The Lincoln Large-Vocabulary Stack-Decoder Based HMM CSR*

Douglas B. Paul

Lincoln Laboratory, MIT
Lexington, Ma. 02173-9108

Abstract

The system described here is a large-vocabulary continuous-speech recognition (CSR) system with results obtained using the Wall Street Journal-based database [15]. The recognizer uses a stack decoder-based search strategy[1, 7, 14] with a left-to-right stochastic language model. This decoder has been shown to function effectively on 20K and 64K-word recognition of continuous speech. It operates left-to-right and can produce final textual output while continuing to accept additional input speech. Thus it need not wait for the end of the sentence and can be structured so that it can accept an unbounded length stream of input speech. The recognizer also features recognition-time adaptation to the user's voice. This system showed improvements of 42% for a 5K vocabulary and 35% for a 20K vocabulary compared to the November 92 evaluation test system.

1. The Basic HMM CSR System

The basic system described here uses two (TM-2) or three (TM-3) observation streams: mel-cepstra, time differential mel-cepstra, and second time-differential mel-cepstra. The system uses Gaussian tied mixture [4, 6] with grand variance pdfs and treats each observation stream as if it were statistically independent of all others. Cross-word sex-dependent triphone models are used to model phonetic coarticulation and a coarse speaker grouping. These triphone models are smoothed with reduced context phone models [20] using Bayesian smoothing weights. Each phone model is a "linear" (no skip transitions) three state HMM. The phone models are trained by the forward-backward algorithm using a bootstrapping procedure which requires only the orthographic transcription. The trainer can also use sentence dependent background models to allow for variation in the training-data recording environment. Both the trainer and the recognizer used the Dragon WSJ dictionary and can use multiple pronunciations for any word. The recognizer extrapolates (estimates) untrained phone models, splits long-duration states to enforce minimum durations, contains an adaptive background model, allows optional intermediate silences between words, performs optional channel compensation, can use any left-to-right stochastic language model (LM), and can adapt to the speaker with or without supervision. The recognizer uses a Viterbi decoder with a ML decision rule. The recognition search is implemented using a stack decoder [1, 7, 14] with a two-pass fast match. The stack decoder includes a proposed CSR-NL interface[10] to access an external LM module.

*This work was sponsored by the Advanced Research Projects Agency. The views expressed are those of the author and do not reflect the official policy or position of the U.S. Government.

2. The Stack Decoder

The stack decoder is organized as described in reference [14]. The basic paradigm used by the stack decoder is:

1. Pop the best theory (partial sentence) from the stack.
2. Apply acoustic and LM fast matches[3, 5] to produce a short list of candidate next words.
3. Apply acoustic and LM detailed matches to the candidate words.
4. Insert surviving new theories into the stack.

This paradigm requires that theories of different lengths be compared. Therefore, the system maintains a least-upper-bound or envelope of all previously computed theory output log-likelihoods (LL_i). (The acoustic log-likelihoods and the envelope are functions of time.)

$$envelope(t) = \max_i LL_i(t)$$

$$StSc_i = \max_t (LL_i(t) - envelope(t))$$

$$t\_exit_i = \operatorname{argmax}_t (LL_i(t) - envelope(t))$$

Theories whose stack score, $StSc$, is less than a threshold are pruned from the stack. The stack entries are sorted by an major sort on most likely exit time, t_exit, and a minor sort on $StSc$. Thus the shortest theories are expanded first which has the net effect of working on a one to two second active region of the input and moving this active region left-to-right through the data.

The "extend each partial theory with one word at a time" approach allows the use of a particularly simple interface to the LM. All requests to the LM are of the form: "Give me the probability of this one-word extension to this theory." This has been exploited in order to place the LM in an external module connected via sockets (pipes) and specified on the on the command line[10]. Since the N-gram LMs currently in use are so trivial to compute, the LM fast match probability is currently just the LM detailed match probability

This stack decoder, since all information is entered into the search as soon as possible, need only pursue a "find the best path and output it" strategy. It is also quite possible to output a list of the several best sentences with minor modifications[13, 14].

Given this search strategy, it is very easy to produce output "on the fly" as the decoder continues to operate on the incoming data. Any time the first N words in all entries on the stack are the same they may be output. (This is the analog of the "confluent node" or "partial traceback" algorithm [21] in a time synchronous decoder.) No future data will alter this partial output.

Similarly, since the active zone moves left-to-right though the data, the stack decoder can easily be adapted to unbounded length input since the various envelopes and the like need only cover the active region. In practice this involves an occasional stop and pass over the internal data shifting it in buffers, altering pointers,

and renormalizing it to prevent underflow, but these are simply problems of implementation, not theory or basic operation.

3. The Fast Match

The acoustic fast match (FM) uses a two pass strategy. Both passes search a left-diphone tree generated from the recognition vocabulary. The first pass takes all theories for which $t\_exit_i = \min_j t\_exit_j$ and combines their log-likelihoods (take their vector maximum for a Viterbi decoder) to create the input log-likelihood for the decoder. This decode produces two outputs: it sets pruning thresholds for the second passes and marks all diphone nodes for words whose FM output log-likelihood exceeds a FM output threshold envelope.

The second pass is applied for every theory which was included in the above combination. It applies the exact log-likelihood from the detailed match as input to the left-diphone tree using the pruning thresholds from the first pass and searching only the marked diphone nodes. The word output log-likelihoods are added to the LM log-probabilities to produce the net word output log-likelihoods. The cumulative maximum of these net log-likelihoods plus a (negative) threshold now produces the FM output threshold envelope. Any word whose output log-likelihood exceeds this threshold envelope is placed on the candidate word list for the detailed match.

Both passes of the fast match use a beam pruned depth-first (DF) search of the diphone tree. (The beam pruning requires a cumulative envelope of the the state-wise log-likelihoods.) The DF search is faster than a time-synchronous (TS) search due to its localized data. At any one time, it only needs an input array (which was used very recently), and output array (which will be used very soon), and the parameters of one phone whereas the TS search must touch every active state before moving to the next time. This allows the DF search to stay in cache (~1 MB on many current workstations) and to page very efficiently. The TS search, in comparison, uses cache very inefficiently and will virtually halt if it begins to page. (A stack search was also tested. Because the operational unit—the diphone—is so small, its overhead canceled any advantages. Its computational locality is also not as good as that of the DF search.)

A goal of recognition system design is to minimize the overall run time without loss of accuracy. In the current system, this minimum occurs (so far) with the relatively expensive fast match described above. It is the largest time consumer in the recognizer. Using generous pruning thresholds that reduce the number of fast match pruning errors to below a few tenths of a percent, this fast match allows only an average of about 20 words of a 20K word vocabulary to be passed to the detailed match.

4. The Detailed Match

The detailed match (DM) is currently implemented as a beam-pruned depth-first searched triphone tree. The tree is precompiled for the whole vocabulary minus the silence phones, but only triphone nodes corresponding to the FM candidate words are searched. The LM log-probabilities are integrated into the triphone tree to apply the information as soon as possible into the search. The beam pruning again requires a state-wise log-likelihood cumulative envelope. Because the right context is not available for cross-word triphones, the final phone is dropped from each word and prepended to the next word.

The silence phones, because they may have very long duration are "continuable"—that is they run for a limited duration and then are placed on the stack for later continuation. They are computed using very small time synchronous decoders so that their state can be placed on the stack to allow the continuation. This allows a finite fixed-size likelihood buffer in each stack entry and reduces decoding delays.

"Covered" theories are pruned from the search[13]. One theory covers another if all entries in its output log-likelihood array are greater than those of the second theory at the corresponding times and its LM probabilities will be the same for all possible extensions.

A covered theory can never have a higher likelihood than its covering theory and is therefore pruned from the search. (This is analogous to a path join in a TS decoder.) For any limited left-context-span LM, such as an N-gram LM, this mechanism prevents the exponential theory growth that can otherwise occur in a tree search.

5. Component Algorithms

This recognition system includes a variety of algorithms which are used as components supporting the major parts described above.

5.1 DF Search Path Termination

It is not always possible to determine when to terminate a search path in a non-TS search because the first path to reach a point in time will not be able to compare its likelihood to the likelihood of any other path. Thus a heavily pruned TS left-diphone tree no-grammar decoder is used to produce a rough estimate of the state-wise envelope for all theories up to the current time. This envelope is used primarily to alter the beam-pruning thresholds of the FM and DM such that the search paths terminate at appropriate times. This decoder requires only a very small amount of computation.

5.2 Bayesian Smoothing

In a number of situations it is necessary to smooth a sparse-data estimate of a parameter with a more robust but less appropriate estimate of the parameter. For instance, the mixture weights for sparse-data triphone pdfs might be smoothed with corresponding mixture weights from the corresponding diphones and monophones[20] or, in an adaptive system, the new estimate based upon a small amount of new data might be smoothed with the old estimate based upon the past data and/or training data. The following smoothing weight estimation algorithm applies to parameters which are estimated as a [weighted] average of the training data.

A standard Bayesian method for combining new and old estimates of the same parameter is

$$x = \frac{N_n}{N_n + N_o} x_n + \frac{N_o}{N_n + N_o} x_o$$

where x is the parameter in question and N is the number of counts that went into each estimate and the subscripts n and o denote new and old. Similarly if one assumes the variance v of each estimate is inversely proportional to N (i.e. $v \propto \frac{1}{N}$),

$$x = \frac{v_o}{v_n + v_o} x_n + \frac{v_n}{v_n + v_o} x_o.$$

The above assumes x_n and x_o to be estimates of the same parameter. However, in the case of smoothing, the purpose is to use data from a different but related old parameter to improve the estimate of the new parameter. For the above examples, x_n might be an estimate from a triphone and x_o from a diphone or monophone, or x_n might be an estimate from the current speaker (i.e. be speaker dependent) and x_o from speaker-independent training data. Thus

$$E[x] = E[x_n] \neq E[x_o].$$

If one assumes that the expected values of x and x_o differ by a zero mean Gaussian representing the unknown bias,

$$E[x] - E[x_o] = G(0, v_d)$$

then a corrected estimate for the old variance is

$$v'_o = v_o + v_d.$$

If we now substitute the new value for v_o and return to the initial form of the estimator,

$$x = \frac{N_n}{N_n + N'_o} x_n + \frac{N'_o}{N_n + N'_o} x_o$$

where

$$N'_o = \frac{N_o N_d}{N_o + N_d}.$$

Note that $N'_o \leq N_d$ and thus the smoothing equation discounts the value of the old data according to N_d which, for the above examples of smoothing, may be determined empirically. This equation can be trivially extended to include multiple old estimates for smoothing a triphone with the left diphone, right diphone, and monophone. In this recognition system, symmetries and linear interpolation across states have been used to reduce the number of N_d's for triphone smoothing from twelve to three. This smoothing scheme has also been tested for speaker adaptation (results below) and might also be used in language modeling.

5.3 Channel Compensation

Blind channel compensation during both training and recognition is performed by first scanning the sentence and averaging the mel-cepstra of the speech frames. This average is then subtracted from all frames (commonly known as mel-cepstral DC removal.) This does not affect either of the differential observation streams.

5.4 Speaker Adaptation

One cannot always anticipate the identity of the user when training a recognizer. The "standard" SI approach is to pool a number of speakers to attempt to provide a set of models which gives adequate performance on a new user. This approach, however, creates models which are rather broad because they attempt to cover any way in which any speaker will pronounce each word rather than the way in which the current speaker will pronounce the words. (This is consistent with the fact that SD models outperform SI models given the same amount of training data.) Speakers may also not be willing to prerecord training or rapid enrollment data and wait for the system to process this data.

One solution for this problem is recognition-time adaptation, in which the recognizer adapts to the current user during normal operation. This solution also has the advantage that the recognizer can track changes in the user's voice and changes in the acoustic environment. The paradigm used here is to initialize the system with some set of models such as SI models or SD models from another speaker, recognize each utterance with the current set of models, and finally use the utterance to adapt the current models to create a new set of models to be used to recognize the next utterance[9, 16]. If the user supplies any information to correct or verify the recognized output, the adaptation can be supervised, otherwise the adaptation will be unsupervised.

The adaptation algorithm used here is a simple smoothed maximum-likelihood scheme:

1. Start with some set of acoustic models, M which have had their DC removed as in channel compensation.
2. Perform channel compensation (mel-cepstral DC removal).
3. Recognize the utterance using the current model, M.
4. Compute the state sequence and alignment using either the corrected text (supervised) or the recognized text (unsupervised).
5. Compute new estimates of the model parameters M_{new} using 1 iteration of Viterbi training.
6. Update the model by smoothing the new estimates of the parameters with old parameters:

$$M = (1 - \lambda)M + \lambda M_{new}$$

7. Go to 2.

The adaptation rate parameter, λ, trades off the adaptation speed and the limiting recognition performance. λ need not be constant—but was held constant in these experiments. Only the Gaussian means of a TM system with a tied variance were adapted in these experiments. (Adapting other parameters will be explored at a later date.)

A number of experiments using simplified phonetic models were performed to evaluate SI starts, cross-sex SD starts, and same-sex SD starts[16] using the RM database[17]. The adaptation helped in all cases, even for unsupervised adaptation of the cross-sex starts which started with a word error rate of 94%. A system which trained SI mixture weights with SD Gaussians and then fixed the weights while training a set of SI Gaussians was also tested in the hope that, once adapted, it would look more like an SD system than a system started with normal SI models. Its unadapted performance was somewhat worse than the normal SI system, but after adaptation, its performance was better than the normal SI system.

The results for our best SI-109 trained system (TM-3, cross-word triphone models) were:

	RM word error rate (sent 76-100)		
System	static	sup adapt	unsup adapt
Best SI-109	5.7%	2.9%	3.1%
SD (control)	1.9%	-	-

std dev=.3–.5%

As can be seen from these results, the adaptation almost halved the error rates for both supervised and unsupervised adaptation. In no case did any system diverge. A Bayesian adaptation scheme based upon the above algorithm was also tested, but was no better than the simple ML algorithm. Unfortunately, the improvement was much less when tested upon the WSJ database (see below).

5.5 Pdf cache

Tied mixture pdfs are relatively expensive to compute and a pdf cache is necessary to minimize the computational load. Each cache location is a function of the state s and the time t. The cache must also be able to grow efficiently upon demand and discard outdated entries efficiently. Algorithms such as hash tables do not grow efficiently and have terrible memory cache and paging behavior. Instead, the pdf cache is stored as a dynamically allocated three dimensional array:

$$pdf[t/T][s][t\%T]$$

where % is the modulo operator. Only the first level pointer array ($[t/T]$) is static, both the $[s]$ pointer arrays and the actual storage locations $[t\%T]$ are allocated dynamically. Outdating is simple: remove all pointer arrays and storage locations for $t/T < t'/T$ (integer arithmetic), allocation occurs whenever a null pointer is traversed, and access is just two pointers to a one dimensional array. It is also a very good match to a depth-first search since such a search accesses the states of a phone sequentially in time for a number of time steps which gives very good memory cache and paging performance. This caching algorithm is used in both the trainer and the recognizer.

5.6 Initialization of the Gaussians

Previously, the Gaussians were initialized as a single mixture by a binary splitting EM procedure. However, it was discovered that these sets of Gaussians tended to be degenerate (i.e. a number

of the Gaussians were identical to at least one other Gaussian). Changing the initialization procedure to a top-1 EM (in effect a K-means that also trains the mixture weights) removed the degeneracy. This did not alter the recognition accuracy, but significantly reduced the mixture summation computation since these sums are observation pruned (only compute the summation terms for the Gaussians within a threshold of the best Gaussian).

5.7 Trainer Quantization Error Reduction

The SI-284 training condition of WSJ1 uses 30M frames of training data and, in the Baum-Welch training procedure, significant fractions of these frames are summed into single numbers. The number of mixture weights (167M, see below) for the largest set of models was so large that only two byte integer log-probs could be allocated to each accumulator. (Quantization in the estimation of the mixture weights flattens the distribution.) Multi-layer sums were used to reduce the quantization error without unduly increasing the the dataspace requirements.

Since there were only a relatively few Gaussians in this system (771), quantization in estimating them was reduced by the use of double-precision accumulators and a change of variable to additionally reduce the error in estimating the variance:

$$var = \frac{1}{N}\left(\sum_i x_i^2 - N\bar{x}^2\right)$$

If one substitutes x_i' for x_i where $x_i' = x_i - \hat{x}$ where $\hat{x}$ is an estimate of $\bar{x}$ will reduce the second term and thus the quantization error. $\bar{x}$ from the previous iteration can be used as $\hat{x}$ in the current iteration.

5.8 Data-Driven Allophonic Tree Clustering

Previous techniques for allophonic tree clustering have generally used a single phonetic rule (simple question) to make the best binary split (according to an information theoretic metric) in the data at each node and some of these techniques alternate between splitting and combining several nodes to minimize the data reduction by forming "compound questions" at the nodes[2]. Another approach is to ask the "most-compound question" from the start.

In this approach, if one is searching for the best split based upon the, for instance, right phonetic context and there are N right phonetic contexts in the triphones assigned to the current node, then there are $2^{(N-1)}$ possible splits. (N can easily be greater than one hundred in some of the nodes near the root of the tree.) Such a search problem can be solved by simulated annealing, genetic search, or multiple quenches from random starts. All three were tried and multiple quenches from random starts appeared to give the highest probability of obtaining the optimum split for a given amount of CPU time. Finally, the pdf weights at each node are smoothed with those of its parent using the Bayesian smoothing described above. This smoothing is carried out from the root down toward each leaf so that, in effect, each node is smoothed by all of the data. The software for this technique has been developed and debugged on the RM database, but we have not yet had sufficient time to test this algorithm on a large vocabulary task. (This algorithm is not currently in use.)

5.9 Parallel Path Search of a Network

In a simple single pass fast match, the fast match network (phonetic tree in this system) must be searched once per theory. This is very expensive because the same network must be searched over the same input data many times. One method for reducing this computation is searching the network once with a technique which computes many inputs in parallel. This search technique represents the data as two data structures: a "max structure" which contains maximum (for a Viterbi search) with a pointer to a "delta structure" which contains a link count and a list of individual deltas such that the sum of the maximum and the deltas gives the individual log-probabilities. A pass over the input data will create one max structure per input time step and fewer delta structures since delta structures can be shared by any number of max structures. Many operations (60–80% in these experiments) of the network decode will share the same delta structure and thus the log-probabilities corresponding to all of the inputs can be computed with just operations on the max structures. When paths represented by max structures with different delta structures join, then operations of linking, unlinking, and/or creation must be performed on the delta structures. The link count is used to garbage collect unlinked delta structures. This algorithm was used for a while in the fast match, but has been replaced by the two pass algorithm described above which is faster and uses less space.

5.10 Gaussian Variance Bound Addition

A well-known problem in ML estimation of Gaussian-dependent variances in Gaussian mixtures is variance values that go to zero. Two common methods for preventing this singularity are lower bounding or using a grand variance. Simple addition of a constant to each variance has been found to be a superior alternative to lower bounding: it is equally trivial to apply and has yielded superior recognition performance on several recognition tasks. For instance, for several tasks using single observation stream Gaussian-dependent variances:

	Error Rate (std dev)	
System	Var lim	Var add
SI-84 CSR	16.7% (.5%)	15.6% (.4%)
Spkr ID[19]	29.0% (1.4%)	26.0% (1.4%)

In both tasks, the performance was improved by over two standard deviations by the use of variance addition. While not needed to insure non-singularity, variance addition was also found to improve recognition in a grand variance system:

	Error Rate (std dev)		
System	none	Var lim	Var add
SI-84 CSR	25.2% (.5%)	20.5% (.5%)	17.5% (.5%)

In spite of the robustness of the estimate of the grand variance, the performance is improved significantly by variance limiting and even more by the variance addition.

Clearly the variance addition is doing something more than just preventing singular variances. One possible viewpoint is that variance addition is a soft limiting function. A simple bound throws away all information about the original variance while the addition retains some of the original information. Another possible view is that the variance addition is providing signal-to-noise (S/N) ratio compensation. Each component of the observation vector contains both useful signal and noise. Variance addition might act like a Wiener filter in adjusting the gain on each component appropriately:

$$-\frac{1}{2}\sum_i \frac{V_i}{V_i + lim}\frac{(x_i-\mu_i)^2}{V_i} = -\frac{1}{2}\sum_i \frac{(x_i-\mu_i)^2}{V_i + lim}$$

where the second term on the left is the normal summation term in the exponent of a diagonal covariance Gaussian and the first term on the left is analogous to a Wiener filter if lim represents the noise power. (In the above systems one would expect the measurement and quantization noise power to be the same in all observation components.) This technique was discovered too late to be included in any of the following recognition results.

6. Recognition Results

The above system has been tested using the WSJ1 database. The primary training condition used here is the "SI-284" condition: 37K sentences from 284 speakers—a total of about 82 hours. The primary test condition in these tests is 5K word non-verbalized punctuation (NVP) closed vocabulary with a perplexity 62 trigram back-off LM[8, 11] using the WSJ0 SI development test data.

The initial tests probed the LM weight using a non-cross-word, non-sex-dependent TM-2 system:

Pdf	x-wd	sx	LM wt	Wd err
TM-2	-	-	4	11.31%
TM-2	-	-	5	10.47%
TM-2	-	-	6	10.44%

p=62 5K NVP word trigram LM, std dev∼.37%

Based upon this result, the LM weight was chosen to be five. (The LM weight is applied to the LM log-probabilities before combining with the acoustic log-likelihoods.)

Next, three different factors in the acoustic modeling—two (TM-2) vs. three (TM-3) observation streams, cross-word triphones (x-wd), and speaker-sex-dependent triphones (sx) were examined to see what their effect would be on this task:

	Pdf	x-wd	sx	Wd err	chg from 1	Δ from 1
1.	TM-2	-	-	10.47%	-	-
2.	TM-3	-	-	9.80%	pdf	.67%
3.	TM-2	-	sx	9.57%	sx	.90%
4.	TM-2	x-wd	-	8.32%	x-wd	2.15%
					chg to 8	Δ to 8
5.	TM-3	-	sx	9.36%	x-wd	1.49%
6.	TM-3	x-wd	-	8.65%	sx	.78%
7.	TM-2	x-wd	sx	7.91%	pdf	.04%
8.*	TM-3	x-wd	sx	7.87%	-	-

p=62 5K NVP word trigram LM, LM wt=5,
std dev ∼.35%, *=Nov 93 eval test system

In the first set of comparisons, (1 vs. 2–4), only one feature (column chg from 1) is added from the simplest system (1) and the change in the error rate is shown in the last column. Similarly, in the last set of comparisons (5–7 vs. 8), only one feature is added to create the most complex system (8). At both ends of the spectrum, cross-word modeling gives the most improvement, sex-dependent triphones an intermediate amount, and the third observation stream the least improvement. Overall, the best system (8) yields a 26% improvement over the simplest system (1) and 42% improvement over the November 92 evaluation test system (SI-84 trained, cross-word semiphone acoustic models: 13.5% word error rate[16]). This best system was chosen for the November 1993 evaluation tests.

The two extreme systems from the above table have also been tested on the 20K word WJS0 recognition task:

	Pdf	x-wd	sx	LM wt	Wd err
1.	TM-2	-	-	5	17.44%
8.*	TM-3	x-wd	sx	5	14.23%

p=160 20K NVP word trigram LM, std dev∼.45%
*=Nov 93 eval test system

This (system 8) is an improvement of 35% over the corresponding November 92 system (21.8% word error rate[16]).

To explore the relative performance gains due to the additional training data in SI-284 over SI-84 and the algorithmic improvements, system 8 was also trained on SI-84 and tested:

System	Training	Wd err (std dev)	Reduction
Nov 92	SI-84	13.5% (.4%)	-
8.	SI-84	9.9% (.4%)	27%
8.*	SI-284	7.9% (.3%)	42%

p=62 5K NVP word trigram LM, *=Nov 93 eval test system

Similarly for 20K word recognition:

System	Training	Wd err (std dev)	Reduction
Nov 92	SI-84	21.8% (.5%)	-
8.	SI-84	16.9% (.5%)	22%
8.*	SI-284	14.2% (.4%)	35%

p=160 20K NVP word trigram LM, *=Nov 93 eval test system

In both cases, about two-thirds of the improvement is due to the algorithmic improvements and about one-third is due to the increased training data (about 16 to 82 hours).

System 7 (TM-2, x-wd, sx) was selected for testing the adaptation algorithm using the WSJ1 S4 adaptation development test set. (System 8 was used for the adaptation evaluation test.) Each speaker uttered about one hundred sentences which are scored in groups of twenty five to show the adaptation:

	Sentences			
System	1–25	26–50	51–75	76–100±
Static	7.3%	9.0%	9.6%	8.6%
Sup adapt	8.0%	8.9%	8.0%	8.1%
Unsup adapt	7.3%	8.6%	8.1%	7.5%
Sup adapt/static	110%	99%	83%	94%
Unsup adapt/static	100%	96%	84%	87%

p=62 5K NVP word trigram LM, LM wt=5, std dev ∼.7%

The adaptation improves the error rates, although far less than the halving of the error rate observed for the RM task.

7. Discussion And Conclusions

Some of the performance improvement over the November 92 system has come at a significant size penalty:

System	Phones	States	Mix Wts	Size
Nov 92	17K semi	26K	13M	26 MB
Nov 93 (8)	73K tri	220K	170M	340 MB

(The weights are stored as two byte log-integers.) Since the trainer requires two copies (mixture weights plus reestimation accumulators) this totals 680MB. The total size of the trainer is about 830MB and the recognizer about 500MB. These are significantly larger than the RAM on any of our machines, but both the trainer and the recognizer have been optimized under that assumption resulting in only moderate speed loss due to paging. However, these systems are still larger than is desirable.

Quantization error in the trainer is very subtle. The Baum-Welch training algorithm is sufficiently stable that it will only be found if one specifically looks for it. The primary effect in the systems described above is a "flattening" of the pdfs through an effective upper-bounding of the mixture weight accumulation sums.

The stack-decoder has been shown to be an effective decoder for large vocabulary CSR both here and elsewhere[1]. Because it efficiently combines all information into a single unified search and it makes a zonal left-to-right pass though the input data, it can produce the recognized output continuously as more data is input as well as handle unbounded length input. Most of the computation in the above CSR is consumed by the acoustic fast match.

(The stack itself is a very efficient mechanism for limiting the number of theories which must be expanded.) Thus the largest future speed-ups will probably result from faster fast matches. Significant speed-ups have already resulted from a mixture of strategies, such as pdf caching and covered theory elimination, and implementations which use the machine architectures efficiently without compromising portability.

The Bayesian smoothing fills in a long-standing gap in the smoothed triphone scheme[20] The smoothing weights must be computed by deleted interpolation[1] which requires at least several instances of the triphone being smoothed or estimated by some non-data-driven method. The non-data-driven methods have generally been ad-hoc[12, 20]. This gives theoretical support for a functional form based upon the amounts of data available to train each object and the objects similarity. This smoothing approach has also been tested in acoustic adaptation and could likely be used in language modeling.

The data-driven allophonic tree clustering is unique in that it uses only acoustic similarity and not phonetic features in its clustering process. This allows more complex decision rules than do phonetic features and might yield better clusters than phonetic feature-based clusterings. (One would expect it to "derive" many phonetic rules in its operation.) As yet, it has not been adequately tested.

By exploiting a time-space trade-off, the parallel search technique is able to speed up computation of multiple inputs to a probabilistic network. While this technique is not currently in use in this CSR system, it might be useful elsewhere.

Finally, variance addition should be useful as a simple technique to reduce the error rate in many Gaussian mixture (or multiple Gaussian) based systems. Many standard techniques for dealing with varying S/N in the observation components perform a linear transform on the observation vector and then drop some of the resulting components. This all-or-nothing dropping of components throws away some signal with the noise. In contrast, variance addition attempts to weight each term according to its value. This technique appears to be related to the technique of "diagonal loading" (adding a constant to the diagonal of a matrix) that is sometimes used to increase the stability and/or noise immunity of a covariance matrix prior to inversion[18].

From a mechanical point of view, variance addition appears to be inhibiting noise induced splitting of the Gaussians. Unsupervised clustering methods such as EM attempt to find a set of Gaussians which best describes the distribution of the training data whether the distribution is due to signal or noise. If an infinite number of Gaussians and an infinite amount of training data were available, there would be no problem since the mixture weights would compensate for any noise induced splitting. However, in real systems both are finite and the noise induced splitting consumes Gaussians to better model the noise at the expense of modeling the signal. Thus, by reducing this splitting, the available Gaussians are better able to model the signal while using larger variances to model the noise.

The above-described CSR system is well suited to handle the large vocabulary CSR problem. Many problems still need work—speed, size, accuracy, and robustness, to name a few—but we will continue to chip away at them.

References

[1] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," IEEE Trans. Pattern Analysis and Machine Intelligence, PAMI-5, March 1983.

[2] L. Bahl, P. V. de Souza, P. S. Gopalakrishnam, D. Nahamoo, M. A. Picheny, "Decision Trees for Phonological Rules in Continuous Speech," ICASSP 91, Toronto, May 1991.

[3] L. Bahl, S. V. De Gennaro, P. S. Gopalakrishnam, R. L. Mercer, "A Fast Approximate Acoustic Match for Large Vocabulary Speech Recognition," IEEE Trans. Speech and Audio Processing, Jan. 1993.

[4] J.R. Bellegarda and D.H. Nahamoo, "Tied Mixture Continuous Parameter Models for Large Vocabulary Isolated Speech Recognition," Proc. ICASSP 89, Glasgow, May 1989.

[5] L. S. Gillick and R. Roth, "A Rapid Match Algorithm for Continuous Speech Recognition," Proceedings June 1990 Speech and Natural Language Workshop, Morgan Kaufmann Publishers, June, 1990.

[6] X. D. Huang and M.A. Jack, "Semi-continuous Hidden Markov Models for Speech Recognition," Computer Speech and Language, Vol. 3, 1989.

[7] F. Jelinek, "A Fast Sequential Decoding Algorithm Using a Stack," IBM J. Res. Develop., vol. 13, November 1969.

[8] S. M. Katz, "Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer," ASSP-35, pp 400-401, March 1987.

[9] E.A. Martin, R. P. Lippmann, D. B. Paul, "Dynamic Adaptation of Hidden Markov Models for Robust Isolated-Word Speech Recognition," Proc. ICASSP 88, New York, NY, April 1988.

[10] D. B. Paul, "A CSR-NL Interface Architecture," Proc. ICSLP 92, Banff, Alberta, Canada, Sept. 1992.

[11] D. B. Paul, "Experience with a Stack Decoder-Based HMM CSR and Back-Off N-Gram Language Models," Proc. DARPA Speech and Natural Language Workshop, Morgan Kaufmann Publishers, Feb. 1991.

[12] D. B. Paul, "The Lincoln Tied-Mixture HMM Continuous Speech Recognizer," ICASSP 91, Toronto, May 1991.

[13] D. B. Paul, "Algorithms for an Optimal A* Search and Linearizing the Search in the Stack Decoder," ICASSP 91, Toronto, May 1991.

[14] D. B. Paul, "An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model," Proc. ICASSP 92, San Francisco, California, March 1992.

[15] D. B. Paul and J. M. Baker, "The Design for the Wall Street Journal-based CSR Corpus," Proc. ICSLP 92, Banff, Alberta, Canada, Sept. 1992.

[16] D. B. Paul and B. F. Necioglu, "The Lincoln Large-Vocabulary Stack-Decoder HMM CSR," ICASSP 93, Minneapolis, April 1993.

[17] P. Price, W. Fisher, J. Bernstein, and D. Pallett, "The DARPA 1000-Word Resource Management Database for Continuous Speech Recognition," ICASSP 88, New York, April 1988.

[18] C. M. Rader, personal communication.

[19] D. A. Reynolds, personal communication.

[20] R Schwartz, Y. Chow, O. Kimball, S. Roucos, M. Krasner, and J. Makhoul, "Context-Dependent Modeling for Acoustic-Phonetic Recognition of Continuous Speech," Proc. ICASSP 85, Tampa, FL, April 1985.

[21] J. C. Spohrer, P. F. Brown, P. H. Hochschild, and J. K. Baker, "Partial Backtrace in Continuous Speech Recognition," Proc. Int. Conf. on Systems, Man, and Cybernetics, 1980.