UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and Machine Intelligence 2019-2020

Module Coursework Feedback

Module Title: Speech Recognition

Module Code: MLMI 2

Candidate Number: H801L

Coursework Number: Final Project

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

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

Marker's Comments:

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

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

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

Continuous speech recognition on TIMIT dataset

Candidate number: H801L

11th January 2021

1 Introduction

In this project we build continous speech, speaker independent phone recognition system. In Section 2 we explore and evaluate application of GMM-HMMs models for this purpose. In Section 3, we consider ANN hybrid configuration, where a single ANN is used in place of state GMMs.

1.1 Performance evaluation

As opposed to isolated word recognition, in continuous speech we have to align the recognition sentence with the utterance transcript. In such case, we are not only concerned with minimising substitution (S) or deletion (D) errors but also insertion (I) errors. Therefore for evaluation of models we use accuracy as defined in Equation 1.

$$Accuracy = \frac{N - D - S - I}{N} \times 100\% \tag{1}$$

2 Monophone GMM-HMM

In our experiments we consider GMM-HMM systems with simple phone-loop grammar. In the following sub-sections we will investigate how choice of feature vectors, HMM initialisation, number of GMM components, and context-dependence affects performance.

2.1 Initial setup

For the baseline model, training starts by invoking *HInit*, which computes initial parameter estimates for each HMM model (48 HMMs, one per phone). Afterwards, Baum-Welch re-estimation, on the isolated phone data, is invoked with *HRest*. Baum-Welch conducts soft assignment of phones to states, therefore it can be more effective then Viterbi alignment of HInit, as in real speech there are no hard boundaries between phones.

Conducting isolated unit training at a phone level is tricky as the labeling is susceptible to human errors. For this reason, isolated phone initialisation is followed by embedded training invoked with *HERest*. We perform 4 Baum-Welch parameter re-estimations before increasing number of GMM components per state for more flexible output representation. Figure 1 shows how accuracy of the model, evaluated on the test set, changes with different stages of training.

2.2 Tuning HVite

The results presented in Figure 1 were obtained using the Viterbi decoder invoked by *HVite*. However, to make model evaluation and future model comparisons meaningful we had to first set beam search width and penalty insertion parameters.

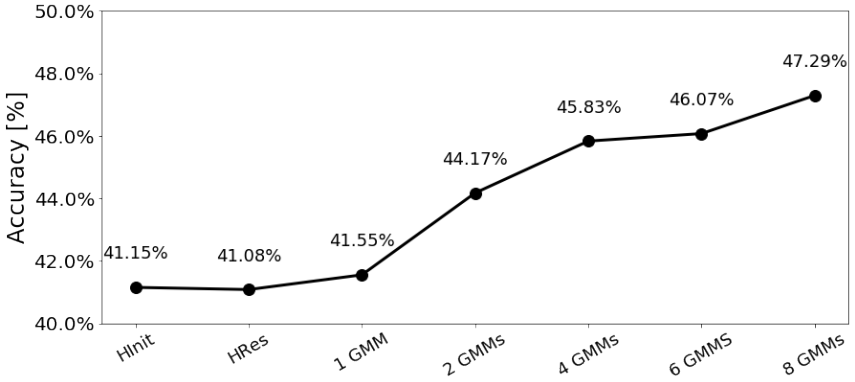

Figure 1: Test accuracy of GMM-HMM system evaluated at different phases of training.

Beam search pruning Beam search reduces computational time, by only considering models for which the maximum log probability is higher then the the log probability of the best model minus the beam width. Otherwise, the model is deactivated and not considered.

The default beam width in *HTE.phoneloop* was set to 100 and was found to be too small. Beam width of 180 proved to be sufficiently large for all sentences in the testset, sub-train, core test sets, to be decoded, across various models created in our experimentation.

Insertion penalty Insertion penalty is a fixed value added to a token as it transitions from the exit state to a new start state. By setting it to a negative value we can penalise more frequent word transitions thus reducing number of insertions, which can significantly improve model accuracy.

However, the more negative the insertion penalty the more deletion errors we make. We decide to use such insertion penalty which minimises the total error rate (maximises accuracy). Figure 2 shows that deletion and insertion errors are highly sensitive to the insertion penalty. The best accuracy was observed at insertion penalty of -15, 1.28 times higher then with a default setting of 0.

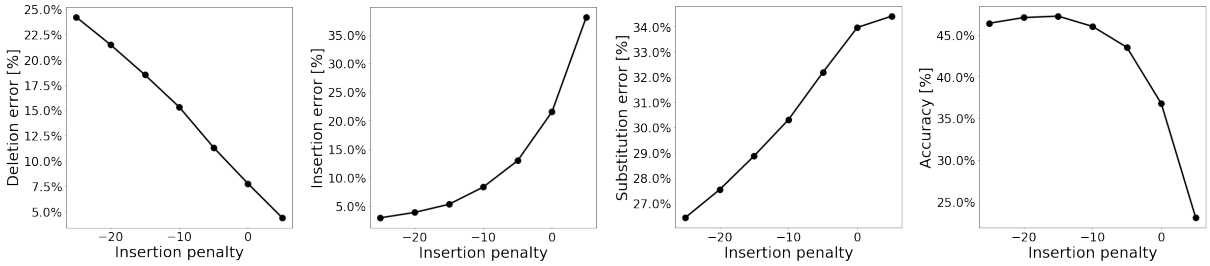

Figure 2: Effect of insertion penalty on accuracy and different error types.

Grammar scale factor Grammar scaling factor could also be tuned, have we used a language model (i.e. N-gram model) alongside HMM acoustic models. It would control recognition performance by varying the amount by which language model probability should be scaled before being multiplied with phone probability as token transits from the end of one phone to the start of another. However, since we use a simple phone-loop network with equal phone priors, we don't specify grammar scale.

2.3 Model initialisation

The *HInit* and *HRes* model initialisation requires manually segmented phone data. For each HMM phone model, we would take all training examples associated with that phone and compute model parameters which maximise likelihood of having generated those phones. An alternative initialisation method is to do a flat-start, where model parameters are set with the same value (i.e. equal transition probabilities and global mean and variance for GMM parameters). For most datasets (TIMIT is an exception) segmented phone data is not available, thus flat-start (or best previous model) is usually used.

Both initialisation methods yielded similar model performances for the initial network setup, as presented in Table 1. To get more conclusive comparison, we have also tested how initialisation affects network performance when best front-end parametrisation, found in next section, is used. Here again, the difference was only marginal. We decide to proceed with FlatStart initialisation for future experiments, however we believe that both approaches would be equally appropriate.

	FBANK + Z		MFCC + D,A,Z	
	HInit + HRest	Flat start	HInit + HRest	Flat start
Accuracy	48.06%	48.01%	63.84%	63.97%
Deletion error	18.0%	17.8%	11.1%	11.5%
Substitution error	28.4%	28.5%	21.1%	20.7%
Insertion error	5.6%	5.7%	4.0%	3.9%

Table 1: Comparison between HInit & HRest and Flat start initialisations for default FBANK front-end and best MFCC front-end as found in the next section.

2.4 Feature vectors

We tested two ways of parametrising speech waveform, 24 channel filter bank parametrisation and 12 Mel-Frequency Cepstral Coefficients (MFCC). Filter bank parametrisation aims to obtain non-linear frequency resolution for the wave form, to emulate the way human ear operates. However, filter bank amplitudes can be highly correlated which reduces their suitability for HMMs with diagonal covariances. MFCC derives its coefficient by applying discrete cosine transform to log filter bank amplitudes which produces decorelated and more compact speech representation. Our experiments show that MFCC performs better then filter banks when used with GMM-HMMs.

Both representations of speech vectors are normalised by log-energy to account for varying volume of utterances. Feature vectors can be extended by computing first and second time derivatives of vector elements. These are meant to compensate for the assumption of HMMs of observations' conditional independence from all previous observations given the state (which is not true for speech).

Figures 3 and 4 present how error rates and accuracy change, as we add time derivatives to feature vectors (2 GMM-HMMs system was used). Better performance of MFCCs seems to be majorly caused by lower substitution error rate, which can be attributed to MFCCs good discrimination. Due to MFCCs compact form and decorelated elements, HMMs can more effectively learn recognising utterances corresponding to the phone they model. It is also apparent that as we include time derivatives, the deletion error rate decreases while insertion errors increase. However, for MFCCs the deletion error decreases by relatively more then insertion error, thus the accuracy improves after adding time derivatives.

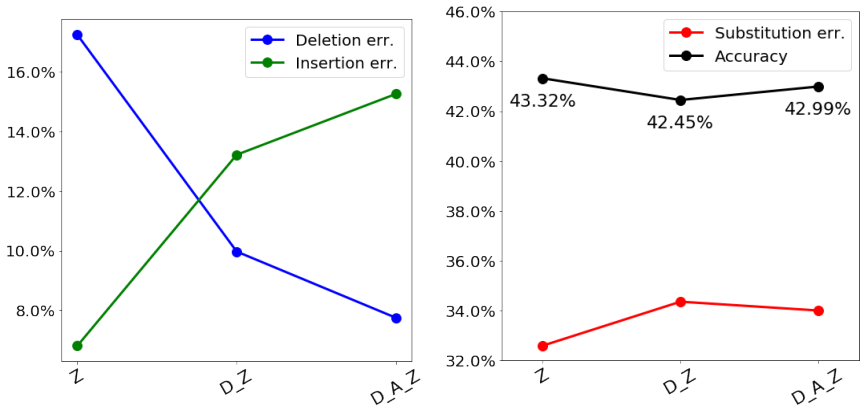

Figure 3: Effect of adding normalised first and second order time derivatives to **filterbank** feature vectors. Insertion penalty set to **-15**

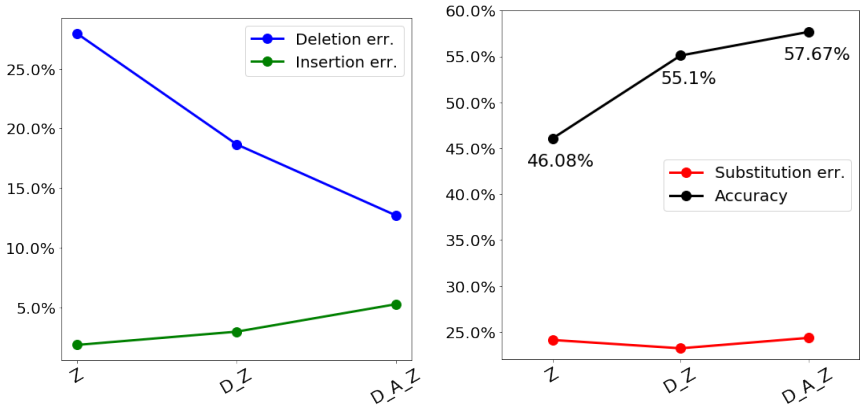

Figure 4: Effect of adding normalised first and second order time derivatives to **MFCC** feature vectors. Insertion penalty set to **-15**

Networks, in Figures 3 and 4, were evaluated using insertion penalty of -15. At this setting we have noted a high discrepancy between insertion and deletion errors for MFCC based network with no deltas. However, Figure 5 where models are evaluated with insertion penalty of -5, proves that observed trends and benefits of time derivatives are consistent when insertion penalty is adjusted.

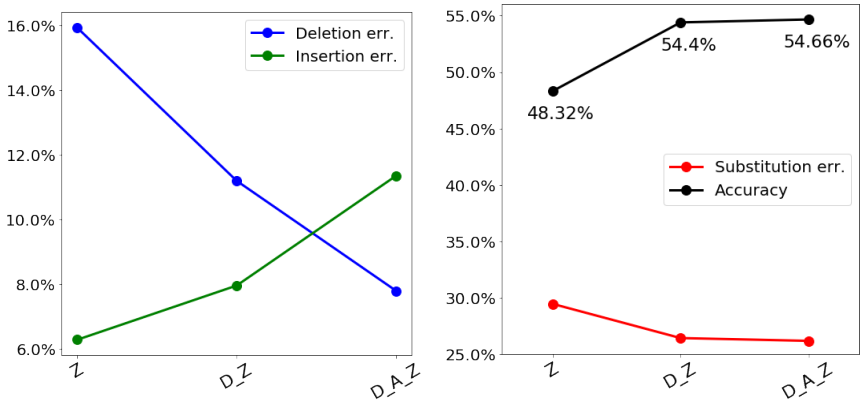

Figure 5: Effect of adding normalised first and second order derivatives to **MFCC** feature vectors. Insertion penalty set to **-5**.

2.5 Number of GMM components

Modeling state output distribution with only a single Gaussian with a diagonal covariance, constrains the shape of the distributions to be unimodal Gaussian. Models can be made more flexible through use of mixture of gaussian, which allows them to learn more assymetric and multi-modal feature vector distributions.

Figure 14 shows that accuracy increases with GMM components as the states' capacity at modeling associated feature vector distribution increases. The positive trend indicates that the model would benefit from even more components (however the training time would increase). In theory with enough GMM components we can learn any output distribution. In practice, however, we expect to observe overfitting at some point. In Figure 14 we see that the generalisation gap increases with number of GMM components, which indicates that as we include more parameters and make the model more flexible it picks up more of trends, which are present in the training data only.

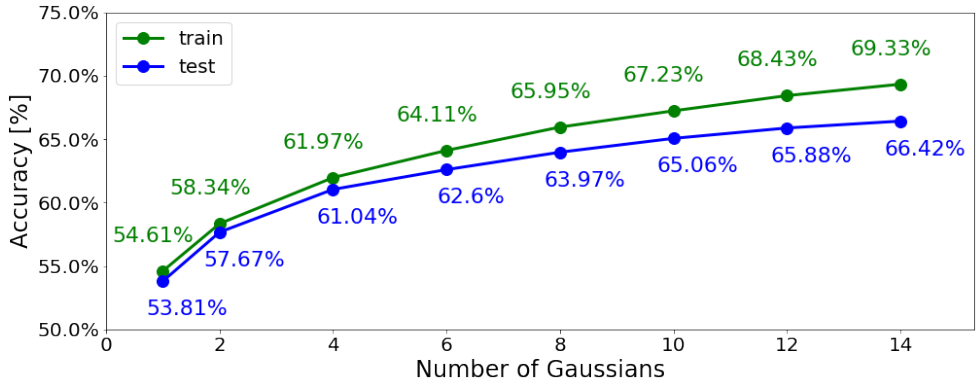

Figure 6: Test and train accuracy increases with the number of GMM components.

2.6 Context dependent triphones

Monophone vs. Triphone Context-independent monophones, considered thus far, benefit from low number of required HMM models (one for each of 48 phones). However, their main disadvantage is low robustness to phone variations, such as co-articulation. An alternative approach is to use context-dependent triphones, for which the feature vectors are much more consistent. However, using triphones requires more HMM models, training of which might be infeasible given relatively small TIMIT dataset size. For this reason we consider tree state tying, one of the clustering techniques to reduce number of system parameters.

Tree state tying In this technique we first transform pretrained monophone HMMs to triphone models and cluster them together. Then, we consider variety of split questions and itteratively choose state splits which offers the highest log likelihood gain. This leads to a top-down sequential optimisation of the network. We can control number of final clusters with two threshold parameters, RO and TB. RO defines the minimum number of training examples each cluster should have, which is important to ensure that each state output distribution is well estimated. TB determines the minimum gain in log-likelihood that we want from a split. When building large vocabulary systems, tree state tying is a better clustering technique then an alternative data-driven method, as it can be used to synthesise models for unseen triphones.

Choosing RO and TB Tree state clustering happens for a model with single gaussian per state. However, when, at the later stage of training, more Gaussian components are added, the number of parameters per state increases proportionally. Therefore it is important to tune number of clusters for a desired GMM model. We conduct our optimisation for GMM with 12 components. In Figure 7 we investigate how accuracy and number of clusters vary with RO, while TB is kept constant at $TB = 800$. There is little variation in accuracy with RO values, with highest accuracy for $RO = 200$. Number of clusters drops as RO is increased beyond 200, however no accuracy gain is observed. This suggests that the number of examples per class at $RO = 200$ is sufficient for robust output distribution estimation and there is no need to set this threshold to a larger value.

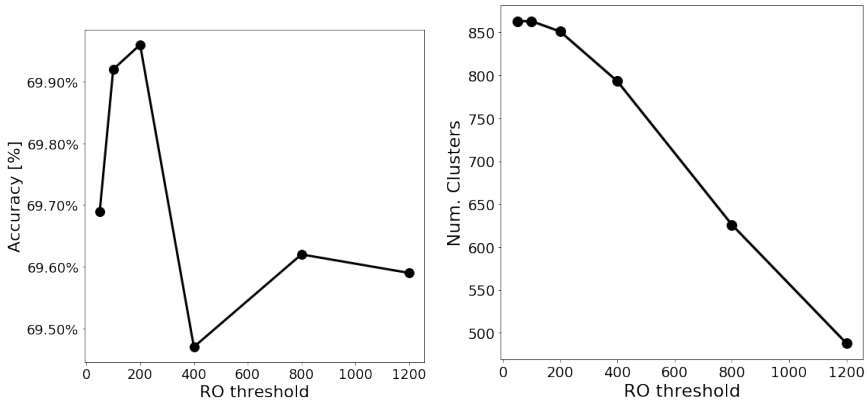

Figure 7: Effect of RO threshold on system accuracy and final number of clusters.

At RO set to 200, and optimum TB parameter for GMM with 12 components is found at $TB = 800$, as shown in Figure 8. Such RO and TB combination reduced the number of clusters (for seen triphones) from 60201 to 851.

Figure 9 shows that the more GMM components we use, the lower the number of optimal clusters is. This is because the number of parameters needed to estimate, grows with with the number of GMM components, thus requiring more data per cluster for robust estimation. However, the amount of training data is fixed, which is why the maximum number of clusters needs to be constrained. Additionally, the higher the number of GMM components, the higher the flexibility of each state at representing distribution of feature vectors, therefore we don't require as many state clusters to fit the data well.

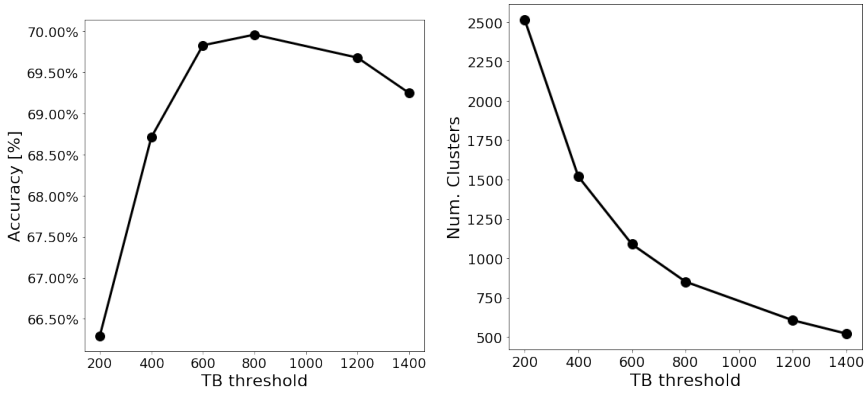

Figure 8: Effect of TB threshold on system accuracy and final number of clusters.

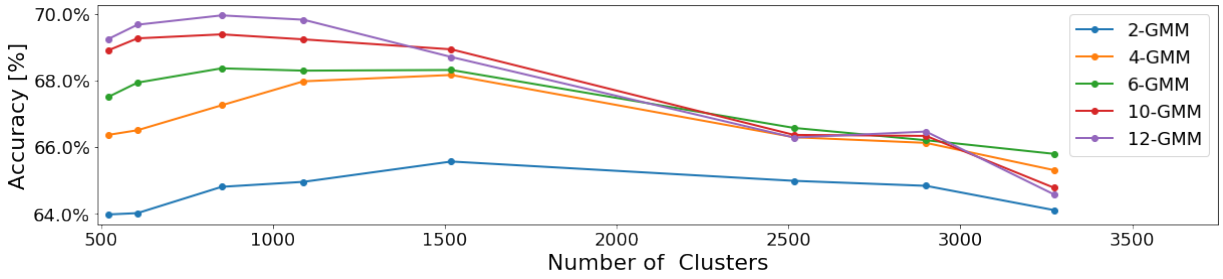

Figure 9: Final accuracy achieved by systems with different number of GMM components, as the number of final state clusters is varied.

3 ANN-HMM

In GMM-HMM systems each state has its own GMM to represent associated output distribution. In ANN-HMM hybrid configuration, all GMMs are replaced with a single ANN network which takes a frame window feature vector (together with any context frames), as an input, and classifies which state the frame came from. These posterior probabilities are divided by training set prior to obtain likelihood, such that the resultant ANN-HMM system can be used for recognition in the same way as GMM-HMM systems, using HVite.

We expect the ANN based state output distributions to perform better, because ANNs can form highly complex and non-linear mappings between inputs and outputs. Moreover, all network parameters are trained on all training examples, rather then only on train data associated with a given state. Finally, both DNN and RNN architectures can effectively utilise additional temporal data through context width or recurrent connections.

3.1 Single Hidden Layer DNN-HMM

Feature type & context width We start by considering a single hidden layer DNN, to investigate how choice of front-end parametrisation and context width affects performance. Figure 10 shows that when no context is used, time derivatives significantly improve phone recognition accuracy. Time derivatives are still useful when medium ($\{-2,-1,0,1,2\}$) and large contexts ($\{-4,-3,-2,-1,0,1,2,3,4\}$) are used, however, the performance boost is incrementally smaller. This was expected as both context window and time derivatives convey temporal information about feature vectors, therefore by extending the context window the information overlap is greater.

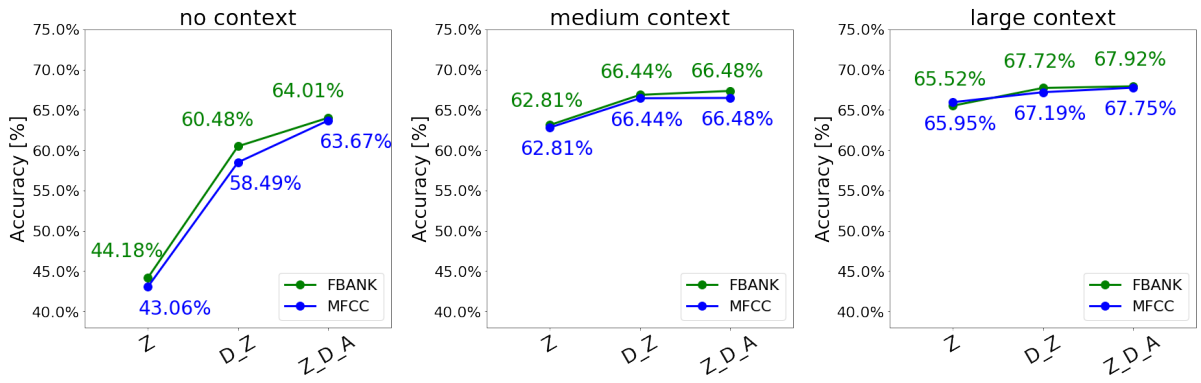

Figure 10: Comparing accuracies of systems with different context window width and input feature vectors.

It also appears that the gain from extending context-window is higher then the gain coming from including deltas in feature vectors, i.e. accuracy for no-context 'Z_D_A' systems is lower than accuracy of large-context 'Z' systems.

Another observation to be made is that the fbank based systems achieve, in general, higher accuracy then MFCC. ANNs do not require uncorrelated inputs, thus the transformation used to obtain MFCCs from filter bank amplitudes appears redundant. Moreover, filter banks might be performing better, because they retain the speech information which is lost during transformation to MFCCs and which could be utilised by the ANN.

For future experimentation we choose fbank parametrisation with D and A deltas.

Frame-level classification To evaluate quality of learned state distributions we can look at accuracy of frame level classification. For a system using monophone target units, the final output layer learns to classify, from which of 144 states, a given frame came from (144 states, because there are 3 states for each of 48 phone HMMs). Therefore the higher the frame-level classification accuracy, the better system has learned to represent output distribution of each state. This should correspond with higher *HVite* phone-level test accuracy the system can achieve. As expected, Frame classification accuracy trends in Figures 11 and 12 are consistent with those in Figure 10.

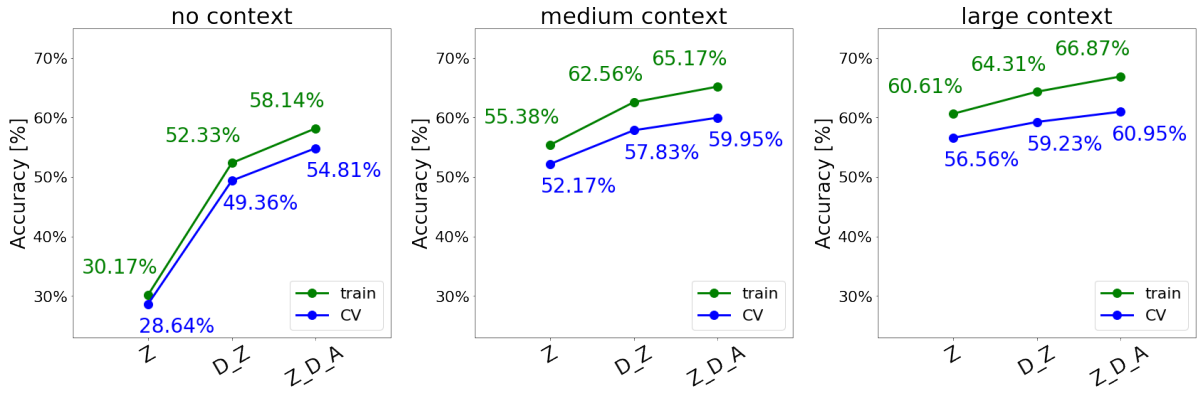

Figure 11: Comparing **Frame classification** accuracies for **FBANK** based systems with different context window width and input feature vectors.

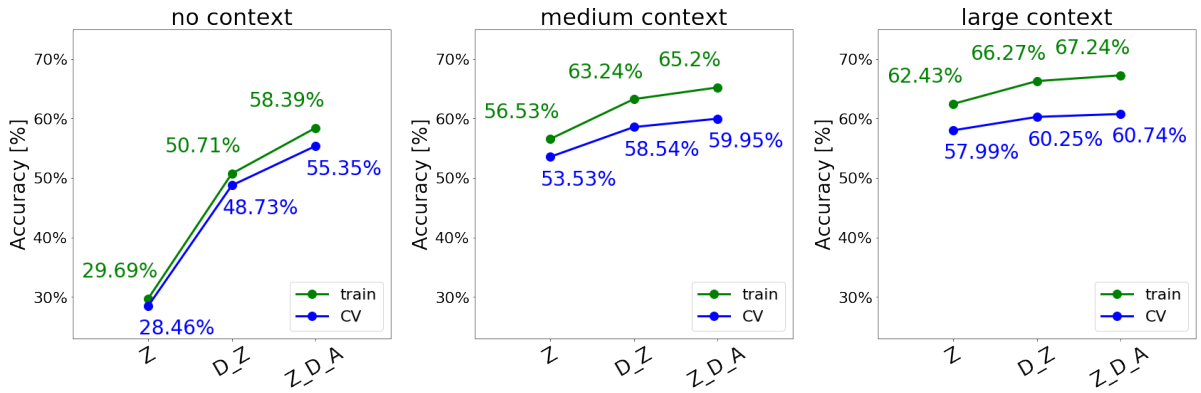

Figure 12: Comparing **Frame classification** accuracies for **MFCC** based systems with different context window width and input feature vectors.

3.2 Varying Depth of DNN-HMM

In the previous experiment we have considered DNN networks with a single hidden layer. By including more layers we can expand model capacity and improve the representative power of DNN at modeling states output distributions.

Figures 13 and 14 show how final frame-classification and phone recognition accuracies, respectively, change with number of layers. In both plots similar trends can be observed. We see that the train accuracy increases up to layer 4 and test accuracy up to layer 5, which is due to **increasing representative power** of the network. We also notice that the **generalisation gap** increases with number of hidden layers, which is also shown in Figure 15, where gap between train and CV accuracies grows at a faster rate for a deeper network. This suggest that networks with increased model capacity pick up more patterns present in training data only. We can reduce generalisation gap by increasing the default L2 weight, however, as presented in Table 2, this resulted in little to no benefits. Other regularisation techniques such as early stopping are already used. Alternatively, we could pretrain each layer individually or increase amount of training data.

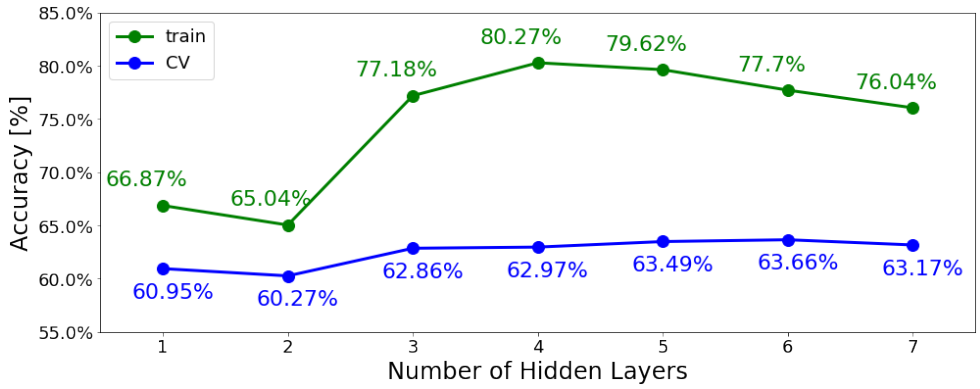

Figure 13: **Frame classification** accuracy variation with number of DNN hidden layers.

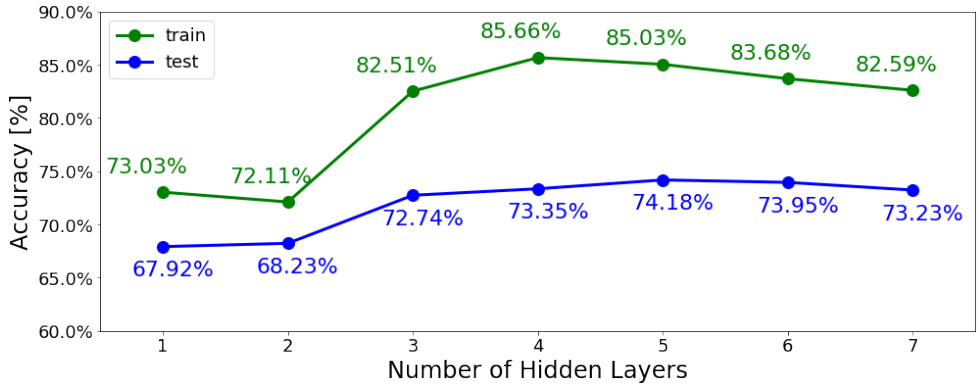

Figure 14: **Phone recognition** accuracy variation with number of DNN hidden layers.

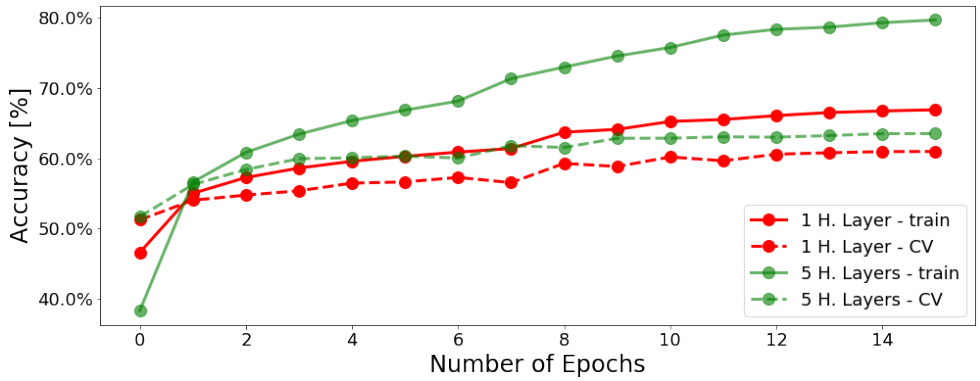

Figure 15: Comparison training frame-classification accuracy and the size of generalisation gap for DNN networks with 1 and 5 hidden layers.

	Frame classification		Phone recognition
	train	CV	Core test
$L2 = 0.0010$ (default)	79.78%	63.40%	74.18%
$L2 = 0.0015$	78.91%	63.32%	73.73%
$L2 = 0.0020$	78.16%	63.72%	73.69%
$L2 = 0.0030$	77.76%	63.67%	73.55%
$L2 = 0.0040$	78.40%	63.76%	73.71%
$L2 = 0.0050$	76.66%	63.40%	73.23%

Table 2: Frame classification and phone recognition accuracies at different L2 regularisation.

Figures 13 and 14, also show, that for higher number of layers both train and test accuracies decrease. We would expect to observe decrease in test accuracy due to **overfitting**, however, the training accuracy should continue increasing. The pattern of decreasing train accuracy with depth can be explained by two factors. First, as the network grows in parameters the optimisation problem becomes harder and training takes longer, however, simultaneously, the network overfits and early stopping terminates training before the deep networks could achieve their high expected train set accuracy.

We have also tested networks with 8, 9, and 10 hidden layers (see Table 3), however for these, the training has not converged, achieving low 5% train accuracy after first epoch and thereafter. We believe that such behaviour, could have been caused by large number of parameters making optimisation hard and primarily by the depth of the network causing issue of **vanishing and exploding gradients**.

	Frame classification		Phone recognition
	train	CV	Core test
8 hidden layers	5.17%	6.04%	7.19%
9 hidden layers	5.17%	6.04%	7.19%
10 hidden layers	5.17%	6.04%	7.19%

Table 3: No training occurred for deep DNNs

3.3 Triphone Target unit

Similarly to GMM-HMMs, ANN-HMMs can use context-dependent triphone HMM models, to make our system more robust to phone variations. For this purpose, we perform frame alignment with 851 clustered states found in Section 2.6. This increases number of parameters in the output layer, as now we conduct softmax over 851 rather than 144 classes. Figure 16 compares train and CV frame classification accuracy, across epochs, for monophone and triphone target units. Since there are almost 6 times more classification classes, the triphone target classification task is much harder, thus triphone model has lower frame classification accuracy. However, towards later epochs, the gap between train accuracies becomes smaller. This is because output distributions of context dependent triphones states have much lower variance and the discriminative power of each state is higher.

Table 4 compares phone recognition accuracies achieved by the network with monophone and triphone target units. Model with triphone HMMs performs better on all sets.

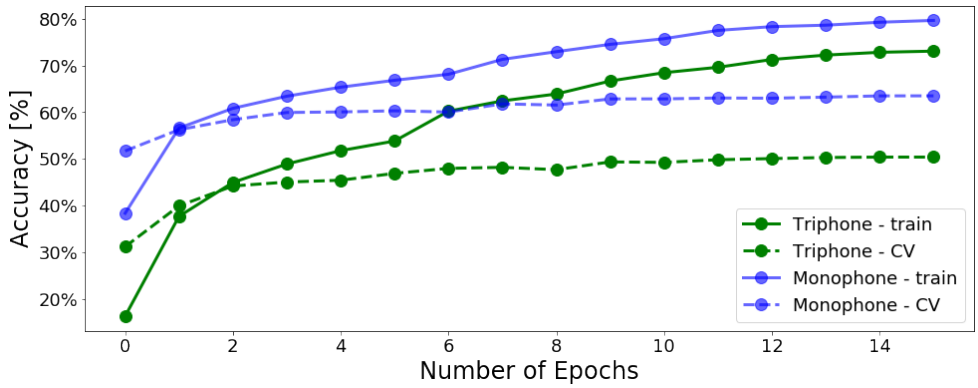

Figure 16: Comparing evolution of training and CV accuracy for **triphone** and **monophone** based 5 hidden layer DNN networks.

	test set	core-test set	sub-train set
Accuracy - Monophone target	75.22%	74.18%	85.03%
Accuracy - Triphone target	78.54%	77.17%	92.12%

Table 4: Comparison of phone recognition accuracies evaluated for test, coretest, and subtrain sets.

3.4 RNN-HMMs

DNNs don't make any prior assumptions about sequentially of the data, and the amount of temporal information they can extract from the context window is limited. For example, DNNs are not robust at handling different speaker rates, infering longer term dependencies, or identifying temporal trends at different timescales [5]. RNNs are an alternative ANN architecture, which treats data sequentially and can alleviate some of these issues. In RNNs the output of a layer is a function of the input and the back history vector which summarises information from previous frames. Such architecture allows to encode the dynamic contextual information in the recurrent weight matrix which is shared across time steps, thus making the contextual window to be flexible (rather then fixed one as in DNNs).

Number of unfolding steps Figures 17 and 18 show how frame classification and phone recognition accuracies, respectively, vary with number of unfolding step for a single RNN layer with one FC layer added after the RNN and before the output layer. As the number of unfolding steps increases up to 20/25 steps, the model performance increases. This is due to the fact that the shared weight matrix can learn to decide how and what information from previous frames should be extracted to benefit discrimination of the state the current frame came from.

However, as the number of unfolding steps gets even higher, the performance decreases. This could be caused by the fact that RNNs are ineffective at learning long temporal dependencies due to vanishing gradients [6]. If the gradients at each layer are less then one, by the time gradient backpropagates to far away layers it will be close to zero, thus preventing the context from those frames to be encoded in the weight matrix. However, during forward pass those early frames will contribute to the back history matrix which can introduce noise and reduce accuracy. Alternatively, if the gradients are higher then 1, the gradients may explode which destabilises training, as earlier layers significantly modify shared weight matrix between iterations.

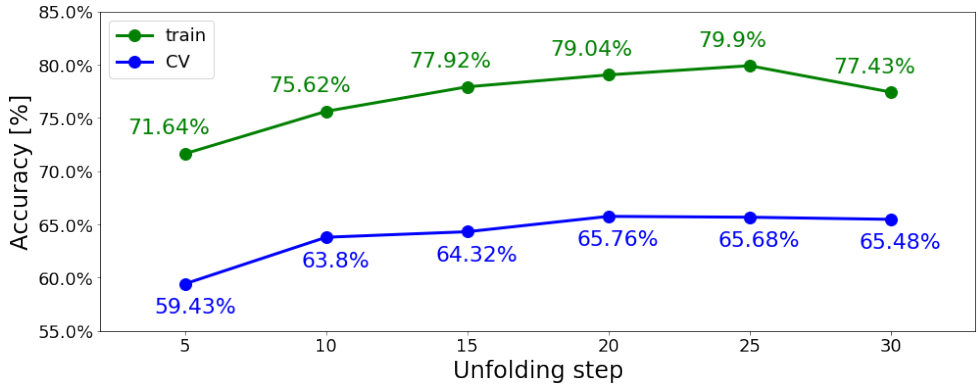

Figure 17: Change in train and CV frame classification accuracy with number of unfolding steps used in the RNN network. At each setting, 5 future frames are considered, with the rest being from the past.

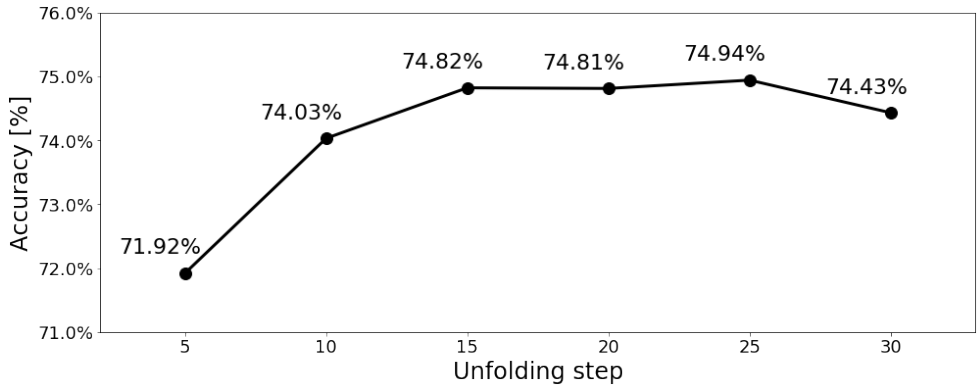

Figure 18: Change in phone recognition accuracy with number of unfolding steps used in the RNN network. Evaluated on main test set.

Triphone vs monophone state targets In Figures 17 and 18 we have used monophone target units to speed-up training and decoding phases. However, we expect RNN-HMM

systems to also benefit from triphone HMM models, due to the same reasoning as presented in Sections 2.6 and 3.3. Table 9 shows that Triphone based RNN achieved 4.52% better (core test) accuracy then models with monophone units.

	monophone target	trihone target
Accuracy	73.77%	78.19%

Table 5: Accuracy of RNN-HMM with 25 unfolding steps, evaluated on coretest set.

RNN vs. DNN Figures 19 and 20 compare training accuracy evolution for RNN and DNN networks. They show that 1 layer RNN has achieved comparable representative power to 5 hidden layer DNN, despite the fewer model parameters used by RNN, see Table 6. The performance of RNN can be attributed to more effective utilisation of temporal information from the data and a large depth when unfolded. Additionally, the lower number of model parameters could have boosted network's generalisation power, which is manifested in higher CV accuracy, as compared to DNN network.

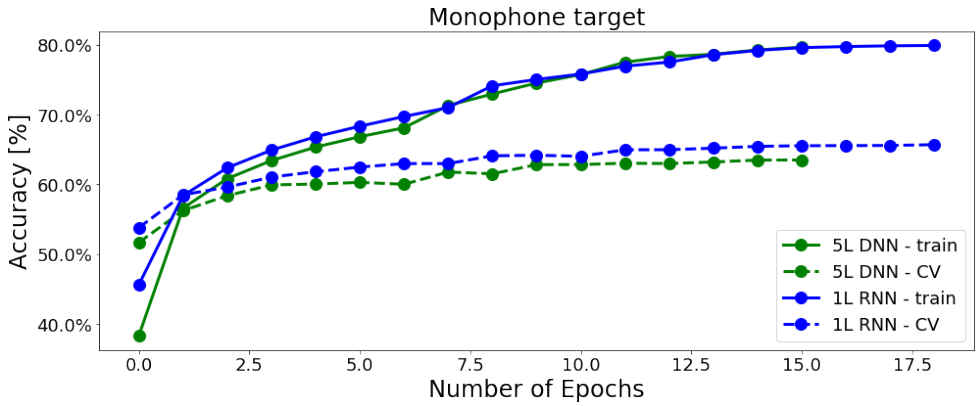

Figure 19: Comparing evolution of training and CV accuracy for **monophone** target based 5L DNN and 1L RNN networks.

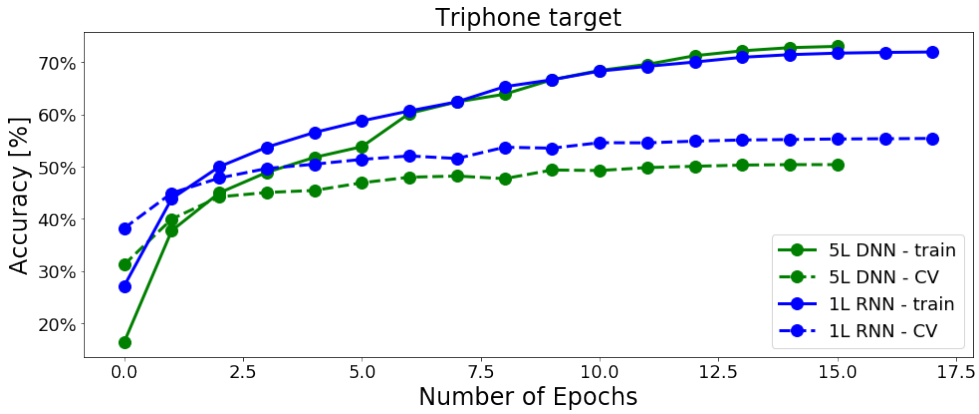

Figure 20: Comparing evolution of training and CV accuracy for **triphone** target based 5L DNN and 1L RNN networks.

	Monophone target		Triphone target	
	5L DNN	1L RNN	5L DNN	1L RNN
Number of Parameters	1,399k	609k	1,754k	965k

Table 6: Comparing number of parameters for different DNN and RNN networks.

3.5 Deep RNNs - extension

In theory shallow NN can represent same functions as deep NN, however deep NN can do it more efficiently by exploiting a hierarchical function formulation [7]. RNN, unrolled in time, can be seen as a deep feed forward neural network. However, simultaneously, they can be perceived as a shallow ones, if we consider that a speech frame at time t is processed only by a single non-linear layer before contributing to RNN output at time t. Therefore, in this section we consider different ways of increasing depth of RNNs in hope of improving their modelling capacity.

Input-to-hidden depth Increasing depth of the network, by adding FC layers before RNN could act as a way of preprocessing feature vectors and projecting them to a higher dimensional space [8]. In general, more abstract features are less sensitive to noise in data which should make the RNN more robust [9]. Working on more abstract features should also improve RNNs ability to learn temporal information as the relationship between abstract features can be represented more easily [10].

Figure 22 shows how core test set accuracy varies with number of FC layers appended before the RNN. We tested RNNs with 15 unfolding steps after observing decreasing trend in accuracy for network with 25 unfolding steps. We hypothesised that the network, with 25 unfolding steps and additional FC layers at the front, was too deep and suffered from the depth related problems mentioned in the previous section. However, lower number of steps has only partially alleviated the problem, as the accuracy increased with first FC layer, but then started decreasing.

Another, factor, which made the optimisation harder for this architecture is increased number of parameters. FC layers feed a vector of 500 element to the RNN layer, which has now weight matrix with 1000x500 parameters.

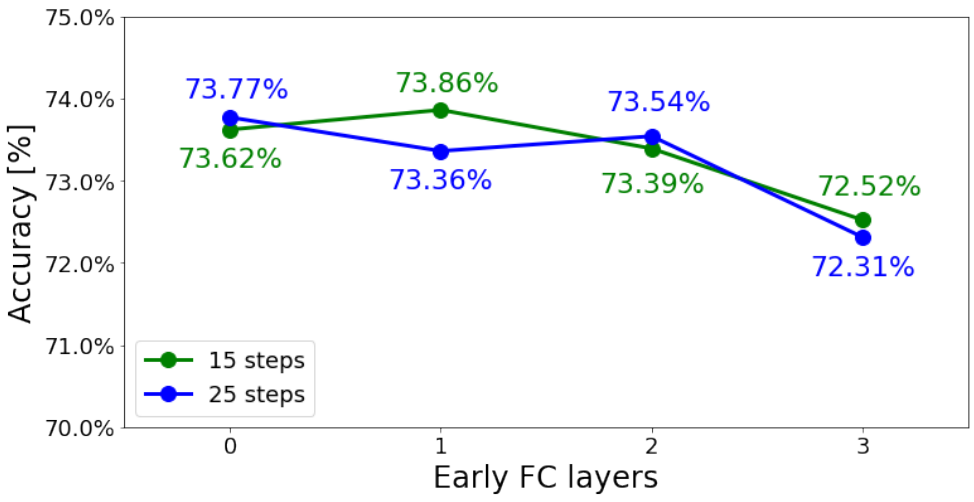

Figure 21: Accuracy variation as more FC layers are appended **before** the RNN.

Hidden-to-output depth Depth of RNN can be also enlarged by adding FC layers following the RNN output. We expect such architecture to be able to form more complex mappings between RNNs outputs, containing temporal information, and the final output layer of the network. This has been shown empirically to be important for RNNs performance [10]. Addition of FC layers between RNN and the output should also make the model more robust to variations in the hidden states of RNN.

We have observed small but consistent improvement to Core test set accuracy, see Figure 22. However, frame classification accuracy, in Table 7, displays no consistent trends. Perhaps, the more non-linear mapping makes the frame-level miss-classifications less severe (i.e. frames are classified to come from phone-states similar to the true state), which helps the Viterbi decoder in forming more sensible phone sequences.

We have extended the best deep RNN to a network with triphone target units and it achieved slightly better accuracy (78.24%) then the simple triphone RNN network (78.19%).

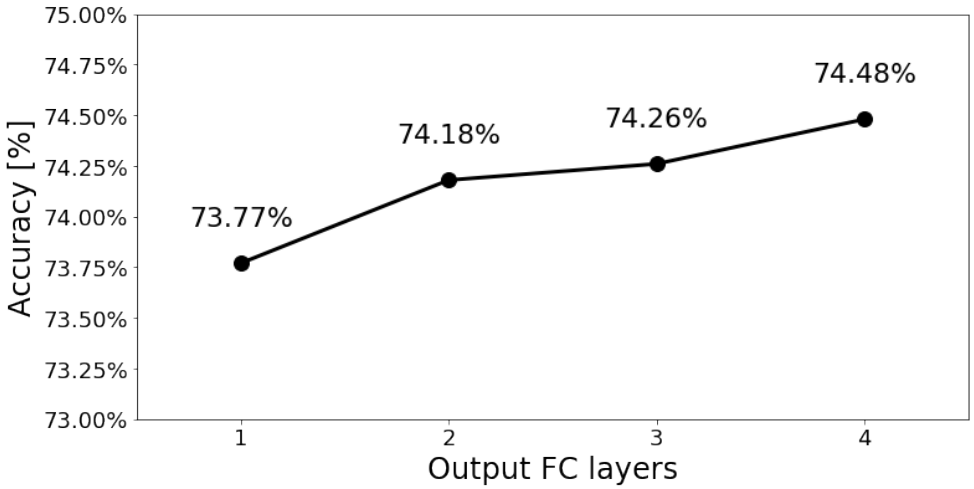

Figure 22: Accuracy variation as more FC layers are appended **after** the RNN.

	Frame classification		Phone recognition
	train	CV	Core test
1L RNN + 1L DNN	79.87%	65.68%	73.77%
1L RNN + 2L DNN	79.97%	65.08%	74.18%
1L RNN + 3L DNN	77.28%	65.11%	74.26%
1L RNN + 4L DNN	81.91%	65.18%	74.48%
Tri. 1L RNN + 4L DNN	75.05%	53.89%	78.24%

Table 7: Adding more FC layers **after** the RNN

Stacked RNNs Perhaps the most intuitive way of extending depth of RNNs if by stacking several recurrent layers together. As suggested in [11], the primary benefit of such architecture is the ability to operate at different time scales. Stacking RNNs creates a temporal hierarchy where bottom layers operate at low time scales (i.e. correlations across frames) and later layers at higher time scales (i.e. correlations across phones). Stacked RNNs can also make better use of parameters, as these can be distributed across layers [12].

Unfortunately, as presented in Table 8, the accuracy decreases with more RNN layers for both frame-classification and phone recognition. We hypothesise that this decreasing trend is common with RNN training problems we saw earlier, where RNNs inability to maintain long temporal dependencies via vanishing gradients and the training instability due to exploding gradients reduces the performance as we go deeper. This problem could be alleviated with gradient clipping, pre-training, highway connections, or through LSTM architecture which we explore in the next section.

	Frame classification		Phone recognition
	train	CV	Core test
1L RNN-HMMs	79.87%	65.68%	73.77%
2L RNN-HMMs	78.09%	65.29%	72.04%
3L RNN-HMMs	75.14%	63.13%	70.75%

Table 8: Accuracy for stacked RNNs with 25 unfolding steps per layer.

3.6 LSTMs

The limitation of RNN networks is their inability to form long range contexts due to vanishing gradients and unstable training due to exploding gradients. These issues can be addresses with LSTM architecture, which uses memory cell and gating to control how frames contribute to the memory state and creates shortcut paths for gradients.

Figure 23 compares accuracies for 1 layer RNN, 1 layer LSTM and stacked 2 layer LSTM, all using triphone target units. The plots show that train set frame-classification accuracy is much higher for LSTM networks. LSTMs ability to learn long temporal dependencies means that it can form stronger context to better discriminate which state a frame came from. It also uses 4 times more parameters then RNN, which increases its modeling capacity.

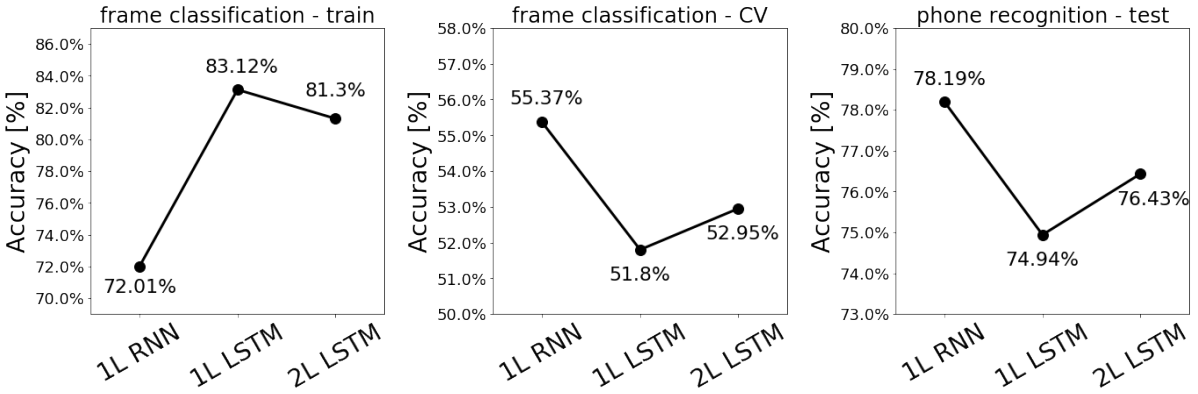

Figure 23: Comparison of accuracies for RNN and LSTM networks in triphone ANN-HMM systems.

However, the CV frame-classification and phone-recognition accuracies are lower for LSTMs. This could be due to low regularisation or poor hyperparameters selection. We hypothesise, that long temporal context might be detrimental to model performance, if the frames are correlated only locally. We also note, that data in test and train sets comes from different speakers, thus LSTMs could be learning long temporal patterns present in

training data, which are not generalisable to test data. This could explain why the gap between train and CV accuracies is larger then for RNNs.

Following, success of stacking FC layers after the recurrent unit, we have repeated this for the LSTM network. Figure 24 shows that gain from an extra FC layer before the output layer is greater for LSTM then it was for RNN. Potentially, due to LSTMs better ability to propagate gradients. However, the accuracy for model with three FC layers added before the output was lower then with 2 FC layers.

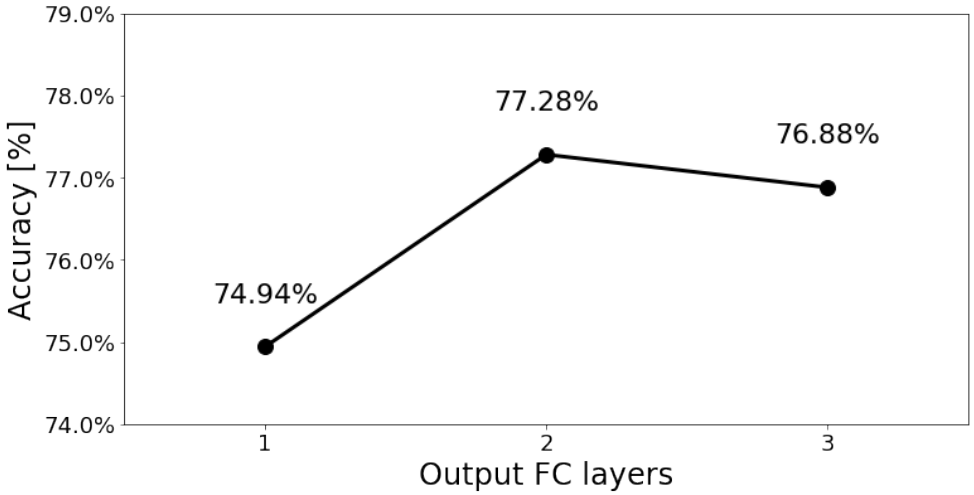

Figure 24: Accuracy for LSTM networks with different number of FC layers preceding the output layer.

4 Conclusion

We have discussed experimental results throughout the report. In this section we summarise merits of different design decisions and choose the best acoustic model studied.

Feature vectors For GMM-HMM models, MFCC parametrisation was much better choice for speech waveform representation, then filter banks. MFCC feature vectors offered better frame discrimination, thanks to more compact form and approximately decorelated coefficients. For hybrid ANN-HMM setup uncorrelated data is not required. ANNs with filter bank parametrisation achieved slightly higher accuracy, which is related to the fact that when forming MFCC from filter bank amplitudes we lose some information which could have been, otherwise, utilised by the ANN.

Including time derivatives in feature vectors improved accuracy in all systems. Time derivatives allowed for temporal information to be included within each feature vector, thus partially compensating for the HMM's assumption of output's temporal independence given its state.

Number of GMM components In GMM-HMM systems increasing number of Gaussian components improved accuracy, by improving states' capacity at representing their associated output distribution. However, we have observed an increasing generalisation gap and we expect overfitting to eventually occur at some high number of GMM components.

Higher number of GMM components also improved accuracy for triphone context-dependent HMM systems. However, the more GMM components we use, the fewer state cluster we should keep to ensure robust parameter estimation in each state and to prevent overfitting.

Better accuracy of GMM-HMM system have also indirectly contributed to ANN-HMMs system performance. Their acccuracy was dependent on the quality of the initial GMM-HMM based frame alignment provided.

Context dependence Triphone based systems can achieve much lower intra-state variance of speech frames thus leading to better discriminative power of each state. For both GMM-HMM and ANN-HMM systems, we have noted 4% to 5% increase in accuracy when changing from monophone to triphone HMMs. To make estimation of parameters feasible, however, we used tree state tying, to reduce number of states from 60201 to 851.

GMMs vs. ANNs We have observed significantly higher accuracy for all ANN-HMM systems as compared to GMM-HMM systems. This can be primarly attributed to three causes. First, ANNs use a signle network trained on all data samples, while in GMM-HMM systems, each GMM uses only a subset of the data. Second, ANNs are better at utilising temporal information via context window or recurrent units. Third, deep ANNs can learn more complex and more non-linear relationships between inputs and outputs then GMMs.

DNN vs. RNN RNN networks were able to achieve comparable accuracy to DNN networks despite lower number of parameters. However, their deep unfolded structure posed difficulties with gradient propagation. LSTM architecture solves this problem, however we found LSTMs to converge to models with large generalisation gap and worse test performance. We hypothesised, that high capacity of LSTMs made them overfit to patterns found only in the training data, especially that TIMIT dataset uses different speakers for train and test datasets. We believe that larger dataset would solve most of our problems in training of ANN-HMMs.

4.1 Best acoustic model

Overall, we have found that Triphone ANN-HMM system, with RNN layer followed by 4 FC layers architecture, achieved highest accuracy of 79.27% on the test set.

	test set	core test set
Accuracy	79.27%	78.24%

Table 9: Phone recognition accuracy for best model found.

References

[1] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu,G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland,The HTK Book (for version 3.5). Cambridge University EngineeringDepartment, 2009.

[2] G.E. Hinton, L. Deng, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, and B. Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. IEEE Signal Processing Magazine, pp. 2–17, November 2012.

[3] Sutskever, I. and Hinton, G. (2010). Temporal-Kernel Recurrent Neural Networks. Neural Networks, 23(2), pp.239–243.

[4] X. He, L. Deng, and W. Chou, "Discriminative Learning in Sequential Pattern Recognition — A Unifying Review for Optimization-Oriented Speech Recognition," IEEE Signal Processing Magazine, vol. 25, no. 5, pp. 14–36, 2008.

[5] Zia, T. and Zahid, U. (2018). Long short-term memory recurrent neural network architectures for Urdu acoustic modeling. International Journal of Speech Technology, 22(1), pp.21–30.

[6] Felix A. Gers, et al., Learning to Forget: Continual Prediction with LSTM, 2000

[7] Bengio, Y. (2009). Learning deep architectures for AI. Found. Trends Mach. Learn., 2(1), 1–127.

[8] Chan, William, and Ian Lane. "Deep recurrent neural networks for acoustic modelling." arXiv preprint arXiv:1504.01482 (2015).

[9] Goodfellow, I., Le, Q., Saxe, A., and Ng, A. (2009). Measuring invariances in deep networks. In NIPS'09, pages 646–654

[10] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to Construct Deep Recurrent Neural Networks," in International Conference on Learning Representations, 2014.

[11] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in Advances in Neural Information Processing Systems, 2013, pp. 190–198.

[12] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling," in INTERSPEECH, 2014.

[13] Zhang, Chao Woodland, Philip. (2018). High Order Recurrent Neural Networks for Acoustic Modelling. 10.1109/ICASSP.2018.8461608.