Applications of Projected Belief Networks (PBN)

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 (LF) that can be trained by gradient ascent as a probability density function (PDF) estimator and classifier. The PBN is derived from a feed-forward neural network (FF-NN) by finding the generative network that implements the probability distribution with maximum entropy (MaxEnt) consistent with the knowledge of the distribution at the output of the FF-NN. The FF-NN, from which the PBN is derived, is a complementary feature extractor that exactly recovers the PBN's hidden variables. This paper presents a multi-layer PBN and a deterministic PBN that are tested using a subset of MNIST data set. When the deterministic PBN is combined with the dual FF-NN, it forms an auto-encoder that achieves much lower reconstruction error on testing data than the equivalent conventional network and functions significantly better as a classifier.**

I. Introduction

A. Background and Motivation

Discriminative neural networks have dominated machine learning for decades. The performance of generative networks lags behind because they need to model the generative process underlying the data, a much harder task than discrimination [1]. Yet, interest in generative models persists because a model of the underlying process is useful, as exemplified by variational autoencoders (VAE) [2], and generative adversarial network [3] (GAN) which have sparked considerable interest. While the generative task is harder, given time and effort, generative models can perform as well as classifiers as their discriminative counterparts. For example, when Hinton's deep belief network (DBN) was published, the DBN worked better than comparable fully-connected (non-convolutional) feed-forward networks [4]. While training algorithms have been developed for VAE and DBN, the likelihood functions (LF) are not available in closed-form, so need to be approximated, using stochastic variational methods in the case of VAE [2], or Monte Carlo approximations in the case of DBN [5]. The projected belief network (PBN) is a new type of generative network with tractable LF that generates data layer-wise from hidden variables similar to a deep latent Gaussian model (DLGM). But, in contrast to other generative models, the PBN is related to a feed-forward neural network (FF-NN) by a duality relationship [6]. The dual FF-NN, which is here called dual analysis network (DAN), exactly recovers the hidden variables of the PBNs data generation process. With tractable LF, the PBN has the potential to enable a new class of generative models and algorithms.

B. Main Idea

The projected belief network (PBN) was previously introduced as a dual counterpart to a feed-forward neural network (FF-NN) [6]. The PBN is derived from a FF-NN by asking the following question: *knowing the FF-NN and the distribution of the output variables (features) of the FF-NN, what is the maximum entropy (MaxEnt) distribution of the visible data consistent with the given features distribution?* The PBN is the generative network that implements this MaxEnt distribution [6]. Not surprisingly, the PBN uses the same network weights as the FF-NN from which it is derived, and employs a special "activation" function that gives it its unique properties. A deterministic version of the PBN is created if instead of generating random data in each layer, the conditional mean is propagated. The deterministic PBN is the complementary network to the DAN and combined with the DAN forms a new type of auto-encoder.

C. Paper Contributions

The PBN has been previously introduced [6]. Novel contributions of this paper include (a) experimental results comparing PBN with other models as a function of data dimension, (b) the detailed description of a multi-layer PBN, (c) the treatment of the issue of sampling efficiency, (d) the conceptual comparison of PBN with the VAE, and (e) the description of a deterministic PBN and its application as an auto-encoder, and experiments showing significant improvements over a conventional auto-encoder of the same structure.

II. Projected Belief Networks (PBN)

A. PBN Exact Form

Figure 1 illustrates a two-layer PBN in its exact, asymptotic, and deterministic forms. It can be easily extended to more layers. Near the bottom of the figure is the dual analysis network (DAN), a conventional feed-forward network employing an activation function $\lambda_n(\ )$ in layer n. Optionally, an energy statistic (ES), denoted by $e = t(\mathbf{x})$ is extracted from the input of each layer. The figure illustrates both data generation by different forms of the PBN (left to right) and feature extraction by the DAN (right to left). Data generation originates by a feature generating distribution $g(\mathbf{z}_2)$, then continues layer by layer. In layer n of the exact form of the PBN, (top), the activation function and bias (if used) are inverted, and the

978-9-0827-9703-9/19/$31.00 ©2019 IEEE

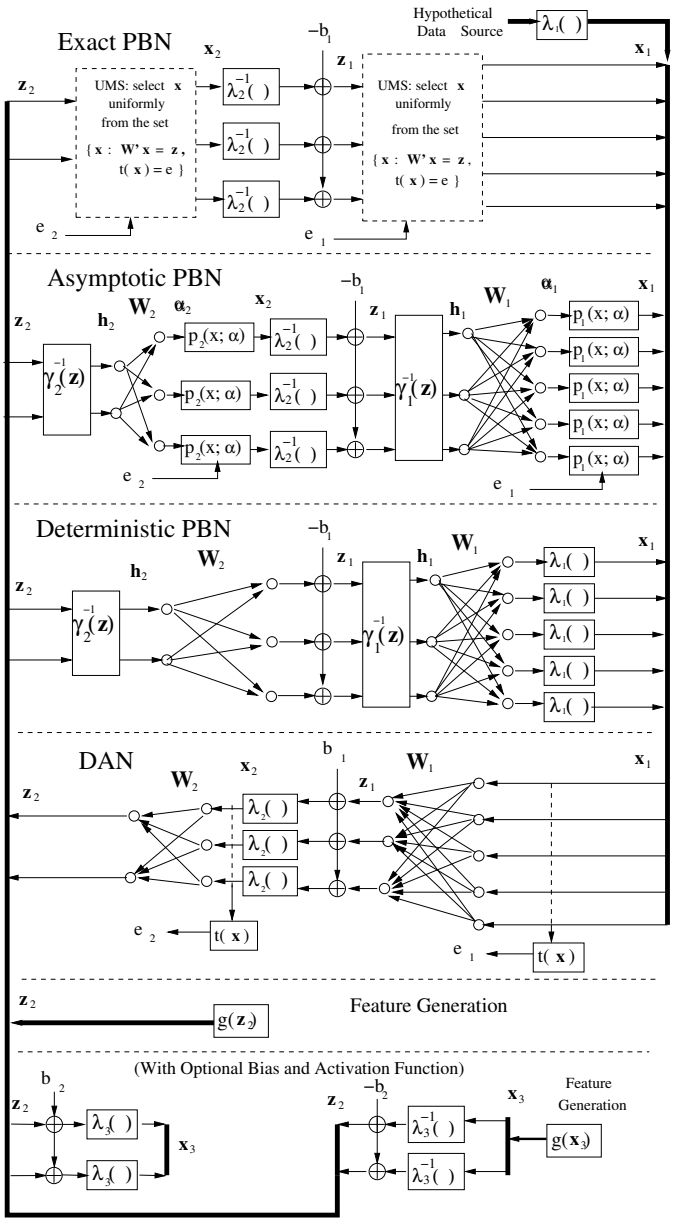

Fig. 1. A 2-layer PBN in three forms, exact, asymptotic, and deterministic, and the corresponding dual analysis network (DAN).

feature $\mathbf{z}_n$ is presented to the "UMS" block in which a sample $\mathbf{x}$ is drawn randomly from the set $\mathcal{M}_n(\mathbf{z}_n, e_n)$ defined by

$$\mathcal{M}_n(\mathbf{z}_n, e_n) = \{\mathbf{x} : \mathbf{W}_n'\mathbf{x} = \mathbf{z}_n,\ t_n(\mathbf{x}) = e_n, \quad \mathbf{x} \in \mathcal{X}_n\}, \tag{1}$$

where $\mathcal{X}_n$ is the input range of layer n and $e_n = t_n(\mathbf{x})$ is the optional ES. The sample $\mathbf{x}$ must be drawn with uniform distribution, so that no member of $\mathcal{M}_n(\mathbf{z}_n, e_n)$ is more likely to be drawn than any other. The sampling procedure is therefore called uniform manifold sampling (UMS) [7].

By the definition of UMS, the DAN will exactly recover the variables $\mathbf{z}_2$, $\mathbf{z}_1$. When the PDF of $\mathbf{z}_2$ is known, denoted by $g(\mathbf{z}_2)$, then the PBN generates samples the PDF:

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

where $\mathbf{x}_n$ is the input data to layer n ($\mathbf{x}_1$ is the visible data), T represents the DAN, $|\mathbf{J}_{\mathbf{z}_1\mathbf{x}_2}|$ is the determinant of the Jacobian of the 1:1 mapping from $\mathbf{z}_1$ to $\mathbf{x}_2$, and ϵ is the sampling efficiency, to be explained below.

Notice the absence of integral signs in (2) - the distribution does not require integrating out the hidden variables, as is necessary in other layered generative models. This is due to the fact that the hidden variables of the DAN are deterministically derived from the visible data, not jointly distributed. Note also that in (2) there appears a set of reference distributions, one for each layer. The distribution $p(\mathbf{x}_n; H_{0,n})$ is the maximum entropy (MaxEnt) reference distribution for layer n and $p(\mathbf{z}_n; H_{0,n})$ is the corresponding feature distribution[1]. This reference distribution depends on $\mathcal{X}_n$, the data range of layer n input, which in turn depends on the activation function used in the previous layer - note that the input (visible data) is assumed to have been created using $\lambda_1(\ )$. We consider three data ranges: $\mathbb{R}^N$, $\mathbb{P}^N$, and $\mathbb{U}^N$, where N represents input data dimension of a generic layer, $\mathbb{R}^N$ is the unlimited case, $\mathbb{P}^N$ is the positive quadrant ($0 \le x_i$), and $\mathbb{U}^N$ is the unit hypercube ($0 \le x_i \le 1$). The MaxEnt reference distribution for each data range $\mathcal{X}$ is given in Table I. The primary computational challenge in computing (2) is calculating the denominator terms $p(\mathbf{z}_n; H_{0,n})$. More is provided in the references [8], [9], [10], [7], [6], [11].

$\mathcal{X}$	$p(x;\alpha)$	$\lambda(\alpha)$	$t(\mathbf{x})$	$p(\mathbf{x}; H_0)$
$\mathbb{R}^N$	$\frac{e^{-(x-\alpha)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}}$ (Gauss.)	α	$\sum_i x_i^2$	$\frac{e^{-t^2(\mathbf{x})/2}}{(2\pi)^{-N/2}}$
$\mathbb{P}^N$	$\alpha e^{-\alpha x}$ (Expon.)	$1/\alpha$	$\sum_i x_i$	$e^{-t(\mathbf{x})}$
$\mathbb{U}^N$	$\left(\frac{\alpha}{e^\alpha - 1}\right) e^{\alpha x}$ (TED)	$\frac{e^\alpha}{e^\alpha - 1} - \frac{1}{\alpha}$	none	1

TABLE I
GENERATING DISTRIBUTIONS $p(x;\alpha)$, EXPECTED VALUE OF GENERATING DISTRIBUTIONS $\lambda(\alpha)$, ENERGY STATISTICS (ES) $t(\mathbf{x})$, AND REFERENCE HYPOTHESES $p(\mathbf{x}; H_0)$ FOR FOR DATA RANGES $\mathbb{R}^N$, $\mathbb{P}^N$, AND $\mathbb{U}^N$. THIS TABLE CONCERNS A SINGLE LAYER AND $\mathbf{x}$ IS ASSUMED TO BE THE VISIBLE DATA FOR THE GIVEN LAYER LAYER WITH DIMENSION N AND RANGE $\mathbf{x} \in \mathcal{X}$.

Optionally, a bias and activation function can be appended to the DAN (bottom of Figure 1), producing feature $\mathbf{x}_3$. In this case, the data generation process begins with the generating distribution $g(\mathbf{x}_3)$, and the activation function and bias must be inverted. Also, equation (2) must be modified by replacing $g(\mathbf{z}_2)$ with $|\mathbf{J}_{\mathbf{z}_2\mathbf{x}_3}|\ g(\mathbf{x}_3)$.

[1] $p(\mathbf{z}_n; H_{0,n})$ is the theoretical PDF of the layer output when the layer input is distribued according to $p(\mathbf{x}_n; H_{0,n})$.

B. PBN Asymptotic Form

It has been shown that the UMS sampling process can be closely approximated by a network layer resembling a sigmoid belief network [7]. To arrive at the asymptotic PBN (see Figure 1), the UMS blocks are replaced by a nonlinear function $\mathbf{h}_n = \gamma_n^{-1}(\mathbf{z}_n)$, matrix multiplication $\boldsymbol{\alpha}_n = \mathbf{W}_n\mathbf{h}_n$, then generation from distributions $p_n(x;\alpha)$, which are given in Table I as a function of $\mathcal{X}_n$. The expected value of these distributions (given α) is denoted by $\lambda_n(\alpha)$, which corresponds to the activation functions used in the DAN at the output of layer $n-1$. Interestingly, for $\mathbb{U}^N$, $\lambda_n(\alpha)$ is the mean of the truncated exponential distribution (TED), which is similar to the sigmoid function [7]. Central to the theoretical analysis of a PBN layer is the function $\gamma_n(\mathbf{h}_n) = \mathbf{W}'_n\lambda(\mathbf{W}_n\mathbf{h}_n)$. To compute a layer of a PBN, this function needs to be inverted: $\mathbf{h}_n = \gamma_n^{-1}(\mathbf{z}_n)$, which requires an iterative algorithm, but might have no solution (See Section II-E).

C. The PBN for $\mathbb{R}^N$ and Relationship to VAE

The VAE is currently a well-studied generative model [12], [2]. The "variational" aspect of VAE has to do with approximating and training the LF, but the VAE is essentially an implementation of DLGM [2]. Thus, both PBN and VAE are layered generative models. The main difference is that the PBN is based on an explicit feed-forward analysis network (the DAN), so the latent variables can be deterministically computed from the visible data. So, once a visible data sample has been generated by the PBN, all the hidden variables can then be exactly recovered by a single pass of the DAN. The VAE on the other hand is an stochastic layered generative model, so the latent variables of the VAE are jointly distributed with the visible data. For this reason the LF of the VAE is only available as an integral over the hidden variables. But, this distinction is moot because when looking at the asymptotic form of the PBN, an approximation that is very good as has been demonstrated [7], we see that the PBN *behaves* like a traditional layered stochastic generative model.

A network layer of a DLGM is composed of an arbitrary non-linear function followed by additive correlated noise [2]. A network layer of an asymptotic PBN, on the other hand, is composed of a non-linear function $\gamma_n^{-1}(\mathbf{z})$, followed by multiplication by matrix $\mathbf{W}_n$, then the layer output is produced by the generating distributions. Function $\gamma_n^{-1}(\mathbf{z})$ and the generating distributions depend on the range of the layer output variable and are given in Table I. When $\mathbf{x} \in \mathbb{R}^N$, the generating distribution is Gaussian, so and is implemented by adding independent Gaussian noise[2]. This produces a type of DLGM. But, the Gaussian noise in an asymptotic PBN must be added after a linear transformation, whereas for DLGM it is added after an arbitrary transformation. It is not clear what this distinction means to the ultimate PDF estimation capability, and can only be discovered by future experiments. Note also that for the DLGM, the activation function is taken to be part of the "arbitrary non-linear function" , whereas in the PBN, the activation function $\lambda_n(\ )$ is defined for the dual DAN and determines the function $\gamma_n^{-1}(\mathbf{z})$ used in the PBN. In holding to the MaxEnt principle, for a given data range $\mathcal{X}$, the activation function $\lambda()$ is fixed, and therefore $\gamma_n^{-1}(\mathbf{z})$ is fixed. But, if one is willing to give up this MaxEnt distinction, there is flexibility in choosing $\lambda()$ so long as it is invertible (for example use *softplus*, not *relu*).

In summary, both DLGM and PBN are layered generative networks and it is not clear from the above comparison which structure is better or more general. It is clear, however, that the PBN under special conditions (i.e. for $\mathcal{X} = \mathbb{R}^N$) approximates a type of DLGM and has a closed-form LF which is especially efficient to compute for this case (see [10] Section IV.C, page 2821). Future work is planned to compare DLGM and PBN in practice.

D. PBN Deterministic Form

The deterministic form of the PBN is obtained from the asymptotic form by replacing $p_n(x;\alpha)$ by their expected values $\lambda_n(\alpha)$. Interestingly, $\lambda_n(\alpha)$ cancels $\lambda_n^{-1}(\alpha)$, leaving $\gamma_n^{-1}(\ )$ as the only non-linearities, except at the visible layer. This resulting PBN is a deterministic dual to the DAN, which exactly recovers the hidden values. An arbitrary activation function $\lambda_n(\alpha)$ can be used as long as $\gamma_n(\mathbf{h}_n)$ is defined using the same function. Note that $\lambda_n(\alpha)$ must be invertible, so activations functions like *softplus* can be used, but not *relu*.

E. Sampling Efficiency

The sampling efficiency ϵ is the fraction of times that the PBN successfully creates a sample of visible data and depends on the feature generating distribution $g(\mathbf{z})$ and whether exact (UMS) or deterministic generation is used. A sampling failure occurs in a UMS block if the set $\mathcal{M}_m(\mathbf{z}_n, e_n)$ has no members, or in the asymptotic or deterministic PBN if $\gamma_n^{-1}(\mathbf{z}_n)$ has no solution. When sampling fails, it is necessary to re-start the process by drawing another feature value. Sampling efficiency, either for UMS or for deterministic PBN, can be driven towards 1.0 though training, as will be demonstrated below.

F. PBN Initialization and Training

In order to initialze the PBN so it has high sampling efficiency, the weight matrices should be initialized by principal component analysis (PCA) of the input data prior to the activation function [3]. Scaling and bias are then used to provide good "activation" of $\lambda_n(\ )$. In this paper, two types of PBN training are used - deterministic auto-encoder training and maximum likelihood (ML) training. In auto-encoder training, the DAN is combined with the deterministic PBN to form an auto-encoder (a clockwise circular path at the bottom of Figure 1). Training is accomplished using back-propagation to minimize total square reconstruction error. Note that the parameters appear in both PBN and DAN, so the derivative

[2]This can be easily extended to correlated noise by introducing a matrix multiplification between the layers.

[3]When data is already constrained to the range $[0,\ 1]$, as it is in the MNIST corpus, it is useful to "gaussianify" the data, mapping to $\mathbb{R}^N$ prior to PCA analysis (See Section III-A).

has two terms. It is critical to have high sampling efficiency for auto-encoder training. In the experiments, ϵ approaches very nearly 1.0 after the first training epoch, even for testing data.

In ML training, the log of equation (2) is trained for highest average value by gradient ascent. We used a special "uniform assumption" training in which the optional activation function (bottom of Figure 1) is applied to compress the data to the range [0,1], and the feature distribution $g(\mathbf{x}_3)$ is ignored. Ignoring the feature distribution is tantamount to assuming that $g(\mathbf{x}_3) = 1$, the uniform distribution. Interestingly, by training this way, a network is produced that, in fact, produces feature data $\mathbf{x}_3$ that is independent uniformly distributed - the simplifying assumption becomes fulfilled.

III. Classification Experiments

We now compare PBN with a Gaussian mixture model (GMM) in a simple classification task.

A. Reduced MNIST Data Description

For the following experiments, just three characters "3", "8", and "9", of the MNIST handwritten data corpus were used. Four pixel down-sampling rates were chosen: 1:1, 2:1, 3:1, and 4:1, resulting in data dimensions of 784, 196, 100, and 49. Since MNIST pixel data is coarsely quantized in the range [0,1], a dither was applied to the pixel values[4]. To create data in $\mathbb{R}^N$, the inverse sigmoid function was then applied in order to create "gaussianified" data with most pixel values in the range -10 and 10.

B. The 1-layer PBN

We revisit the 1-layer PBN, which was previously introduced [6]. The results of 1-layer PBN experiments are relevant to determine if the PBN should be exended to a second layer. In a multi-layer PBN, a given layer acts as a PDF model for the features of the up-stream layer. So, it seems that there is no advantage to adding a layer to a PBN if a GMM works better than the added layer. The idea, then is to test a 1-layer PBN against a GMM as a function of dimension. This experiment is data-set dependent, so the results here apply only to MNIST. As a performance benchmark, the GMM was applied to the "gaussianified" data in $\mathbb{R}^N$, using both diagonal (GMM-D), and full (GMM-F) covariance matrices [5]. A separate 1-layer PBN was initialized using PCA, then trained for each data class to maximize the mean log-likelihood using gradient ascent with "ADAM" optimization and L2 regularization using "uniform assumption" training (Section II-F). After training, the final activation function was removed, then $g(\mathbf{z}_1)$ was modeled as a GMM. For $N = 49, 100, 196, 784$, the number of hidden units (columns of matrix $\mathbf{W}$) were 12, 16, 30, and 34, respectively.

[4]For pixel values above 0.5, a small exponential-distributed random value was subtracted, but for pixel values below 0.5, a similar random value was added.

[5]To avoid singularities, the diagonal elements of the covariance matrices were multiplied by the factor $(1 + \delta)$, where $\delta = 0.3, 0.3, 0.5$, and 0.6 for dimensions 49, 100, 192, and 784, respectively.

Results of the experiment are shown in Figure 2. The PCA-initialized PBN, with no further training are reported as "PBN-P", and with training as "PBN-G". When comparing "PBN-P" with "PBN-G", we can conclude that ML training greatly improves a PBN. This means that the PDF model offered by a 1-layer PBN is more than a just a re-packaged type of Gaussian model or PCA. The next observation is that the PBN performs better than GMM-F above $N = 100$. Both GMM-F

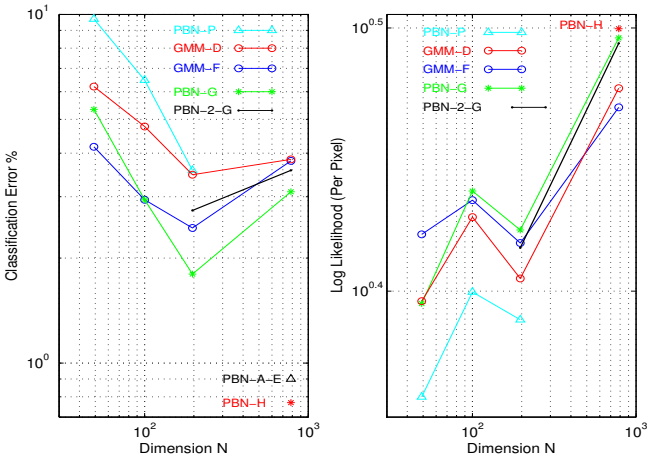

Fig. 2. Model comparison as a function of data dimension.

and PBN can model pixel correlation, GMM-F explicitly using the covariance matrices, and PBN implicitly by decorrelating the features, as was noted at the end of Section II-F. But, PBN requires MN parameters, versus the MN^2 parameters required for the GMM. This may explain the advantage of PBN above $N = 100$. The average sampling efficiency for PBN-G was 0.72, 0.85, 0.77, and 0.52 for $N = 49, 100, 192, 784$, respectively. The worst case change in per-pixel log-likelihood, is 0.007, so sampling efficiency in Figure 2 can be essentially ignored.

C. Multi-layer PBN

The 1-layer PBNs for $N = 196$ and 784 were extended to a second layer with 16 and 18 hidden units, respectively. The 2-layer PBNs were then trained with an assumption of uniform distribution for $g(\mathbf{x}_3)$, then the final activation function was removed and GMM was used to model the final feature PDF $g(\mathbf{z}_2)$. Sampling efficiencies were 0.55 and 0.70, respectively, also negligible. Performance is shown in Figure 2 as "PBG-2-G" and shows worse performance with respect to 1-layer PBN-G. This could have been predicted based on Figure 2 because the feature dimension is much less than 100. Extending the PBN to a second layer would only be effective if the first layer feature dimension is much larger.

IV. Auto-Encoder (A-E) Experiments

In the next experiment, a multi-layer deterministic PBN together with the DAN are used as an A-E and compared with a standard A-E network of the same structure. The full 28×28 ($N = 784$) data was used. Separate A-Es were trained

Nodes	Act	Type	E-Train	E-Test	Class
32-12	T	A-E	7.40	10.39	1.94%
32-12	S	A-E	6.73	10.79	2.97%
32-12	T	**PBN**	8.63	**9.04**	**1.27%**
36-16	T	A-E	5.84	8.21	2.57%
36-16	S	A-E	5.26	8.26	2.51%
36-16	T	**PBN**	6.96	**7.40**	**1.70%**
32-16-9	P	A-E	8.27	15.3	4.4%
32-16-9	P	**PBN**	9.95	**11.25**	**0.90%**

TABLE II
TOTAL SQUARE ERROR FOR AUTO-ENCODER TASK. ACTIVATION FUNCTIONS (ACT) ARE TED (T), SIGMOID (S) AND SOFTPLUS (P)

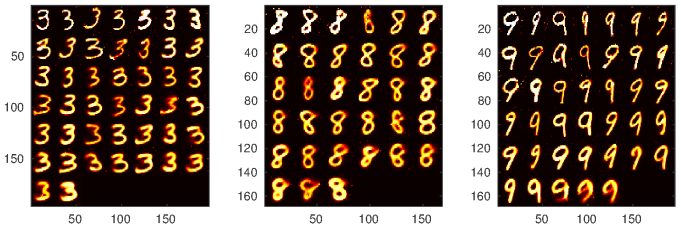

Fig. 3. Data synthesized from deterministic PBN and sorted in oder of decreasing likelihood value.

.

on each data class to minimize total square error by back-propagation. ADAM optimization and L2 regularization was used for both network types. TED (T), sigmoid (S) and softplus (P) activation functions were tried. The average squared error was measured for testing and training data and is listed in Table II. Although the conventional A-E attained a lower squared error on the training data, it fared much worse on the test data. In contrast, the PBN had similar squared error on both sets, significantly out-performing the standard A-E - which can probably be attributed to (a) that fact that the PBN uses the same weights for reconstruction and analysis, and thereby implements the same task with half the parameters, and (b) the reconstruction (PBN) is the perfect complement to the analysis network (DAN). Using L2-regularization for conventional A-E did not change this. The A-E performance for TED and sigmoid was similar, but training took longer for TED. Sampling efficiency for PBN was 100 percent (no samples that failed reconstruction) for training, and about 99.9% (typically 1 sample or less failed) on the test data. The good generalization of the PBN A-E suggests using it as a classifier based on minimum reconstruction error, which we tried. The results are shown in Table II in column "Class". PBN performed significantly better than A-E, attaining a very respectable 0.9%, which handily out-performs the standard PBNs in Figure 2 (denoted by "PBN A-E").

The deterministic PBN is also useful to generate entirely synthetic data, In Figure 3, examples were generated by training a GMM on the features (i.e. output of the DAN), then passing synthetic features through the PBN. The configuration "32-16-9" with softplus activation was used. The synthetic samples are sorted in order of decreasing likelihood (starting from top left), demonstrating the a benefit of a tractable likelihood function. The quality of these samples suggests using the deterministic PBN in a generative adversarial network (GAN) - but differing from a standard GAN in the posession of a tractable LF.

V. CONCLUSIONS

In this paper, a multi-layer PBN has been described, in its standard, asymptotic, and deterministic forms. Experiments comparing a 1-layer PBN with a GMM on a reduced subset of MNIST show that PBN out-performs GMM only above a dimension of about 100, which would suggest using a 2-layer PBN when the output dimension of the first layer is large. This paper also described a deterministic multi-layer PBN for the first time and it has been experimentally found to be superior to a standard auto-encoder when generalizing to test data both in terms of reconstruction error and classifier performance.

REFERENCES

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

[2] 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.

[3] 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.

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

[5] R. Salakhutdinov and I. Murray, "On the quantitative analysis of deep belief networks," *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 2008.

[6] 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.

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

[8] P. M. Baggenstoss, "The PDF projection theorem and the class-specific method," *IEEE Trans Signal Processing*, pp. 672–685, March 2003.

[9] S. M. Kay, A. H. Nuttall, and P. M. Baggenstoss, "Multidimensional probability density function approximations for detection, classification, and model order selection," *IEEE Transactions on Signal Processing*, vol. 49, pp. 2240–2252, Oct 2001.

[10] P. M. Baggenstoss, "Maximum entropy PDF design using feature density constraints: Applications in signal processing," *IEEE Trans. Signal Processing*, vol. 63, June 2015.

[11] P. M. Baggenstoss, "Evaluating the RBM without integration using pdf projection," in *Proceedings of EUSIPCO 2017, Island of Kos, Greece*, Aug 2017.

[12] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. Cambridge, MA: MIT press, 2016.