A GAUSSIAN MIXTURE MODEL LAYER JOINTLY OPTIMIZED WITH DISCRIMINATIVE FEATURES WITHIN A DEEP NEURAL NETWORK ARCHITECTURE

Ehsan Variani[1*], *Erik McDermott*[2], *Georg Heigold*[2]

[1]Johns Hopkins Univ., Baltimore, MD USA
[2]Google Inc., USA

variani@jhu.edu {erikmcd,heigold}@google.com

ABSTRACT

This article proposes and evaluates a Gaussian Mixture Model (GMM) represented as the last layer of a Deep Neural Network (DNN) architecture and jointly optimized with all previous layers using Asynchronous Stochastic Gradient Descent (ASGD). The resulting "Deep GMM" architecture was investigated with special attention to the following issues: (1) The extent to which joint optimization improves over separate optimization of the DNN-based feature extraction layers and the GMM layer; (2) The extent to which depth (measured in number of layers, for a matched total number of parameters) helps a deep generative model based on the GMM layer, compared to a vanilla DNN model; (3) Head-to-head performance of Deep GMM architectures vs. equivalent DNN architectures of comparable depth, using the same optimization criterion (frame-level Cross Entropy (CE)) and optimization method (ASGD); (4) Expanded possibilities for modeling offered by the Deep GMM generative model. The proposed Deep GMMs were found to yield Word Error Rates (WERs) competitive with state-of-the-art DNN systems, at the cost of pre-training using standard DNNs to initialize the Deep GMM feature extraction layers. An extension to Deep Subspace GMMs is described, resulting in additional gains.

Index Terms— Deep neural networks, feature extraction, classification.

1. INTRODUCTION

The recent gains in the field of ASR obtained from the use of DNNs rather than GMMs are typically attributed to the greater ability of DNNs to extract useful discriminative features automatically [1] [2]. In particular, [1] carefully examines the GMM vs. DNN picture as progressively more complex features are engineered into the GMM architecture. Eventually, the GMM results improve quite a bit compared to the no-engineering baseline, but still trail the DNN results. There is some evidence [3] that higher DNN layers represent features that are more invariant to speaker differences. These studies, among others, suggest that the fundamental breakthrough with DNNs derives from this superior feature extraction. However, there is no need to limit "deep" feature extraction to DNN architectures. Previous work proposed the general concept of *Discriminative Feature Extraction (DFE)*, in which both the feature extraction and classification modules are optimized jointly [4]. One can view the many successful DNN results as special cases of DFE, with the final DNN layer(s) being the classifier, and the preceding layers corresponding to the feature extractor. From this perspective, there is no reason to limit the classifier to be a DNN layer, itself closely related to a single Gaussian model with a globally pooled covariance matrix [5]

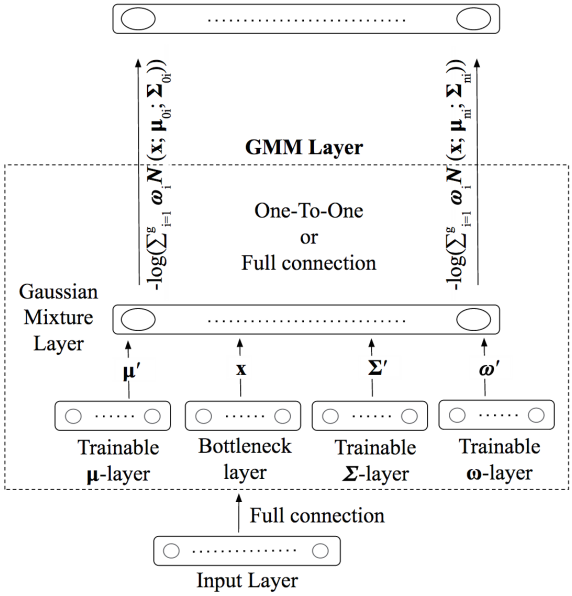

Fig. 1. The Gaussian Mixture Model (GMM) layer: n nodes with g Gaussian components per node, and associated μ, Σ & ω parameters.

[6]. This last observation makes the link to more structured classifiers based on generative models such as GMMs. Previous work [7] [8] [9] explored this possibility by separately training DNNs to generate "bottleneck" features which are fed into independently trained GMM systems. This offers the possibility of preserving familiar GMM techniques for optimization (such as use of the Expectation-Maximization (EM) algorithm) and adaptation, on top of the richer "deep" features from the DNN. These hybrid approaches, however, did not address the possibility of joint optimization of the DNN features together with the GMM backend. That is exactly what the Deep GMM proposed here aims to address. The benefits of previously investigated hybrid systems still apply, but are now amplified by a new capacity for joint optimization.

We note that a related "Deep GMM" was proposed in [10] in which successive layers of log-linear models are built on *top* of GMM-derived posteriors; this work did not evaluate joint optimization of the GMM parameters. Moreover [11] investigated the joint training of the bottleneck features with a sequence training objective, while the GMM parameters are separately trained using the EM algorithm. Finally, [12] proposed a deeply factored GMM, operating on original input features and not involving DNNs.

*Research conducted as an intern at Google, USA

2. A GMM LAYER FOR DEEP LEARNING

The GMM layer, shown in Fig. 1, embeds a set of Gaussian Mixture Models into a DNN. The layer is designed so that it can easily be imported to any part of the deep structure. Throughout this paper, the GMM layer is mainly used as the last layer. Each node in the *Gaussian Mixture sub-layer* has g Gaussian components. The number of GMM nodes in the layer is equal to the number of classes (or states). For each state s, the corresponding node in the GMM layer outputs the negative log likelihood $-log(p(x|s))$, $p(x|s)$ being produced by a mixture of Gaussians,

$$p(x|s) = \sum_{i=1}^{g} \omega_{si} \mathcal{N}(x, \mu_{si}, \Sigma_{si}). \quad (1)$$

Here μ_{si} and Σ_{si} represent the mean vector and covariance matrix of the i^{th} Gaussian within state s, and ω_{si} is the corresponding weight.

The GMM layer has four additional sub-layers. *A linear bottleneck sub-layer* of d nodes feeds its output to all nodes in the GMM layer as the input feature x (of dimensionality d). The notation *GMM(d, g)* is used for a GMM layer with given input dimensionality and number of Gaussians per state. Three parameter sub-layers, the μ-*layer*, Σ-*layer*, and ω-*layer* store real-valued parameters for each node, representing the standard GMM parameters, or transformed versions thereof. Each of these sub-layers has the same number of nodes as the GMM layer, to which they are connected one-to-one.

The actual model parameters are derived by transforming the stored values to respect the parameter constraints of a Gaussian Mixture distribution, detailed below. To avoid confusion between actual model parameters and stored values, the parameters in the trainable layers are referred to as μ', Σ' and ω'. Each node stores a supervector corresponding to the model parameters. For a GMM(d, g) layer, each node in the μ-layer stores the supervector $\mu'_s = [\mu'_{s1}, ..., \mu'_{sg}]$, where μ'_{si} is the vector stored for component i of state s. Similarly, Σ-layer and ω-layer store corresponding supervectors Σ'_s and ω'_s. In the forward pass, these parameters are transformed to the corresponding GMM parameters in the GMM layer. In the backward pass, the corresponding gradients are fed back to update the parameters.

Forward Pass. In the forward pass, the GMM layer outputs the negative log likelihood, $L(x, s) = -log(p(x|s))$, where $p(x|s)$ is the Gaussian Mixture distribution from Eq. (1), with parameters μ, Σ, and ω. To ensure that the parameters of the GMM layer obey the constraints of a Gaussian mixture, appropriate activation functions are applied within the GMM layer for each state s and component i:

- μ-layer: There is no constraint on μ values, so $\mu_{si} = \mu'_{si}$.
- Σ-layer: The parameters stored in this layer, Σ', correspond to the diagonal components of the covariance matrix. To impose the semi-positivity of these parameters, the exponential activation function was used, i.e. $\Sigma_{si} = \exp(\Sigma'_{si})$.
- ω-layer: To impose positivity and the sum-to-one property of the mixing weights, this layer stores real values but applies a *softmax* transform:

$$\omega_{si} = \frac{\exp(\omega'_{si})}{\sum_{i=1}^{g} \exp(\omega'_{si})}.$$

The input to the GMM layer is first fed to the bottleneck sub-layer. This layer is a linear layer which changes the dimensionality of input features and reduces the correlation between features. The latter is jointly enforced with the diagonal covariance matrix assumption. The bottleneck layer outputs a d-dimensional vector which will be fed to the GMM layer as input feature vector. The number of nodes in this layer can significantly affect the overall performance.

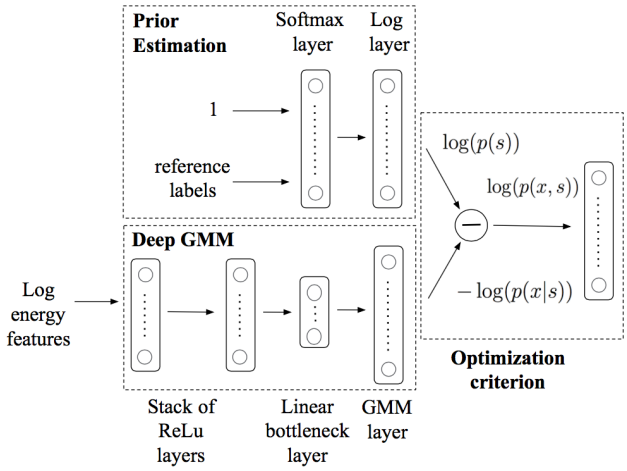

Fig. 2. GMM optimization within overall NN structure.

Backward Propagation of Gradients. The partial derivatives of each parameter back-propagated by the GMM to the rest of the network are as follows. The index i corresponds to the GMM component and index j denotes the dimension, so $i = 1, ..., g$ and $j = 1, ..., d$.

$$\begin{aligned}
\frac{\partial L(x,s)}{\partial \mu'_{sij}} &= \pi_i(x,s)\{\frac{x_j - \mu_{sij}}{{\sigma_{sij}}^2}\} \\
\frac{\partial L(x,s)}{\partial \sigma'_{sij}} &= \pi_i(x,s)\{(\frac{x_j - \mu_{sij}}{\sigma_{sij}})^2 - 1\} \\
\frac{\partial L(x,s)}{\partial \omega'_{si}} &= \pi_i(x,s) - \omega_{si} \\
\frac{\partial L(x,s)}{\partial x} &= -\sum_{i=1}^{g} \frac{\partial L(x,s)}{\partial \mu_{si}}
\end{aligned} \quad (2)$$

with

$$\pi_i(x,s) = \frac{\omega_{si}\mathcal{N}(x, \mu_{si}, \Sigma_{si})}{\sum_{k=1}^{g} \omega_{sk}\mathcal{N}(x, \mu_{sk}, \Sigma_{sk})} \quad (3)$$

which is the contribution of each Gaussian distribution in the mixture. Detailed derivation of the above equations can be found in [13, 14, 15, 16, 17].

3. TRAINING A NEURAL NETWORK WITH GMM LAYER

Fig. 2 shows the structure of the embedded GMM within a Neural Network (NN) architecture during optimization with Cross Entropy (CE) [2]. There are three modules in this figure. The top module estimates priors online during CE training. This is done by inputting a constant value of 1 and the reference frame label into the softmax. The logarithm of the estimated prior distribution, $log(p(s))$, is then calculated.

The GMM layer is used as the last layer of the network. If the GMM layer is used after the stack of hidden layers, the corresponding structure is referred to as *Deep GMM* (DGMM), otherwise it will be called *Shallow GMM*. The GMM layer outputs $-log(p(x|s))$ for

each frame of the data. By subtracting this value from the logarithm of the prior, the logarithm of the joint distribution can be formed:

$$log(p(x,s)) = log(p(s)) - (-log(p(x|s))). \quad (4)$$

The state posteriors for CE training can then be derived using

$$p(s|x) = \frac{\exp(\log(p(x,s)))}{\sum_s \exp(\log(p(x,s)))}. \quad (5)$$

In this paper all experiments used CE training. However, the proposed design allows the use of any optimization criterion, such as Maximum Likelihood (ML), CE, or Discriminative Sequence Training [18] [19]. (Use of ML to optimize the entire structure requires care, since the Jacobian of the bottleneck features has to be handled; this is not detailed here).

The trainable sub-layers are initialized as follows. The μ-layer values are initialized to random numbers derived from a normal distribution, $\mathcal{N}(0,1)$. All parameters in the Σ-layer are initialized with a constant value of 0 (corresponding to a transformed value of 1) and the parameters in the ω-layer are initialized with a uniform value of $1/g$. We considered the effect on training of each parameter separately. Our observation was that training all parameters jointly achieves the best performance, but requires more overall training steps than first training just the means and weights while fixing the variances, before then training all parameters together.

4. EXPERIMENTS

The main concept behind the DGMM model is the *joint* optimization of *feature extraction* and *classification*. Since GMMs primarily function on the classification side, the "depth" of the model proposed here primarily operates on the feature extraction side. These two are then jointly optimized in the DGMM structure. The experiments are designed to carefully examine the effect of each of these components. DNNs can also be seen as joint models with both deep structure and a final classification layer. Both jointly optimized GMMs and DNNs are compared in matched conditions for shallow and deep structures. Further on, the Subspace GMM (SGMM) model is investigated using the GMM layer.

Datasets. The experiments described here used two anonymized datasets: *Icelandic (is-is)*, consisting of 60 hours of read speech, simulating Voice Search queries, and *American English (en-us)*, consisting of 3000 hours of spontaneous speech from anonymized, human-transcribed Voice Search data. The test sets for each language are separate anonymized, human-transcribed Voice Search datasets of about 25 hours each.

The training examples are log energy features obtained from the concatenation of 26 frames, 20 left and 5 right. Frames consist of 40-dimensional log-energies extracted from a 25 ms window shifted over the audio samples in 10 ms increments. Each training step has a batch size of 200 examples. For both en-us and is-is, the DNN baseline uses 8 hidden layers, each with 2560 nodes. The is-is system uses 2432 context-dependent output states (resulting in about 55 M parameters in total) while the en-us system uses 13568 output states (for a total of about 85 M parameters). The DGMM model hidden layers are initialized using the baseline DNN model, unless explicitly mentioned otherwise. The rectified linear units were used for the hidden layers. ASGD training used 100 multi-core machines.

Effect of Joint Optimization. In this experiment we evaluated three configurations on is-is. The first configuration is the baseline DNN, trained to convergence. The second system is a DGMM with a GMM(272,5) layer and 7 hidden layers. The hidden layers are initialized using the corresponding baseline DNN model and then kept frozen during training. This simulates the *separate optimization* approach where feature extraction is done *independently* of classification. The third system, the *jointly optimized* DGMM, uses the same structure as the second one, with the difference that all the layers are trained together. Both DGMM systems have about 49 M parameters in all, but only 7.3 M of those are trained in the 2nd system. Table 1 summarizes the results of this experiment. The jointly optimized DGMM clearly outperforms the separately optimized DGMM.

System	baseline DNN	separately optimized DGMM	jointly optimized DGMM
state acc	60.75	59.01	62.66
WER	15.6	14.6	12.2

Table 1. Separate versus joint optimization of model and features in deep architectures (is-is).

The results for the two DGMM systems suggests that joint training successfully extracted features beneficial for GMM training. However, a counter-argument is that the gain is due to the greater number of trainable parameters in the jointly trained DGMM. To address this possibility, we compared the two approaches using a shallow structure. A GMM(39, 21) layer was used without any hidden layers. In the first experiment, we used PLP + LDA features extracted outside the model as input to the GMM layer. In this case, the GMM layer doesn't have the bottleneck sub-layer. In the second experiment, we used log energies as the input features and a bottleneck sub-layer of size 39 as feature extractor. Each system here has about 4.0 M parameters in total. Table 2 shows the results, as well as the number of training steps and CE cost at the time of of evaluation. The high WERs here suggest that optimization of shallow, randomly initialized GMMs using ASGD is not very effective compared to conventional GMM initialization techniques and ML-based training using the EM algorithm. Nonetheless, the joint system outperforms the separate system significantly, using a smaller number of steps. This argues against the number of trainable parameters having a dominant effect in the previous experiment, highlighting instead the importance of joint optimization.

Front Topology	CE cost	num steps	state acc	WER
PLP+LDA	10.12	33.92 M	33.35	51.7
Log Energy	8.92	17.8 M	44.7	29.3

Table 2. Separate versus joint optimization of features and classification for shallow GMMs (is-is).

Effect of DGMM depth. To examine the effect of depth more carefully we conducted another experiment comparing DGMM models with different numbers of hidden layers. Three structures with 1, 3 and 7 hidden layers, and 2560 nodes per layer, were considered. The GMM layer, GMM(272,5), is used right after the last hidden layer in each structure. Table 3 summarizes the results. Performance significantly improves with depth, suggesting that the deep structures possess superior feature extraction ability. In addition, the

# hidden layers	#params	CE cost	#steps	state acc	WER
1	10 M	10.02	19.74 M	47.43	32.4
5	36 M	8.00	2.75 M	59.44	14.8
7	49 M	7.62	1.4 M	62.66	12.2

Table 3. Effect of depth on DGMM performance (is-is).

number of training steps significantly decreases as the structure becomes deeper. This supports the discussion in [20] on the learning efficiency of deep structures over shallow ones. Note, however, that the total number of parameters is not matched across configurations.

Feature Extraction vs Classification Power in NNs & GMMs. In this section, the tradeoff between deep feature extraction capacity and classification power is examined by comparing GMMs and NNs with both shallow and deep architectures with matched total numbers of parameters. All four models have roughly the same number of parameters, 54 M. The shallow NN has 3936 hidden nodes in a single hidden layer while the deep NN is the baseline DNN described earlier. The shallow GMM has a GMM(272, 41) layer without any hidden layers beyond the 272-node linear bottleneck layer. For the DGMM, we used 7 hidden layers of 2560 nodes with a GMM(480, 5) layer. All four models were trained starting from randomly initialized weights. Table 4 summarizes the results. For both GMMs and NNs, the deep structure outperforms the corresponding shallow structure in terms of number of training steps as well as state accuracy and WER. The shallow NN outperforms the shallow GMM by a large margin. The performance difference between deep and shallow architectures is much larger for GMMs than for NNs. It might be that optimized deep feature extraction is inherently more important for GMMs than for NNs. However, as observed earlier, the high WER for the shallow GMM suggests that optimizing shallow, randomly initialized GMMs using ASGD is not very effective. In any case, deep structure leads to much better optimization and/or feature extraction capability for the GMM system: the jointly optimized DGMM outperforms the shallow GMM structure significantly. The DGMM also outperforms the DNN.

	Shallow		**Deep**	
	GMM	NN	GMM	NN
state acc	30.66	53.38	55.67	54.06
WER	71	18.2	15.6	16.1
CE cost	10.21	1.33	7.33	1.18
#training steps	9.4 M	23 M	4.2 M	8 M

Table 4. NN and GMM results using shallow and deep structures, with matched # of parameters, and no pre-training (is-is).

Use of pre-training; large scale ASR results. Table 5 shows results comparing DNNs and DGMMs with matched numbers of parameters for the large-scale en-us task, as well as for is-is, with some use of pre-training. The DNN models have 8 hidden layers with 2560 nodes per layer; the is-is DNN was initialized cross-lingually from an existing en-us DNN. The DGMMs share the input layer and the first 7 hidden layers with their corresponding DNN model. A GMM(480, 5) layer was used for is-is and a GMM(80, 15) layer was used for en-us. The DGMM models are initialized from the corresponding DNN models. The DGMM training results for en-us are not fully converged. For both tasks, the proposed DGMMs yield WERs competitive with state-of-the-art DNN systems.

Dataset	is-is (60 hours)		en-us (3,000 hours)	
	DGMM	DNN	DGMM	DNN
state acc	61.69	60.75	57.4	59.2
WER	13.3	15.6	12.0	11.7

Table 5. Results for pre-trained DNNs and DGMMs.

Deep Subspace-GMM (DSGMM) The Subspace GMM (SGMM) extends GMMs by introducing a shared subspace, v_s, among mixture components and two linear transforms M and W which are *tied* among all mixtures [21]. For mixture s, the mean parameters are $\mu_s = Mv_s$ and the mixture weights are derived by applying the softmax activation on Wv_s. In [21], the variances are not modeled with a subspace, due to a cited difficulty with training.

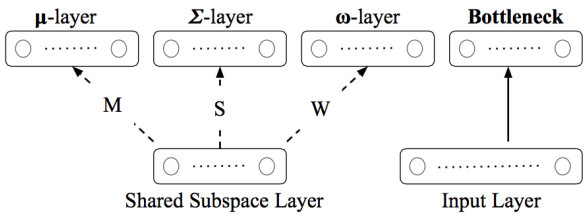

Fig. 3. Integrating a Subspace Gaussian Mixture Model (SGMM) into a DGMM architecture: the Deep SGMM (DSGMM).

The SGMM model can easily be applied to the GMM layer, as shown in Fig. 3. The *shared subspace layer* is a trainable layer that stores a shared subspace, v_s. There are three linear transforms, M, S, and W, corresponding to the three model parameters, means, variances and mixing weights. These transforms are *tied* across states. The subspace layer can be *shared* by one, two, or all three sub-layers above. This is shown by dotted connections in the figure. We examined three configurations in Table 6. All these DSGMM models share the same deep structure of 7 hidden layers (baseline DNN) and a GMM(80, 5) layer. The subspace dimensionality is the same as that of the input feature, 80. The only difference is in the way that models share the subspace layer. The parameters that do not share the subspace have their own trainable layer instead. The experiments show that the DSGMM model outperforms the baseline DGMM model of similar size. The configuration that uses the subspace only for the mean parameters slightly outperforms other models in terms of state accuracy and WER. This required fewer training steps than in the other configurations. Sharing the subspace among all parameters also outperforms the baseline DGMM, but a significantly larger number of training steps is required than for other models. Sharing the subspace among just means and weights outperforms sharing the subspace among all parameters.

	baseline DGMM	shared mean	shared mean-weight	shared all
state acc	60.63	62.38	61.8	61.13
WER	14.9	12.7	13.3	13.7
# training steps	2.1 M	0.3 M	1.8 M	4.2 M

Table 6. Deep Subspace GMM experiments (is-is).

5. CONCLUSIONS

This paper proposed a GMM layer as an alternative to the softmax layer in a DNN structure. The resulting Deep GMM system combines the advantages of deep feature extraction and GMM representation. Joint training of this model with ASGD produced results competitive with DNNs on two Voice Search tasks, particularly for the smaller task. Extension of this model to Deep Subspace GMMs resulted in additional improvements.

Acknowledgments

The authors would like to thank our colleagues Rajat Monga and Ke Yang for their valuable help and design ideas.

6. REFERENCES

[1] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 24–29.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, November 2012.

[3] A. Mohamed, G. Hinton, and G. Penn, "Understanding how deep belief networks perform acoustic modelling," in *Proc. IEEE ICASSP*, March 2012, pp. 4273–4276.

[4] A. Biem, S. Katagiri, E. McDermott, and B.-H. Juang, "An Application of Discriminative Feature Extraction to Filter-Bank Based Speech Recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 2, pp. 96–110, 2001.

[5] G. Heigold, *A log-linear discriminative modeling framework for speech recognition*, Ph.D. thesis, RWTH Aachen University, 2010.

[6] G. Heigold, H. Ney, P. Lehnen, T. Gass, and R. Schluter, "Equivalence of generative and log-linear models," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 19, no. 5, pp. 1138–1148, 2011.

[7] D. Yu and M. L. Seltzer, "Improved bottleneck features using pretrained deep neural networks," in *INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association, Florence, Italy, August 27-31, 2011*, 2011, pp. 237–240.

[8] T. Sainath, B. Kingsbury, and B. Ramabhadran, "Auto-encoder bottleneck features using deep belief networks," in *Proc. IEEE ICASSP*, 2012, pp. 4153–4156.

[9] L. Deng and J. Chen, "Sequence classification using the high-level features extracted from deep neural networks," in *Proc. IEEE ICASSP*, 2014.

[10] K. Demuynck and F. Triefenbach, "Porting concepts from DNNs back to GMMs," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 356–361.

[11] Matthias Paulik, "Lattice-based training of bottleneck feature extraction neural networks.," in *INTERSPEECH*, 2013, pp. 89–93.

[12] Aaron van den Oord and Benjamin Schrauwen, "Factoring variations in natural images with deep gaussian mixture models," in *Advances in Neural Information Processing Systems*, 2014, pp. 3518–3526.

[13] E. McDermott, *Discriminative Training for Speech Recognition*, Ph.D. thesis, Waseda University, School of Engineering, March 1997.

[14] M. Schuster, "On supervised learning from sequential data with applications for speech recognition," *Daktaro disertacija, Nara Institute of Science and Technology*, 1999.

[15] E. McDermott, "Discriminative prototype-based methods for speech recognition," in *Handbook for Neural Networks in Speech Processing*, pp. 159–216. Artech House, Morwood, MA, USA, 2000.

[16] E. McDermott, T.J. Hazen, J. Le Roux, A. Nakamura, and S. Katagiri, "Discriminative training for large vocabulary speech recognition using Minimum Classification Error," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, no. 1, pp. 203–223, January 2007.

[17] C. Bishop, *Neural networks for pattern recognition*, Oxford University Press, 1995.

[18] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *Proc. IEEE ICASSP*, 2009, pp. 3761–3764.

[19] E. McDermott, G. Heigold, P. Moreno, A. Senior, and M. Bacchiani, "Asynchronous stochastic optimization for sequence training of deep neural networks: Towards big data," in *Proc. Interspeech*, 2014.

[20] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI," *Large-scale kernel machines*, vol. 34, pp. 1–41, 2007.

[21] D. Povey, L. Burget, M. Agarwal, P. Akyazi, K. Feng, A. Ghoshal, O. Glembek, N. Goel, M. Karafiát, A. Rastrow, R. C. Rose, P. Schwarz, and S. Thomas, "Subspace gaussian mixture models for speech recognition," in *Proc. IEEE ICASSP*. IEEE, 2010, pp. 4330–4333.