The Projected Belief Network Classfier : both Generative and Discriminative.

Paul M. Baggenstoss
Fraunhofer FKIE, Fraunhoferstrasse 20
53343 Wachtberg, Germany
Email: p.m.baggenstoss@ieee.org

***Abstract*—The projected belief network (PBN) is a layered generative network with tractable likelihood function, and is based on a feed-forward neural network (FF-NN). It can therefore share an embodiment with a discriminative classifier. In this paper, a PBN is constructed with special output prior that serves as a discriminative cost function. The result is a single network that both fully discriminative and fully generative. Training is possible with varying degrees of generative and discriminative influence. A convolutional PBN classifier is tested on spectrograms of spoken commands. It is shown that the network displays excellent qualities from either the discriminative or generative viewpoint. Excellent random data synthesis and visible data reconstruction from hidden variables deep within the network is shown, while classifier performance approaches that of a regularized discriminative network.**

I. Introduction

A. Background and Motivation

Much has been published on the comparison of generative and discriminative classifiers. The widespread view is that discriminative classifiers generalize better when sufficient labeled training data is available [1]. Despite their success, it has been recognized that discriminative methods have flaws, vividly demonstrated by *adversarial sampling* [2], a technique in which small, almost imperceptible changes to the input data cause false classifications. Because generative classifiers are based on a model of the underlying data distribution, they are immune to adversarial sampling and can complement discriminative classifiers. As a result, there are a large number of methods that seek to combine generative and discriminative classifiers [3], [4], [5], [6], [7], [8], [1], or to combine discriminative and generative training [1], [9], [10] The weakness of generative classifiers stems from the need to estimate the data distribution, a very difficult task that is unecessary when just classifying between known data classes [11]. Deep layered generative networks are a step in the right direction because they can model complex data generation processes, but they have a serious flaw: the data distribution, also called likelihood function (LF) is intractible because the hidden variables are jointly distributed with the input data and must be integrated out. Such networks need to be trained using surrogate cost functions such as contrastive divergence to train restricted Boltzmann machines [12], [13], and Kullback Leibler divergence to train variational auto-encoder (VAE) [14], or an adversarial discriminative network to train generative adversarial networks (GAN) [15].

In summary, there is a need for better generative models with tractable LF that can be combined with discriminative approaches. The newly introduced layered generative network called projected belief network (PBN) stands out as a potentially better choice to achieve these goals. The PBN is a deep layered generative network (DLGN), so can model complex generative processes, but differs from all other DLGNs in three ways. First, it has a tractable likelihood function, so can be trained directly. Using the tractable LF, it can detect out-of-set samples (outliers that are outside of the set of training classes). Second, the PBN is based on a feed-forward neural network (FF-NN), so it can share an embodiment with a discriminative classifier (i.e. it is a single network that is both a complete generative model and a discriminative classifier). Third, the discribution of the output layer (output variables of the last layer) is embedded as a separate factor in the likelihood function, so can be used to inject discriminative behavior into the network without compromising the generative model. For these three reason, the PBN is a more direct way to introduce the advantages of generative models into a discriminative classifier, or vice-versa.

B. Main Idea

The goal of this paper is to construct a layered generative network with tractable likelihood function that is at the same time a fully discriminative classifier. As we stated, the PBN is based on a feed-forward neural network (FF-NN). Figure 1 shows a simple 3-layer feed-forward neural network (FF-NN). Each layer l consists of a linear transformation (represented by

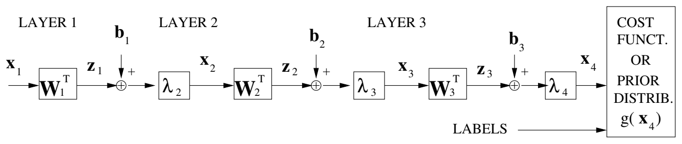

Fig. 1. A feed-forward neural network (FF-NN). This FF-NN can be a discriminative classifier if λ_4 is the *softmax* function and the output box is the cross-entropy cost function. It can also be a generative model if viewed as PBN and the output box is the output prior distribution $g(\mathbf{x}_4)$.

matrix $\mathbf{W}_l$), a bias $\mathbf{b}_l$ and an activation function $\lambda_{l+1}(\,)$. The linear transformation can be fully-connected or convolutional, but must have total output dimension lower than the input dimension. The output layer is required to have a total dimension equal to the number of classes. Therefore, if the final activation function were **softmax**, then this network could be trained as a traditional classifier using cross-entropy cost function. On the other hand, it can also be viewed as a projected belief network (PBN) [16], [17]. As such, it has the likelihood function (see [17]) given by

$$p_p(\mathbf{x}_1;T,g) = \frac{1}{\epsilon}\,\frac{p(\mathbf{x}_1;H_{0,1})}{p(\mathbf{z}_1;H_{0,1})}\,|\mathbf{J}_{\mathbf{z}_1\mathbf{x}_2}| \cdot \frac{p(\mathbf{x}_2;H_{0,2})}{p(\mathbf{z}_2;H_{0,2})}\,|\mathbf{J}_{\mathbf{z}_2\mathbf{x}_3}|\,\frac{p(\mathbf{x}_3;H_{0,3})}{p(\mathbf{z}_3;H_{0,3})}\,|\mathbf{J}_{\mathbf{z}_3\mathbf{x}_4}|\,g(\mathbf{x}_4), \quad (1)$$

where ϵ is the *sampling efficiency* [17] that we can assume to be 1.0, $p(\mathbf{x}_l;H_{0,l})$ is the assumed prior distribution for the input to layer l, denoted by $\mathbf{x}_l$, $p(\mathbf{z}_l;H_{0,l})$ is the distribution of $\mathbf{z}_l$ under the assumption that $\mathbf{x}_l$ is distributed according to $p(\mathbf{x}_l;H_{0,l})$, and where $g(\mathbf{x}_{L+1})$ is the assumed prior for the output of a network. The PBN is trained by maximizing the mean of the log of (1) using stochastic gradient ascent. To be a PBN, however, the network must have decreasing dimension in each layer. In the following, we will describe how to create a prior $g(\mathbf{x}_{L+1})$ so that the same network can be viewed as a discriminative classifier and as a generative PBN. In other words, the prior $g(\mathbf{x}_{L+1})$ will assume the role of the cross-entropy cost function, so will result in a generative/discriminative network.

II. Technical Approach

A. Output Non-Linearity and Prior

In order to create a PBN that is compatible with a discriminative classifier, we used the truncated exponential distribution (TED) activation function (non-linearity) [17], [19] given by $\lambda(\alpha) = \frac{e^{\alpha}}{e^{\alpha}-1} - \frac{1}{\alpha}$, which is similar to sigmoid, producing output in [0, 1]. We used a TED prior output distribution $g(\mathbf{x}_{L+1})$ given by $g(\mathbf{x};\boldsymbol{\alpha}) = \prod_i \left(\frac{\alpha_i}{e^{\alpha_i}-1}\right) e^{\alpha_i x_i}$, where $\boldsymbol{\alpha}$ depends on the class labels (the ground-truth label for input data $\mathbf{x}_1$). The relationship is $\alpha_i = 2C(l_i - .5)$, where $[l_1, l_2 \ldots l_M]$ are the one-hot label encodings, so α_i has values C or $-C$.. Recall the output dimension M is also the number of classes. An approximation to this prior for large C is to add a value of $log(C)$ to the LF when output x_i matches l_i, and a value of $log(C) - C$ if it is the logical inverse of l_i. Training the network to maximize the average of the log of (1) can be interpreted as discriminative (through term $g(\mathbf{x}_{L+1})$) and generative through the remaining terms. The degree of discriminative training can be varied by changing C.

B. MaxEnt Reconstruction and Synthesis

We now investigate a distinctly generative property of the PBN : visible data reconstruction from hidden variables. Input data can be randomly synthesized or reconstructed from the output of any layer of the FF-NN.

Unlike other generative networks, the PBN is not an explicit generative network, it operates implicitly by "backing up" through a FF-NN. In each layer, the FF-NN operates on the input $\mathbf{x}$ by dimension-reducing linear transformation $\mathbf{z} = \mathbf{W}'\mathbf{x}$. To "back up", it is necessary to determine the set $\mathcal{M}(\mathbf{z})$ of possible input samples $\mathbf{x}$ that "could have" produced $\mathbf{z}$. In other words,

$$\mathcal{M}(\mathbf{z}) = \{\mathbf{x} : \mathbf{W}'\mathbf{z} = \mathbf{x}\}.$$

A sample is selected from $\mathcal{M}(\mathbf{z})$ with probability density proportional to the prior distribution $p_0(\mathbf{x})$. This is called "MaxEnt" sampling because $p_0(\mathbf{x})$ is the maximum entropy prior given the range of $\mathbf{x}$. It is also called uniform manifold sampling (UMS) because under certain conditions, $p_0(\mathbf{x})$ has constant value on $\mathcal{M}(\mathbf{z})$. Sampling requires a type of Markov chain Monte-Carlo (MCMC) [19]. For deterministic reconstruction, we select $\mathbf{x}$ to be the centroid of $\mathcal{M}(\mathbf{z})$, which is also the conditional mean $\hat{\mathbf{x}}|\mathbf{z} = \mathbb{E}(\mathbf{x}|\mathbf{z})$.

This conditional mean can be found in closed form for a range of MaxEnt priors [18], [17], [19]. It is given by $\hat{\mathbf{x}}|\mathbf{z} = \lambda\left(\mathbf{W}\hat{\mathbf{h}}\right)$, where $\hat{\mathbf{h}}$ is the solution of the equation

$$\mathbf{W}'\lambda\left(\mathbf{W}\mathbf{h}\right) = \mathbf{z}. \quad (2)$$

This solution is guaranteed to exist as long as $\mathbf{x}$ is in the support $p_0(\mathbf{x})$ and is also the saddle-point for the saddle-point approximation to $p_0(\mathbf{z})$ [18]. For the simplest case of Gaussian MaxEnt prior, the activation function is linear, $\lambda(\alpha) = \alpha$, and the reconstruction is just least-squares, $\hat{\mathbf{x}}|\mathbf{z} = \mathbf{W}\left(\mathbf{W}'\mathbf{W}\right)^{-1}\mathbf{z}$. For positive-valued data, we use the truncated Gaussian prior [18], and for data in [0, 1], we use the uniform prior [18].

Starting at any layer output, one can proceed in the backward direction up the network, always increasing the dimension, until the visible data is reconstructed. Note that $\hat{\mathbf{h}}$ is only guaranteed to exist if $\mathbf{z} = \mathbf{W}'\mathbf{x}$ for some $\mathbf{x}$ in the support $p_0(\mathbf{x})$. But, when reconstructing from more than one layer, this requirement is not always met, so the reconstruction chain could fail (see *sampling efficiency* in [17]). However, after the network is trained, reconstruction failure is rare [17], and often means the input sample is mal-formed.

Note that there are two possible methods to reconstruct $\mathbf{x}$ from $\mathbf{z}$, (a) random sampling in $\mathcal{M}(\mathbf{z})$ by MCMC , and (b) deterministically selecting the centroid $\hat{\mathbf{x}}|\mathbf{z}$. In the following, we use the approach of random sampling the last 2 layers, then deterministic reconstruction back to the visible data.

C. PBN Properties

The proposed method differs significantly with other methods of combining the roles of generative and discriminative networks that are available in the field because the discriminative influence is added into the output prior and does not disturb the "purity" of the generative network. In other words, there is no compromise between generative and discriminative training or structure, they are both contained in one network and one cost function!

When reconstructing visible data from hidden variables, the synthesized data, when applied to the feed-forward network, produces exactly the same hidden variables. This property of hidden variable recovery is interesting and is true of no other layered generative network.

When the discriminative cost function is “satisfied” (the training data is almost completely separated), then the generative cost dominates, so the network becomes the best possible PBN that at the same tiome separates the data. This can be seen as a generative regularization effect.

III. Classification of Spectrograms of Words Commands

A. Data set

The data was selected to be at the same time relevant, realistic, and challenging. We selected a subset of the Google speech commands data [20], choosing three pairs of difficult to distinguish words: “three, tree”, “no, go”, and “bird, bed”, sampled at 16 kHz and segmented into into 48 ms Hanning-weighted windows shifted by 16 ms. We used log-MEL band energy features with 20 MEL-spaced frequency bands and 45 time steps, representing a frequency span of 8 kHz and a time span of 0.72 seconds. The input dimension was therefore $N = 45 \times 20 = 900$. From each of the six classes, we selected 500 training samples, 150 validation samples, at random. The remaining samples were used to test, averaging about 1500 per class or about a total of 10000 testing samples.

B. Network

A separate network was trained on each word pair. The networks had $L = 5$ layers. The first layer was convolutional with 36 21×16 convolutional kernels using “valid” border mode and 6×4 downsampling (not pooled, just down-sampled), thus producing 36 5×2 output feature maps, or a total output dimension of 360. The remaining 3 layers were fully-connected with 100, 32, and 16 hidden neurons. The output layer had 2 neurons, matching the number of classes. Note that we sought to reduce the dimension in each layer by at least a factor of 2. The layer output activation functions were linear, TG, TG, TG, and softmax, where TG is the truncated Gaussian activation [18] (similar to softplus) : $\lambda_{TG}(\alpha) = \alpha + \frac{\mathcal{N}(\alpha)}{\Phi(\alpha)}$, where $\mathcal{N}(x) \triangleq \frac{e^{-x^2/2}}{\sqrt{2\pi}}$ and $\Phi(x) \triangleq \int_{-\infty}^{x} \mathcal{N}(x)$. The TG activation is the theoretical activation function for reconstructing data from a linear transformation applied to positive-valued data under the truncated Gaussian prior distribution [18], however in behavior, it is similar to softplus $\lambda_{SP}(\alpha) = \frac{1}{1+e^{-x}}$.

C. Classification Results

The networks were first initialized with random weights and trained as a standard discriminative deep neural network (DNN) with dropout regularization of 0.2, 0.1, and 0.1 applied to the output of the first through third layers (dimensions 360, and 100, 36, respectively) and L-2 regularization. No data augmentation (such as random shifting) was used. Classification accuracy for the DNN is given in Table I.

Next, the DNNs trained as described above were used as the initial network for the PBN, which was trained by maximizing the mean likelihood function (1) with output prior distribution parameter $C = 2000$. No data augmentation was used, and no regularization was used.

	”three-tree”		”no-go”		”bird-bed”	
	train	test	train	test	train	test
DNN	1.00	0.870	1.00	0.874	1.00	0.960
PBN	.991	0.881	0.992	0.810	0.978	0.946

TABLE I
Classification accuracy for the thre class pairs.

The classification results for the PBN on the three class pairs are shown in Table I where they can be compared with the DNN. Note that the PBN has only slightly lower accuracy than the DNN, and in fact has higher accuracy on one class pair. It should also be kept in mind that the PBN was trained without regularization of any kind, wheras the DNN used both L2 and dropout regularization.

D. Reconstruction Results

It has been established that the PBN has lost little in terms of classification performance. It will now be determined what has been gained in terms of generative power. The first thing that comes to mind is the reconstruction of visible data from the hidden variables. Using the method of Section II-B, we reconstructed data from the DNN’s hidden variables, starting with 1 layer, then 2 layers, etc. Results are shown in Figure 2. After the first layer, some resemblance can be seen, but after

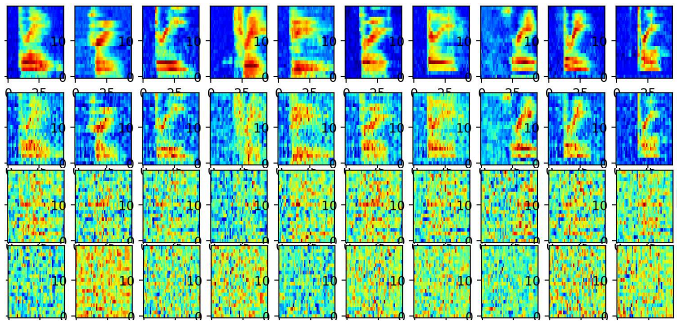

Fig. 2. Samples of spoken word commands “three” and “tree” reconstructed from standard DNN hidden variables. From top: original spectrograms, then the same reconsructed from first through third layer. Reconstruction from layers 4 and 5 was not possible due to sampling efficieny of zero.

that, the images are unrecognizeable. This is how the network sees the data through the hidden variables. The noisy images, when used as input data will produce exactly the same hidden variables at the given layer as the original input sample.

The reconstruction experiment was repeated for the trained PBN. Results are shown in Figure 3. Reconstructing from the output of the layers produces reconstructions with excellent quality, but gradually decreasing sharpness. Reconstruction from the fifth layer (a feature bottleneck of dimension 2) produces either a “standard” “three” or a “standard” “tree”. Note that this network was not trained for lower reconstruction error, but instead to maximize (1). The reconstruction power of the network comes as a side-effect and can be tapped into anywhere in the network. A standard auto-encoder, in contrast, is trained to reconstruct for a fixed network length.

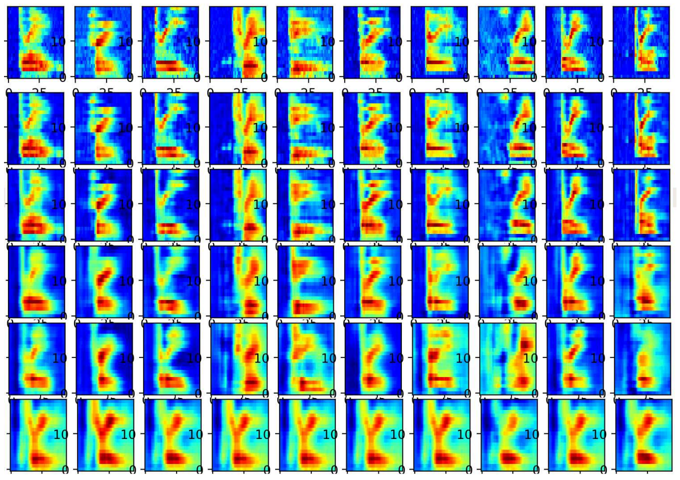

Fig. 3. Samples of speech commands “three, three” reconstructed using PBN. From top: original spctrograms, then the same reconsructed from output of first through fifth layer. Hidden variable dimensions are 360, 100, 32, 16, and 2.

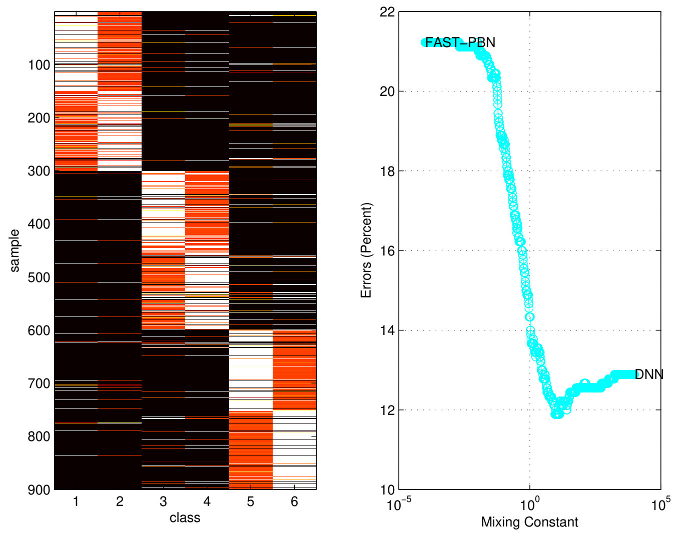

Fig. 4. Left: PBN output probabilities for the six classes. Right: Classification error in percent as a function of mixing constant for a mixture of PBN with DNN.

E. Classifying between class pairs

A second exercise in “generative capability” is the classification between class pairs using models trained separately on just one pair. This exercise is obviously not possible using the discriminative classifiers trained on the class pairs, but must be trained on all six classes. For the PBN, discrimination between members of the class pairs occurs in the output layer and does not require computing the LF because these are independent terms in (1). To compute the LF for just one member class, it is necessary to set the parameter α of the output distribution to the respective one-hot encoding. The results of the 6-class PBN experiment are shown in Figure 4. The probability output of the PBN classifier is shown on the left side. Notice that there are a significant number of errors between class pairs. Total classification error was 21 percent.

A discriminative DNN was trained on the 6-classes using the same network structure as the two-class networks, but with increased neuron counts of 48, 150, 48, 24, and 6 using dropout and L2 regularization. The classification performance of the DNN was 12.9 percent, significantly lower than the multiple PBN classifier. It is easy to explain why the PBN performed almost as well in the class-pairs, but much worse in the 6-class experiment. This is because the class-pair PBN is a single model, whereas for the 6-class problem, three separate generative models are required. This introduces errors resulting from imbalances in the models. Here is an opportunity, however, to improve the result by mixing the two classifier outputs. On the right side of Figure 4, we show the classification error percentage as a function of the mixing constant, showing the transition from PBN only (left) to DNN only (right). At a certain value, there is a dip in error, providing the optimal mixing value. As has been shown [19], a hybrid generative model can be created at this “optimal” point.

F. Random Synthesis

As a final demonstration of generative power, we synthesized entirely random events by starting with random data equal in dimension to the PBN output layer, in this case dimension-2. Data was synthesized at the point prior to the output activation function using Gaussian random variables. Results are shown in Figure 5 for the class pair “three” and “tree”. The synthetic samples appear realistic and are diverse,

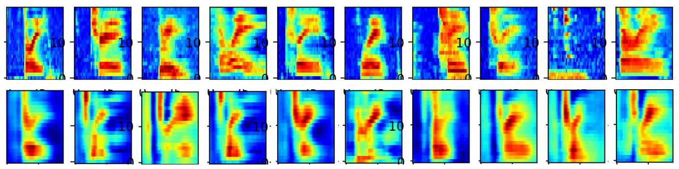

Fig. 5. Top: ten training samples randomly selected from “three” and “tree” spoken word commands. Bottom: randomly synthesized data from trained PBN. There is no relationship to the selected training samples on top.

showing variations in time shift, dilation, and other qualities. This means that the PBN has indeed learned much about the data generation process.

G. Implementation and Applications

The PBN was implemented in Python using Theano symbolic expression compiler [21]. The primary computational challenge is the solution of a symmetric linear system with dimension $M \times M$, where M is the total output dimension of a layer. This must be solved for each iteration in the solution of (2). This was parallelized on the GPU, one processor for sample in a mini-batch. The computational time for an epoch was 1.1 seconds. This was only about an order of magnitude slower than training the DNN. All results were obtained using PBN Toolkit [1].

IV. Conclusions and Future Work

In this paper, a projected belief network (PBN), which is a purely generative layered network, was trained as a generative-discriminative classifier. This was achieved using a label-dependent prior for the output features. Since the PBN is based on a feed-forward neural network (FF-NN), it can share an embodiment with a discriminative deep neural network (DNN). Using a single parameter, the network can be trained either as a generative PBN, or as a discriminative DNN, or any point in between. When reconstructing visible data from the hidden variables, it was shown that the DNN had very poor ability to reconstruct, even from initial layers, whereas when training jointly with the PBN, the reconstruction greatly improved. The PBN classifier had comparable classification performance to the discriminative DNN despite using no regularization, yet provided generative power from three standpoints: visible data reconstruction from hidden variables, random data synthesis, and classification of out-of set samples.

The results in this paper open up several questions for future work. For example, the PBN appears not to respond well to dropout or L2 regularization, but the unregularized PBN classifier performed on par with the regularized DNN. Is there a way to regularize the PBN to achieve further improvements? How can the strucure of the PBN be improved, for example with longer or shorter networks? Can the PBN be used in adversarial networks? Are there other, better prior distributions to use? Are there better methods to initialize the PBN? How does classification through reconstruction error compare to LF classification?

References

[1] J. Lasserre, C. Bishop, and T. Minka, "Principled hybrids of generative and discriminative models," vol. 1, pp. 87– 94, 07 2006.

[2] C. Mayer and R. Timofte, "Adversarial sampling for active learning," *arXiv:1808.06671, to appear WACV 2020*, 2019.

[3] T. Jaakkola and D. Haussler, "Exploiting generative models in discriminative classifiers," tech. rep., Dept. of Computer Science, Univ. of California, 1998.

[4] R. Raina, Y. Shen, A. Ng, and A. McCallum, "Classification with hybrid generative /discriminative models," in *Proceedings of NIPS (Neural Information Processing Systems) 2004*, 2004.

[5] S. Fine, J. Navratil, and R. Gopinath, "Enhancing gmm scores using SVM hints," in *Proceedings of the 7th European Conference on Speech Communication and Technology (EuroSpeech)*, 2001.

[6] A. Fujino, N. Ueda, and K. Saito, "A hybrid generative/discriminative approach to semi-supervised classifier design," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 20, 2005.

[7] A. Holub, M. Welling, and P. Perona, "Hybrid generative-discriminative visual categorization," in *Proceedings of the International Journal of Computer Vision*, vol. 77, pp. 239–258, 2008.

[8] A. Bosch, A. Zisserman, and X. Muoz, "Scene classification using a hybrid generative/discriminative approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 4, pp. 712–727, 2008.

[9] T. Minka, "Discriminative models, not discriminative training," *Microsoft Research Ltd, technical report*, 2005.

[10] C. Bishop and J. Lasserre, "Generative or discriminative? getting the best of both worlds," *Bayesian Statistics*, vol. 8, pp. 3–24, 2007.

[11] V. Vapnik, *The Nature of Statistical Learning*. Springer, 1999.

[12] M. Welling, M. Rosen-Zvi, and G. Hinton, "Exponential family harmoniums with an application to information retrieval," *Advances in neural information processing systems*, 2004.

[13] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," in *Neural Computation 2006*, 2006.

[14] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 1278–1286, PMLR, 22–24 Jun 2014.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[16] P. M. Baggenstoss, "On the duality between belief networks and feed-forward neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2018.

[17] P. M. Baggenstoss, "Applications of projected belief networks (pbn)," in *Proceedings of EUSIPCO 2019*, (La Coruña, Spain), Sep 2019.

[18] P. M. Baggenstoss, "A neural network based on first principles," in *arXiv:2002.07469, submitted to ICASSP 2020*, (Barcelona, Spain), Sep 2020.

[19] P. M. Baggenstoss, "Uniform manifold sampling (UMS): Sampling the maximum entropy pdf," *IEEE Transactions on Signal Processing*, vol. 65, pp. 2455–2470, May 2017.

[20] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv:1804.03209*, 2018.

[21] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, Y. Bengio, "Theano: A CPU and GPU Math Expression Compiler," Proceedings of the Python for Scientific Computing Conference (SciPy) 2010.

[1] http://class-specific.com/pbntk. A copy of the data is also available at this link