TRAINING MANY SMALL HIDDEN MARKOV MODELS

G. E. Hinton — Gatsby Computational Neuroscience Unit, University College London, 17 Queen Square, London WC1N 3AR.

A. D. Brown — Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3G4 Canada.

1 INTRODUCTION

This paper describes research in progress on two quite different ways of training systems that are composed of many small Hidden Markov Models (HMM's). The first is a purely discriminative method in which all of the parameters of all the HMM's are adjusted to optimize classification performance. The second is an unsupervised method in which many little HMM's are used to model the probability density of a single set of sequences.

HMM's have been very successful in automatic speech recognition, mainly because there is an efficient way of fitting an HMM to data: the forward-backward algorithm and the Baum-Welch reestimation formulas. Despite this success, HMM's have several major limitations as models of sequential data. They represent the recent history of the sequence using a single, discrete K-state multinomial. The efficiency of the Baum-Welch restimation algorithm depends on this fact, but it severely limits the representational power of the model. The hidden state of a single HMM can only convey $\log_2 K$ bits of information about the recent history. If the generative model had a *distributed* hidden state representation [15] consisting of M variables each with K alternative states it could convey $M \log_2 K$ bits of information, so the information bottleneck scales linearly with the number of variables and only logarithmically with the number of alternative states of each variable. This suggests that it would be much better to use generative models composed of many small HMM's whose outputs are somehow combined to produce a sequence.

A second limitation of HMM's is that they have great difficulty in learning to capture long-range dependencies in a sequence [3]. In the case of natural language, there are many examples of word agreements which span a large portion of a sentence. This should be much easier to model in a system that is composed of many small HMM's since each HMM can then be concerned with a specific type of long-range regularity and its limited memory capacity does not get used up by having to deal with all the other intervening regularities in the sequence.

2 TRAINING MULTIPLE HMM'S IN THE SPIRIT OF BACKPROPAGATION

During the 1990's, many new machine learning algorithms were created by taking an existing algorithm that contained a Gaussian distribution and replacing the Gaussian with some other distribution, typically a mixture of Gaussians [11, 9] or a Laplacian [2, 14]. This method of generating new algorithms is far from being exhausted, but more creativity is now required in spotting the Gaussians that can be fruitfully replaced.

2.1 Discriminative training with one HMM per class

Bridle demonstrated interesting links between backpropagation and HMM's [5]. Consider a network with one output unit per class and make the output for class k on training case c be determined by the "softmax" function:

$$p_k^c = \frac{\exp(z_k^c)}{\sum_l \exp(z_l^c)} \tag{1}$$

where z_k^c is the total input to unit k from some other, adaptive computation and l is an index over all possible classes. If the aim is to maximize the log probability of the correct answer, we can train z_k^c by changing each parameter, θ_k, that determines z_k^c in proportion to the derivative of our objective function:

$$\frac{\partial \log p_{\mathrm{ans}(c)}^c}{\partial \theta_k} = (\delta_{\mathrm{ans}(c),k} - p_k^c)\frac{\partial z_k^c}{\partial \theta_k} \tag{2}$$

where $\mathrm{ans}(c)$ is the correct answer on training case c and $\delta_{\mathrm{ans}(c),k}$ equals 1 if $\mathrm{ans}(c) = k$ and 0 otherwise.

Bridle assumed that each z_k^c represented the log probability of observation sequence c under HMM_k and he showed that $\exp(z_k^c)$ could be computed by a particular kind of recurrent neural network that implemented the forward part of the forward-backward algorithm. Backpropagating the derivatives provided by equation 2 through this recurrent network allows all of the parameters of all of the HMM's to be trained discriminatively.

An obvious extension of Bridle's approach is to use several HMM's for each class. If we then define $\exp(z_k^c)$ to be the sum of the probabilities produced by the HMM's used for class k this extension is vacuous because it amounts to using a single, larger HMM with a partitioned state space. If, however, we multiply together the probabilities produced by the HMM's for class k we have a model that is potentially much more powerful. This is equivalent to adding in the log domain:

$$z_k^c = \sum_m z_{k,m}^c \tag{3}$$

where m is an index over the HMM's used for class k. Discriminative training thus has **two** major advantages: It uses all the parameters for discrimination and it allows a distributed representation of the hidden state over many small HMM's, which is an exponentially more efficient way of capturing the mutual information over time if the data contains many independent or almost independent temporal regularities.

2.2 Using pairs of HMM's to define features

Instead of using one or several HMM's per class, we propose to use many different pairs of relatively simple HMM's and to train each pair to extract a temporally extended feature that is useful for discriminating between classes. We arrived at this method by considering what the hidden units do in a standard backpropagation net that has one hidden layer and output units that obey equation 1. Each hidden unit, j, produces an output y_j which is given by:

$$y_j^c = \frac{1}{1 + \exp\left(-b_j - \sum_i w_{ji} x_i^c\right)} \tag{4}$$

where b_j is the bias of hidden unit j, w_{ji} is the weight on the connection from input unit i to hidden unit j, and x_i^c is the activity of input unit i. We can interpret y_j^c as the posterior probability that the input data was generated by one of two competing Gaussians that have the same covariance matrix. Without further loss of generality we can assume that this matrix is the identity matrix. If the Gaussians have means $\mu_\mathbf{j}^+$ and $\mu_\mathbf{j}^-$ in the input space and mixing proportions π_j^+ and π_j^- the posterior probability of the Gaussian with mean $\mu_\mathbf{j}^+$ given input vector $\mathbf{x}^c$ is given by:

$$\begin{aligned} y_j^c &= \frac{\pi_j^+ \exp\left(-\frac{1}{2}\sum_i (x_i^c - \mu_{ji}^+)^2\right)}{\pi_j^- \exp\left(-\frac{1}{2}\sum_i (x_i^c - \mu_{ji}^-)^2\right) + \pi_j^+ \exp\left(-\frac{1}{2}\sum_i (x_i^c - \mu_{ji}^+)^2\right)} \\ &= \frac{1}{1 + \exp\left(-\log\frac{\pi_j^+}{\pi_j^-} - \frac{1}{2}\sum_i (x_i^c - \mu_{ji}^-)^2 - (x_i^c - \mu_{ji}^+)^2\right)} \end{aligned} \tag{5}$$

Equation 5 is identical to equation 4 if we set:

$$b_j = \log\frac{\pi_j^+}{\pi_j^-} + \frac{1}{2}\sum_i (\mu_{ji}^-)^2 - (\mu_{ji}^+)^2 \ , \qquad w_{ji} = \mu_{ji}^+ - \mu_{ji}^-$$

The fact that the activity of a hidden unit can be interpreted as the posterior probability distribution over a mixture of two Gaussians immediately suggests several generalizations. Instead of using a pair of Gaussians for each hidden unit we could use m Gaussians in the mixture and represent the posterior for each multinomial hidden unit by m numbers that sum to one[1]. A different extension is

[1] The extreme version of this approach is to have a single multinomial hidden unit, which is known in the literature as a normalized Gaussian Radial Basis Function network.

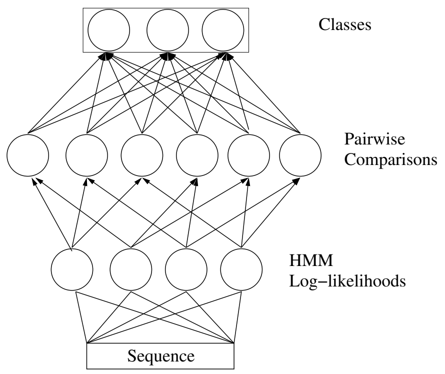

Figure 1: A backpropagation network with the log-probabilities of HMM's as inputs and hidden units which represent the relative probability of a sequence under pairs of HMM's.

to replace the two Gaussians by two HMM's which allows standard backpropagation to be applied to sequential data. Equation 4 then becomes

$$y_j^c = \frac{1}{1 + \exp\left(-\log p(\mathbf{x}^c|\mathrm{HMM}_j^+) + \log p(\mathbf{x}^c|\mathrm{HMM}_j^-)\right)} \tag{6}$$

where $\mathbf{x}^c$ is now a sequence. Note that this is quite different from the idea of using a neural network as a more sophisticated output model for each hidden state of an HMM. We are putting multiple HMM's inside a neural network as opposed to putting multiple neural networks inside an HMM.

When pairs of HMM's are used to define hidden features, it generally requires much more computation to get the probability of the input sequence under each HMM than to compute the probabilities of classes given the hidden activities. It therefore makes computational sense to reuse each HMM in defining many different features (figure 1). With m HMM's it is possible to define $n = m(m-1)/2$ features and although the n posterior probabilities only have m degrees of freedom they should nevertheless be in a form that makes it easier to compute the class probabilities.

Each hidden feature, j, learns weights, v, to all of the output units. These weights represent the logarithm of a probability distribution over all possible classes. The activity of the hidden unit represents a multiplicative coefficient on the log probability distribution which is equivalent to an exponent on the distribution itself. The combination rule for the outputs of the hidden features is:

$$z_k^c = \sum_j y_j^c v_{kj} \tag{7}$$

This is equivalent to multiplying together the distributions specified by each hidden feature. To simplify the interpretation of the hidden units, we can use the hyperbolic tangent function to determine the hidden activities. This simply scales and shifts the posterior probabilities that would be produced by the logistic in equation 6.

$$y_j^c = \tanh\left(\log p(\mathbf{x}^c|\mathrm{HMM}_j^+) - \log p(\mathbf{x}^c|\mathrm{HMM}_j^-)\right) \tag{8}$$

Each hidden unit then outputs one probability distribution over classes if one of its two HMM's fits the data much better than the other, and a complementary distribution if the other HMM fits much better. When there is no clear winner, the hidden unit uses the posterior distribution over the two HMM's to geometrically interpolate between the two distributions.

3 PRELIMINARY RESULTS OF THE DISCRIMINATIVE ALGORITHM

One of the benefits of using more HMM's than classes is that no single HMM is necessarily responsible for identifying each class. Several HMM's may discover features useful for discriminating one class from the others. A network consisting of many HMM's should, therefore, be less susceptible to noise than one in which a single HMM is assigned to model each class. This insight is the basis for the recent interest in subband-based speech and speaker recognition [4, 12, 1], where rather than trying to build models of all the spectral information at once, classification is performed on independent frequency subbands and the decisions of the subband models are combined to make a global decision. Here we present a simple example of speaker classification to demonstrate this effect. However, rather than explictly modelling spectral subbands, we provide all the channel information to each of the constituent models and allow the learning procedure to determine how the HMM's specialize on parts of the spectrum.

Using data from the CSLU Speaker Recognition corpus [7], we took examples of two individuals uttering strings of connected digits. The telephone quality speech, sampled at 8kHz was processed with a bank of 24 Mel frequency, triangular bandpass filters, with centers between 55 and 3655 Hz. The data was collected in multiple sessions under different recording conditions, but for the purposes of this experiment training and test sets were drawn randomly from all recording sessions so that the training and testing sets were drawn from the same distribution. We used 64 cases for training and 32 for testing. Thus, this experimental set up is not reflective of a a realistic speaker verification task where the training and testing conditions are expected to differ.

The task was to learn to discriminate two speakers with and without artificial bandlimited noise added to the spectral information. We compared a network with 6, 3-state HMM's and $\binom{6}{2}$ hidden units, with one that had 2, 8-state HMM's and a single hidden unit. Such a model is equivalent to discriminatively training two HMM's as described by Bridle [5]. The two models are matched for parameters as closely as possible given the integer number of hidden states in the respective HMM's, and the number of parameters is kept quite small. Since we are only modelling the conditional class densities and not the data density it is very easy to overfit by using too many parameters.

In the first test condition no noise was added to the data set and in multiple training sessions of the two models they both generalized to the training data equally well. In the second condition, 4 adjacent channels of each training and test sequence were corrupted with 0db Gaussian noise. The frequencies of the 4 adjacent channels were chosen at random in each sequence. As expected, the model with more small HMM's generalizes better to test data than the one with just one HMM per class.

4 PRODUCTS OF HMM'S

It is also possible to train multiple HMM's in a non-discriminative way. This is more appropriate for tasks such as novelty detection in which a model is trained on one set of sequences and is then used to decide if a test sequence comes from the same distribution. We first formulate a generative model in which the outputs of many separate HMM's are combined to produce a sequence and then we adapt the parameters of all the HMM's to maximize the likelihood of the observed sequences or to optimize some other similar objective function.

One such generative model assumes that all of the HMM's choose paths through their state spaces independently and then, at each time in the sequence, the hidden states selected by each HMM jointly determine the observation. This way of combining HMM's is known as a Factorial HMM [8] and is shown in Fig. 2.

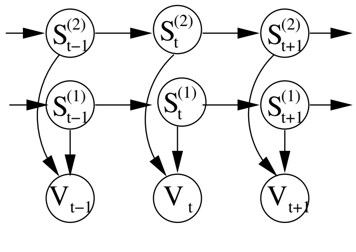

Figure 2: Factorial HMM

In a causal belief network each local probability distribution can be independently estimated given the posterior distribution of the hidden variables conditioned on the evidence. However, it is exponentially expensive to compute this posterior distribution exactly because observing the visible variables induces dependencies among the hidden variables. Ghahramani and Jordan [8] handle this problem by approximating the posterior with a factored, variational distribution.

There is a very different, non-causal, way of combining multiple HMM's which we call a "Product of HMM's" or PoHMM. As before, we allow each HMM to independently choose a path through its state space. But we also allow each HMM to independently generate an output symbol at each time step. If all of the HMM's happen to produce exactly the same sequence, we output that sequence. Otherwise we try again. It is immediately apparent that this is not an efficient way of generating sequences. It is, nevertheless, a perfectly legitimate generative model because, given the individual HMM's, it does define a probability distribution over sequences. Although it is hard to generate from a PoHMM, inferring the hidden states is much easier than in a factorial HMM because, given the data, the hidden states of different HMM's are conditionally independent. Conversely, when the data is unobserved, the hidden states of different HMM's are marginally dependent[2] so this generative model is just the opposite of a the factorial HMM which has conditional dependence and marginal independence.

The PoHMM generative model is equivalent to multiplying together the distributions over sequences defined by the individual HMM's and then renormalizing (Fig. 3):

- Parameters: $\Theta = \{\theta\}_{m=1}^{M}, \theta = \{A_m, B_m, \pi_m\}$
- Observed Variables: $V^{1:T} = \{V^0 \dots V^T\} \in \mathcal{V}$
- Hidden Variables: $S_m^{1:T} = \{S_m^0 \dots S_m^T\} \in \mathcal{S}_m$

$$P_m(V^{1:T}, S_m^{1:T}|\theta_m) = P(S_m^0|\pi_m)\prod_{t=2}^{T} P(S_m^t|S_m^{t-1}, A_m)\prod_{t=1}^{T} P(V^t|S_m^t, B_m) \quad (9)$$

$$P_m(V^{1:T}|\theta_m) = \sum_{\mathcal{S}_m} P_m(V^{1:T}, S_m^{1:T}|\theta_m) \quad (10)$$

$$P(V^{1:T}|\Theta) = \frac{\prod_{m=1}^{M} P_m(V^{1:T}|\theta_m)}{Z(T,\Theta)}, \quad (11)$$

$$Z(T,\Theta) = \sum_{U^{1:T}\in\mathcal{V}} \prod_{m=1}^{M} P_m(U^{1:T}|\theta_m), \quad (12)$$

where θ_m is the set of parameters for each HMM in the product, and π_m, A_m and B_m are the parameterizations of the initial state, the transition and the output distributions respectively. The summation in (10) is tractable due to the Markov property of the hidden states and can be efficiently computed, but the partition function, Z, a summation over all possible observed strings, does not have such a nice decompostion. The existence of Z in the denominator of (11) makes it intractable to compute the exact gradient of the log likelihood of the observed data *w.r.t* the parameters, so it appears to be very hard to fit a PoHMM to data. Gibbs sampling can be used to estimate the derivatives of the partition function but this is very slow and noisy. Fortunately, there is an alternative objective function for learning whose gradient can be approximated accurately and efficiently [10]. It has been shown that optimizing this alternative objective function leads to good generative models for non-sequential data [10] and Brown and Hinton [6] have shown that the same approach works for PoHMM's when the data consists of sequences of discrete symbols. In the following sections we summarize the results of Brown and Hinton [6] and then describe preliminary results of applying PoHMM's to speech.

[2]Although the PoHMM generative model chooses paths through the different HMM's independently, it rejects all choices that do not result in identical sequences over the visible variables and this rejection induces dependencies among the hidden paths in the successful attempts.

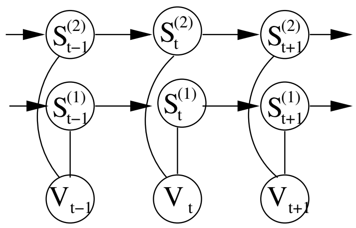

Figure 3: Product of HMM's

4.1 Training products of HMM's using contrastive divergence

Maximizing the log likelihood of the data is equivalent to minimizing the Kullback-Leibler divergence $KL(P^0||P^\infty)$ between the observed data distribution, P^0, and the equilibrium distribution, P^∞, produced by the generative model[3]. Instead of simply minimizing $KL(P^0||P^\infty)$ we minimize the "contrastive divergence" $KL(P^0||P^\infty) - KL(P^1||P^\infty)$, where P^1 is the distribution over one-step reconstructions of the data that are produced by running a Gibbs sampler for one full step, starting at the data. The advantage of using the contrastive divergence as the objective function for learning is that the intractable derivatives of the partition function cancel out and, if we are prepared to ignore a term that turns out to be negligible in practice [10], it is easy to follow the gradient of the contrastive divergence using the following procedure:

1. Calculate each model's gradient $\frac{\partial}{\partial \theta_m} P(V^{1:T}|\theta_m)$ on a sequence using the forward-backward algorithm.
2. For each model, pick a path from the posterior distribution over paths through state space.
3. At each time step, get the distribution over symbols specified by the hidden state for that time step in the path chosen for each HMM. Multiply these output distributions together and renormalize to get the reconstruction distribution at that time step.
4. Draw a sample from the reconstruction distribution at each time step to get a reconstructed sequence. Compute each model's gradient on the new sequence $\frac{\partial}{\partial \theta_m} P(\hat{V}^{1:T}|\theta_m)$
5. Update the parameters:

$$\Delta\theta_m \propto \frac{\partial \log P(V^{1:T}|\Theta)}{\partial \theta_m} - \frac{\partial \log P(\hat{V}^{1:T}|\Theta)}{\partial \theta_m}$$

[3]We call this distribution P^∞ because one way to get exact samples from it is to run a Gibbs sampler for an infinite number of iterations

There is an intuitive justification for the contrastive divergence objective function. The one-step reconstructions of the data are always, on average, more probable under the PoHMM than the data itself unless the model is perfect or the one-step Markov Chain used for producing the reconstructions fails to mix. The increase in probability caused by reconstructing the data from the model is a way of measuring how much the model would like to change the data distribution. A perfect model would still allow an individual reconstruction to be different from the individual sequence from which it was produced, but the distributions of the two and hence their average probabilities would be the same. So by minimizing the extent to which the reconstructions are more probable than the data we are eliminating the tendency of the model to prefer sequences that have lower probability in the data distribution than the actual data.

To compute the gradient of the HMM we use an EM-like trick. Directly computing the gradient of an HMM is difficult due to the fact that all the parameters are coupled through their influence on the hidden states. If the HMM were visible and the hidden states were known then the gradient of the log-likelihood for each parameter would decouple into an expression involving only local variables. As in EM, we use the posterior distribution over the hidden states in place of actual values by using the identity:

$$\frac{\partial}{\partial \theta} \log P(V^{1:T}|\theta) = \frac{\partial}{\partial \theta} \big\langle \log P(V^{1:T}, S^{1:T}|\theta) \big\rangle_{P(S^{1:T}|V^{1:T})} \tag{13}$$

This says that if we compute the posterior of the HMM using the forward-backward algorithm we can take the gradient of the complete data log-likelihood using the sufficient statistics of the hidden variables in place of actual values.

A second optimization trick which we have used is to re-parameterize the probabilities of the HMM, using the softmax function. Working in this domain allows us to do unconstrained gradient descent over the real numbers. Doing gradient optimization directly in the probability domain would involve the more difficult proposition of constraining the parameters to the probability simplex. An added advantage of this re-paramaterization is that the probabilities cannot go to zero anywhere. It is clearly desirable in the PoHMM framework that none of the individual HMM's assigns zero probability to a sequence, as this would effectively veto the other HMM's.

As an example we look at the gradient rule for the transition probabilities of an HMM, $P(S^t = j|S^{t-1} = i) = A_{ij}$. If we re-parameterize using the softmax function:

$$A_{ij} = \frac{\exp(a_{ij})}{\sum_j \exp(a_{ij})}. \tag{14}$$

Taking the derivative with respect to a_{ij} yields

$$\frac{\partial}{\partial a_{ij}} \big\langle \log P(V^{1:T}, S^{1:T}) \big\rangle = \sum_{t=1}^{T} \langle S^t = j, S^{t-1} = i \rangle - (\sum_{t=1}^{T} \langle S^{t-1} = i \rangle) A_{ij}, \tag{15}$$

As before the angle brackets indicate an expectation with respect to the posterior of the hidden states. This has the intuitive interpretation that the derivative for the softmax parameter a_{ij} regresses

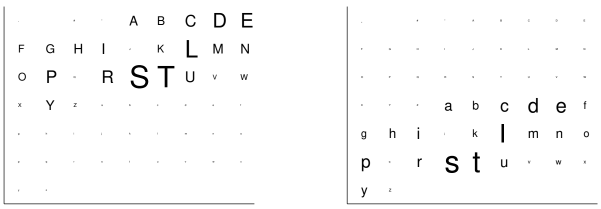

Figure 4: An 'eye-chart' diagram of the output distributions of the 2-state HMM in the PoHMM. Each chart corresponds to a single state's output distribution and the size of each symbol is proportional to the probability mass on that symbol.

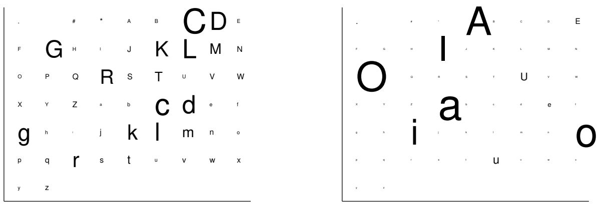

Figure 5: Eye-chart diagram of the output distributions of two of the states of the 30 state HMM

toward the point where A_{ij} is equal to the expected transition probability under the posterior. If we set the derivative to zero and solved this equation directly, we would recover the Baum-Welch update equation.

5 RESULTS ON A TOY TEXT-MODELING PROBLEM

To demonstrate the relative merits of a product of HMM's versus a single HMM, we tried modelling character strings from a corpus of English text, but we modified the task to better demonstrate the advantages of a product model. Rather than training the model on a single case, or mixed case text, we trained it on data in which the characters in a sentence were either all upper case or all lower case. Thus there really are independent factors underlying this sequence: the binary decision of upper case or lower case and the statistics of the case-independent letter sequences.

We used 8600 sentences[4] and converted them to all upper and all lower case to yield over 17,000 training sentences. 56 symbols were allowed: 4 symbols for space and punctuation, 26 upper and 26 lower case letters. We compared a single HMM with 32 hidden states against a product of a 2 state and a 30 state hidden Markov model. In the product model the 2 state HMM learns to differentiate upper and lower case. It 'votes' to put probability mass on the upper or lower case letters respectively (Fig. 4), and it enforces the continuity through its transition matrix. Then the 30-state

[4]from Thomas Hardy's "Tess of the d'Urbervilles" available from Project Gutenberg (http://www.gutenberg.net)

HMM need only learn the case-independent statistics of the characters and the fact that the upper and lower case characters are analogous, placing proportional amounts of probability mass on the two halves of the symbol set. In Fig. 5 we see an example of two of the big HMM's 30 hidden states. Its output distributions are symmetric over the upper and lower case letters, indicating that it has left the modelling of case to the smaller 2-state HMM model.

By contrast, the single HMM has to partition its data space into two parts, one each for upper and lower case. In effect it has to model the caseless letter statistics with a much smaller number of hidden states. This can be seen in Fig. 6a) where the observation distributions of the 32 states fall into 3 categories: punctuation, upper case, and lower case. Similarly we can see in the transition matrix (Fig. 6b) that the upper case states only transition to upper case states and likewise for the lower case states.

We cannot compute the log likelihood of a string under the PoHMM becasue of the intractable partition function, but we can easily compute the probability of a single symbol conditioned on the other symbols in a sentence. This leads to a simple, interesting test of the models which we refer to as the "symmetric Shannon game". In the original Shannon game [13], a prediction of the next symbol in a sequence is made given the previous N symbols. In the symmetric Shannon game the model is given both past and future symbols and is asked to predict the current one. We can compute this distribution exactly since we need only normalize over the missing symbol and not all strings of symbols. For models based on directed acyclic graphs, such as an HMM, it is easy to compute the probability of the next symbol in a sequence given the symbols so far. Somewhat surprisingly, this is not true for undirected models like a PoHMM. If the data after time t is missing, the posterior distribution over paths through each HMM up to time t depends on how easily these paths can be extended in time so as to reach agreement on future data.

Table 7 shows a comparison of several PoHMM models with a single large HMM. They were scored on a set of 60 hold-out sentences with an equal number of upper and lower case. The product of a 2-state and 30-state HMM with 2728 parameters, while capturing the componential structure we were hoping for, does not outperform a single 32 state HMM which has been roughly matched for the number of parameters (2848 parameters). This is mainly an optimization problem, because if we train a 2-state model alone and a 30-state model on uni-case text, and then use their parameters to initialize the PoHMM then it does much better than the single HMM. If we use a product of many, simple HMM's then the optimization problem is eased. A product of 10, 4-state HMM's, which has still fewer parameters (2440), performs as well as a hand initialized product of 2 HMM's. Increasing, the number of HMM's in the product provides further improvements while the parameters and computation time scale linearly with the number of HMM's in the model.

6 AN EXTENSION FOR BIGGER ALPHABETS OF SYMBOLS

One concern that we have about the PoHMM is that each HMM has it's own output distribution over the data, which could include many parameters if there are a large number of symbols. One way to deal with this is to add an extra layer of shared hidden features between the hidden variables of

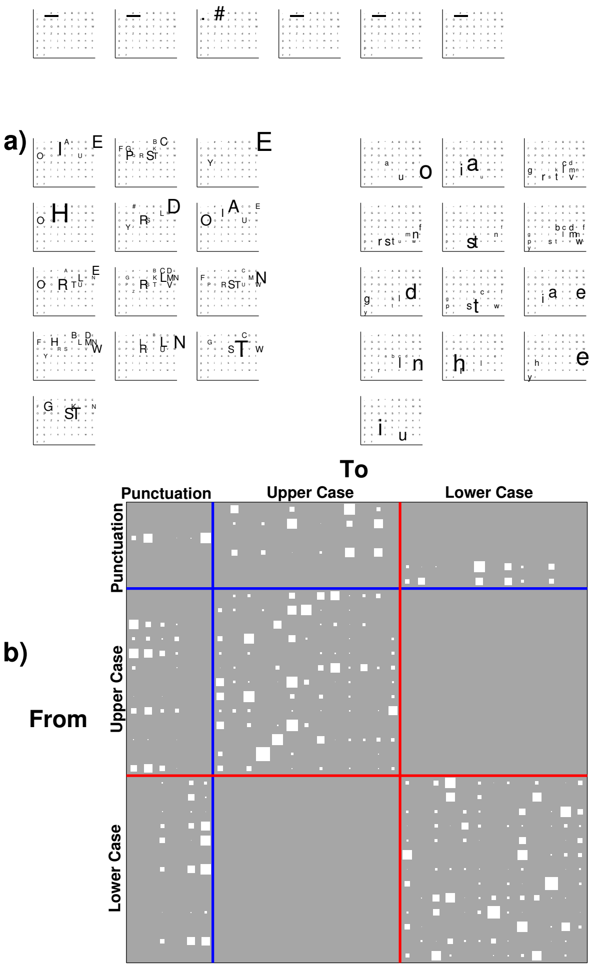

Figure 6: The 32 state HMM a) the observation probabilities of the HMM b) a diagram of the transition matrix where the area of the square indicates the probability of going to a state.

Model	Sym. Shannon (bits)
PoHMM 40 x 4-states	1.96
PoHMM 20 x 4-states	2.06
PoHMM 10 x 4-states	2.13
PoHMM (2-state + 30-state, pre-initialized)	2.14
32 State HMM	2.46
PoHMM (2-state + 30-state random initialization)	2.73

Figure 7: Symmetric Shannon scores for several PoHMM models and a single large HMM

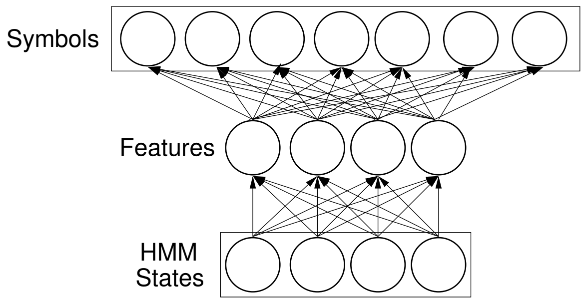

Figure 8: Output model of the HMM's

the HMM and the output symbols. Sharing the output model features among the HMM's, greatly reduces the number of free parameters in the PoHMM and it has the benefit that similarities between symbols learned by one model do not have to be re-learned again and again in the other models. Each HMM retains it's own transition distribution and it's own weights from it's hidden states to the hidden features[5].

We parameterize the output model as a two layer network, with a linear hidden layer and a softmax non-linearity in the output layer (Fig. 8). Note that we do not constrain the hidden layer values to be positive or sum to one. They may be positive or negative. If we constrained the hidden features to be a proper probability distribution then this would be equivalent to inserting a single discrete valued stochastic variable between the hidden variable and the visible variable of the HMM. This is not as powerful a representation as allowing the hidden features to take on independent real values. The formula for such an output model is given by:

$$P(V|S;\theta_m) = \sigma(\mathbf{s}'U_mW) \tag{16}$$

Where we treat the hidden state, $\mathbf{s}$, as a column vector of indicator variables – a one in the position

[5]This resembles the technique of using a common pool of learned Gaussian distributions and allowing each node to create its own output distribution by learning its own set of mixing proportions for the shared Gaussians.

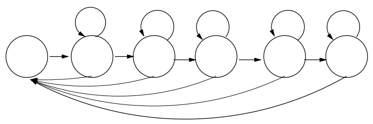

Figure 9: State transition diagram of the HMM's in the PoHMM used for acoustic modelling. The states were organized in a loop, with left to right transitions and a self transition whose probabilities were learnable. In addition there was a small minimum probability of transitioning back to the initial state (indicated by thin arrows).

of the discrete state which the hidden variable takes. σ is the softmax function. U is the matrix of weights which the states of model m place on the hidden features and W is the matrix shared hidden features. Interestingly, this output distribution is also a product model. The columns of W are linearly combined in the log domain and then pushed through the softmax function to get a probability distribution. The rows of U are the weights that each state puts on these basis distributions.

There are two ways that we can regularize or constrain the output model. One way is to create a bottle neck by using a small number of hidden features. This is equivalent to decomposing the stochastic output matrix as the product of two lower rank matrices. The other way is to use a large number of hidden features, but use another regularizer on the output weights forcing them to be small. Thus, the hidden features are restricted to be soft distributions over the output symbols. We have applied this technique to a task involving symbol sequences it improves the generalization performance (Brown and Hinton, 2001).

7 PRELIMINARY RESULTS ON MULTIBAND ACOUSTIC MODELLING

We applied the learning algorithm for PoHMM's to an acoustic modelling task in which there was just one speaker. Our aim was simply to see how the various HMM's specialized on different aspects the task. The PoHMM contained 10 HMM's each of which had 6 hidden states with state transitions constrained as shown in figure 9. Each hidden state has its own, learned diagonal Gaussian output model.

The PoHMM was trained on raw high resolution spectrograms. Using speech from the TIMIT database sampled at 16 kHz, an FFT was taken on speech windows of 64ms (1024 samples) at a frame rate of 24ms (384 samples). The $\log$ of the 513 unique spectral magnitude coefficients were used to represent the speech at each timestep.

The individual HMM's specialize in different frequency bands. They do this by using Gaussian output models that have tight variances on frequency dimensions that they care about and broad variances on frequency dimensions that they do not care about. One way of showing how an HMM specializes is to indicate, for each pixel in the spectrogram, when that HMM is constraining the value of the pixel more strongly than any other HMM. For each HMM we chose the state with the highest posterior probability at each time and among these states we found, for each frequency, the one that had the lowest variance on that frequency dimension of its Gaussian output model. Figure 10 shows a spectrogram and, for two of the HMM's, the pixels in the spectrogram that are dominated by the HMM. Notice that it is quite possible for an HMM to have different hidden states that specialize in somewhat different frequency bands so it can learn to model structure that moves from one frequency band to another.

8 CONCLUSIONS

We have described both discriminative and non-discriminative methods of fitting systems in which single large HMM's are replaced by many small HMM's. Although the fitting is slower, the increased representational power of multiple small HMM's means that the same number of parameters can produce better models for both discrimination and density estimation. When applied to speech, the small HMM's have a tendency to specialize in different frequency bands without requiring any prior specification of which bands to use.

Acknowledgements

We thank Zoubin Ghahramani, Sam Roweis, Brian Sallans and Chris Williams for helpful discussions. This work has been supported by the Gatsby Charitable Foundation and NSERC.

References

[1] L. Basacier and J. Bonastre. Subband architecture for automatic speaker recognition. *Signal Processing*, 80:1245–1259, 2000.

[2] A. Bell and T.J. Sejnowski. An information-maximisation approach to blind separation and blind deconvolution. *Neural Computation*, 7:1129–1159, 1995.

[3] Y. Bengio and P. Frasconi. Diffusion of credit in Markovian models. In G. Tesauro, D.S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 553–560, Cambridge, MA, 1995. MIT Press.

[4] H. Bourlard and S. Dupont. Subband-based speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1251–1254, Munich, Germany, April 1997.

[5] J.S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 211–217, San Mateo, CA, 1990. Morgan Kaufmann.

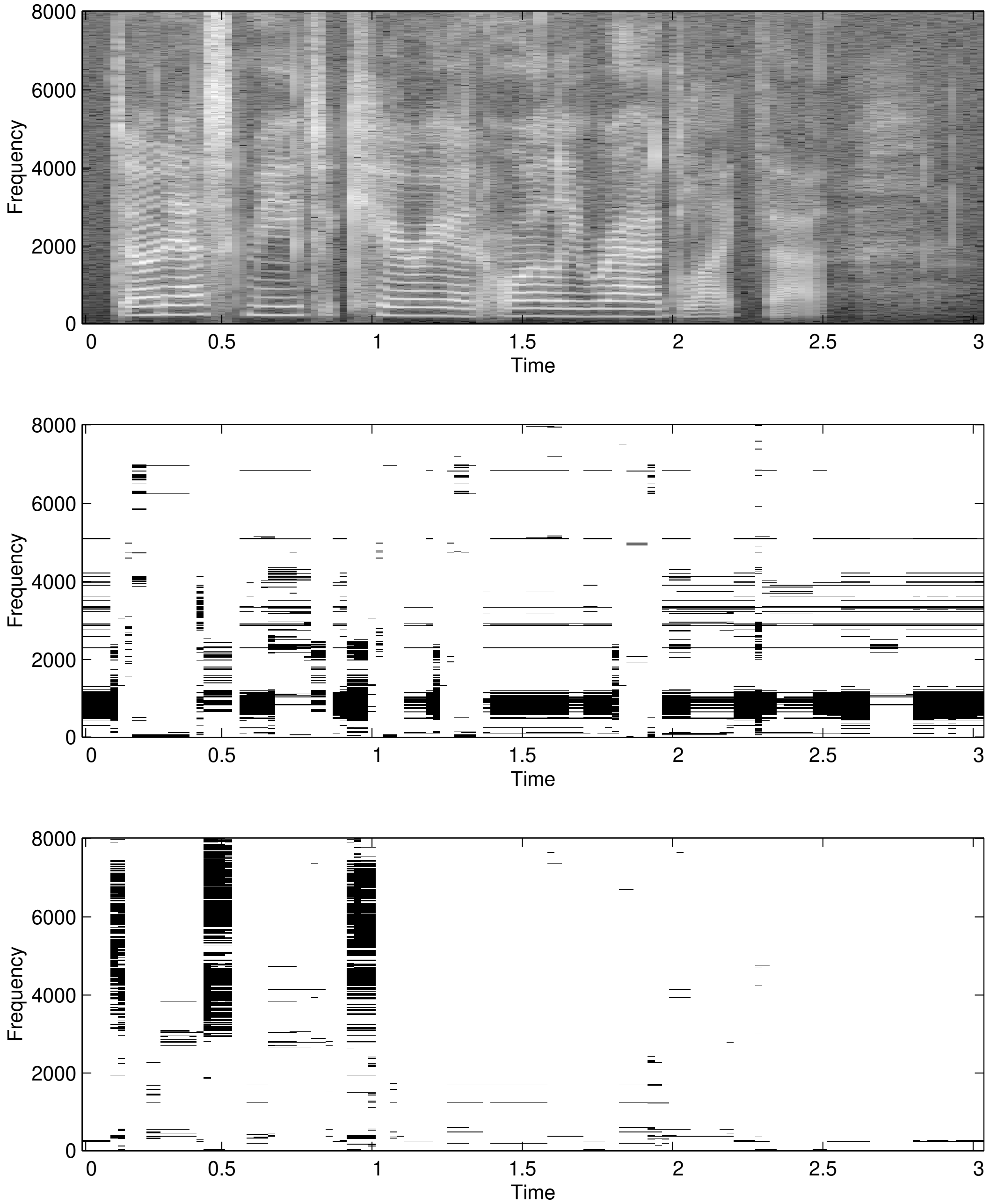

Figure 10: A spectrogram and the pixels on which two of the HMM's dominate. Notice that the first HMM has specialized to model structure at low frequencies.

[6] A.D. Brown and G.E. Hinton. Products of hidden Markov models. In T. Jaakkola and T. Richardson, editors, *Artificial Intelligence and Statistics 2001*, pages 3–10, San Francisco, CA, 2001. Morgan Kaufmann.

[7] R. Cole, M. Noel, and V. Noel. The CSLU speaker recognition corpus. In *Proceedings of ICSLP*, Sydney, Australia, 1998.

[8] Z. Ghahramani and M.I. Jordan. Factorial hidden markov models. *Machine Learning*, 29(2/3):245–273, November 1997.

[9] T. Hastie and R. Tibshirani. Discriminant analysis by Gaussian mixtures. *Journal of the Royal Statsistical Society B*, 58(1):155–176, 1996.

[10] G.E. Hinton. Training products of experts by minimizing contrastive divergence. Technical Report GCNU TR 2000-004, Gatsby Computational Neuroscience Unit,University College London, London, UK, 2000.

[11] S.J. Nowlan and G.E. Hinton. Simplifying neural networks by soft weight sharing. *Neural Computation*, 4:173–193, 1992.

[12] L. K. Saul, M. G. Rahim, and J. B. Allen. A statistical model for robust integration of narrowband cues in speech. *Computer Speech and Language*, 2001. In press.

[13] C. E. Shannon. Prediction and entropy of printed english. *Bell System Techncial Journal*, 27:623–656, July, October 1948.

[14] R. Tibshirani. Regression selection and shrinkage via the lasso. *Journal of the Royal Statistical Society B*, 58(1):267–288, 1996.

[15] C.K.I. Williams and G. E. HInton. Mean field networks that learn temporally distorted strings. In D.S. Touretzky, J.L. Elman, T.J. Sejnowski, and G.E. Hinton, editors, *Connectionist Models Proceedings of the 1990 Summer School*, pages 18–22, San Mateo, CA, 1991. Morgan Kaufmann.