1 GMM – HMMs

Exercise 5.3: Flat Start

- Set-up: Compare FlatStart initiation with HInit/HRest initiation in terms of Phone Error Rate (PER) for context independent GMM – HMMs. Use `step-decode` to decode TIMIT test set using different Insertion Penalties. Front-end parameterisations: FBK and MFCC; with delta parameters: _Z, _D and _A_D; and Gaussian components per state ranging from 8 to 32.
- Experiment results:

MFC	Feature type			# Gaussian Components		Total
	_Z	**_D**	**_D_A**	**8-20**	**21-32**	
Times FlatStart Preferred	85.71%	28.57%	85.71%	75%	55.55%	66.66%
Min./Max. PER difference	-0.30% 0.02%	-0.08% 0.10%	-0.60% 0.07%	-0.60% 0.10%	-0.20% 0.07%	-0.60% 0.10%
Min. PER	47.73% (FlatStart)	36.48% (FlatStart)	31.34% (FlatStart)	32.44% (FlatStart)	31.34% (FlatStart)	31.34% (FlatStart)

- Observations: Overall, experiment results show no mayor difference between using FlatStart or HInit/HRest initiations in terms of PER, with the biggest difference being less than 1% (-0.62%) favourable to FlatStart. However, FlatStart seems to be the preferred initiation for most configurations, specially for MFCC front-end parameterisation (66.66% of the times). Specifically, FlatStart is clearly preferred when using static feature type _Z and 2nd delta parameters _D_A (85.71%), as well as for systems with 8 to 20 Gaussians per state (75%). In contrast, for FBK front-end parameterisation test result reveal no preference across all configurations, with system using FlatStart having the lowest PER.

- Conclusions: The initiation step aims to produce a prototype system for the HMM. Here an appropriate HMM topology (e.g.: 3-state left-to-right HMM) has more importance than any initial parameter estimates. In FlatStart, all the 1-Gaussian component HMMs are initiated with the same parameters, that is the global mean and covariance matrix of the training set. Hence FlatStart seems less expensive and more practical than HInit/HRest initiation. Recall HInit/HRest requires using a prior prototype model definition (e.g.: from best system in a similar database) as generator of speech vectors or some prior transcriptions to later produce initial parameter estimates by averaging over states. For this reason, FlatStart is often used for large-scale datasets, such as TIMIT. Another advantage of FlatStart is that it assigns soft boundaries for monophone boundaries during the Baum-Welch Training. We will use FlatStart as initiation method for future experiments.

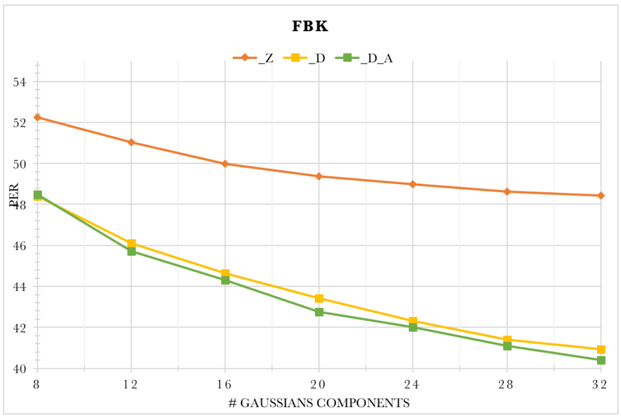
Exercise 5.4: Other Front-End Parameterisations

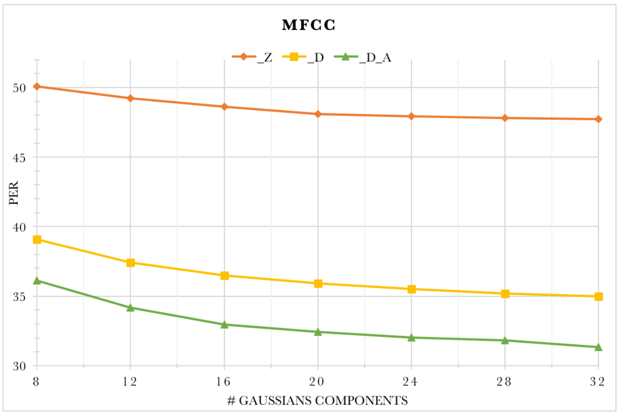
- Set-up 1: Compare monophone systems while adding delta parameterisations _D and _D_A for both front-end parameterisations FBK and MFCC in terms of PER. Use `step-decode` to decode TIMIT test-set using Insertion Penalty -12.5 and number of Gaussian Components has kept fixed to a maximum of.
- Experiment results:

PER	Feature type		
	_Z	**_D**	**_D_A**
FBK	52.25%	48.40% (-7.36%)	48.49% (-7.19%)
MFCC	50.06%	39.07% (-21.19%)	36.12% (-27.19%)

- Observations: For both front-end parameterisations FBK and MFCC adding differential coefficients translates into a substantial improvement in performance. For FBK, a 7.36% decrease (with respect to _Z) and 21.19% decrease for MFCC are observed with just the addition of 1st order delta parameters. For MFCC, using 2nd order delta parameters achieve a further decrease in PER of 27.19% (w.r.t _Z).

- Set-up 2: Compare performance of monophone systems with parametrisations _Z, _D and _D_A, while increasing the number of Gaussian components per state from 8 to 32 for both front-end parameterisations FBK and MFCC. Use `step-decode` with Insertion Penalty -12.5
- Experiment results 2:



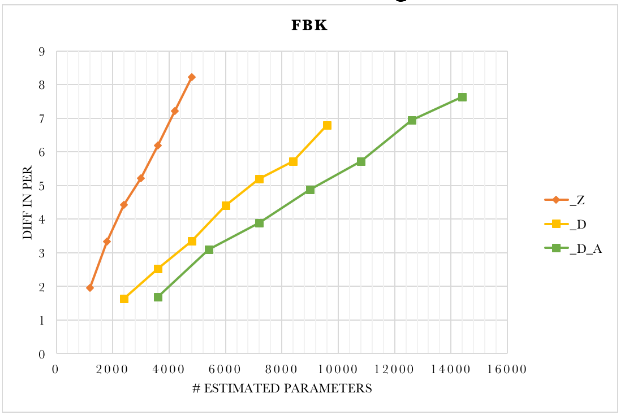


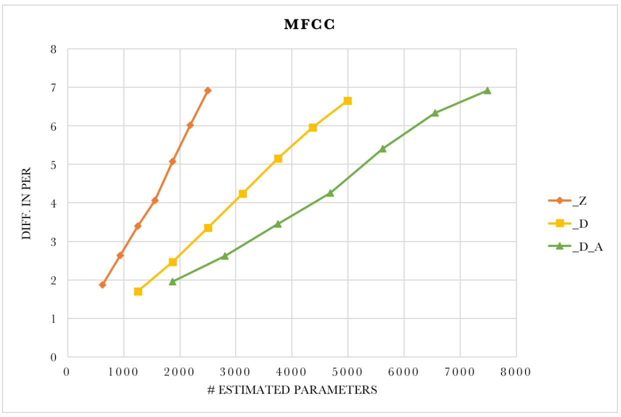
- Observations: It can be observed that for all feature types (_Z, _D and _D_A for FBK and MFCC) as the number of Gaussian mixture components of the output distribution increases, the more flexible the model is and hence PER in the TIMIT test-set decreases. Conversely, for a fixed number of Gaussian components, as a general trend adding dynamic information through the feature type improves the performance significantly. HMMs assume all output feature vectors are independent from each other, which is not true for speech. Hence, delta differential parameters add dynamic information to the system, adding some temporal context to it.

It is worth noticing:

- For MFCC parameterisation, there is very little change, when increasing the number of components from 20 to 32 in all three feature types.
- For FBK, that increasing the feature dimension from 50 to 75 in FBK (by adding 2nd order differentials) does not make a difference in terms of performance.

To explain these, the graphs below show the difference in PER between decoding TIMIT test-set and a subset of the training as the number of parameters estimated (i.e.: model complexity)





increases:

We see that as the complexity increases the difference is higher this difference in PER is. If we set a threshold for overfitting the training set at 4%, we see how most systems start overfitting with 16 Gaussian components or more. Also, models with lower feature dimension and hence less number of parameters tend to overfit faster. They start overfitting with a smaller number of mixture components, hence lowering their generalisation ability.

In all these experiments, the best performing front-end parameterisation is MFCC with _D_A. Hence we use a single Gaussian context-independent GMM-HMM with MFCC and _D_A and FlatStart as base phone for the constructing triphones in the next section.

Exercise 5.5: Triphone Decision Tree State Tying

Realisation of a particular phone may differ depending on its phonetic context due to co-articulation. To deal with this, context independent (CI) phone models (monophones) can be extended to context dependent (CD) models, triphones, which for a given base phone take the preceding and following phones into account. The number of possible triphones is 108,288 = (47+sil)x(47)x(47+sil) . These are called logical triphones. Each triphone is modelled by a 3-state left-to-right GMM–HMM. In order to improve the trainability of the system we need to reduce the number of triphone models by tying estimated parameters between states. `step-xwtri` does this using tree based clustering of states. `ROVAL` and `TBVAL` control the depth of these state-clustering trees and hence the final number of physical triphone models used in the recognition task.

- `ROVAL` is a threshold for outlier state removal. This avoids states from forming singleton clusters just from being different to the rest of states. The way it does that is by at each node removing questions that could lower the state occupancy below `ROVAL`.
- `TBVAL` is a threshold of the minimum log-likelihood increase. At each node a question (about the context of the triphone) will be ask unless, it produces an increase in log-likelihood lower that `TBVAL` or it causes the lower the state occupancy to fall below `ROVAL`.

Generally, the higher `TBVAL` is the shorter the tree and less state-clusters are form. This causes the state occupancy to be higher (on average), making the role of `ROVAL` less important; and final number of physical phones to be lower.

The best way to figure out the clustering thresholds that provide the appropriate number of clusters for a given number of Gaussian mixture components per state is by binary search. First, we test triphone systems with small fixed `ROVAL` (<100) and in a sense let the clustering trees grow with no restriction other than the corresponding `TBVAL`. Once the best `TBVAL` values are identified for each number of Gaussian mixtures, we tune the `ROVAL` value remove outliers and improve the quality of the tree-clustering.

For fixed `ROVAL` equal to 75, we obtain the following: (best in **bold**)

	TBVAL	850	950	1000	1050	1150	1250
#Gaussian per State	8	31.17	31.63	31.22	**31.17**	**31.13**	31.79
	12	30.83	30.91	31.28	**30.75**	30.98	**30.8**
	16	**30.08**	30.55	**30.32**	30.46	30.38	30.64
	20	30.7	30.27	**30.11**	30.36	**30.04**	30.47
	24	31.14	**30.08**	30.35	30.49	**30.23**	31.03
#Physical models		2797	2384	2198	2032	1790	1527

For example, fixed the number of Gaussian mixture components to 12:

TBVAL

1050			1250		
#Physical models	ROVAL	PER	#Physical models	TBVAL	PER
2032	75	30.75	1527	75	30.8
2049	100	**30.46**	1520	100	30.96
2022	150	30.76	1515	150	30.99

Hence the best performing triphone system for 12 Gaussian mixture models is ROVAL 100 and 1050 TBVAL with number of physical triphone models 2049.

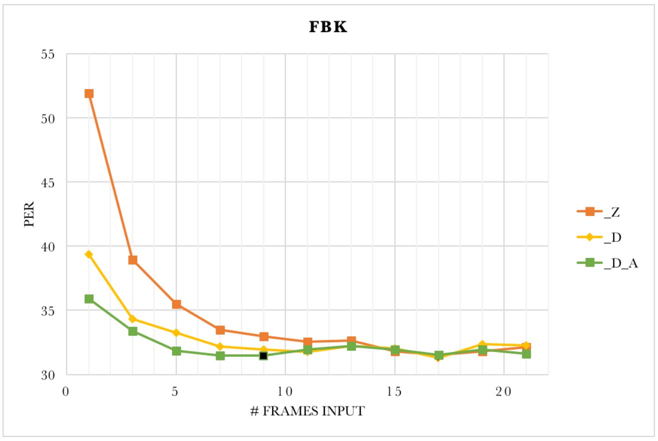
Experiments show the best configuration of ROVAL and TBVAL usually produces a number of physical models in the range of 1800 to 2200. By adding Gaussian components, in general, performance of the system is improved. However, from 24 Gaussian components onwards the relative improvement of the system slows down.

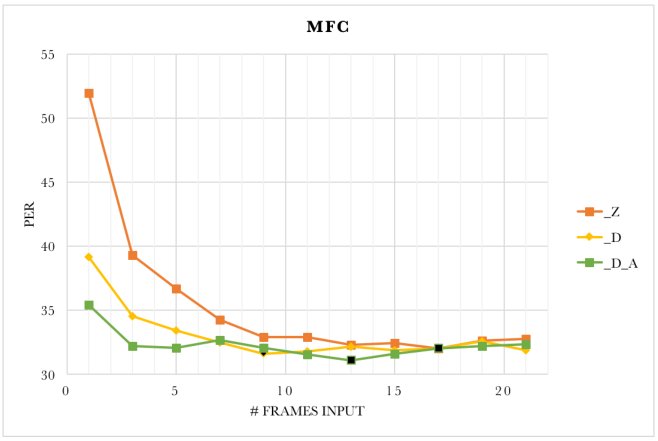
The best performing triphone system with ROVAL 200 and TBVAL 1000 (which results in 2165 physical models) and 20 Gaussian components has a PER of 29.50%, which an improvement over 2% with respect to the best performing context independent GMM – HMM.

2 DNN – HMMs

Exercise 6.3: Single Layer Experiments

- Set-up: Compare the PER of context-independent single hidden layer DNN–HMMs with parametrisations _Z, _D and _D_A while increasing the `CONTEXTSHIFT` (i.e.: the number of frames in input acoustic window) from 0 to ±10. In all this cases the acoustic context is assumed to be symmetric. Number of nodes per layers is kept constant at 500. After optimizing the insertion penalty to -4, use `step-decode` to decode TIMIT test-set.
- Experiment Results:



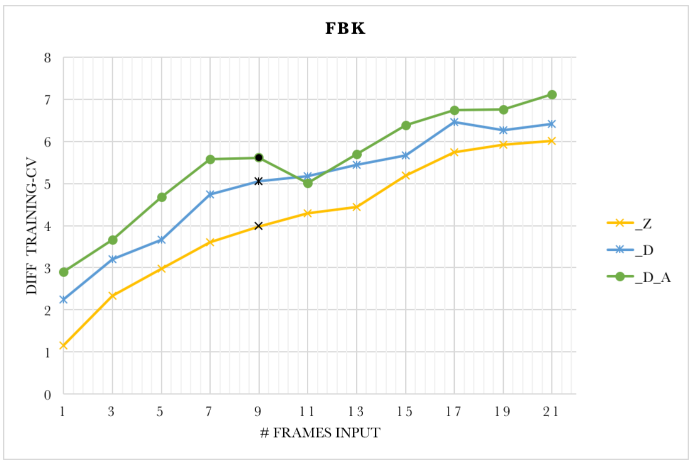


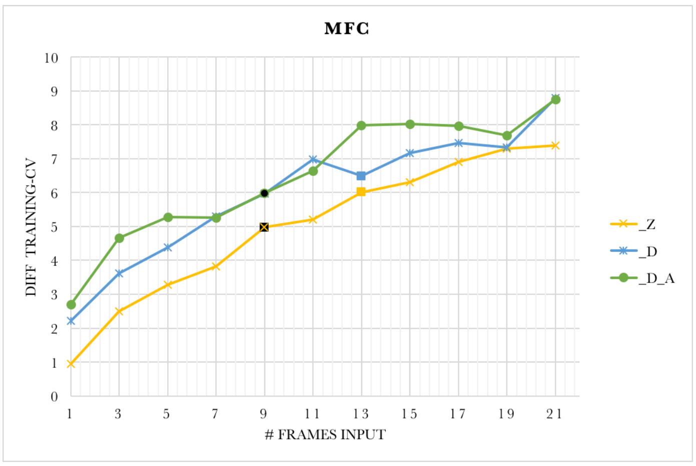
In **black** points where for a given feature type PER reaches a minimum.

- Observations: Overall, we observe that for all feature types (_Z, _D and _D_A for FBK and MFCC) as the size of the acoustic input window increases, the PER in the TIMIT test-set decreases. Conversely, for a fixed input acoustic window, as a general trend adding dynamic information through the feature type improves the performance.

  It is worth noticing from these graphs that in both FBK and MFCC, that the decrease in PER we see in _Z when incrementing the number of frames from 1 to ±2 is the same drop that occurs when adding 1st order delta parameters to _Z. Similarly, the decrease in PER we see in _Z when incrementing the number of frames from 1 to ±3-4 is the same drop that occurs when adding 2nd order delta parameters to _Z. This indicates that incrementing the input acoustic window, allows hidden layer of the feedforward neural network to obtain 1st and 2nd order dynamic information.

  For size of the input acoustic window higher than ±4, we observe very small relative improvement across all feature types. For feature types _D and _D_A, the PER even increases. This is a sign of the neural network overfitting the TIMIT training-set. To examine this closely, the graphs on the top of next page show the difference between training and cross validation (CV) frame classification performance in the last epoch of `step-dnntrain`.





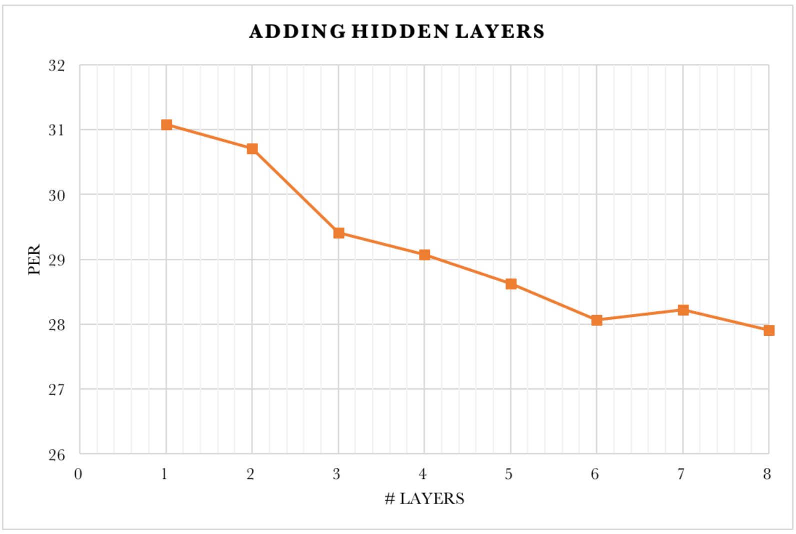
From these we see that as general trend the difference between training and CV increases as the number of frames in context or the feature dimension increases. From 9 frames, onwards (input acoustic window higher than ±4) this difference becomes bigger indicating the overfitting of the Neural Network to the training set.

- Extra experiments: I tried using an asymmetric `CONTEXTSHIFT` (-6 to 2) and it showed an improvement in performance of 1.47% with respect to ±4.

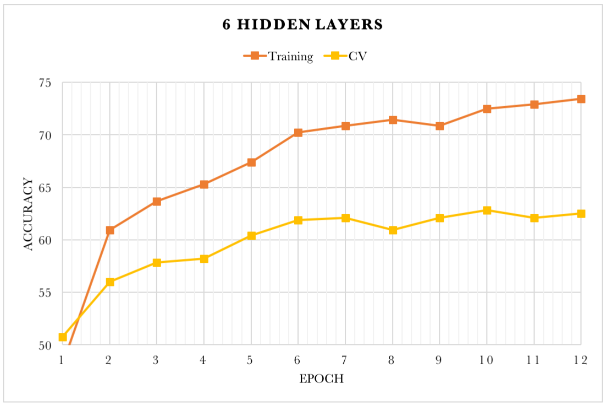
  In all these experiments, the best performing combination of front-end parameterisation and size of acoustic window is MFCC with _D_A and ±6 `CONTEXTSHIFT`. We will use these settings in the following sections.

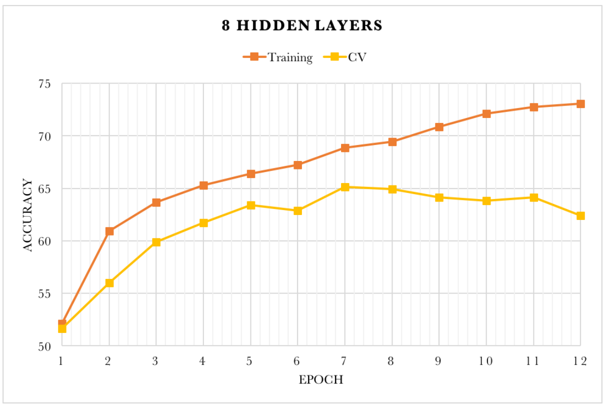
Exercise 6.4: Adding Multiple Layers

- Set-up: Compare performance of context-independent DNN–HMMs with number multiple hidden layers ranging from 1 to 7. After optimizing the insertion penalty to -4, use `step-decode` to decode TIMIT test-set. Front-end parameterisation is MFCC with _D_A, `CONTEXTSHIFT` ±6, number of nodes per layers is kept constant at 500 and the `PTWEIGHTDECAY` and `FTWEIGHTDECAY` keep default value 0.001.
- Experiment results:



- Observations: In general, it can be observed that as we increase the number of hidden layers, the performance of the DNN–HMM improves. With each new the number of estimated parameters/weights grows linearly with it, increasing the flexibility of the system. However, it can be observed that the relative improvement with each new layer becomes smaller from 5 layers onwards, reaching a minimum at 6 layers. To examine this closely, the graphs below show training and cross validation (CV) frame classification accuracies of a 6 and 8 hidden layers DNN–HMM over the epochs during `step-dnntrain`.





Even though the difference between training and CV accuracy increases after each epoch for both DNN–HMMs, the CV accuracy steadily increases in the 6-hidden layer system, whereas in the 8-hidden layer system CV accuracy drops in the last 4 epochs. This indicates the 8-hidden layer system is overfitting the training set as the number of estimated parameters is too high in comparison with size of the data base.

- Extra experiments: To prevent the overfitting of the 8-hidden layer system, I tuned the L2 regularisation parameters, `PTWEIGHTDECAY` and `FTWEIGHTDECAY`, obtaining and improvement in PER of 0.83% with `PTWEIGHTDECAY = 0.0001` and `FTWEIGHTDECAY = 0.001`. In this case, the training/CV against plot resembling the 6-hidden layer system plot above.

  To prevent the system from overfitting we will use 5-hidden layer DNN–HMMs for the next section.

Exercise 6.5: Triphone Target units

The goal is to increase the output layer to be more context dependent. Remember, the output layer is produced by a soft-max activation function, that assigns the probability to each phone model. If we use monophones as phone models, the output layer consists of 48+sil nodes. However, if instead we use triphones as target units, it increases to the number of nodes to the number of physical triphones from the state-clustering trees. If this number is too high, the data-base might not be large enough to allow for parameter estimates to give appropriate generalisations of unseen triphones, hence causing overfitting. To set this up we need to use `step-align` to produce state-level alignments based on a GMM – HMM triphone system.

For this part, we use MFCC as front-end parameterisation with _D_A, `CONTEXTSHIFT` ±6, number of nodes per layers is kept constant at 500 and 5 hidden layers. We want compare performance of context-dependent DNN–HMM based on the best performing context-dependent GMM–HMM with `ROVAL` 200 and `TBVAL` 1000 and 20 Gaussian components with this same context-dependent GMM–HMM. After optimizing the insertion penalty to -8; and tuning the `PTWEIGHTDECAY` and `FTWEIGHTDECAY` to 0.1 and 0.0001 respectively we use `step-decode` to decode the TIMIT test-set and a subset of the training set. We obtain:

200-1000-20	PER TEST	PER SUB	Difference	TRAIN	CV
DNN	24.26	16.81	9.45	59.75	50.05
GMM	29.5	11.97	17.53		
Difference	-5.24	4.84			

This goes to prove that DNN–HMMs can generalise better that than GMM–HMMs if configuration parameters such as hidden layers, `CONTEXTSHIFT`, input feature type and the L2 regularisation parameters, `PTWEIGHTDECAY` and `FTWEIGHTDECAY;` are tuned correctly (as well as the Insertion Penalty).

- Extra experiments: In addition, implementing triphone realignment translates into an improvement in performance of 0.5%.

3 Extensions

Exercise 7.1: Biphone Models

We can extend context independent phone systems to triphone model so they take into account its phonetic context (preceding and following phones), we can also contract context dependent biphones. In particular, left (ae-dh) and right (dh+ae) context.

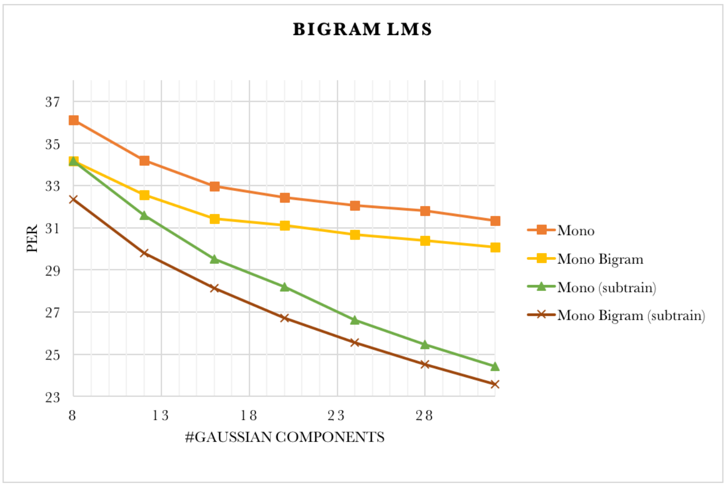
If we trained two (left and right) biphone GMM–HMM with `ROVAL` 200 and `TBVAL` 1000 and 20 Gaussian components and align them to produce a context-dependent DNN-HMMs with biphone target units we get:

200-1000-20	PER TEST	PER SUB	Difference	TRAIN	CV
DNN-rc	24.97	15.10	9.87	64.29	52.65
GMM-rc	29.78	17.43	12.35		
Difference	-4.81	-2.33			
DNN-lc	24.91	16.47	8.44	63.98	52.89
GMM-lc	30.37	17.32	13.05		
Difference	-5.46	-0.85			

In general, we observe that biphone systems do a better job that monophones and slight worse than triphones. Experiment results show no preference between right and left context biphones.

Exercise 7.2: Bigram Language Model

A quick Grammar Scale search reveals that it is optimal value is 2.0. The table below shows the improvement observed when using a bigram language model for GMM–HMMs as the number of Gaussian components increases:



Similarly, we can decode our DNN–HMMs with a modified HTE.phoneloop file. For a DNN-HMM with MFCC with _D_A as input feature, `CONTEXTSHIFT` ±6, number of nodes per layers is kept constant at 500 and 5 hidden layers and the L2 regularisation parameters, the `PTWEIGHTDECAY` and `FTWEIGHTDECAY` equal to 0.001, we obtain: 28.75% PER in test set (and 18.75% in PER in subtrain) which is an improvement with respect to the unigram model of 1.34% in the test set.

This goes to show that Language models phone system to perform better in Large Vocabulary Speech recognition data-bases. However, if we try to increase the language model to 4-gram model, lots of sparsity will populate the data which will cause improvement of the system to be smaller.