Acoustic Modeling Using Deep Belief Networks

Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton

***Abstract*—Gaussian mixture models are currently the dominant technique for modeling the emission distribution of hidden Markov models for speech recognition. We show that better phone recognition on the TIMIT dataset can be achieved by replacing Gaussian mixture models by deep neural networks that contain many layers of features and a very large number of parameters. These networks are first pre-trained as a multi-layer generative model of a window of spectral feature vectors without making use of any discriminative information. Once the generative pre-training has designed the features, we perform discriminative fine-tuning using backpropagation to adjust the features slightly to make them better at predicting a probability distribution over the states of monophone hidden Markov models.**

***Index Terms*—Acoustic modeling, deep belief networks (DBNs), neural networks, phone recognition.**

I. Introduction

AUTOMATIC speech recognition (ASR) has evolved significantly over the past few decades. Early systems typically discriminated isolated digits or yes/no, whereas current systems can do quite well at recognizing telephone-quality, spontaneous speech. A huge amount of progress has been made in improving word recognition rates, but the core acoustic modeling has remained fairly stable, despite many attempts to develop better alternatives.

A typical ASR system uses hidden Markov models (HMMs) to model the sequential structure of speech signals, with each HMM state using a mixture of Gaussians to model a spectral representation of the sound wave. The most common spectral representation is a set of Mel frequency cepstral coefficients (MFCCs) derived from a window of about 25 ms of speech. The window is typically advanced by about 10 ms per frame, and each frame of coefficients is augmented with differences and differences of differences with nearby frames.

One research direction involves using deeper acoustic models that contain many layers of features. The work in [1] proposes a hierarchical framework where each layer is designed to capture a set of distinctive feature landmarks. For each feature, a specialized acoustic representation is constructed in which that feature is easy to detect. In [2], a probabilistic generative model is introduced where the dynamic structure in the hidden vocal tract resonance space is used to characterize long-span contextual influence across phonetic units.

Feedforward neural networks have been used in many ASR systems [3]–[5]. Inspired by insights from [6], the TRAP architecture [7] models a whole second of speech using a feature vector that describes segments of the temporal evolution of critical-band spectral densities within a single critical band. Sub-word posterior probabilities are estimated using feedforward neural networks for each critical band and these probabilities are merged to produce the final estimate of the posterior probabilities using another feedforward neural network. In [8], the split temporal context system is introduced which modifies the TRAP system by including, in the middle layer of the system, splits over time as well as over frequency bands before the final merger neural network.

Feedforward neural networks offer several potential advantages over GMMs:

- Their estimation of the posterior probabilities of HMM states does not require detailed assumptions about the data distribution.
- They allow an easy way of combining diverse features, including both discrete and continuous features.
- They use far more of the data to constrain each parameter because the output on each training case is sensitive to a large fraction of the weights.

The benefit of each weight in a neural network being constrained by a larger fraction of training cases than each parameter in a GMM has been masked by other differences in training. Neural networks have traditionally been trained purely discriminatively, whereas GMMs are typically trained as generative models (even if discriminative training is performed later in the training procedure). Generative training allows the data to impose many more bits of constraint on the parameters (see below), thus partially compensating for the fact that each component of a large GMM must be trained on a very small fraction of the data.

MFCCs, GMMs, and HMMs co-evolved as a way of doing speech recognition when computers were too slow to explore more computationally intensive approaches. MFCCs throw away a lot of the information in the sound wave, but preserve most of the information required for discrimination. By including temporal differences, MFCCs partially overcome the very strong conditional independence assumption of HMMs, namely that successive frames are independent given the hidden state of the HMM. The temporal differences also allow diagonal covariance Gaussians to model the strong temporal covariances by reducing these particular pairwise covariances to individual coefficients. As we shall see, a Fourier transform based filterbank, densely distributed on a mel-scale, is a potential alternative to MFCCs for models that can easily model correlated features [7].

Manuscript received September 23, 2010; revised January 06, 2011; accepted January 10, 2011. Date of publication January 31, 2011; date of current version December 16, 2011. The research was supported in part by a gift from Microsoft Research and in part by grants from the Natural Sciences and Engineering Research Council of Canada and the Canadian Institute for Advanced Research. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Nelson Morgan.

The authors are with the University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: asamir@cs.toronto.edu; gdahl@cs.toronto.edu; hinton@cs.toronto.edu).

Digital Object Identifier 10.1109/TASL.2011.2109382

GMMs are easy to fit to data using the EM algorithm, especially when they have diagonal covariance matrices, and with enough components they can model any distribution. They are, however, statistically inefficient at modeling high-dimensional data that has any kind of componential structure. Suppose, for example, that N significantly different patterns can occur in one sub-band and M significantly different patterns can occur in another sub-band. Suppose also that which pattern occurs in each sub-band is approximately independent. A GMM requires NM components to model this structure because each component must generate both sub-bands (each piece of data has only a single latent cause). On the other hand, a model that explains the data using multiple causes only requires $N + M$ components, each of which is specific to a particular sub-band. This exponential inefficiency of GMMs for modeling factorial structure leads to ASR systems that have a very large number of Gaussians, most of which must be estimated from a very small fraction of the data.

In this paper, we focus on a more powerful alternative to GMMs for relating HMM states to feature vectors. In particular, we reconsider the use of multi-layer, feed-forward neural networks that take a window of feature vectors as input and produce posterior probabilities of HMM states as output.

Previous instantiations of the neural network approach have used the backpropagation algorithm to train the neural networks discriminatively. These approaches coincided nicely with a trend initiated by [9] in which generative modeling is replaced by discriminative training. Discriminative training is a very sensible thing to do when using computers that are too slow to learn a really good generative model of the data. As generative models get better, however, the advantage of discriminative training gets smaller[1] and is eventually outweighed by a major disadvantage: the amount of constraint that the data imposes on the parameters of a discriminative model is equal to the number of bits required to specify the correct labels of the training cases, whereas the amount of constraint for a generative model is equal to the number of bits required to specify the input vectors of the training cases. So when the input vectors contain much more structure than the labels, a generative model can learn many more parameters before it overfits.

The benefit of learning a generative model is greatly magnified when there is a large supply of unlabeled speech in addition to the training data that has been labeled by a forced HMM alignment. We do not make use of unlabeled data in this paper, but it could only improve our results relative to purely discriminative approaches.

Naturally, many of the high-level features learned by the generative model may be irrelevant for making the required discriminations, even though they are important for explaining the input data. However, this is a price worth paying if computation is cheap and *some* of the high-level features are very good for discriminating between the classes of interest.

The main novelty of our work is to show that we can achieve consistently better phone recognition performance by "pre-training" a multi-layer neural network, one layer at a time, as a *generative* model of the window of speech coefficients. This pre-training makes it easier to optimize deep neural networks that have many layers of hidden units and it also allows many more parameters to be used before overfitting occurs. The generative pre-training creates many layers of feature detectors that become progressively more complex. A subsequent phase of discriminative fine-tuning, using the standard backpropagation algorithm, then slightly adjusts the features in every layer to make them more useful for discrimination. The big advantage of this new way of training multi-layer neural networks is that the limited amount of information in the labels is not used to design features from scratch. It is only used to change the features ever so slightly in order to adjust the class boundaries. The features themselves are discovered by building a multi-layer generative model of the much richer information in the window of speech coefficients, and this does not require labeled data.

Our approach makes two major assumptions about the nature of the relationship between the input data, which in this case is a window of speech coefficients, and the labels, which are HMM states produced by a forced alignment using a pre-existing ASR model. First, we assume that the discrimination we want to perform is more directly related to the underlying causes of the data than to the individual elements of the data itself. Second, we assume that a good feature-vector representation of the underlying causes can be recovered from the input data by modeling its higher-order statistical structure.

It is easy to dream up artificial tasks in which our two assumptions are entirely wrong, but a purely discriminative machine learning approach, such as a support vector machine with a polynomial kernel, would still work very well. Suppose, for example, that the label assigned to a window of speech coefficients is simply the parity of two particular components of the binarized data. Our approach would almost certainly fail because this information is unlikely to be preserved in the high-level features and, even if it is preserved, it will be in a much more complex form than the simple form it had in the data. Our claim is that, because of the highly structured way in which speech is generated, this artificial task is exactly what ASR is not like, and machine learning methods that do not make use of the huge difference between these two tasks have no long term future in ASR.

II. Learning a Multilayer Generative Model

There are two very different ways to understand our approach to learning a multi-layer generative model of a window of speech coefficients. In the *directed* view, we fit a multilayer generative model that has infinitely many layers of latent variables, but uses weight sharing among the higher layers to keep the number of parameters under control. In the *undirected*, "energy-based" view, we fit a relatively simple type of learning module that only has one layer of latent variables, but then we treat the activities of the latent variables as data and fit the same type of module again to this new "data." This can be repeated as many times as we like to learn as many layers of latent variables as we desire.

[1]It is impossible for a discriminatively trained system to produce better estimates of the posterior probability ratio of two classes than the ratio of the probabilities of generating the data from a mixture of two, class-specific, generative models if these models and their mixing proportions are correct.

A. **Undirected** *View*

The simple learning module used in the undirected view is called a "Restricted Boltzmann Machine" (RBM). It is a bipartite graph in which visible units that represent observations are connected to hidden units that learn to represent features using undirected weighted connections. An RBM is restricted in the sense that there are no visible-visible or hidden-hidden connections. In the simplest type of RBM, the binary RBM, both the hidden and visible units are binary and stochastic. To deal with real-valued input data, we use a Gaussian–Bernoulli RBM in which the hidden units are binary but the input units are linear with Gaussian noise. We will explain the Gaussian–Bernoulli RBM later after first explaining the simpler binary RBM.

In a binary RBM, the weights on the connections and the biases of the individual units define a probability distribution over the joint states of the visible and hidden units via an energy function. The energy of a joint configuration is

$$E(\mathbf{v},\mathbf{h}|\theta) = -\sum_{i=1}^{\mathcal{V}}\sum_{j=1}^{\mathcal{H}} w_{ij}v_ih_j - \sum_{i=1}^{\mathcal{V}} b_iv_i - \sum_{j=1}^{\mathcal{H}} a_jh_j \quad (1)$$

where $\theta = (\mathbf{w},\mathbf{b},\mathbf{a})$ and w_{ij} represents the symmetric interaction term between visible unit i and hidden unit j while b_i and a_j are their bias terms. $\mathcal{V}$ and $\mathcal{H}$ are the numbers of visible and hidden units.

The probability that an RBM assigns to a visible vector $\mathbf{v}$ is

$$p(\mathbf{v}|\theta) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u}}\sum_{\mathbf{h}} e^{-E(\mathbf{u},\mathbf{h})}}. \quad (2)$$

Since there are no hidden-hidden connections, the conditional distribution $p(\mathbf{h}|\mathbf{v},\theta)$ is factorial and is given by

$$p(h_j = 1|\mathbf{v},\theta) = \sigma\left(a_j + \sum_{i=1}^{\mathcal{V}} w_{ij}v_i\right) \quad (3)$$

where $\sigma(x) = (1+e^{-x})^{-1}$. Similarly, since there are no visible-visible connections, the conditional distribution $p(\mathbf{v}|\mathbf{h},\theta)$ is factorial and is given by

$$p(v_i = 1|\mathbf{h},\theta) = s\sigma\left(b_i + \sum_{j=1}^{\mathcal{H}} w_{ij}h_j\right) \quad (4)$$

Exact maximum-likelihood learning is infeasible in a large RBM because it is exponentially expensive to compute the derivative of the log probability of the training data. Nevertheless, RBMs have an efficient *approximate* training procedure called "contrastive divergence" [10] which makes them suitable as building blocks for learning DBNs. We repeatedly update each weight w_{ij} using the difference between two measured, pairwise correlations

$$\Delta w_{ij} \propto \langle v_ih_j\rangle_{\text{data}} - \langle v_ih_j\rangle_{\text{reconstruction}}. \quad (5)$$

The first term on the right-hand side (RHS) of (5) is the measured frequency with which visible unit i and hidden unit j are on together when the visible vectors are samples from the training set and the states of the hidden units are determined by (3). The second term is the measured frequency with which i and j are both on when the visible vectors are "reconstructions" of the data vectors and the states of the hidden units are determined by applying (3) to the reconstructions. Reconstructions are produced by applying (4) to the hidden states that were computed from the data when computing the first term on the RHS of (5).

The contrastive divergence learning rule does not follow the maximum-likelihood gradient. Understanding why it works at all is much easier using the directed view, so we defer the explanation to the next section. After learning the weights in an RBM module, we use the states of the hidden units, when driven by real data, as the data for training another module of the same kind. This process can be repeated to learn as many layers of features as we desire. Again, understanding why this greedy approach works is much easier using the directed view.

For Gaussian–Bernoulli RBMs[2] the energy of a joint configuration is

$$E(\mathbf{v},\mathbf{h}|\theta) = \sum_{i=1}^{\mathcal{V}} \frac{(v_i - b_i)^2}{2} - \sum_{i=1}^{\mathcal{V}}\sum_{j=1}^{\mathcal{H}} w_{ij}v_ih_j - \sum_{j=1}^{\mathcal{H}} a_jh_j. \quad (6)$$

Since there are no visible-visible connections, the conditional distribution $p(\mathbf{v}|\mathbf{h},\theta)$ is factorial and is given by

$$p(v_i|\mathbf{h},\theta) = \mathcal{N}\left(b_i + \sum_{j=1}^{\mathcal{H}} w_{ij}h_j, 1\right) \quad (7)$$

where $\mathcal{N}(\mu, V)$ is a Gaussian with mean μ and variance V. Apart from these differences, the inference and learning rules for a Gaussian–Bernoulli RBM are the same as for a binary RBM, though the learning rate needs to be smaller.

B. **Directed** *View*

In the undirected view, it is easy to say what we do, but hard to justify it. In the alternative "directed" view, the learning algorithm is more complicated to explain, but much easier to justify [11].

Consider a sigmoid belief net [12] that consists of multiple layers of binary stochastic units as shown in Fig. 1. The higher "hidden" layers represent binary features and the bottom, "visible," layer represents a binary data vector (we will show how to handle real-valued data later). Generating data from the model is easy. First, binary states are chosen for the top layer of hidden units using their biases to determine the log odds of choosing 1 or 0. Given the binary states of the units in layer k, binary states are chosen in parallel for all of the units in layer $k-1$ by applying the logistic sigmoid function $\sigma(x) = (1+e^{-x})^{-1}$ to the total input received from the layer above plus the unit's own bias

$$p\left(h_i^{(k-1)} = 1|\mathbf{h}^{(k)},\mathbf{W}^{(k)}\right) = \sigma\left(b_i^{(k-1)} + \sum_j w_{ij}^{(k)}h_j^{(k)}\right) \quad (8)$$

where $\mathbf{h}^{(k)}$ is the vector of binary states for layer k and $h_j^{(k)}$ is its j^{th} element. $\mathbf{W}^{(k)}$ is the matrix of weights from layer k to layer $k-1$, $w_{ij}^{(k)}$ is an element of that matrix, and $b_j^{(k)}$ is the

[2]To keep the equation simple, we assume that the Gaussian noise level of all the visible units is fixed at 1. We also normalize the input data to have a fixed variance of 1 for each component over the whole training set.

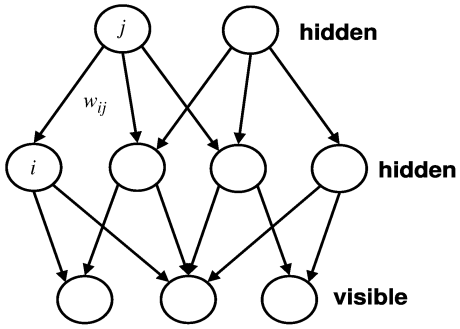

Fig. 1. Multi-layer sigmoid belief net composed of stochastic binary units.

bias of unit j in layer k. The vector of states of the visible units, $\mathbf{v}$, is also called $\mathbf{h}^{(0)}$.

Now consider the problem of adjusting the weights on the top-down connections so that the model is more likely to generate the binary training data on its visible units. Performing gradient ascent in the expected log probability of generating the training data is very simple if we can get unbiased samples of the hidden states from their posterior distribution given an observed data vector

$$\Delta w_{ij} \propto \left\langle h_j^{(k)} \left(h_i^{(k-1)} - p\left(h_i^{(k-1)} = 1 | \mathbf{h}^{(k)}, \mathbf{W}^{(k)} \right) \right) \right\rangle \quad (9)$$

where the angle brackets denote an expectation over the training data. If we want to ensure that every weight update increases the log probability of the data, we need to use a very small learning rate and we need to average over many samples from the posterior. In practice, it is much more efficient to use a larger learning rate on small mini-batches of the data.

Unfortunately, getting unbiased samples from the exponentially large posterior appears to be intractable for all but the smallest models.[3] Consider, for example, the posterior in the first hidden layer. This posterior is the normalized product of a complicated, non-factorial prior created by the layers above and a complicated non-factorial "likelihood term" created by the observed states of the visible units. When generating data from the model, the units in the first hidden layer are, by definition, conditionally independent given the states of the units in the layer above. When inferring the posterior, however, they are not conditionally independent given the states of the units in the layer below, even with a uniform prior, due to the phenomenon of "explaining away" [13].

Approximate samples from the posterior can be obtained by using a slow Markov chain Monte Carlo method [12] or a fast but crude approximation [14]. There is, however, one very special form of sigmoid belief net in which sampling from the posterior distribution in every hidden layer is just as easy as generating data from the model. In fact, inference and generation are the same process, but running in opposite directions.

1) Learning With Tied Weights: Consider a sigmoid belief net with an infinite number of layers and with tied symmetric weights between layers as shown in Fig. 2. In this net, the posterior in the first hidden layer is factorial: the hidden units are independent given the states of the visible units. This occurs because the correlations created by the prior coming from all of the layers above exactly cancel the anti-correlations in the likelihood term coming from the layer below [11]. Moreover, the factorial posterior can be computed by simply multiplying the visible vector by the transposed weight matrix and then applying the logistic function to each element

$$p\left(h_j^{(1)} = 1 | \mathbf{v}, \mathbf{W}\right) = \sigma\left(b_j^{(1)} + \sum_i w_{ij} v_i\right). \quad (10)$$

Notice that this simple operation computes the normalized product of the prior and the likelihood terms, not the likelihood term, which is considerably more complicated.

Once the posterior has been sampled for the first hidden layer, exactly the same process can be used for the next hidden layer. So inference is extremely easy in this special kind of network. Learning is a little more difficult because every copy of the tied weight matrix gets different derivatives. However, we know in advance that the expected derivatives will be zero for very high level layers. This is because the bottom-up inference process is really a Markov chain that eventually converges to its stationary distribution in the higher layers. When it is sampling from its stationary distribution, the current weights are perfect for explaining the samples, so, on average, there is no derivative. When the weights and biases are small, this Markov chain converges rapidly and we can approximate gradient ascent in the log likelihood quite well by just adding the derivatives for the first two layers [10].

The tied weights mean that the process of inferring $\mathbf{h}^{(2)}$ from $\mathbf{h}^{(1)}$ is the same as the process of generating $\mathbf{v}$ from $\mathbf{h}^{(1)}$. Consequently, $\mathbf{h}^{(2)}$ can be viewed as a noisy but unbiased estimate

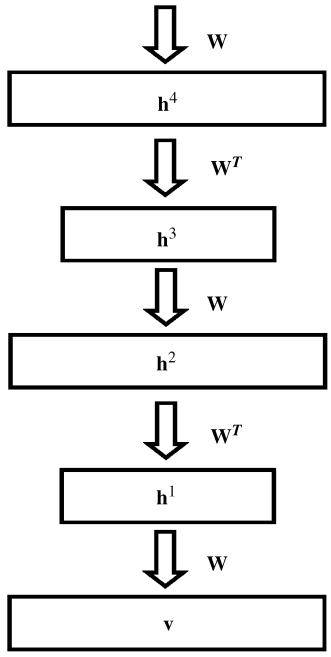

Fig. 2. Infinite sigmoid belief net with tied weights. Alternate layers must have the same number of units. The tied weights make inference much simpler in this net than in a general sigmoid belief net.

[3]In this respect, mixture models appear to be far superior. The exact posterior over the mixture components is easy to compute because it only has as many terms as the number of components. This computational simplicity, however, comes at a terrible price: The whole data vector must be generated by a single component. As we shall see later, it is possible to achieve an equally simple posterior while allowing multiple simultaneous causes.

of the probabilities for the visible units predicted by $\mathbf{h}^{(1)}$. Similarly $\mathbf{h}^{(3)}$ can be viewed as a noisy estimate of the probabilities for the units in the first hidden layer predicted by $\mathbf{h}^{(2)}$. We can use these two facts and (9) to get an unbiased estimate of the sum of the derivatives for the first two layers of weights. This gives the following learning rule which is known as "contrastive divergence" [10]:

$$\begin{aligned}\Delta w_{ij} &\propto \left\langle h_j^{(1)}\left(v_i - h_i^{(2)}\right) + h_i^{(2)}\left(h_j^{(1)} - h_j^{(3)}\right)\right\rangle \\ &\propto \left\langle v_i h_j^{(1)}\right\rangle - \left\langle h_i^{(2)} h_j^{(3)}\right\rangle \end{aligned} \tag{11}$$

where the angle brackets denote expectations over the training data (or a representative mini-batch).

As the weights and biases grow, it makes sense to add the derivatives for more layers [15] and for learning really good generative models this is essential [16]. For the purpose of creating sensible feature detectors, however, even a rather poorly tuned generative model is good enough, and the learning rule in (11) is sufficient even when the weights get quite large. The approximate maximum likelihood derivatives produced by this learning rule become highly biased, but they are cheap to compute and they also have very low variance [17] which is important for allowing a high learning rate when the derivatives are estimated from small mini-batches of data. These issues are discussed further in [18].

2) Learning Different Weights in Each Layer: Now that we have efficient inference and learning procedures for an infinite sigmoid belief net with tied weights, the next step is to make the generative model more powerful by allowing different weights in different layers. First, we learn with all of the weight matrices tied together. Then we untie the bottom weight matrix from the other matrices and freeze the values of its weights. This frozen matrix is now called $W^{(1)}$. Then, keeping all the remaining matrices tied together, we continue to learn the higher matrices, treating the inferred state vector $\mathbf{h}^{(1)}$ in just the same way as we previously treated $\mathbf{v}$. This involves first inferring $\mathbf{h}^{(1)}$ from $\mathbf{v}$ by using $\left(W^{(1)}\right)^T$ and then inferring $\mathbf{h}^{(2)}$, $\mathbf{h}^{(3)}$, and $\mathbf{h}^{(4)}$ in a similar bottom-up manner using W or W^T. The inference for the higher hidden layers produces unbiased samples given $\mathbf{h}^{(1)}$, but the simple inference method no longer gives an exactly unbiased sample for $\mathbf{h}^{(1)}$ because the higher weight matrices are no longer equal to $W^{(1)}$ so they no longer create a prior that exactly cancels the correlations in the likelihood term. However, the posterior for $\mathbf{h}^{(1)}$ is still approximately factorial and it can be proved that if we continue to infer $\mathbf{h}^{(1)}$ as if the higher matrices had not changed, then learning improves a variational lower bound on the log probability of the training data [4].[4]

We can repeat the process of freezing and untying the lowest copy of the currently tied weight matrices as many times as we like, so we can learn as many layers of features as we desire. When we have learned K layers of features, we are left with a directed generative model called a "deep belief net" (DBN) that has K different weight matrices between the lower layers and an infinite number of higher layers that all use the K^{th} weight matrix or its transpose. We then simply throw away all layers above the K^{th} and add a final "softmax" layer of label units representing HMM states. We also jettison the probabilistic interpretation that was used to justify the generative learning, and we treat the whole system as a feedforward, deterministic neural network. This network is then discriminatively fine-tuned by using backpropagation to maximize the log probability of the correct HMM state.[5]

For the softmax final layer, the probability of label l given the real-valued activations of the final layer of features (which we call $\mathbf{h}^{(K)}$ even though they are no longer binary sampled values) is defined to be

$$p\left(l|\mathbf{h}^{(K)}\right) = \frac{\exp\left(b_l + \sum_i h_i^{(K)} w_{il}\right)}{\sum_m \exp\left(b_m + \sum_i h_i^{(K)} w_{im}\right)} \tag{12}$$

where b_l is the bias of the label and w_{il} is the weight from hidden unit i in layer K to label l. The discriminative training must learn the weights from the last layer of features to the label units, but it does not need to create any new feature detectors. It simply fine-tunes existing feature detectors that were discovered by the unsupervised "pre-training."

To model real values in the visible layer, we simply replace the binary unit by linear units with Gaussian noise that has a variance of 1. This does not change $p(\mathbf{h}^{(1)}|\mathbf{v})$ and the distribution for visible unit i given $\mathbf{h}^{(1)}$ is a Gaussian with unit variance and mean μ_i given by

$$\mu_i = b_i^{(0)} + \sum_j w_{ij}^{(1)} h_j^{(1)}. \tag{13}$$

This type of generative pre-training followed by discriminative fine-tuning has been used successfully for hand-written character recognition [10], [11], [19], dimensionality reduction [20], 3-D object recognition [21], [22], extracting road maps from cluttered aerial images [23], information retrieval [24], [25] and machine transliteration [26]. As we shall see, it is also very good for phone recognition.

III. Using Deep Belief Nets for Phone Recognition

In order to apply DBNs with fixed input and output dimensionality to phone recognition, we use a context window of n successive frames of speech coefficients to set the states of the visible units of the lowest layer of the DBN. Once it has been pre-trained as a generative model, the resulting feedfoward neural network is discriminatively trained to output a probability distribution over the possible labels of the central frame. To generate phone sequences, the sequence of predicted probability distributions over the possible labels for each frame is fed into a standard Viterbi decoder.

Strictly speaking, since the HMM implements a prior over states, we should divide out the prior from the posterior distribution over HMM states produced by the DBN, although for our particular task it made no difference.

[4]The proof assumes that each layer of weights is learned by following the maximum likelihood derivative, rather than using the contrastive divergence approximation.

[5]In a convenient abuse of the correct terminology, we sometimes use "DBN" to refer to a feedforward neural network that was initialized using a generatively trained deep belief net, even though the feedforward neural network is clearly very different from a belief net.

IV. Experimental Setup

A. TIMIT Corpus

Phone recognition experiments were performed on the TIMIT corpus.[6] We used the 462 speaker training set and removed all SA records (i.e., identical sentences for all speakers in the database) since they could bias the results. A separate development set of 50 speakers was used for tuning all of the meta parameters, such as the number of layers and the size of each layer. Results are reported using the 24-speaker core test set, which excludes the development set.

The speech was analyzed using a 25-ms Hamming window with a 10-ms fixed frame rate. In most of the experiments, we represented the speech using 12th-order MFCCs and energy, along with their first and second temporal derivatives. For some experiments, we used a Fourier-transform-based filter-bank with 40 coefficients distributed on a mel-scale (and energy) together with their first and second temporal derivatives.

The data were normalized so that, averaged over the training cases, each coefficient or first derivative or second derivative had zero mean and unit variance. We used 183 target class labels (i.e., three states for each one of the 61 phones). After decoding, the 61 phone classes were mapped to a set of 39 classes as in [27] for scoring. All of our experiments used a bigram language model over phones, estimated from the training set.

B. Computational Setup

Training DBNs of the sizes used in this paper is quite computationally expensive. Training was accelerated by exploiting graphics processors, in particular GPUs in a NVIDIA Tesla S1070 system, using the CUDAMAT library [28]. A single pass over the entire training set (an "epoch") for a model with five hidden layers and 2048 units per layer took about 6 minutes during pre-training of the topmost layer and about 11 minutes during fine-tuning the whole network with backpropagation. A single GPU learns about 20 times faster than a single 2.66-GHz Xeon core.

V. Experiments

For all experiments, we fixed the parameters for the Viterbi decoder. Specifically, we used a zero word insertion probability and a language model scale factor of 1.0.

All DBNs were pre-trained with a fixed recipe using stochastic gradient decent with a mini-batch size of 128 training cases. For Gaussian-binary RBMs, we ran 225 epochs with a fixed learning rate of 0.002 while for binary-binary RBMs we used 75 epochs with a learning rate of 0.02.

The theory used to justify the pre-training algorithm assumes that when the states of the visible units are reconstructed from the inferred binary activities in the first hidden layer, they are reconstructed stochastically. To reduce noise in the learning, we actually reconstructed them deterministically and used the real values (see [18] for more details).

For fine-tuning, we used stochastic gradient descent with the same mini-batch size as in pre-training. The learning rate started at 0.1. At the end of each epoch, if the substitution error on the development set increased, the weights were returned to their values at the beginning of the epoch and the learning rate was halved. This continued until the learning rate fell below 0.001.

During both pre-training and fine-tuning, a small weight-cost of 0.0002 was used and the learning was accelerated by using a momentum of 0.9 (except for the first epoch of fine-tuning which did not use momentum). [18] gives a detailed explanation of weight-cost and momentum and sensible ways to set them.

Figs. 3 and 4 show the effect of varying the size of each hidden layer and the number of hidden layers. For simplicity, we used the same size for every hidden layer in a network. For these comparisons, the number of input frames was fixed at 11.

The main trend visible in Figs. 3 and 4 is that adding more hidden layers gives better performance, though the gain diminishes as the number of layers increases. Using more hidden units per layer also improves performance when the number of hidden layers is less than 4, but with more hidden layers the number of units has little effect provided it is at least 1024. The advantage of using a deep architecture is clear if we consider the best way to use a total of 2048 hidden units: It is better to use two layers of 1024 or four layers of 512 than one layer of 2048.

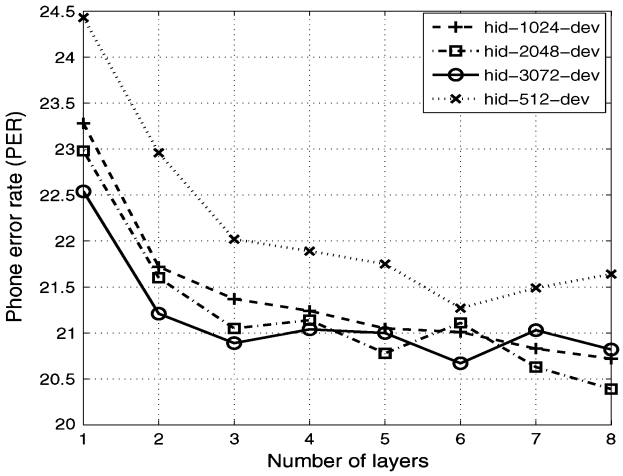

Fig. 3. Phone error rate on the development set as a function of the number of layers and size of each layer, using 11 input frames.

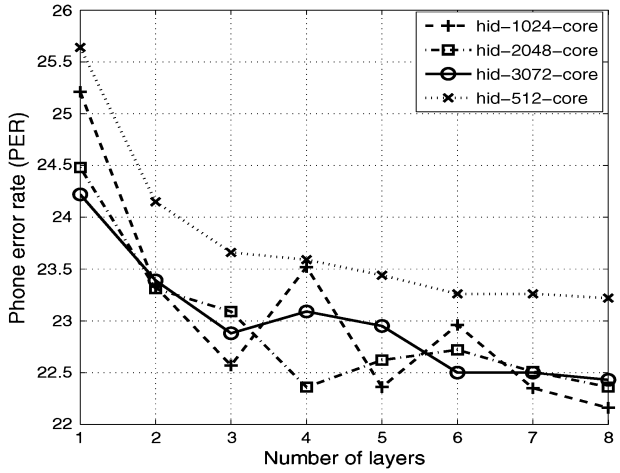

Fig. 4. Phone error rate on the core test set as a function of the number of layers and size of each layer, using 11 input frames.

[6] http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1.

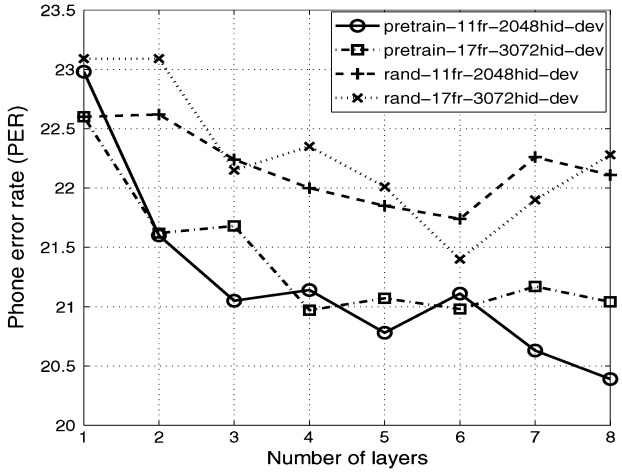

Fig. 5. Phone error rate on the development set as a function of the number of hidden layers using randomly initialized and pretrained networks.

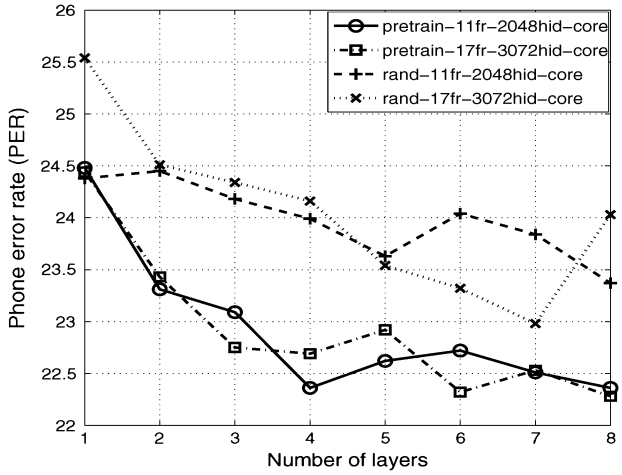

Fig. 6. Phone error rate on the core test set as a function of the number of hidden layers using randomly initialized and pretrained networks.

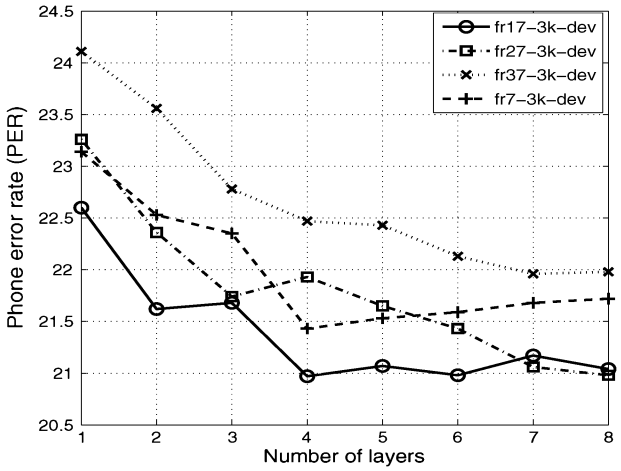

Fig. 7. Phone error rate on the development set as a function of the number of layers, using 3072 hidden units per layer.

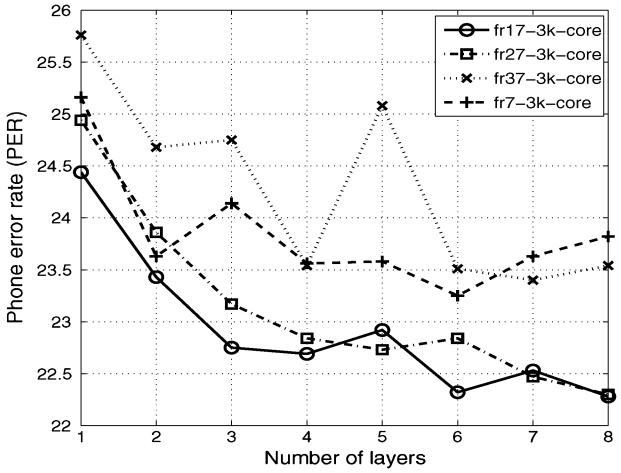

Fig. 8. Phone error rate on the core test set as a function of the number of layers, using 3072 hidden units per layer.

To benefit from having many hidden layers, it is necessary to do generative pre-training. With a single hidden layer of 2048 units, generative pre-training gives a phone error rate of 24.5% and exactly the same fine-tuning algorithm started from small random weights gives 24.4%. So generative pre-training does not help. Adding a second hidden layer causes a larger proportional increase in the number of trainable parameters than adding a third hidden layer because the input and output layers are much smaller than the hidden layers and because adjacent hidden layers are fully connected. This large increase in capacity makes the model far more flexible, but it also makes it overfit far more easily. Figs. 5 and 6 show that for networks that are not pre-trained (but still use early stopping), these two effects apparently cancel out, whereas for pre-trained networks there is less overfitting so extra layers help. Although the advantage of pre-training is not as large as for some other tasks [20], [29], it is still required to gain an advantage from using extra hidden layers.

Fixing the number of hidden units per layer to 3072 and varying the number of frames in the input window (Figs. 7 and 8 and also Figs. 3 and 4) shows that the best performance on the development set is achieved using 11, 17, or 27 frames and this is true whatever the number of hidden layers. Much smaller (7 frames) and much bigger (37 frames) windows give significantly worse performance. The range from 110 ms to 270 ms covers the average size of phones or syllables. Smaller input windows miss important discriminative information in the context, while networks with larger windows are probably getting distracted by the almost irrelevant information far from the center of the window. The TRAP [7] architecture successfully uses 1-s-long windows, but it dedicates separate networks to model different parts of the spectrum which simplifies the learning task. Larger windows would probably work better using triphone targets which provide the network with more information about the context and make the peripheral frames more relevant.

Since all but the output layer weights are pre-trained, it could be helpful to introduce a "bottleneck" at the last layer of the DBN to combat overfitting by reducing the number of weights that are not pre-trained. A bottleneck layer followed by a softmax is exactly equivalent to using a distributed output code for each class and then making the class probabilities

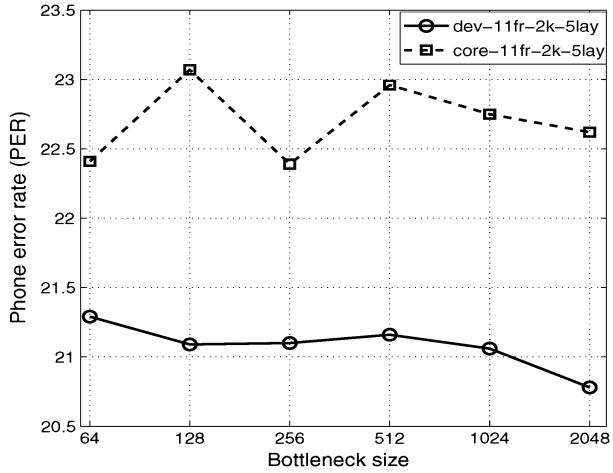

Fig. 9. Effect of the size of the bottleneck on the phone error rate for a typical network with 11 input frames and five hidden layers of 2048 units per layer (except for the last hidden layer).

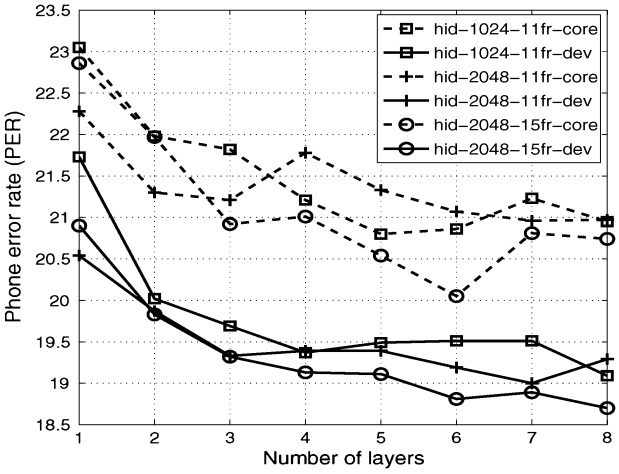

Fig. 10. Phone error rates as a function of the number of layers, using a context window of filter bank coefficients as input features to the DBN.

proportional to the exponentiated squared difference between the code for a class and the activity pattern in the bottleneck layer [30]. Fig. 9 shows that having a bottleneck layer does not actually improve PER for a typical network with five hidden layers of 2048 units and an input window of 11 frames of MFCCs.

In all of the previous experiments MFCCs were used as the input representation. MFCCs attempt to reduce the dimensionality of the input by eliminating variations that are irrelevant for recognition and spoon-feeding the recognizer with a modest-sized input representation that is designed to make recognition easy. With a more powerful learning procedure, better recognition performance can be achieved by using a less pre-processed input representation consisting of 40 filter-bank coefficients augmented with temporal differences and differences of differences. Fig. 10 shows that a DBN is capable of making good use of the more detailed information available in this larger input representation. For the DBN system that performed best on the development set, the phone error rate on the TIMIT core test set was 20.7%.

TABLE I
REPORTED RESULTS ON TIMIT CORE TEST SET

Method	PER
Stochastic Segmental Models [31]	36%
Conditional Random Field [32]	34.8%
Large-Margin GMM [33]	33%
CD-HMM [34]	27.3%
Augmented conditional Random Fields [34]	26.6%
Recurrent Neural Nets [35]	26.1%
Bayesian Triphone HMM [36]	25.6%
Monophone HTMs [37]	24.8%
Heterogeneous Classifiers [38]	24.4%
Triphone HMMs discriminatively trained w/ BMMI [39]	22.7%
Monophone Deep Belief Networks(DBNs) (this work)	**20.7%**

Table I compares the best performing DBN model with previously reported results on the TIMIT core test set.[7]

VI. CONCLUSION AND FUTURE WORK

So far as we know, the work reported in this paper was the first application to acoustic modeling of neural networks in which multiple layers of features are generatively pre-trained. Since then, our approach has been extended to explicitly model the covariance structure of the input features [40]. It has also been used to jointly train the acoustic and language models using the full utterance rather than a local window of frames [41]. It has also been applied to a large vocabulary task [42] where the competing GMM approach uses a very large number of components. In this latter task it gives a very large advantage relative to the GMM.

We are currently exploring alternatives input representations that allow deep neural networks to see more of the relevant information in the sound-wave, such as very precise coincidences of onset times in different frequency bands. We are also exploring ways of using recurrent neural networks to greatly increase the amount of detailed information about the past that can be carried forward to help in the interpretation of the future.

ACKNOWLEDGMENT

The authors would like to thank J. Glass, J. Bridle, L. Deng and G. Penn for helpful discussions and the two anonymous reviewers for improving the paper.

REFERENCES

[1] A. Jansen and P. Niyogi, "A hierarchical point process model for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2008, pp. 4093–4096.
[2] L. Deng, D. Yu, and A. Acero, "Structured speech modeling," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 14, no. 5, pp. 1492–1504, Sep. 2006.
[3] Y. Bengio, "Artificial neural networks and their application to sequence recognition," Ph.D. dissertation, Dept. of Comput. Sci., McGill Univ., Montreal, QC, Canada, 1991.
[4] H. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*. Norwell, MA: Kluwer, 1993.
[5] N. Morgan, Q. Zhu, A. Stolcke, K. Sonmez, S. Sivadas, T. Shinozaki, M. Ostendorf, P. Jain, H. Hermansky, D. Ellis, G. Doddington, B. Chen, O. Cretin, H. Bourlard, and M. Athineos, "Pushing the envelope—Aside," *IEEE Signal Process. Mag.*, vol. 22, no. 5, pp. 81–88, Sep. 2005.

[7]In [8] a PER of 21.48% is reported on the **complete** test set of TIMIT. The speech units used are not the same as the standard TIMIT definitions and their method would very probably give a worse result using the standard speech units.

[6] J. Allen, "How do humans process and recognize speech?," *IEEE Trans. Speech Audio Process.*, vol. 2, no. 4, pp. 567–577, Oct. 1994.
[7] H. Hermansky and S. Sharma, "Traps—Classifiers of temporal patterns," in *Proc. Int. Conf. Spoken Lang. Process. (ICSLP)*, 1998, pp. 1003–1006.
[8] P. Schwarz, P. Matejka, and J. Cernocky, "Hierarchical structures of neural networks for phoneme recognition," in *Proc. ICASSP*, 2006, pp. 325–328.
[9] P Brown, "The acoustic-modeling problem in automatic speech recognition," Ph.D. dissertation, Carnegie-Mellon Univ., Pittsburgh, PA, 1987.
[10] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1711–1800, 2002.
[11] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.
[12] R. Neal, "Connectionist learning of belief networks," *Artif. Intell.*, vol. 56, no. 1, pp. 71–113, 1992.
[13] J. Pearl, *Probabilistic Inference in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.
[14] G. E. Hinton, P. Dayan, B. J. Frey, and R. Neal, "The wake-sleep algorithm for self-organizing neural networks," *Science*, vol. 268, pp. 1158–1161, 1995.
[15] M. A. Carreira-Perpignan and G. E. Hinton, "On contrastive divergence learning," in *Proc. Artif. Intell. Statist.*, 2005.
[16] T. Tieleman and G. Hinton, "Using fast weights to improve persistent contrastive divergence," in *Proc. 26th Int. Conf. Mach. Learn.*, New York, 2009, pp. 1033–1040, ACM New York.
[17] C. Williams and F. Agakov, "An analysis of contrastive divergence learning in Gaussian Boltzmann machines," Inst. for Adaptive and Neural Computation, Univ. of Edinburgh, Edinburgh, U.K., Tech. Rep. EDI-INF-RR-0120, 2002.
[18] G. E. Hinton, A practical guide to training restricted Boltzmann Machines Machine Learning Group, Univ. of Toronto, Toronto, ON, Canada, Tech. Rep. 2010-003, 2010.
[19] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems 19*, S. B. , J. Platt, and T. Hoffman, Eds. Cambridge, MA: MIT Press, 2007.
[20] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Sci.*, vol. 313, pp. 504–507, 2006.
[21] V. Nair and G. E. Hinton, "3-d object recognition with deep belief nets," *Advances in Neural Information Processing Systems*, vol. 22, pp. 1339–1347, 2009.
[22] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.Sc. thesis, Dept. of Comput. Sci., Univ. of Toronto, Toronto, ON, Canada, 2009.
[23] V. Mnih and G. E. Hinton, "Learning to detect roads in high-resolution aerial images," in *Proc. Eur. Conf. Comput. Vis.*, 2010.
[24] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[25] R. Salakhutdinov and G. E. Hinton, "Semantic hashing," *Int. J. Approximate Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.
[26] T. Deselaers, S. Hasan, O. Bender, and H. Ney, "A deep learning approach to machine transliteration," in *Proc. EACL Workshop Statist. Mach. Translat.*, 2009, pp. 233–241.
[27] K. F. Lee and H. W. Hon, "Speaker-independent phone recognition using hidden markov models," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 37, no. 11, pp. 1641–1648, Nov. 1989.
[28] V. Mnih, Cudamat: A CUDA-based matrix class for Python Dept. of Comput. Sci., Univ. of Toronto, Toronto, ON, Canada, Tech. Rep. UTML TR 2009-004, Nov. 2009.
[29] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. Int. Conf. Mach. Learn.*, 2007, pp. 473–480.
[30] I. Sutskever and G. E. Hinton, "Using matrices to model symbolic relationships," *Adv. in Neural Inf. Process. Syst.*, vol. 21, pp. 1593–1600, 2008.
[31] V. V. Digalakis, M. Ostendorf, and J. R. Rohlicek, "Fast algorithms for phone classification and recognition using segment-based models," *IEEE Trans. Signal Process.*, vol. 40, no. 12, pp. 2885–2896, Dec. 1992.
[32] J. Moris and E. Fosler-Lussier, "Combining phonetic attributes using conditional random fields," in *Proc. Interspeech*, 2006, pp. 597–600.
[33] F. Sha and L. Saul, "Large margin Gaussian mixture modeling for phonetic classification and recognition," in *Proc. ICASSP*, 2006, pp. 265–268.
[34] Y. Hifny and S. Renals, "Speech recognition using augmented conditional random fields," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 17, no. 2, pp. 354–365, Feb. 2009.
[35] A. Robinson, "An application to recurrent nets to phone probability estimation," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 298–305, Mar. 1994.
[36] J. Ming and F. J. Smith, "Improved phone recognition using Bayesian triphone models," in *Proc. ICASSP*, 1998, pp. 409–412.
[37] L. Deng and D. Yu, "Use of differential cepstra as acoustic features in hidden trajectory modelling for phonetic recognition," in *Proc. ICASSP*, 2007, pp. 445–448.
[38] A. Halberstadt and J. Glass, "Heterogeneous measurements and multiple classifiers for speech recognition," in *Proc. ICSLP*, 1998.
[39] T. N. Sainath, B. Ramabhadran, and M. Picheny, "An exploration of large vocabulary tools for small vocabulary phonetic recognition," in *Proc. IEEE Autom. Speech Recognition Understanding Workshop*, 2009.
[40] G. E. Dahl, M. Ranzato, A. Mohamed, and G. E. Hinton, "Phone recognition with the mean-covariance restricted Boltzmann machine," *Adv. Neural Inf. Process. Syst. 23*, pp. 469–477, 2010.
[41] A. Mohamed, D. Yu, and L. Deng, "Investigation of full-sequence training of deep belief networks for speech recognition," in *Proc. Interspeech'10*, 2010.
[42] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large vocabulary speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, accepted for publication.

Abdel-rahman Mohamed received the B.Sc. and M.Sc. degrees from the Electronics and Communication Engineering Department, Cairo University, Giza, Egypt, in 2004 and 2007, respectively. He is currently pursuing the Ph.D. degree at the University of Toronto, Toronto, ON, Canada.

In 2004, he was with the Speech Research Group, RDI Company, Egypt. Then he joined the ESAT-PSI Speech Group, KULeuven. His research focus is in developing machine learning techniques for automatic speech recognition.

George E. Dahl received the B.A. degree in computer science, with highest honors, from Swarthmore College, Swarthmore, PA, and the M.Sc. degree from the University of Toronto, Toronto, ON, Canada, where he is currently pursuing the Ph.D. degree, with a research focus in statistical machine learning.

His current main research interest is in training models that learn many levels of rich, distributed representations from large quantities of perceptual and linguistic data.

Geoffrey Hinton received the Ph.D. degree in artificial intelligence from the University of Edinburgh, Edinburgh, U.K., in 1978.

He spent five years as a faculty member at Carnegie-Mellon University, Pittsburgh, PA, and he is currently the Raymond Reiter Distinguished Professor of Artificial Intelligence at the University of Toronto, Toronto, ON, Canada, where he directs the program on Neural Computation and Adaptive Perception funded by the Canadian Institute for Advanced Research.

Prof. Hinton is a fellow of the Royal Society and an honorary foreign member of the American Academy of Arts and Sciences. He was one of the researchers who introduced the back-propagation algorithm. His other contributions include Boltzmann machines, distributed representations, time-delay neural nets, mixtures of experts, variational learning, and deep belief nets.