word Count: 3,972[a]
Submitted: January 16, 2022

[a]Excluding code snippets, footnotes, references, and the appendix.

Contents

1 Introduction

This report examines the evolution of HMM based automatic speech recognition models. The first section looks at GMM-HMM models, which use hidden Markov models to model phone generation, with emission distributions given by Gaussian mixture models with variable number of components. The second section then moves to ANN-HMM models, which follow a similar methodology, except now the emission distributions are modelled by artificial neural networks. In this section the type and specification of these neural networks are changed to investigate which combinations give the best accuracy.

In both sections, other adaptations are also considered, such as how including phone context through triphones can improve accuracy, and the relative performances of different input vector types. Finally, the final section will go on to look at whether more advanced models such as TDNN-HMMs and LSTM-HMMs can give practical improvements.

1.1 Features and Front-End Parameterisations

Feature Types:

The first feature types considered in this practical are filter banks (FBANKs) with 24 channels, each carrying information about the amplitudes of a different frequency band present in the signal, calculated by Fourier transforming the waveform and convolving the amplitudes with a triangular filter in the frequency domain. In this way, each 10ms waveform can be transformed to a 24 length vector (not including extra front-end parameterisations, see below) of elements $\{m_j\}_{j=1}^{24}$. Mathematically, if the window of the signal is $x(t)$ and the j^{th} triangular filter in the frequency domain is $h_j(f)$, the j^{th} vector element is given by:

$$m_j = \mathcal{F}\left[x(t)\right] * h_j(f) \tag{1}$$

One issue with filter banks is that the amplitudes of different frequency bands are very correlated. Therefore, an alternative is to use Mel-Frequency Cepstral Coefficients (MFCCs) as the input features. These now use the logarithm of the FBANK amplitudes as coefficients in a discrete Cosine transform, in order to give a user-set number of *uncorrelated* vector elements for each 10ms signal window:

$$c_i \propto \sum_j \ln(m_j) \cos\left(\frac{i\pi}{N}(j - \frac{1}{2})\right) \tag{2}$$

For the simplest MFCC case, 12 coefficients are chosen.

Parameterisations:

For the FBANK feature types, there are also three types of input vector parameterisation that can be taken:

1. The first is the simplest form, with 24 channels representing 24 filters to convolve the signal with in the Fourier domain. However, there is also the additional step that all the vectors contain sentence based mean removal. These vectors are denoted as FBANK Z.

2. Differentials can then also be appended to the base vectors to increase the information they store about the signal. The first differential relies on a delta window size Θ, and is defined by [5]:

$$d_t = \frac{\sum_{n=1}^{\Theta} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{\Theta} n^2} \tag{3}$$

   Adding the first differential (referred to as *delta coefficients*) to the mean-removed filter bank features is denoted FBANK DZ, and adds an additional 24 elements, making the input vectors now 48 dimensional.

3. Finally, the last parameterisation used in this practical for filter banks is to use the second differential, the *acceleration coefficients*, which are defined as above except for replacing c_t with d_t. These add an additional 24 elements again, and can only be used if the delta coefficients are already in use. Therefore these input vectors are now 72 dimensional, and are denoted FBANK DAZ.

For the MFCC feature types, the same parameterisations as above also hold. However, there is an additional term E that represents the addition of each signal's normalised log-energy to its input vector [5].

1. MFCC EZ is the most basic mean-removed parameterisation and has 13 elements (12 coefficients and one for log-energy).

2. MFCC EDZ then has 26 dimensions due to the addition of first differentials.

3. MFCC EDAZ has 39 dimensions to account for second and first differentials.

1.2 Train and Test Sets

The corpus used for this practical is the TIMIT Acoustic-Phonetic Continuous Speech Corpus created by Texas Instruments and MIT researchers [1]. The corpus is a collection of 6300 spoken sentences from 630 speakers, with a 16kHz speech waveform file associated to each utterance. Importantly, each utterance is labelled with the corresponding transcription, that has been verified by a person.

Of the 6300 sentences, 1344 are held back for testing, and the rest used for training.

1.3 Brief Overview of Model Training and Prediction

Training[1]**:**

The base of all of the models considered in this practical is a stack of hidden Markov models connected in a looped fashion to allow for continuous speech. Each HMM represents one of 39 phones (as defined by the Carnegie Mellon University pronouncing dictionary), with an additional silence phone. Within each of the stacked HMMs, there are 3 nodes (subphones), with transition probabilities defined both between subphones (to form a single phone of variable length) and between phone HMMs (to form a word as a sequence of variable numbers of phones).

The emission probabilities from each of the nodes are then given by GMMs (Section 2) or ANNs (Section 3), and in each case training of the transition probabilities is done in a generative fashion using Baum-Welch training - iteratively finding the parameters that give optimal log-likelihoods of the labelled phones given the input waveforms.

In the case of GMM-HMMs, every 4 iterations of Baum-Welch the number of Gaussian mixtures is increased by 2, up to the desired number of mixtures, in order to speed up training.

Prediction:

In order to make predictions of the phones present in a waveform, the Viterbi decoding algorithm can be used, because each of the HMMs in the looped stack represents a phone (and each of the three nodes within them a sub-phone), so finding the most probable path through the HMM model given the signal is equivalent to finding the most probable string of phones present in the signal. In this way we can write the predicted phone sequence $w^* := \{w^*\}_{t=1}^T$ as (where the input signal is given by vectors $\{\vec{x}_{t=1}^T\}$, see Section 1.1 above):

$$w^* = \arg\max_w P(\{\vec{x}\}_{t=1}^T|w)P(w) = \arg\max_w \left[\ln P(\{\vec{x}\}_{t=1}^T|w) + \ln P(w)\right] \tag{4}$$

It is also possible to influence the decoding to take certain strategies, for example adding a term to the log joint distribution that penalises long predicted sequences can help improve accuracy by keeping the number of word insertions low. Mathematically, this is is accomplished via:

$$w^* = \arg\max_w \left[\ln P(\{\vec{x}\}_{t=1}^T|w) + \ln P(w) + \gamma_{ins} \ln n(w)\right] \tag{5}$$

for $n(w)$ the number of phones in the predicted sequence, and γ_{ins} the word insertion penalty. The $P(w)$ term is the language model, and can be used to favour some phones over others. In the case of this practical however, all phones are a priori treated as equally probable, and hence the language model is a uniform distribution (e.g. independent of the phone sequence).

[1] A further explanation can be found in Appendix A.1

2 GMM-HMM Models

2.1 Varying the Insertion Penalty

When evaluating the performance of models used for Automatic Speech Recognition (ASR), there are two measurements to consider. The first is the correctness, which simply refers to the percentage of phones in a length of speech that are correctly predicted. However, there is ambiguity here, as the length of speech is not fixed for ASR, and so a model could achieve a very high percentage correctness by inserting many extra phones to increase the chances that the true ones are included. This is therefore not a suitable measure to judge the efficacy of the models, and so a new measure, the *word error rate* is used, defined as:

$$\text{WER} = \frac{I + S + D}{N} \tag{6}$$

for I the number of insertions, S the number of substitutions, D the number of deletions, and N the total number of words. From here the accuracy is then defined as:

$$\text{Accuracy} = 1 - \text{WER} \tag{7}$$

Note that this means the accuracy can be negative, if the sum in the numerator of Equation (6) is greater than N. The accuracy provides a more useful interpretation of model performance and will therefore be favoured above correctness for this report. From its definition, it is clear that higher accuracy can be achieved if the number of insertions is kept low (all else being equal). Therefore, the first aspect of this practical was to vary the insertion penalty and evaluate which penalty gives highest accuracy in the trade-off between keeping insertions low and allowing for model flexibility. As a rule of thumb, it is a good idea to keep the number of word insertions similar to the number of deletions, so that the net change of word number in Equation (6) is close to 0.

Figure 1 shows the percentage accuracy and percentage correctness for varying word insertion penalty using a GMM-HMM with MFCC EDAZ inputs. The optimal word insertion penalty was found to be -15 as this value maximised the accuracy, and so for the rest of this section (Part 1), this is used as the insertion penalty unless otherwise stated.

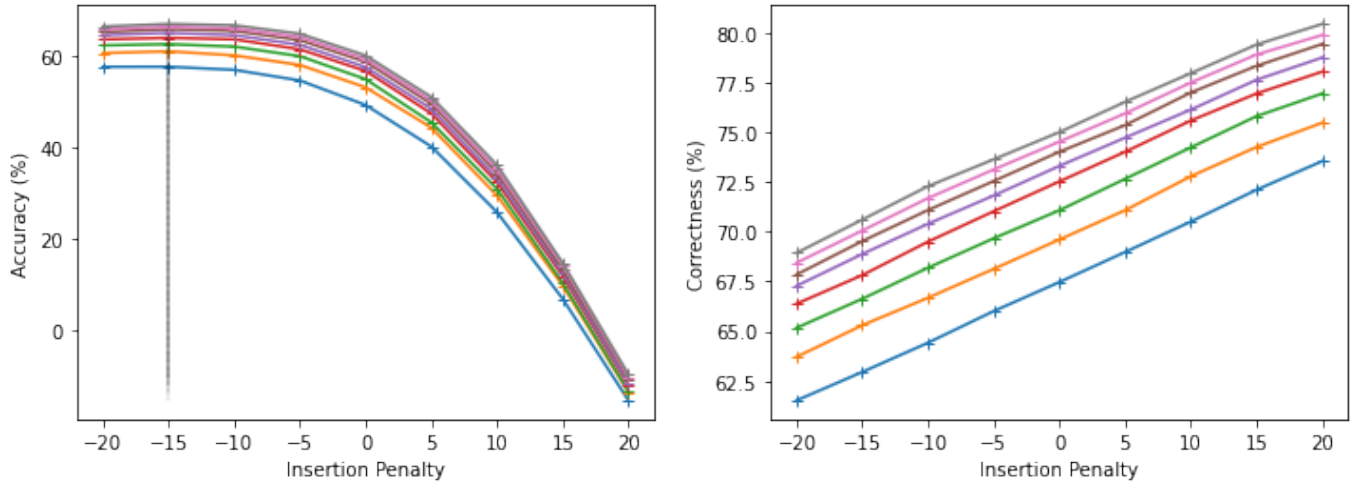

Figure 1: Plots of both the accuracy and correctness against word insertion penalty for a GMM-HMM model with varying numbers of GMM components (between 2 in blue, and 16 in grey). The input features to this model were Mel frequency Capstral coefficients with logarithmic frame energy, delta coefficients, and acceleration coefficients appended (MFCC EDAZ). The black dashed line was used to locate the optimal insertion penalty of the values used, and was found to be -15 in all cases.

```
../tools/steps/step-mono -NUMMIXES 8 ../convert/mfc13d/env/environment_E_D_A_Z
    ../results/mfc13d_E_D_A_Z_8_False/mono/

../tools/steps/step-decode -BEAMWIDTH 200 -INSWORD -15
    /homes/bjtk2/MLMI2/pracgmm/results/mfc13d_E_D_A_Z_8_False/mono hmm84
    ../results/initial/mfc13d_E_D_A_Z_8_False/mono/decode-hmm84--15-200-False/
```

Listing 1: Example code to build and then test a GMM-HMM model with 8 Gaussian components (increased incrementally by 2 every 4 iterations of Baum-Welch training). For decoding, the insertion penalty was set to -15 and the the beam width to 200. In this example the input had MFCC features with EDAZ parameterisation and a flat start. Figure 1 can be created by changing the -NUMMIXES number, Figure 2 by adding -FLATSTART after the step-mono command, and Figure 3 by changing ../convert/mfc13d/env/environment_$E_D_A_Z$ to ../convert/{features}/env/environment_{parameterisation}.

2.2 Complexity of the GMMs

Whilst Figure 1 demonstrates that the more Gaussian mixtures in the GMM the higher the accuracy, it does not clearly show how that accuracy increases with mixture number. Therefore, Figure 2 shows the accuracy as a function of mixture number for only the -15 word insertion penalty (as it was found to be the maximum from Figure 1), for the same MFCC EDAZ inputs. It is clear from this plot that increasing the number of mixtures does improve the accuracy, but with diminishing returns, as the rate of increase gradually flattens out to 0. Therefore, when factoring in the computational costs of using a larger number of mixtures, it seems sensible to use a value that is high enough to give acceptable accuracy, whilst being low enough that the models train quickly. From here on, this value is taken to be 8 mixtures.

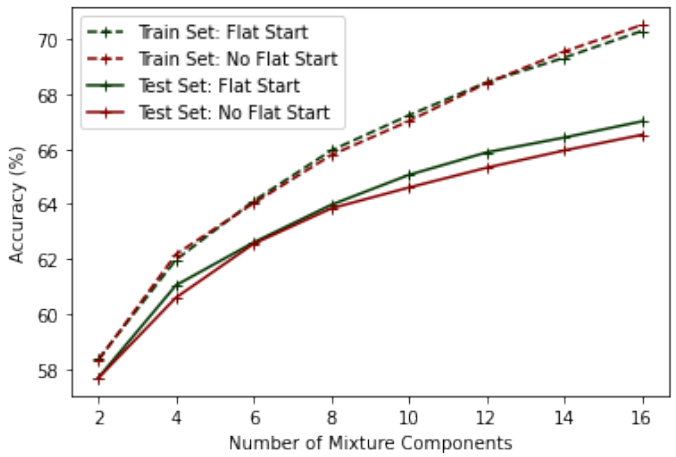

Figure 2: A clearer comparison of how the number of Gaussian mixture components effects the accuracy of the GMM-HMM. The dashed lines show the performance when testing on a subset of the training data. The increased difference between training and test accuracies with number of components demonstrates that the more complex models are prone to overfit. Again the feature type was MFCCs with the EDAZ front-end parameterisation.

2.3 Flat Start vs TIMIT-bootstrapping

There are two schemes for initialising the transition probabilities prior to Baum-Welch training [4]. The first is the flat start, and is the simplest approach of initialising all transition probabilities to

be equal, and setting Gaussian emission distributions to have mean and variances derived from the training corpus. The Baum-Welch algorithm then iteratively deviates from this uniform distribution to improve the log-likelihood via the Expectation-Maximisation (EM) algorithm. Alternatively, a more involved method for initialising the probabilities can be used, which uses the provided phone proportions from the phonetically-aligned TIMIT corpus, with repeated iterations of the Viterbi algorithm to converge the model parameters before a single Baum-Welch training iteration. This second method therefore requires the training corpus to have phone labeled phone alignments.

As can be seen from Figure 2 above, there is little difference in training and testing accuracies between the flat start and TIMIT-bootstrapping model initialisation procedures. Therefore, as the flat start is computationally faster to run, and slightly improves the accuracy, flat starts were used from here-on-in in the practical.

2.4 Comparing Features and Parameterisations

As described in the Introduction, there are different ways to encode speech to pass into the GMM-HMM (and ANN-HMM) models. The first option to choose is whether to use filter bank (FBANK) features or Mel-frequency cepstral coefficients (MFCC) features, and from there the front-end parameterisation needs to be chosen. Figure 3 was created to examine how each of these feature and parameter combinations performed when using the GMM-HMM model with 8 Gaussian components and an insertion penalty of -15 (which was previously found to be the best for GMM-HMM models).

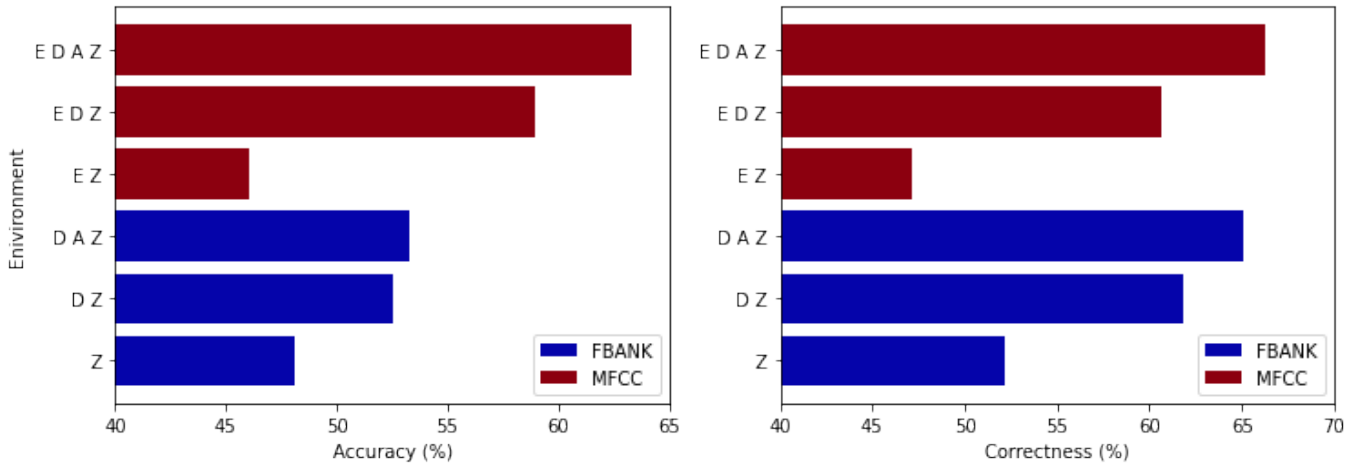

Figure 3: A comparison of front-end parameterisations for both the FBANK and MFCC feature types using 8 Gaussian mixtures, -15 insertion penalty, and a flat start. The MFCC EDAZ input was found to be by far the best, and therefore is used predominantly in the rest of this section.

The plot of accuracy in Figure 3 (Left) shows firstly that including first and second differentials improves performance, and secondly that the MFCC features perform better than the FBANK features for the same parameterisation. This is because although using FBANK features gives higher dimension input vectors (for the same parameterisation), each vector's elements are more highly correlated than MFCC inputs' elements. Therefore GMMs perform less well with FBANK features because each of the Gaussian components are assumed to be diagonal [10] (and hence uncorrelated), and thus more components are needed to effectively represent clusters of vectors with highly correlated dimensions. As a result, for the remainder of this section, the MFCC EDAZ input type is used unless stated otherwise.

2.5 Adding Context: Triphone Tying

So far only monophone models have been considered and no context accounted for. However, this could be too much of a simplification, as phones often are not completely independent of their neighbours in speech. As a consequence of how muscles in the vocal tract and mouth/nose move dynamically in between phones, the same phone in one context may sound slightly different in another, an effect known as co-articulation. Therefore a more accurate representation is to consider not only the current phone but also the ones preceding and following it, as in this way the different variants of the same phone can be identified from their context. This is achieved through using triphones, such as *s-p+iy* for the *p* phone in *speech*, as the prior *s* and proceeding *iy* sounds change how the *p* phone is pronounced.

However, an issue with using triphones is the size of the models it produces due to the combinatoric way in which they are combined. Therefore to reduce the number of states, triphones which have similar contexts can have their HMM parameters tied, so that in training the model now needs to optimise fewer parameters. There have been a number of suggestions on how tying can be accomplished, some requiring expert knowledge of co-articulation, but the method used in this practical is to use decision tree state tying [2].

Decision tree state tying is a top-down process, which starts with 1 state, and then iteratively branches out to create new states, stopping when the creation of new states either leads to nodes with too few triphones occupying them, or does not increase the log-likelihood sufficiently. In this way it is an unsupervised method, and does not require any expert knowledge.

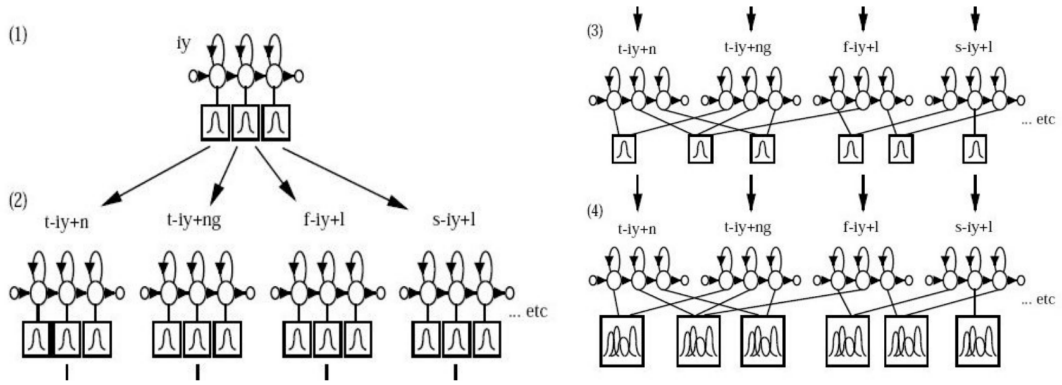

Figure 4: A figure depicting the process of moving from monophones to triphones, including tying the triphone states to reduce the number of model parameters. Figure taken from [2].

The diagram in Figure 4 above shows the method of going from a pretrained monophone GMM-HMM model to a tied triphone one. From steps 1 to 2, the monophone models with 1 Gaussian mixture are cloned for each of the triphone states, and between steps 2 and 3 the triphone states deemed most similar by decision tree tying have their parameters tied. Once the number of states has been reduced, the number of Gaussian mixtures (and hence the number of parameters) can be increased, as shown between 3 and 4. The effects of triphone tying were thus investigated using this approach for the GMM-HMM model with MFCC EDAZ inputs and multiple decision tree parameters, and this is shown below in Figure 5.

Figure 5 shows the variation of test accuracy when using triphones with decision tree tying and various parameters. The RO command defines outlier threshold, which sets the minimum occupancy of a tree state and prevents outliers with unusual acoustic properties from forming clusters with populations less than RO. The parameter denoted TB refers to the value given to the TB command, which dictates the minimum increase in log-likelihood from splitting any node in the tree on the next

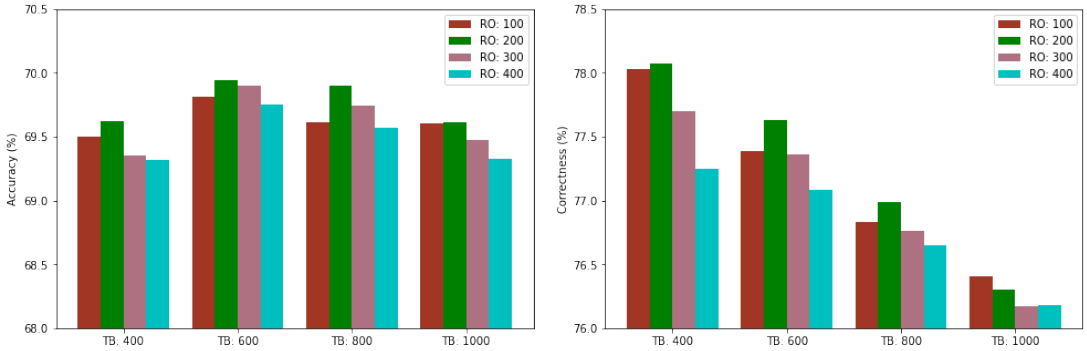

Figure 5: A plot demonstrating how changing the RO and TB values in Triphone decision tree state tying affects accuracy and correctness on the test-set. The combination of tested parameter values that maximised the accuracy was (RO: 200, TB: 600). However, the differences compared to other combinations are very small, and largely within 0.5%.

tying, before the process of building the decision tree is stopped. In this way, TB can be thought of as defining the depth or complexity of the tree. Both a high RO and a high TB parameter restrict the size of the tree and promote fewer tied states - if both were set to 0, then the number of tied states would equal the number of triphone states (108,289), and if both tend to very large values, the number of tied states approaches the monophone model number of 144: 48 phones x 3 sub-phones for each (this was tested by increasing RO and TB to very large numbers and observing the converged number of tied states).

From Figure 5 it seems that setting very high or very low values of TB and RO reduces the accuracy, which suggests that the low values are causing overfitting (as too many tied states lead to data sparsity), and the too high values are over simplifying the model and returning to monophone-like performance. However, having said this, the accuracy seems somewhat insensitive to the chosen parameter values, as the differences seen in accuracy in the figure above are still small and all better than the monophone model accuracy (see Table 1). The optimal combination was found the be RO: 200 and TB: 600, and hence these values are used for the remainder of the practical where triphone tying is involved.

Features	Triphone Tying Parameters	Number of states (sub-phones)	Test Accuracy
Monophone	-	48 x 3 = 144	63.84%
Triphone	TB: 600, RO: 200	1,092	69.94%

Table 1: Comparison between monophone and triphone features for the MFCC EDAZ input type with 8 Gaussian mixtures in the GMM. For reference, before decision tree tying there were 108,289 triphone states, which would make the triphone model dramatically more complex than if tying was not used. The number of states included sub-phone states in each phone's HMM. Hence for monophones, there were 48 phones each with 3 sub-phone states, giving 144 states in total.

```
../tools/steps/step-xwtri -NUMMIXES 8 -ROVAL 200 -TBVAL 600
    ~/MLMI2/pracgmm/results/initial/mfc13d_E_D_A_Z_8_True/mono hmm14
    ../results/triphones/mfc13d_E_D_A_Z_8_True/xwtri_200_600/
```

Listing 2: Example code snippet that is used to execute triphone decision tree tying. In this example the RO value was 200 and the TB value 600. The model used to align the unclustered triphones was the MFCC EDAZ model with a flat start, as represented by 'mfc13d_E_D_A_Z_8_True', and the clustered triphone model has 8 GMM components. The 'step-decode' step can then be used on this triphone model as done previously to test the model.

3 ANN-HMM Models

A novel approach suggested in [6] is to transfer from using GMM-HMM systems to ANN-HMM systems, which pass the input features into an artificial neural network that outputs HMM state probabilities in order to produce generative likelihoods[2]. This section discusses models of this kind, and compares them to the GMM-HMMs.

3.1 Testing Insertion Penalty and the effect of Adding Layers

Similarly to above in Section 2, the effect of different insertion penalties was first investigated to establish which gives best performance. Figure 6 (Left) shows the variation of accuracy as a function of insertion penalty, and demonstrates that the optimal value has now increased to $\gamma_{ins} = -5$, indicating that the ANN-HMM models are less likely to insert phones into the predicted sequence in order to artificially improve correctness. The right-hand plot of Figure 6 shows both the testing and training (tested on a subset of the training set) accuracy as a function of hidden layer number in the deep neural network. Increasing the number of hidden layers from 1 to 5 steadily increases the accuracy, but with a decreasing rate of improvement as the higher layer models begin to overfit. In fact, when the performance was evaluated for 8 hidden layers (with results omitted from the plot to keep axis scales tighter), a test accuracy of 5.45% was reached, which is a huge fall from the accuracies of fewer layers, showing a great extent of overfitting. Also noticeable is the significantly greater performance on the training data, which demonstrates how the additional complexity of the ANN architecture causes more overfitting than with HMMs (see Figure 2 for the HMM train/test comparison).

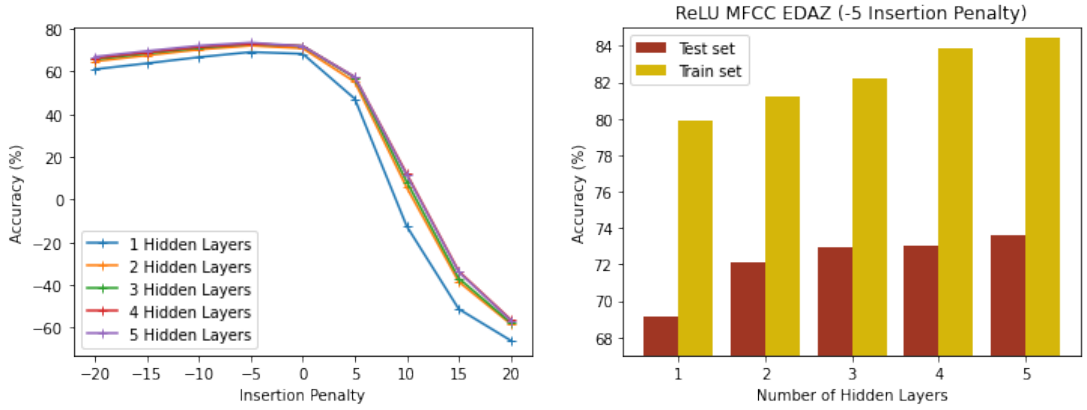

Figure 6: (Left) A plot of the variation of accuracy with insertion penalty for an ANN-HMM model with varying number of hidden layers in the ANN. (Right) Plot of the accuracy as a function of hidden layer number for only the -5 insertion penalty. In this right-hand plot the accuracy is shown both for the test set and a subset of the training data. For both plots, the input features were MFCCs with EDAZ front-end parameterisation, and the ANNs use a ReLU activation function in their hidden layers.

The diminishing returns from adding hidden layers is again demonstrated in Figure 7, which further compares using Sigmoid and ReLU activation functions on hidden layer nodes. Due to the near convergence of test accuracy at 5 hidden layers, and the propensity of many layer networks to overfit to the training data, 5 hidden layers is taken as the optimal architecture, as is using ReLU activation functions.

[2]A more detailed explanation of how this works can be found in the Appendix A.2

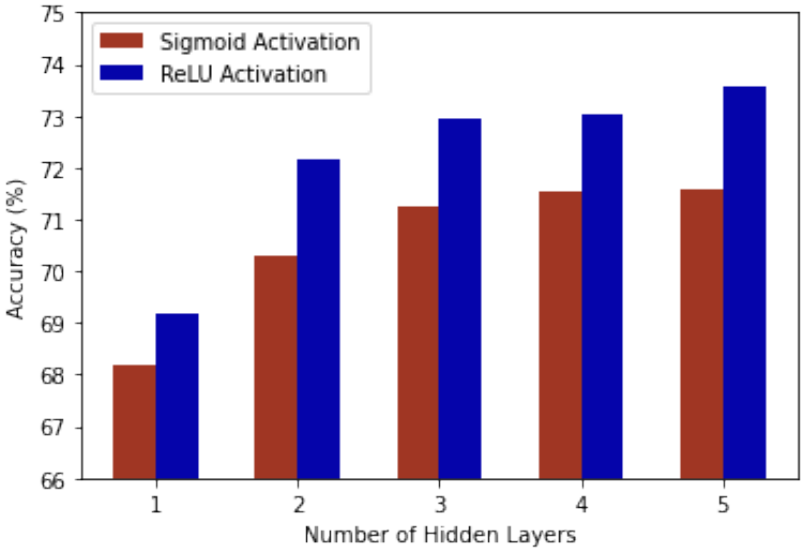

Figure 7: Comparison between using Sigmoid and ReLU activation functions in the hidden layers plotted as a function of hidden layer number, again for the MFCC EDAZ input types. It is clear from this plot that the ReLU gives better accuracy regardless of the number of hidden layers, and both the ReLU and sigmoid activation functions demonstrate the same convergence of accuracy with number of hidden states.

3.2 Varying Context Windows

A context window of size τ is a set of frames that are used as context for the current frame at time t, defined as all frames in the discrete time interval $\{t-\tau, ..., t, ..., t+\tau\}$. Figure 8 shows how varying the context window width affects the accuracy when using an ANN-HMM model with 1 hidden layer (for computational speed) and MFCC EDAZ input parameterisations.

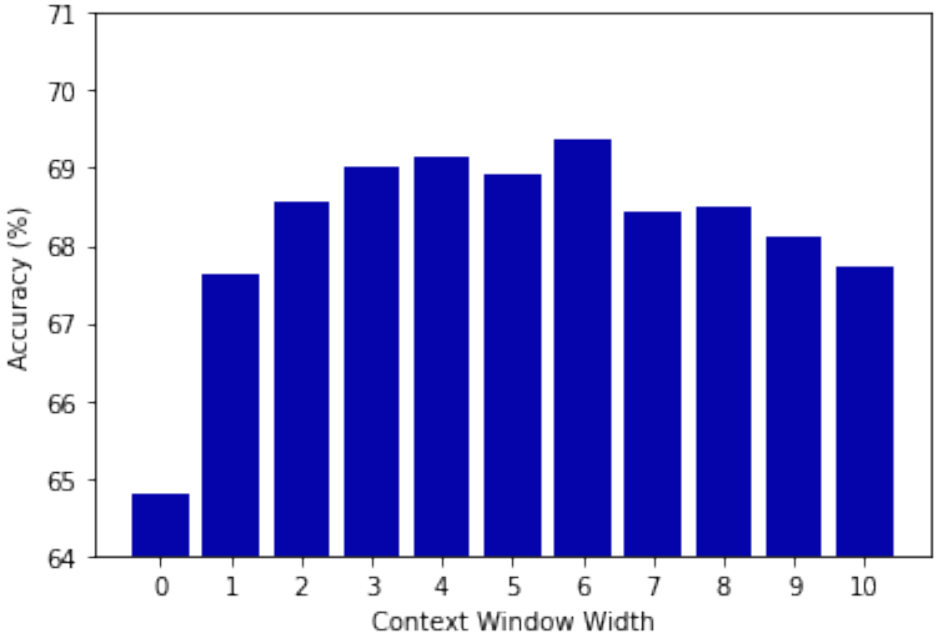

Figure 8: Accuracies of the DNN-HMM with a single hidden layer as a function of the context window width (a single hidden layer is used used to examine context window effects because it is computationally faster to train than networks with multiple hidden layers). The input type was using MFCC features with EDAZ parameterisation.

It is evident that not including any additional context at all hinders the performance, as each frame is shorter than a single phone, and so considering only the current frame makes it more difficult for the network to distinguish between phones containing similar sounding sub-phones. However, interestingly, it also seems to suggest that phones do not have long-term historical/future dependencies, as a 3 to 6 frame context gives optimal accuracy, and due to a single phone being constructed of multiple frames, it is likely that this will only represent a couple of phones. From a theoretical point of view this makes sense, as the phones heard in a word have little to do with the word's actual meaning, so historical and future context matters far less than in a natural language processing situation for example, where often words are far more likely to occur in one context than another. If only recent context gives insight into the current phone's classification, then using large context widths is likely diluting the information that they add, and hence performance begins to slowly decrease as context window width gets too large. This problem of dilution from high-dimension inputs could be addressed by using a more advanced way to encode historical/future context than simply concatenating frames, and this is discussed with RNNs in Section 3.5.

3.3 Comparing Features and Parameterisations

Similarly to in Section 2, the input feature types and parameterisations are now tested for the ANN-HMMs, and the relative performances plotted in Figure 9.

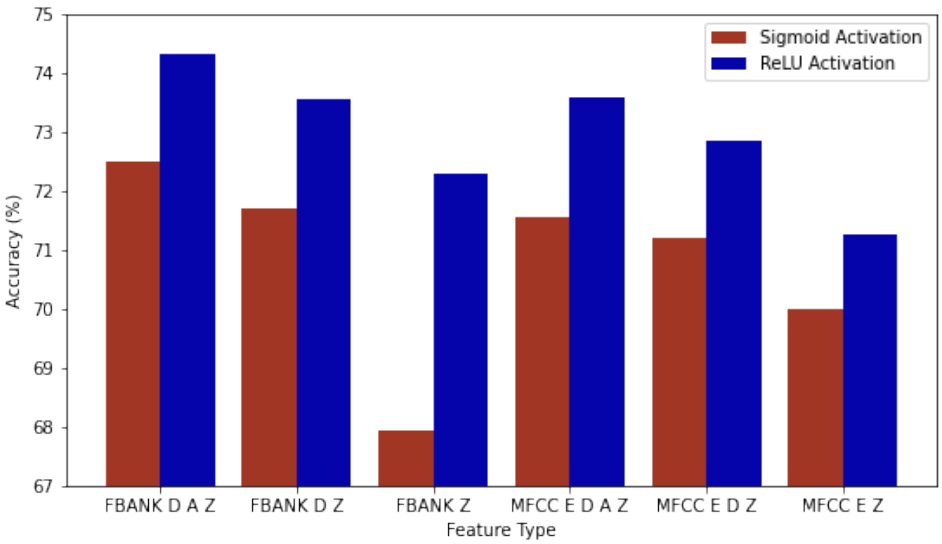

Figure 9: Comparison of feature types and front-end parameterisations for the ANN-HMM models. Unlike previously with the GMM-HMM models (see Figure 3), the FBANK feature type with DAZ parameterisation now gives slightly better accuracy than the MFCC EDAZ input type. However, for the same front-end parameterisation, the difference in accuracies between feature types is still small.

Figure 9 gives a different behaviour to that observed previously, as now FBANK input features give improved accuracies and the use of MFCCs is not as important. This is because DNNs are less vulnerable to correlated input vectors, and hence the greater number of dimensions (and consequent amount of information) in the FBANK features now present a small edge over MFCCs once correlation is not an issue. However, even now this difference between feature type accuracies for the same parameterisations is small, as DNNs handle correlated inputs by mapping the input vector into a higher-dimension space, which make any prior correlations somewhat redundant. Therefore, as MFCCs are constructed as a de-correlated version of FBANKs, they hold similar (but slightly less) information once mapped into the higher-dimensions of the hidden layers, and thus lead to similar predictions [7].

```
~/MLMI2/pracgmm/tools/steps/step-align ~/MLMI2/pracgmm/exp/MFC_E_D_A_Z/mono hmm84
    ~/MLMI2/pracgmm/exp/MFC_E_D_A_Z/align-mono-hmm84

../tools/steps/step-dnntrain -vv -GPUID 0 -MODELINI
    ../exp/new_models/DNN-1L.ReLU.MFCC_E_D_A_Z.ini
    ../convert/mfc13d/env/environment_E_D_A_Z
    /homes/bjtk2/MLMI2/pracgmm/exp/MFC_E_D_A_Z/align-mono-hmm84/align/timit_train.mlf
    /homes/bjtk2/MLMI2/pracgmm/exp/MFC_E_D_A_Z/mono/hmm84/MMF
    /homes/bjtk2/MLMI2/pracgmm/exp/MFC_E_D_A_Z/align-mono-hmm84/hmms.mlist
    ../results/aligned_with_MFCC_E_D_A_Z/DNN-1L.ReLU.MFCC_E_D_A_Z/
```

Listing 3: Example code snippet used to align and then train a DNN-HMM model (using MFCC EDAZ inputs). The '1L' corresponds to the DNN having 1 hidden layer (the model architecture was adapted from the models provided in HTK), and the 'ReLU' corresponds to a ReLU activation function being used for hidden layer nodes. The alignment step uses the pre-trained GMM-HMM model for that input parameterisation to provide the target labels for DNN training, in order to speed up training. The DNN training is then done by stochastic gradient descent with L_2 regularisation. In order to change context window size, a new model specification must be made, with the different context window size contained within the '.ini' file, and the number of hidden layers can be changed in a similar fashion.

3.4 Triphone Tying

The final test conducted on DNN-HMM models was to incorporate triphone state tying using the decision tree method discussed in Section 2.5. For comparability, and because they provided the optimal results on GMM-HMM models, the RO: 200, TB: 600 parameters were used, resulting in the same triphone states as before.

Input Features	Test Accuracy	No. States
Monophones	74.33%	144
Triphones (RO: 200, TB: 600)	77.81%	1092

Table 2: Comparison of the best tested monophone DNN model (the DNN with 5 hidden layers and FBANK DAZ input type) with the same DNN model using triphone input features. The number of states here is the number of output nodes of the DNN, and corresponds to the number of unique nodes for sub-phones.

Table 2 shows that once again, the extra information provided by including triphones improves accuracy, for as long as there are not too many tied triphone states (which is achieved by choosing high enough values for RO and TB).

3.5 Recurrent Neural Networks (RNNs)

Instead of using a deep feedforward network with context windows to model the emission distribution, a recurrent neural network can be used to give more complex weighting to historical and future frames. What is more, if it is the case that the current phones are largely independent of the phones long before/after them, then the vanishing gradients characteristic inherent to RNNs should not be an issue for this practical. The amount of context is controlled by the unfolding parameter, which defines how many frames the RNN trains on, and makes a single layer RNN similar in architecture

to a deep network with number of layers equal to the unfolding parameter. The effect of unfolding parameter is explored in Table 3 below.

Unfolding Parameter:

Model	Unfolding Parameter	Test Accuracy
RNN 1 Layer (FBANK DAZ)	10	74.09%
·	15	74.92%
·	**20**	**75.24**%
·	25	74.92%
·	30	74.79%

Table 3: Performance of a 1 layer RNN model with FBANK input features (DAZ parameterisation), varying with unfolding parameter. Of the unfolding parameters tested, the optimum was found to be 20.

In these investigations, the future context is fixed at 5 frames, meaning that the RNN training includes 5 frames in the future, and $n-5$ frames in the past, where n represents the unfolding parameter. This means that the optimal context found was to account for 15 historical and 5 future frames, which is different to what was found when looking at context window widths in the previous DNN-HMM models, where worse results were observed when 7 or more frames were used each for past and future context (i.e. 14 or more in total). The reason for this must therefore lie in the difference between how an RNN handles context compared to the context-dependent DNN - instead of increasing the input dimensions by adding context frames, the RNN encodes the context by adding it to the current frame being considered, thus keeping dimensions lower. The weighting to the different time contexts is also different in RNNs, as frames nearer in time have greater influence on predictions due to vanishing gradients, which could help to regulate to what extent uninformative frames longer in the past/future dilute the input's information.

Including Triphones:

In the models tested so far, adding phone context through using triphones has shown to improve the accuracy. Therefore we once again include them, and use tying to keep the number of HMM model parameters feasible. For the 1 layer RNN model with an unfolding parameter of 20 and an insertion penalty of -5, triphone tying was found to give a **test set accuracy of 79.27**% with the FBANK DAZ inputs.

4 Extension: LSTM and TDNN Models

The first model to be considered in the extension was the Long Short-Term Memory (LSTM) model. The shortcoming of RNNs is that, although they provide historical context, the nature of how recurrence is treated means that the weighting of the historical inputs decreases rapidly as more new inputs are given to the network. Therefore they fail to effectively take into account inputs that were more than approximately 5-10 time lags before [3]. LSTMs were introduced as a solution to this vanishing error problem, as they use an architecture of gates in order to selectively forget and remember information from previous frames based on what they deem important during training. With this framework, historical inputs can be remembered for time lags in excess of 1000 discrete time steps [3], a major improvement over RNNs.

Another modification that can be used to again provide time context in the speech signal is a Time Delay Neural Network (TDNN), which replicates a fully-connected layer for different 'time-bins' (stacks of sub-sampled frames taken at different times within $t = [\tau_{first}, ..., t, ..., \tau_{last}]$) [9]. In this way, the early layers of a TDNN analyse localised temporal patterns in the data, and the later layers analyse patterns over greater time periods[3]. The purpose of this structure is to automatically weight the importance of context frames to the current input's classification, whilst taking a long temporal context[4].

Model	Monophone Test Accuracy	Triphone Test Accuracy
LSTM 1L-ReLU FBANK DAZ	73.18%	77.07%
LSTM 2L-ReLU FBANK DAZ	73.31%	77.61%
TDNN FBANK DAZ	76.73%	**81.01**%

Table 4: Comparison of accuracies for the more complex models used in the extension. Surprisingly, the performance of the LSTMs were worse than for the RNNs considered in Section 3 (Table 3 shows RNN performance for monophones only). What is less surprising is that the time delay neural network performs better than RNNs, and gave the best accuracy observed in the investigations when triphones were used.

The results in Table 4 show the performances of the LSTM and TDNN architecture models. The TDNN model in this case had 5 hidden layers, similarly to the DNNs tested in Section 3, with ReLU activation functions on hidden layer nodes. The results show that the LSTMs do not perform as well as the RNNs, or in fact the 5 hidden layer DNN, despite their ability to recall far more historical information. A possible explanation for this is that although some short-term historical context can improve performance (as the trialled RNNs performed better than the DNNs), the amount of context provided by LSTMs is unnecessary, and a drop in performance is caused as the LSTMs only have 1 or 2 fully connected layers after the LSTM cell, in contrast to the DNN's 7 layers (6 after the input layer). The TDNNs, however, performed well, and achieved the highest test accuracy of the investigations. This could perhaps be attributed to the combination of flexibility in handling acoustic context whilst still retaining the same high number of hidden layers as the DNNs.

[3] A more in-depth discussion of TDNN architectures can be found in Appendix A.3

[4] The model used had 6 layers (4 hidden) and used sub-sampling to provide context over 28 frames, $t = [-16, 12]$. In order of increasing granularity the sub-sampling was:
$\{0\}, \{-7, +2\}, \{-3, +3\}, \{-3, +3\}, \{-1, +2\}, \{-2, -1, 0, +1, +2\}$.

5 Conclusion

A selection of the optimal model results from each section are summarised by Table 5 below.

Model	Input Type	Monophone Acc.	Triphone Acc.
GMM-HMM (8 components, no FS)	FBANK DAZ	53.26%	57.15%
GMM-HMM (8 components, no FS)	MFCC EDAZ	63.84%	69.94%
GMM-HMM (8 components, FS)	MFCC EDAZ	63.97%	70.02%
DNN-HMM (1 HL, no context)	FBANK DAZ	64.81%	72.49%
DNN-HMM (1 HL, ±6 CW, ReLU)	FBANK DAZ	69.36%	75.68%
DNN-HMM (5 HLs, ±6 CW, Sigmoid)	FBANK DAZ	71.56%	74.85%
DNN-HMM (5 HLs, ±6 CW, ReLU)	FBANK DAZ	74.33%	77.81%
DNN-HMM (5 HLs, ±6 CW, ReLU)	MFCC EDAZ	73.58%	75.77%
RNN-HMM (20 Unf)	FBANK DAZ	75.24%	79.27%
LSTM-HMM (2 L, ReLU)	FBANK DAZ	73.31%	77.61%
TDNN-HMM (5 HL, ReLU)	**FBANK DAZ**	**76.73%**	**81.01%**

Table 5: Summary of model performances on monophones and triphones. Key: FS = Flat Start (nos FS = Viterbi alignment); HL = hidden layer; CW = context window width; Unf = unfolding parameter.

There are a few general trends that can be observed from these results. The first is that all of the ANN-HMM models performed better than the GMM-HMM models, especially when the deep neural networks had more than 1 hidden layer. It also shows that MFCC features perform better for GMM-HMM models than FBANK features due to the uncorrelated input vector elements. However, for ANN-HMM models FBANK features perform better because neural networks are less susceptible to correlated inputs, and so the extra information provided by the higher dimensionality of FBANK features improved performance. In both cases, including the delta and acceleration coefficients increased the information given by the inputs and improved performance.

Also apparent is that including context from frames before and after the current frame in general improves the test accuracy, although including too many context frames in a naive way can lead to overfitting and decrease performance, as observed with the DNNs. The optimal set-up that was found was therefore using FBANK DAZ parameterisations for an ANN-HMM model with triphone states, where the artificial network has a way of encoding frame contexts without greatly increasing input dimensions.

References

[1] Garofolo, John S. et al. *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. 1993. DOI: 10.35111/17GK-BN40. URL: https://catalog.ldc.upenn.edu/LDC93S1.

[2] S. J. Young, J. J. Odell, and P. C. Woodland. "Tree-Based State Tying for High Accuracy Acoustic Modelling". In: *Proceedings of the Workshop on Human Language Technology*. HLT '94. Plainsboro, NJ: Association for Computational Linguistics, 1994, pp. 307–312. ISBN: 1558603573. DOI: 10.3115/1075812.1075885. URL: https://doi.org/10.3115/1075812.1075885.

[3] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to Forget: Continual Prediction with LSTM". In: *Neural Computation* 12.10 (Oct. 2000), pp. 2451–2471. DOI: 10.1162/089976600300015015. URL: https://doi.org/10.1162/089976600300015015.

[4] Keith Vertanen. "Baseline WSJ acoustic models for HTK and Sphinx: training recipes and recognition experiments". In: (Jan. 2006).

[5] Steve J. Young et al. *The HTK Book Version 3.4*. Cambridge University Press, 2006.

[6] Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton. "Acoustic Modeling Using Deep Belief Networks". In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.1 (2012), pp. 14–22. DOI: 10.1109/TASL.2011.2109382.

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

[8] Irina Kipyatkova. "Experimenting with Hybrid TDNN/HMM Acoustic Models for Russian Speech Recognition". In: *Speech and Computer*. Ed. by Alexey Karpov, Rodmonga Potapova, and Iosif Mporas. Cham: Springer International Publishing, 2017, pp. 362–369.

[9] Florian Kreyssig, Chao Zhang, and Philip C. Woodland. "Improved TDNNs using Deep Kernels and Frequency Dependent Grid-RNNs". In: *CoRR* abs/1802.06412 (2018).

[10] P.C. Woodland C. Zhang. *TIMIT Speech Recognition Practical Handbook (Version 7.0)*. 2021.

A Appendix

A.1 A brief overview of GMM-HMMs

A GMM-HMM system outputs the most likely phone sequence in a spoken utterance by using a generative model for the observed speech frames. To do this, a HMM 'state'[5] is created per phone, and the Viterbi decoding algorithm then used to find the most probable path through the looped-HMM system to recover the phone sequence. In this way the model can be thought of as maximising $P(\{S_t\}_{t=1}^T|\{x_t\}_{t=1}^T)$, which therefore requires us to calculate the likelihood $P(x_t|S_t)$ for each t, where S_t is a phone state in the HMM stack.

In order to calculate the likelihood, a GMM is used, with a different set of parameters that depend on the state. The GMM is trained iteratively by gradually increasing the number of Gaussian components while conducting 4 rounds of Baum-Welch training for each, as described in the Introduction.

Figure 10 shows the looped architecture, with the phone states stacked together, and each being represented by 3 sub-phone HMM states.

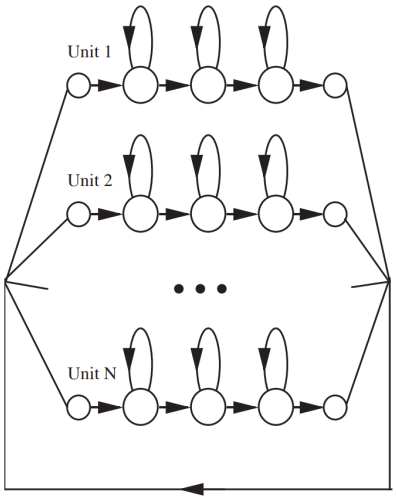

Figure 10: Diagram of the structure of a HMM system used in this practical. For the GMM-HMM, each of the sub-phone states in the phone triplets have an associated GMM, with unique parameters (assuming no parameter tying). For monophones there are 48 such states, and for triphones this increases exponentially, hence parameter tying is employed (the GMMs and sub-phone transition probabilities for tied states share the same parameters). Figure taken from [10].

A.2 A brief overview of ANN-HMMs

In order to explain the ANN-HMM system, it makes sense to compare it to how the GMM-HMM model works. The GMM model finds the likelihood, i.e. the probability of the observations given each phone's HMM state, and then the HMM Viterbi algorithm decodes the most probable state sequence from there in a generative fashion. In contrast, the ANN takes the frames of features derived from the utterance as input, and outputs the posterior probability of the phone state given the observation directly, and so cannot be directly incorporated into the HMM system in the same way.

Therefore, Bayes' rule is used to transform the posterior into the likelihood, using relative frequency of phones seen in training as the prior distribution of phone states. Now that $P(x_t|S_t)$ has been

[5]More accurately this 'state' is actually a sub-HMM with 3 states used to account for the non-uniformity of phone lengths

recovered using the ANNs, they can be inserted in the same way as the GMMs to model the HMM nodes' emission distributions.

The one thing that I have failed to mention thus far is how the ANNs are trained. Unlike GMMs which are unsupervised and learn through expectation maximisation, ANNs require supervision to train, and thus need corresponding phone labels for each of the phones in the utterance sequence. In order to save these labels being generated by hand, alignment can be used as an 'almost-as-good' alternative [6]. Alignment uses the results of a previously trained GMM-HMM system (which did not require labelled training data) as the assumed ground-truth, and allows for the error function to be calculated for SGD and backpropagation. This is why the *step-align* model is used before training any ANN in the scripts supplied by HTK.

A.3 Time Dependent Neural Network (TDNN) Architecture

Figure 11 shows a diagram of a possible TDNN base structure, and outlines how the time granularity hierarchy of TDNNs works. In the input layer, each frame is very local (in fact it is unique to a single frame in the diagram, but this need not be the case), and therefore captures fine-grain detail about the frame for that context shift. However, whereas a normal DNN with a context shift would now simply splice all of these input frames together, a TDNN instead passes these frames through to one of a set of connected hidden layers. In the first tier, there are fewer fully-connected (FC) layers (4 in this example), and each takes a splice of inputs from different locations of the input layer. Therefore, although each FC layer in the first tier also represent a different time frame, there are fewer of them, yet they still hold information over the whole input period, but the dimensions are lower than simple splicing. This concept is then repeated for the next tier, where the 4 lower FC layers feed into 2 higher FC layers, which therefore now each represent time context for 6 input frames whilst having inputs that are the dimension of 2 frames of the FC layers below. This can be repeated as much as desired, until the output layer is reached (which is in the next tier for the case in the diagram).

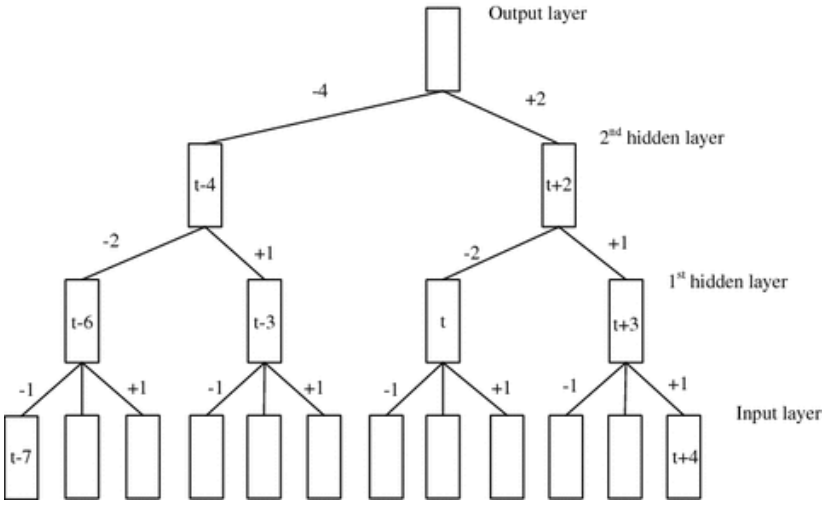

Figure 11: Diagram of a TDNN base structure, in this case looking at a time window of $t \in [-7, +4]$ for current frame t, with 1 context frame taken per shift, and a shift size of 1. For computational speed, sub-sampling is used in this diagram, as although the input layer slides over the whole input frame, the higher layers sample only select time frames from the previous layer. An alternative would be to not sub-sample the higher layers, but let the input layer have more frames per shift to reduce the input layer dimension instead. Figure taken from [8].

The TDNN in this investigation used an input time-frame $t = [-16, 12]$ with 6 layers in total (4 hidden). The sub-sampling in order of increasing granularity was:
$\{0\}, \{-7, +2\}, \{-3, +3\}, \{-3, +3\}, \{-1, +2\}, \{-2, -1, 0, +1, +2\}$.

This means that the output tier's FC layer holds contextual information from 28 frames despite having an input size equal to twice the output dimension of the previous tier's FC layers ($2x500 = 1000$ for the model used in this investigation). Not only does the TDNN reduce the input dimensions compared to using context windows in DNNs, but it also provides a means of automatically gauging context importance due through parameter training. If only frames near t are influential, then the weights connecting to frames far from t in the lowest tier will be nearer 0, and the model will not be negatively affected.