CODEBOOK DESIGN FOR ASR SYSTEMS USING CUSTOM ARITHMETIC UNITS

Xiao Li, Jonathan Malkin, Jeff Bilmes

SSLI Lab, Department of Electrical Engineering
University of Washington, Seattle

ABSTRACT

Custom arithmetic is a novel and successful technique to reduce the computation and resource utilization of ASR systems running on mobile devices. It represents all floating-point numbers by integer indices and substitutes a sequence of table lookups for all arithmetic operations. The first and crucial step in custom arithmetic design is to quantize system variables, preferably to low precision. This paper explores several techniques to quantize variables with high entropy, including a reordering of Gaussian computation and a normalization of Viterbi search. Furthermore, a discriminatively inspired distortion measure is investigated for scalar quantization to better maintain recognition accuracy. Experiments on an isolated word recognition show that each system variable can be scalar quantized to less than 8 bits using a standard quantization method, except for the alpha probability in Viterbi search which requires 10 bits. However, using our normalization and discriminative distortion measure, the forward probability can be quantized to 9 bits, thereby halving the corresponding lookup table size. This greatly reduces the memory bandwidth and enables the implementation of custom arithmetic on ASR systems.

1. INTRODUCTION TO CUSTOM ARITHMETIC

Automatic speech recognition (ASR) has unquestionable utility when used in environments without a keyboard. Ideally, one could implement a fully-functioning ASR on as portable a device as a watch, necklace, or pendant. On such devices, however, it is critical to reduce computation and resource utilization. In [1] we introduced and proposed a number of software techniques to this end. One of the most popular one used in practice is the discrete HMM or the discrete mixture HMM [2, 3], where the observation vectors or sub-vectors thereof are quantized and their state likelihoods are obtained efficiently via lookup tables (LUT). The use of LUTs also provides a compact representation of model parameters, which not only saves memory but also reduces run time [4, 5].

Motivated by the low entropy of the internal variables within an ASR system and the efficiency of hardware-based table lookups, we in [6] propose a novel framework of custom arithmetic for high-speed, low-resource ASR systems. Therein, each variable (speech features, model parameters, and all intermediate variables computed within the system) is quantized and a codebook is generated. We then pre-compute each arithmetic operation on its inputs' codewords and quantize the computational result using its output's codebook. In this way, the entire computation is performed using simple and fast ROM lookups. Distinct from [6] which presents a general design methodology for the above framework, this paper focuses on codebook design for internal ASR system variables using custom arithmetic units.

This work was funded by NSF grant ITR/RC-0086032

Codebook design is the first and the key step enabling custom arithmetic. Since the size of a LUT depends on the bit-widths of its inputs and output, the variables are preferably quantized to as low precision as possible. This poses potential challenges, as some variables with relatively high entropy might require huge codebooks, leading to prohibitive table sizes. In the Mahalanobis distance calculation of Gaussian evaluation, for example, the distance is accumulated along the dimension of the features, resulting in a relatively spread-out distribution covering all partial accumulations. In addition, the forward probability in Viterbi search possesses a more fatal problem – the forward pass computes over an arbitrarily long utterance in real applications, making α's distribution unknown to the quantizer at the codebook design stage. A key contribution of this work is to use reordering and rescaling techniques to reduce the entropy of such variables without adversely affecting speech recognition word error rate (WER). Thirdly, with scalar quantization, the distortion measure should ideally be consistent with WER minimization. We propose a discriminatively inspired distortion measure to achieve better compression. As will be seen, these techniques greatly reduce the memory bandwidth and the computational load of our ASR engine.

The rest of the paper is organized as follows: Section 2 discusses two structures for Mahalanobis distance computation. Section 3 presents our Viterbi normalization procedure. Section 4 formulates the discriminative distortion measure. Finally, Section 5 presents experimental results with Section 6 concluding.

2. REORDERING OF LIKELIHOOD COMPUTATION

Log-arithmetic is widely used in practical ASR systems to achieve numerical values with a very wide dynamic range [1]. To this end, the log state-conditioned likelihood $\bar{b}_j(t)$ of the t^{th} observation vector $(x_1(t), x_2(t), ..., x_D(t))$ can be expressed as

$$\bar{b}_j(t) = \bigoplus_{i \in M_j} \left[\bar{w}_i + c_i - \frac{1}{2} \sum_{k=1}^{D} \frac{(x_k(t) - \mu_{ik})^2}{\sigma_{ik}^2} \right]. \quad (1)$$

M_j is the subset of Gaussian components of state j. The variables μ_{ik} and σ_{ik} are the scalar mean and variance a Gaussian. $\bar{w}_i$ is the log responsibility of the i^{th} component, and c_i is a constant.

Two iterative operations are required by Equation (1): one is $e_i(t) \triangleq \sum_{k=1}^{D} d_{ik}(t)$, where $d_{ik}(t) \triangleq (x_k(t) - \mu_{ik})^2/\sigma_{ik}^2$, and the other is $\bar{b}_j(t)$ associated with the log addition. There are two natural strategies for performing quantized accumulation: a linear accumulation and a binary tree.

In linear accumulation, the next value equals the current value combined with an additional input as depicted on the left in Figure 1. In the case of $e_i(t)$, the variable $e_i(t)$ is initialized as zero

[1] In this paper, variables with bars are log numbers. E.g., $\bar{x} = \log x$. Also, $\oplus$ denotes log addition where $\bar{x} \oplus \bar{y} = \log(e^{\bar{x}} + e^{\bar{y}})$.

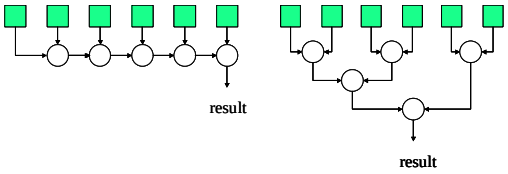

Fig. 1. Linear vs. tree-structure accumulation. Squares refer to operands; circles refer to arithmetic operators to be implemented by table lookup.

and $e_i(t) = e_i(t) + d_{ik}(t)$ is iteratively performed for $k = 1..D$. An alternative to linear accumulation is to use a binary tree, as depicted on the right in Figure 1.

There are different ways of implementing these two schemes with LUTs. First, a separate table could be used for each circle, with the advantage that each table would be customized to its particular distribution of input and output values. This would lead to small tables, but $D-1$ tables are needed in both linear and tree cases. At the other extreme, a single table could account for all circles, but its size might be very large since its input and output distributions would have much higher entropy. Between these two extremes, a separate table could be used for each tree level, leading to $\lceil \log D \rceil$ tables. The potential advantage here is that each tree level can expect to see operand distributions with smaller entropy, and the amount of overall table size might be minimized.

This paper compares three strategies in computing $e_i(t)$. First, we implement the linear case using only one overall shared LUT. Second, we try two different tree accumulation patterns, both with $\lceil \log D \rceil$ LUTs as mentioned above. The two tree strategies differ only at the top level, where the first case adds adjacent elements of the vector $\{d_{ik}(t)\}_{k=1}^{D}$, and the second case instead combines the $d_{ik}(t)$ of each feature with its corresponding delta. The second idea is based on the empirical observation that the dynamic range of $d_{ik}(t)$ is similar between the static features after mean subtraction and variance normalization, and similar also between the deltas. This will cause the outputs of the top level to be more homogeneous, leading to better quantization and representation. When dealing with an incomplete tree (D is not a power of 2), some values are allowed to pass over levels and are added to lower levels of the tree, as shown in Figure 1.

3. NORMALIZATION OF VITERBI SEARCH

In Viterbi search, the forward probability $\alpha_j(t) = P(O_{1:t}, q_t{=}j)$ is calculated for each frame $t = 1..T$ as follows,

$$\bar{\alpha}_j(t) = [\max_i(\bar{\alpha}_i(t-1) + \bar{a}_{ij})] + \bar{b}_j(t). \tag{2}$$

Here we let state 1 and N denote the beginning and ending non-emitting states respectively. The final Viterbi score is evaluated as $\log P(O_{1:T}) = \max_j[\bar{\alpha}_j(T) + \bar{a}_{jN}]$.

As can be seen in Equation (2), the $\bar{\alpha}$ value does not have a bounded dynamic range. Specifically, as T increases the Viterbi score will decrease, something that causes severe problems in codebook design. First, since the utterance length in real applications is unknown at the stage of system design, the $\bar{\alpha}$ values at decode time might not lie in the dynamic range of those values used for quantization at codebook design time. Essentially, it is known that the distribution over $\bar{\alpha}$ has high entropy since it decreases unboundedly with T. While we could assume some upper bound on T and quantize with the $\bar{\alpha}$ distributed accordingly, this would yield an exponentially larger and wasteful codebook where many values are rarely used by short utterances. Therefore, it is highly desirable to have a normalized version of the forward probability, where inference is still valid but the dynamic range is restricted regardless of the utterance length.

Often $\alpha'_j(t) \triangleq P(q_t = j|O_{1:t})$ serves as a normalized forward probability to solve the underflow problem that occurs in fixed-precision floating-point representation [7, 8] with the recursion,

$$\alpha'_j(t) = \frac{1}{s(t)} \sum_i [\alpha'_i(t-1)a_{ij}]b_j(t) \tag{3}$$

producing α' with a representable numerical range, where

$$s(t) \triangleq \frac{P(O_{1:t})}{P(O_{1:t-1})} = \sum_j [\sum_i [\alpha'_i(t-1)a_{ij}]b_j(t)], \tag{4}$$

However, $\alpha'_j(T)$ alone can not serve as the final likelihood score, since the scaling factor $s(t)$ at each frame is different for different words. Obtaining a valid score, according to Equation (4), requires the log values of $s(t)$, $t = 1..T$, to be stored during the forward pass and be summed up at the end, which again brings up the issue of an ever growing dynamic range. One widely used approach to circumvent this problem is to sum up $s(t)$ values of different words and to use this sum as the new scaling factor for that frame. Since at a frame the recursions of different words are normalized by the same scaling factor, the final likelihood score is naturally obtained from the last forward probability. There are potential difficulties, however, with implementing this recursion using custom arithmetic – computing the sum of $s(t)$ values involves significant additional operations with a complexity as much as Viterbi decoding itself.

Therefore, this paper seeks an alternative normalization method. We show that the dynamic range of $\bar{\alpha}_j(t)$ is bounded by linear functions of time. Equation (2) implies that

$$\begin{aligned} \max_j \bar{\alpha}_j(t) - \max_i \bar{\alpha}_i(t-1) &\le \max_{ij} \bar{a}_{ij} + \max_j \bar{b}_j(t) \\ \min_j \bar{\alpha}_j(t) - \min_i \bar{\alpha}_i(t-1) &\ge \min_{ij} \bar{a}_{ij} + \min_j \bar{b}_j(t) \end{aligned} \tag{5}$$

Assuming $\max_j \bar{b}_j(t)$ and $\min_j \bar{b}_j(t)$ are mean ergodic processes, we can obtain the lower and upper bounds of $\bar{\alpha}_j(t)$ as [2]

$$r_l t \le \bar{\alpha}_j(t) \le r_h t \tag{6}$$

where

$$\begin{aligned} r_h &\triangleq \max_{ij} \bar{a}_{ij} + \mathrm{E}[\max_j \bar{b}_j(t)]; \\ r_l &\triangleq \min_{ij} \bar{a}_{ij} + \mathrm{E}[\min_j \bar{b}_j(t)]. \end{aligned}$$

Motivated by (6), we propose a normalized forward probability $\eta_j(t) \triangleq \alpha_j(t)e^{rt}$, where r is a positive constant. The final likelihood score consequently becomes $\max_j[\bar{\eta}_j(T) + \bar{a}_{jN}] = \log P(O_{1:T}) + rT$. First, this final score is valid simply because the offset rT stays the same for all word candidates and hence has no impact on the decoding decision. Second, dynamic programming still applies to the inference with the same computational complexity:

$$\bar{\eta}_j(t) = \max_i[\bar{\eta}_i(t-1) + \bar{a}_{ij}] + \bar{b}_j(t) + r. \tag{7}$$

[2] The lower and upper bounds were derived by writing the two inequalities in (5) for all frames and summing them up respectively. The following ergodic processes assumption was used in the derivation, $\frac{1}{T}\sum_t \max \bar{b}_j(t) = E[\max \bar{b}_j(t)]$, and similarly for the min case.

Finally, the dynamic range of the normalized log forward probability $\bar{\eta}_j(t)$ is controlled by r, since by Equation (6) we have

$$(r_l + r)t \leq \bar{\eta}_j(t) \leq (r_h + r)t. \quad (8)$$

To choose r, we compute the scores of all utterances from the training set evaluated on their own generative word models. And we let $r = -\,\mathrm{E}\,[\log P(O_{1:T}|\text{correct model})/T]$, in an attempt to normalize to zero the highest (or at least the correct-model's) log likelihood score of an utterance. It still might be true that when evaluating utterances with respect to a wrong word model the score decreases as T increases. When this happens, however, it will be for those words with lower partial likelihoods. The scheme, therefore, is analogous to pruning, where we essentially prune away unpromising partial hypotheses by collapsing their likelihoods down to be encoded with a very few number of bits.

4. DISCRIMINATIVE DISTORTION MEASURE

Lacking an analytically well-defined distortion measure to maximize recognition rate, conventional discrete HMM ASR systems often use Euclidean or Mahalanobis distance for vector quantization [7]. However, it is important to investigate new metrics customized to minimize the degradation in recognition accuracy.

As will be shown in Section 5, the forward probability requires the highest bit-width among all system variables and hence has the greatest impact on the memory bandwidth and the total table size. We are therefore particularly interested in further compressing this variable. Forward probabilities are in fact just likelihoods. It turns out that different likelihood magnitudes can be either more or less important for generating the ultimate correct answer. The correct answer will typically have a high likelihood, whereas very wrong answers will typically have low likelihoods and are likely to be pruned away regardless of their relative value. A standard data-driven quantization scheme, however, tends to allocate more bits to a value range only based on its higher probability mass. Since low likelihoods are the more probable (there is only one correct answer), more bits will be allocated to these low scores at quantization time (thereby giving them high resolution). Such a quantization scheme, therefore, can be quite wasteful.

Therefore, we propose to use a discriminatively inspired distortion measure to penalize low-valued forward probabilities. The distortion between a sample $\bar{\eta}_j(t) = x$ and its quantized value $Q(x)$ is defined as

$$D(x, Q(x)) = \frac{(x - Q(x))^2}{f(x)}, \quad (9)$$

where $f(x)$ is strictly positive. In choosing $f(x)$, it is desired that as x increases, the "distance" between x and $Q(x)$ will increase, which will cause more bits to be allocated for higher likelihood scores. $f(x) = s - x$ is such a function, where $s > \max x$ controls the degree of discrimination with smaller s implying higher discrimination. In this work, s was determined empirically on training data.

5. EXPERIMENTS AND RESULTS

5.1. Experimental setup

The quantization experiments were evaluated on NYNEX PhoneBook [9], an isolated word database recorded through telephone channels. The acoustic features are the standard MFCCs plus the log energy and their deltas. Mean subtraction and variance normalization are applied to all features in an attempt to make the system robust to noise and to decrease codebook sizes. The phone-based CHMMs are concatenated together into word-based models according to their pronunciation models, which enables the users to define their own vocabulary by composing a word from phonemes. Our system has 42 phoneme models, each with 4 emitting states except for the silence model. The state probability distribution is a mixture of 12 diagonal Gaussians. The testing set consists of 8 subsets, each with a different vocabulary of 75 words. The final WER is an average over them, where the baseline WER is 2.07%.

$\mathbf{x}$	$x_k(t)$	$\mathbf{s}$	$(x_k(t) - \mu_{i,k})^2$
$--$	$--$	$\mathbf{d}$	$d_{ik}(t)$
$\mathbf{m}$	μ_{ik}	$\mathbf{e}$	$e_i(t)$
$\mathbf{v}$	σ^2_{ik}	$\mathbf{p}$	$c_i - \frac{1}{2}e_i(t)$
$\mathbf{c}$	c_i	$\mathbf{q}$	$\bar{w}_i - c_i - \frac{1}{2}e_i(t)$
$\mathbf{w}$	$\bar{w}_i$	$\mathbf{b}$	$\bar{b}_j(t)$
$\mathbf{a}$	$\bar{a}_{ij}$	η	$\bar{\eta}_j(t); \bar{\eta}_j(t) + \bar{a}_{ij}$

Table 1. System variables in the ASR back-end, followed by their corresponding expressions in Equations (1) and (2)

Based on the analysis in the previous sections, we defined 13 variables to be quantized which are listed in Table 1. The variable $\mathbf{x}$ is the output feature scalar of the front-end, $\mathbf{m}$, $\mathbf{v}$, $\mathbf{c}$, $\mathbf{w}$ and $\mathbf{a}$ are the acoustic model parameters, and $\mathbf{s}$, $\mathbf{d}$, $\mathbf{e}$, $\mathbf{p}$, $\mathbf{q}$, $\mathbf{b}$ and η are other intermediate variables in the back-end system. There are 8 functions and hence 8 potential LUTs associated with these variables:

$$\begin{array}{lcllcl} \mathbf{s} & = & (\mathbf{x} - \mathbf{m})^{\mathbf{2}} & \mathbf{q} & = & \mathbf{w} + \mathbf{p} \\ \mathbf{d} & = & \mathbf{s}/\mathbf{v} & \mathbf{b} & = & \mathbf{b} \oplus \mathbf{q} \\ \mathbf{e} & = & \mathbf{e} + \mathbf{d} & \eta & = & \eta + \mathbf{a} \\ \mathbf{p} & = & \mathbf{c} - \mathbf{e}/\mathbf{2} & \eta & = & \eta + \mathbf{b} - \mathbf{r} \end{array}$$

Note that the max operations in Equation (7) can be implemented easily with integer comparisons, so no extra LUTs are required.

5.2. Quantization experiments and results

We quantized each variable individually with an increasing bit-width, leaving all other variables at full precision. The algorithm used for scalar quantization is LBG [10].

Table 2 shows the minimum bit-width to which a variable can be quantized without any increase in WER. Here we report all the system variables except for the accumulated Mahalanobis distance $\mathbf{e}$ and the forward probability η which will be discussed separately later. We let $\mathbf{q}$ and $\mathbf{b}$ share the same codebook because their value ranges have much overlap. As shown in the table, all the variables can be individually compressed no higher than 6 bits. It suggests that the floating-point representation of the variables of an ASR system is far from compact. It is also interesting to see that all the model parameters $\mathbf{m}$, $\mathbf{v}$, $\mathbf{c}$, $\mathbf{w}$ and $\mathbf{a}$ can each be quantized to no more than 5 bits. Low-precision representation of these parameters substantially reduces the offline storage as well as online computation. In addition, the feature scalar $\mathbf{x}$ and the state likelihood $\mathbf{b}$ can each be quantized to 5 bits, which implies an additional substantial memory savings.

variable	$\mathbf{m}$	$\mathbf{v}$	$\mathbf{w}$	$\mathbf{a}$	$\mathbf{x}$	$\mathbf{s}$	$\mathbf{d}$	$\mathbf{c}$	$\mathbf{p}$	$\mathbf{b}$ ($\mathbf{q}$)
min bit	4	5	2	2	5	6	5	5	6	5

Table 2. Minimum bit-width to which each variable can be quantized without increase in WER

We proposed two approaches in Section 2 to quantize variable $\mathbf{e}$. In our case, the operation $\mathbf{e} = \mathbf{e} + \mathbf{d}$ is repeated 26 times to get the final value of $\mathbf{e}$. Using linear accumulation, it involves only

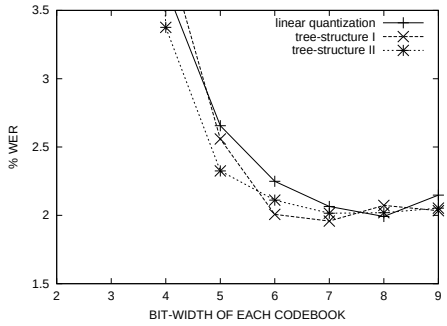

Fig. 2. Single-variable quantization for accumulative variable $\mathbf{e}$

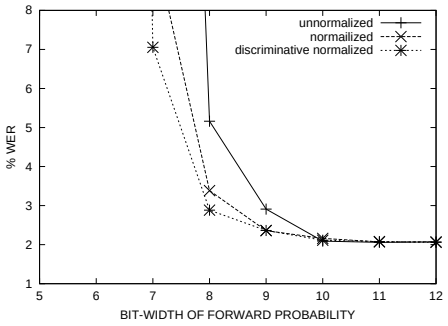

Fig. 3. Single-variable quantization for forward probability

two variables $\mathbf{e}$ and $\mathbf{d}$ and only one table $\mathbf{e} = \mathbf{e} + \mathbf{d}$. Alternately, it can have multiple variables and $\lceil \log 26 \rceil = 5$ different tables generated by the tree-structure accumulation each with a relatively small size. For simplicity, we quantized the variables in each level of the tree structure with the same bit-width. Figure 2 summarizes the results in terms of WER vs. bit-width of each codebook. Tree-structure I denotes the method where adjacent pairs are added at all levels to form the next-level codebook, whereas in tree-structure II, MFCCs are added to their corresponding deltas at the first level. It can be seen that in order to get baseline recognition rate, linear accumulation needs 7 bits and tree-structure schemes need 6 bits for each of its codebooks. The total table size, however, is a different story. To compare the total table size, we only consider the two functions in which $\mathbf{e}$ is involved.[3] Assuming $n_d = 5$, $n_c = 5$ and $n_p = 6$ according to table 1, we need 6 kBytes of lookup tables to realize the operations involving $\mathbf{e}$ using linear accumulation, whereas the required space goes up to almost 10 kBytes for the tree-structure schemes to achieve the same goal. It is worth noting that the feature dimension is fixed at 26 and is relatively low, and the addition operation only changes the dynamic range at a linear scale. This only yields a mild increase in the entropy of $\mathbf{e}$, thereby making the linear accumulation an effective approach.

As stated in Section 3, we applied our normalization to the α probability to reduce its entropy. To show the advantage of the normalization on quantization, we extracted samples of the forward probabilities with and without normalization on the same subset of training data, and generated codebooks based on the Euclidean distance distortion measure for each case. We additionally applied quantization to the normalized Viterbi search using our discriminative distortion measure. As shown in Figure 3, the normalized forward probability obviously outperforms its unnormalized counterpart by saving 1 bit while keeping the baseline recognition rate (thus halving the total table size). In fact, we believe the benefits of normalization would be more conspicuous on a task with longer utterances, such as connected-digit or continuous speech recognition. In addition, the discriminative distortion measure works slightly better than that the normal one.

We therefore chose the linear accumulation in quantizing $\mathbf{e}$ and used our normalization and discriminative distortion measure in quantizing the forward probability. Together with other variable quantization results, we generated codebooks with different bit-width for all system variables, to which an optimization search can be applied to find the best bit-width allocation scheme. Using this fully quantized system, we in [6] introduce a procedure for overall system design and achieve a complete ASR system back-end using only 59 kBytes of ROM tables with only a slight degradation in recognition accuracy. This system using custom arithmetic units has a clock-cycle speedup of 2 to 3 over a system using floating-point arithmetic.

[3] If $\mathbf{e}$, $\mathbf{d}$, $\mathbf{c}$ and $\mathbf{p}$ are quantized to n_e, n_d, n_c and n_p bits respectively, the total size of related tables is $n_e 2^{n_d+n_e} + n_p 2^{n_c+n_e}$ bits.

6. CONCLUSION AND FUTURE WORK

In this work, we presented several computation reordering and rescaling techniques on codebook design for an ASR system using custom arithmetic units. They enable a fully quantized system to process utterances with arbitrary length. We also gave a discriminative distortion measure to further compress the forward probability. The quantization would be further improved if a distortion measure entirely driven by recognition rate could be defined.

We would like to thank Chris Bartels and Gang Ji for proofreading and Carl Ebeling for much helpful advice.

7. REFERENCES

[1] X.Li and J.Bilmes, "Feature pruning in likelihood evaluation of HMM-based ASR systems," in *ASRU*, 2003.

[2] S.Takahashi, K.Aikawa, and S.Sagayama, "Discrete mixture HMM," in *ICASSP*, 1997, vol. 2, pp. 971–974.

[3] V.Digalakis, S.Tsakalidis, C.Harizakis, and L.Neumeyer, "Efficient speech recognition using subvector quantization and discrete-mixture HMMs," *Computer Speech and Language*, vol. 14, pp. 33–46, 2000.

[4] E.Bocchieri and B.K.Mak, "Subspace distribution clustering hidden Markov model," *IEEE Trans. on Speech and Audio Processing*, vol. 9, no. 3, pp. 264–275, 2001.

[5] K.Filali, X.Li, and J.Bilmes, "Data-driven vector clustering for low-memory footprint ASR," in *ICSLP*, 2002.

[6] J.Malkin, X.Li, and J.Bilmes, "Custom arithmetic for high-speed, low-resource ASR systems," in *ICASSP*, 2004.

[7] K-F.Lee, *Automatic speech recognition: the development of the SPHINX system*, Kluwer Academic Publishers, 1989.

[8] M.Jordan and C.Bishop, *An Introduction to Graphical Models*, pre-print, 2001.

[9] J.F.Pitrelli, C.Fong, and H.C.Leung, "PhoneBook: A phonetically-rich isolated-word telephone-speech database," in *ICASSP*, 1995.

[10] Y.Linde, A.Buzo, and R.M.Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Communication*, vol. 28, pp. 84–95, 1980.