# Learning Generative Models Using Structured Latent Variables

by

Yichuan Tang

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

# Abstract

Learning Generative Models Using Structured Latent Variables

Yichuan Tang
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2015

Recent machine learning advances in computer vision and speech recognition have been largely driven by the application of supervised neural networks on large labeled datasets, leveraging effective regularization techniques and architectural design. Using more data and computational resources, performance is likely to continue to improve in the future. Despite these nice properties, supervised neural networks are sometimes criticized because their internal representations are opaque and lack the kind of interpretability that seems evident in human perception. For example, detecting a dog hidden in the bushes by looking at its exposed tail is a task that is not yet solved by discriminative neural networks. Another class of challenging tasks is one-shot learning, where only one training example for a new concept is available to the model for training. It is widely believed that learned prior knowledge must be utilized in order to tackle this problem.

My dissertation tries to address some of these concerns by introducing domain-specific knowledge to standard deep learning models. This domain-specific knowledge is used to specify meaningful latent representation with structure, which forces the model to generalize better under certain scenarios. For example, a generative model with latent gating variables that will "switch off" noisy pixels should perform better when encountering noise at test time. A generative model with latent variables representing 3D surface normal vectors should do better at modeling illumination variations. These are the kinds of relatively simple domain-specific modifications that we explore in this thesis. It is true that we should not rely too much on manual engineering and learn as much as possible. This principle is taken seriously and we strike a balance between using too much laborious engineering on the one hand and learning everything from scratch on the other. Adding structure to deep generative models is not only helpful for computer vision applications, but it is also very effective for unsupervised density learning tasks.

# Acknowledgements

None of this would have been possible without people, the people who have offered me guidance, inspiration and wisdom in these past 5 years. First and foremost, I would like to thank Geoffrey Hinton for being an incredible supervisor, teacher, and motivator. The most important virtues I hope to have learned from Geoff are not academic at all, but rather his passion, determination, and above all his compassion towards others. I would also like to thank my co-supervisor Ruslan Salakhutdinov for his amazing inspiration, guidance and support. Russ has shown me how an intensively positive attitude and incredible work ethic always lead to great things.

I would also like to thank the members of my committee who have given me great feedback: Rich Zemel, Raquel Urtasun, Sanja Fidler, and Bill Freeman. A big thanks must go out to the first class people of the lab throughout the years: Abdel-Rahman, Alex (Graves, Krizhevsky, and Schwing), Chris, Danny, Deep, Fernando, George, Graham, Greg, Hugo, Ilya, Jake, James, Jasper, Jimmy, Jon, Kevin, Laurent, Maks, Marc'Aurelio, Marcus, Nitish, Roger, Ryan (Kiros and Adams), Shenlong, Tijmen, Tyler, Vlad, Vinod, Yanshuai, Yujia, and others. None of the work in the Machine Learning group would happen without the work behind the scenes by Relu Patrascu and Luna Boodram. Finally, no words can express the role played by the people in my life, past to present, in motivating me and leading to the completion of this thesis.

# Contents


**1 Introduction** **1**

1.1 Contributions of This Thesis . . . 2

1.2 Summary of Remaining Chapters . . . 2

**2 Building Blocks** **4**

2.1 The Restricted Boltzmann Machine Family . . . 4

2.1.1 Binary Restricted Boltzmann Machines . . . 4

2.1.2 Gaussian Restricted Boltzmann Machines . . . 9

2.2 Other Models . . . 19

2.2.1 Higher-order Boltzmann Machines . . . 19

2.2.2 Autoencoders . . . 20

2.2.3 Factor Analysis . . . 20

2.3 Deep generative models . . . 22

2.3.1 Deep Belief Network . . . 22

2.3.2 Deep Boltzmann Machines . . . 23

**3 Generative models with structure for vision** **26**

3.1 Robust Boltzmann Machines . . . 27

3.1.1 The Model . . . 28

3.1.2 Properties of the model . . . 29

3.1.3 Inference . . . 30

3.1.4 Learning . . . 31

3.1.5 Experiments . . . 32

3.1.6 Shortcomings . . . 39

3.2 Deep Lambertian Neural Networks . . . 39

3.2.1 The Model . . . 41

3.2.2 Inference . . . 43

3.2.3 Learning . . . 45

3.2.4 Experiments . . . 45

3.2.5 Inference . . . 46

3.2.6 Relighting . . . 47

3.2.7 Recognition . . . 47

3.2.8 Generic Objects . . . 48

3.2.9 Shortcomings . . . 49

3.3 Conclusions and Future Work . . . 50

**4 Density learning with structure** **51**
4.1 Deep Mixture of Factor Analyzers . . . 52
4.1.1 Mixture of Factor Analysers . . . 53
4.1.2 Deep Mixtures of Factor Analysers . . . 54
4.1.3 Inference . . . 56
4.1.4 Learning . . . 57
4.1.5 Experiments . . . 57
4.1.6 Overfitting . . . 58
4.1.7 Qualitative Results . . . 59
4.1.8 High Dimensional Data . . . 59
4.1.9 Low Dimensional Data . . . 61
4.1.10 Natural Images . . . 61
4.1.11 Allocating more components to more popular Factor Analysers . . . 62
4.2 Learning Stochastic Feedforward Neural Networks . . . 62
4.2.1 Stochastic Feedforward Neural Networks . . . 64
4.2.2 Learning . . . 65
4.2.3 Cooperation during learning . . . 66
4.2.4 Experiments . . . 68
4.2.5 Synthetic datasets . . . 68
4.2.6 Modeling Facial Expression . . . 69
4.2.7 Additional Qualitative Experiments . . . 72
4.2.8 Computation Time . . . 74
4.3 Discussion . . . 74

**5 A higher-order generative model: The Tensor Analyzer** **75**
5.1 Related Works . . . 76
5.2 Preliminaries . . . 77
5.2.1 Factor Analyzers . . . 78
5.3 Tensor Analyzers . . . 78
5.4 Inference . . . 80
5.5 Learning . . . 80
5.5.1 Likelihood Computation . . . 82
5.5.2 Annealed Importance Sampling for TA . . . 82
5.5.3 Equality Constraints . . . 85
5.5.4 Mixture of Tensor Analyzers . . . 86
5.6 Experiments . . . 86
5.6.1 Synthetic Data . . . 86
5.6.2 Natural Images . . . 86
5.6.3 Face Recognition with equality constraints . . . 88
5.6.4 Learning with incomplete equality constraints . . . 90
5.6.5 Computation Time . . . 90
5.7 Shortcomings . . . 91

5.8 Conclusions . . . 91

**6 Learning generative models with visual attention** **92**
6.1 Motivation . . . 92
6.2 The Model . . . 94
6.3 Inference . . . 95
6.3.1 Approximate Inference . . . 97
6.4 Learning . . . 99
6.5 Experiments . . . 99
6.5.1 Approximate inference . . . 100
6.5.2 Generative learning without labels . . . 101
6.5.3 Inference with ambiguity . . . 102
6.6 Conclusion . . . 103

**7 Conclusion** **104**

**Bibliography** **106**

# List of Tables


2.1 AIS est. log-probability in nats. Average training log-probs are in parenthesis. FPCD-30 means the model was trained for 30 epochs using Fast Persistent Contrastive Divergence. 9

2.2 Toronto Face Database (TFD) (Susskind, 2011) is a dataset of 100,000 cropped face images. Berkeley database (Martin et al., 2001) contains natural image patches. TIMIT[1] is a database for speech recognition. TFD with translation is created by randomly translating a face within a noise background image. TFD with rotation is created by using random rotations. Higher numbers correspond to better models. . . . . 17

3.1 Peak Signal to Noise Ratio (PSNR) in dB for denoising on Yale faces for a hand-tuned and learned RoBM. The numbers are averages over 40 trials ± the standard error of the mean. . . . . 34

3.2 Recognition results on the AR Face Database. . . . . 39

4.1 Model performance on various high dimensional datasets (nats). TFD-24-Rot is generated by randomly rotating $24 \times 24$ face images in the range of $\pm 45$ deg. Diff-2 and Diff-3 are the gains from going from 1 to 2 layers and from 2 to 3 layers, respectively. For all datasets, Diff-2 and Diff-3 are statistically significant at $p = 0.01$. . . . . 60

4.2 Test set predictive log-likelihood on 4 UCI datasets (nats). Reported results are from 10-fold cross validation on each dataset. MFA results are from our experiments. Other results are from Silva et al. (2011). . . . . 61

4.3 Average test log-likelihood (in nats) of various models learned on 50,000 $8 \times 8$ test patches. **PIX**: independent pixels. **PCA**: Principle Component Analysis. **ICA**: Independent Component Analysis. **GMM**: Mixture of Gaussians with 200 components. **MFA** Mixture of Factor Analysers with 200 components. **MFA-2** Two layer DMFA. MFA and MFA-2 results are from our experiments, other numbers are taken from Zoran and Weiss (2011). GMM's $167.2^*$ is different from the previously reported 164.5 due to the random extraction of test patches. 167.2 was obtained by evaluating the downloaded model of Zoran and Weiss (2011) on our own test patches. . . . . 62

4.4 Average test log-probability density on synthetic 1D datasets. . . . . 69

4.5 Average test log-probability and total training time on facial expression images. Note that for continuous data, these are probability densities and can be positive. . . . . 70

4.6 Recognition accuracy over 5 folds. Bold numbers indicate that the difference in accuracy is statistically significant than the competitor models, for both linear and nonlinear classifiers. 72

6.1 Model architectures of the convolutional neural network used during approximate inference. 99

6.2 Face localization accuracy. $w$: image width; $h$: image height; $s$: image scales; $c$: number of inference steps used. . . . . 101
6.3 Variational lower-bound estimates on the log-density of the Gaussian DBNs (higher is better). . . . . 102

# List of Figures


2.1 Top: MNIST digits and learned filters from a 500 hidden nodes RBM using Fast Persistent Contrastive Divergence (FPCD) for 30 epochs. Bottom: Negative MNIST digits and learned filters from a 500 hidden nodes RBM using FPCD for 30 epochs. Notice some filters have not been used by the model. . . . . 6
2.2 Filters learned by FPCD on zero-meaned Negative MNIST. . . . . 9
2.3 Examples of GRBM trained on 2D synthetic data. The $\lambda$ is fixed to be 0.5 for both the x and y dimensions, and CD-1 is used for training. Best viewed in color. . . . . 13
2.4 A very imprecise density learned by the GRBM due to its affinity for hidden configurations with large means. . . . . 14
2.5 Preprocessing methods. Contrast normalization gets rid of background intensity variations, but does not lose any significant information for building a generative of objects. . . 15
2.6 Contrast normalization and its effect on learning precision. Note that for clarity in presentation, the display intensity range of the two plots are set differently. The "index" value are the standard deviation learnt for each pixel. . . . . 15
2.7 Samples of the model from the negative phase of learning, using contrast normalized data vs. standard data. Samples from the model trained on contrast normalized data (a) are clearly better and sharper compared to (b). . . . . 16
2.8 Activation function of the ssRBM (blue) is very similar to that of SSU's (Relu in black). Best viewed in color. . . . . 19
2.9 The RBM on the left has an equivalent DBN on the right. . . . . 22

3.1 The Robust Boltzmann Machine with real images demonstrating its latent representations. $\tilde{\mathbf{v}}$ is observed, $\mathbf{s}$ and $\mathbf{v}$ are inferred. $\mathbf{g}$ denotes the binary hidden variables of the RBM modeling the structure of the occluder, while $\mathbf{h}$ denotes the binary hidden variables of the RBM modeling the "clean" face. The model uses the higher layer RBMs to separate out the clean face and the occluder/noise. Best viewed in color. . . . . 28
3.2 Graphical model of the Robust Boltzmann Machine. Filled triangles indicate gating of the connection between $v_i$ and $\tilde{v}_i$ by $s_i$. The yellow connections are the weights of the RBM while the green connections are the weights of the GRBM. Each pixel is modeled by three random variables: $v_i$, $\tilde{v}_i$ and $s_i$. Best viewed in color. . . . . 29
3.3 Internal states of the RoBM during learning: columns from left to right represent epochs 1 to 50. The first row is the training data $\tilde{\mathbf{v}}$, the second row is the inferred $\mathbf{v}$, and the third row is the inferred $\mathbf{s}$. 20 Gibbs iterations were run to sample from the posterior. . . 34
3.4 Difference between various denoising algorithms for block occlusion. . . . . 35

3.5 Qualitative comparison of various denoising algorithms for random noise and occlusion. The first row has the original faces and the second row is corrupted with noise. Starting from the third row, we have denoising results from RoBM, RBM, PCA, Wiener filtering, and Nearest Neighbor, respectively. . . . . 36

3.6 Left: Methods are (a) RoBM, (b) RBM, (c) PCA, (d) Wiener, (e) Nearest Neighbor. Right: PSNR versus the number of Gibbs iterations used for sampling from the posterior during inference. . . . . 37

3.7 Recognition rates on the Yale Database as a function of the percentage of pixels corrupted by noise. Random noise with standard deviation of 0.5 was added to the corrupted pixels. Recognition rates on the Yale Database as a function of the percentage of pixels occluded. Block occlusion was applied as in Fig. 3.5(b). . . . . 38

3.8 Intermediate results during RoBM inference. The leftmost images are the test samples. . 38

3.9 Deep Belief Networks pay more attention to illumination variations rather than identity. Left: three person under three different lighting variations. Right: hidden layer activation clusters along the columns (lighting variations). . . . . 40

3.10 Diagram of the Lambertian Reflectance model. $\boldsymbol{\ell} \in \mathbb{R}^3$ points to the light source. $\vec{\mathbf{n}}_i \in \mathbb{R}^3$ is the surface normal, which is perpendicular to the tangent plane at a point on the surface. 41

3.11 Graphical model of the Deep Lambertian Network. The yellow weights model the surface normals while the green weights model the albedo. The arrow in the left figure is the light source direction vector, pointing towards the light source. Note that the light vector is shared for all pixels in the image. Best viewed in color. . . . . 42

3.12 Examples from the Yale B Extended face database. Each row contains samples from an illumination subset. . . . . 46

3.13 Random samples after 50,000 Gibbs iterations of the Deep Belief Network modeling the learned albedo prior. . . . . 47

3.14 **Left**: A single input test image. **Right**: Intermediate samples of estimated albedo during alternating Gibbs sampling: iterations 1 to 50. The albedo was initialized with the visible biases. . . . . 47

3.15 **Left**: Inference results when using only a single test image. The 1st column contains the test images and the 2nd column contains the albedos. **Middle**: Results improve slightly when using an additional test image with a different illumination. **Right**: Using the estimated albedo and surface normals, we show synthesized images under novel lighting conditions. . . . . 48

3.16 Recognition results on the Yale B face database. **NN**: nearest neighbor. **DBN**: Deep Belief Network. **Correlation**: normalized cross correlation. **SVD**: singular value decomposition. **DLN**: Deep Lambertian Network. . . . . 49

3.17 Inference conditioned on test objects, using 50 Gibbs iterations. **Top**: Images of objects under new illumination. **Bottom**: Inferred albedo. It is interesting how the model can estimate the albedo of the dark side of the cup. . . . . 49

4.1 **Left**: Hypothetical distribution of the latent factors of a *single* component. In this case the aggregated posterior is not Gaussian distributed. **Middle**: Illustration of our model for 2D data with each ellipse representing a Gaussian component. The first layer MFA has two components colored blue ($c = 1$) and red ($c = 2$). Their mixing proportions are given by $\pi_c$. For the blue component, we further learn a second layer MFA with three components. For the red component, we learn a separate second layer MFA with two components. We also introduce the second layer component indicator variable $k_c = 1, \ldots, K_c$, where $K_c$ is the total number of the second layer components associated with the first layer component $c$. $K_c$ is specific to the first layer component and need not be same for all $c$. In our example, $K_1 = 3$ and $K_2 = 2$. **Right**: Graphical model of a DMFA. Best viewed in color. . . . . 54
4.2 DMFA improves over MFA. Overfitting occurs during further training of a shallow MFA with increased capacity. Best viewed in color. . . . . 59
4.3 **Top**: training images. **Middle**: samples from MFA. **Bottom**: samples from DMFA. The red box highlights the worst looking samples of the two models while the green box highlights a very realistic looking generated sample. . . . . 60
4.4 Improvements of DMFA over standard MFA on $24 \times 24$ face images vs. the number of first layer components. Gains are observed across different numbers of first layer components. Surprisingly, while the dataset contains thousands of different people, more than 10 mixture components results in overfitting. Best viewed in color. . . . . 61
4.5 *Stochastic Feedforward Neural Networks.* **Left***: Network diagram. Red nodes are stochastic and binary, while the rest of the hiddens are deterministic sigmoid nodes.* **Right***: motivation as to why multimodal outputs are needed. Given the top half of the face* $\mathbf{x}$*, the mouth in* $\mathbf{y}$ *can be different, leading to different expressions.* . . . . 64
4.6 KL divergence and log-likelihoods. Best viewed in color. . . . . 67
4.7 *Three synthetic datasets of 1-dimensional one-to-many mappings. For any given* $x$*, multiple modes in* $y$ *exist. Blue stars are the training data, red pluses are* exact *samples from SFNNs. Best viewed in color.* . . . . 68
4.8 Samples generated from various models. SFNN samples are superior since they capture various expressions while preserving the identity of the conditioning face. . . . . 70
4.9 Plots demonstrate how hyperparameters affect the evaluation and learning of SFNNs. . . 71
4.10 **Left**: Noisy test images $\mathbf{y}$. Posterior inference in SFNN finds $E_{p(\mathbf{h}|\mathbf{x},\mathbf{y})}[\mathbf{h}]$. **Right**: generated $\mathbf{y}$ images from the expected hidden activations. . . . . 72
4.11 Samples generated from a SFNN after training on object and horse databases. Conditioned on a given foreground mask, the appearance is multimodal (different color and texture). Best viewed in color. . . . . 73
4.12 Samples from SFNN trained on rotated faces. . . . . 73

5.1 Illustration of why TA is needed. Three dimensional bases (left) span the pixel space of specific individuals. In order to properly generalize to a novel test subject, tensor interactions of the learned training bases are needed to form a new basis for modeling a test subject. . . . . 76
5.2 Diagram of TA's ($J = 2$) generative process. $\mathbf{\Lambda}_{new} = \mathcal{T} \bar{\times}_2 \mathbf{z}_2$ gives a new factor loading matrix for $\mathbf{z}_1$. $\mathbf{\Lambda}_{new}\mathbf{z}_1 \triangleq \mathbf{T}_{(1)}(\mathbf{z}_2 \otimes \mathbf{z}_1)$ determines the mean of $p(\mathbf{x}|\mathbf{z}_1, \mathbf{z}_2)$. . . . . 79

5.3 **left**: MCMC trace plot for TA learning on synthetic data. **right**:MCMC trace plot for TA learning on high dimensional face images. . . . . 83
5.4 Gibbs sampling quickly converges around 20 iterations. 9 components were randomly picked from a MTA with 200 components trained on natural image patches. For each component, we randomly selected 6 latent factors (one for every color). . . . . 84
5.5 **left**: AIS algorithm for TA. **right**: Experimental validation of AIS on a small model, where 100,000 Monte Carlo samples estimate the "true" log-likelihood. . . . . 85
5.6 TA vs. FA on 2D synthetic datasets. TAs can better model complex densities. . . . . 86
5.7 Natural image patches. (a) Training data. (b) Samples from MTA. (c) Each row contains filters from a different MTA component. (d) Average test log-probability comparisons. ICA: Independent Component Analysis. GRBM: Gaussian Restricted Boltzmann Machine (Hinton and Salakhutdinov, 2006). DBN: Deep Belief Nets (Hinton et al., 2006). Factored Bilinear: (Culpepper et al., 2011). . . . . 87
5.8 Performance of MTA vs. MFA with different numbers of mixture components. . . . . 88
5.9 Tensor Analyzer is able to simultaneously decompose a test image into separate identity and lighting factors. (a) 4 test images with frontal, left, right and top light sources. (b) Random samples with identity factor fixed to the inferred values from the test faces in (a). (c) Random samples with lighting factor fixed to the inferred values. (d) Comparison to other methods. Tensor decomposition methods require additional labels of lighting direction during the test phase. . . . . 89
5.10 UMIST Recognition errors. . . . . 90

6.1 **Left:** The Shifter Circuit, a well-known neuroscience model for visual attention (Olshausen et al., 1993); **Right**: The proposed model uses 2D similarity transformations from geometry and a Gaussian DBN to model canonical face images. Associative memory corresponds to the DBN, object-centered frame correspond to the visible layer and the attentional mechanism is modeled by 2D similarity transformations. . . . . 94
6.2 (a) HMC can easily get stuck at local optima. (b) Importance of modeling $p(\mathbf{u}|\mathbf{v}, \mathcal{I})$. Best in color. 97
6.3 A visual diagram of the convolutional net used for approximate inference. . . . . 98
6.4 Inference process: $\mathbf{u}$ in step 1 is randomly initialized. The average $\mathbf{v}$ and the extracted $\mathbf{x}(\mathbf{u})$ form the input to a ConvNet for approximate inference, giving a new $\mathbf{u}$. The new $\mathbf{u}$ is used to sample $p(\mathbf{v}|\mathcal{I}, \mathbf{u}, \mathbf{h})$. In step 3, one step of Gibbs sampling of the GDBN is performed. Step 4 repeats the approximate inference using the updated $\mathbf{v}$ and $\mathbf{x}(\mathbf{u})$. . . . 100
6.5 Example of an inference step. $\mathbf{v}$ is $24 \times 24$, $\mathbf{x}$ is $72 \times 72$. Approximate inference quickly finds a good initialization for $\mathbf{u}$, while HMC provides further adjustments. Intermediate inference steps on the right are subsampled from 10 actual iterations. . . . . 100
6.6 (a) Accuracy as a function of gaze initialization (pixel offset). Blue curve is the percentage success of at least 50% IOU. Red curve is the average IOU. (b) Accuracy as a function of the number of approximate inference steps when initializing 50 pixels away. (c) Accuracy improvements of HMC as a function of gaze initializations. . . . . 101
6.7 **Left:** Samples from a 2-layer DBN trained on Caltech. **Right:** samples from an updated DBN after training on CMU Multi-PIE *without* labels. Samples highlighted in green are similar to faces from CMU. . . . . 102

6.8 **Left:** Conditioned on different $\mathbf{v}$ will result in a different $\triangle\mathbf{u}$. Note that the initial $\mathbf{u}$ is exactly the same for two trials. **Right:** Additional examples. The only difference between the top and bottom panels is the conditioned $\mathbf{v}$. Best viewed in color. . . . . 103

# Chapter 1

# Introduction

A big part of machine learning is about building systems that are able to intelligently perceive the sensory data in their environment: images, videos, speech and natural language. The key to the success of these systems lies in their ability to extract high-level and semantically meaningful features from the sensory data. There are two major approaches in trying to achieve this goal: supervised learning of discriminative models and unsupervised learning of generative models. The advantage of supervised training is that we are optimizing for a simple final objective (typically classification error). The idea is that good classification performance by the model must mean that useful and meaningful features have been learned. The main disadvantage of supervised training is that curated label data are required for training. On the other hand, generative learning seeks to model the input data, without requiring any labels at all. With generative learning, we assume that if the model can efficiently and faithfully generate the data, we must have learned a good latent representation. In other words, we are trying to solve the perception problem using an analysis-by-synthesis approach. In this approach, perception can be accomplished by first learning a set of good parameters for synthesis, followed by the inference of the latent states of model. Generative models have deficiencies as well, the main challenge is learning high-dimensional data which have complicated data manifolds.

Recent advances in vision and speech recognition have been driven by the successful application of supervised neural networks to large datasets. Using effective regularization (Hinton et al., 2012) and a special 2D spatial architectural design (LeCun et al., 1998), the performance of these systems improves with more data and computational resources (Krizhevsky et al., 2012; Szegedy et al., 2014; He et al., 2015; Deng et al., 2013; Hannun et al., 2014). While these models are popular, they are sometimes criticized for their distributed representation. The hidden layers contain a group of nodes which acts as a population code to represent the input stimulus. This kind of representation lacks concept specificity which is likely to be present for human high-level reasoning. It is reasonable to expect a more semantically meaningful representation will lead to better generalization.

This thesis focuses on improving unsupervised generative models, where domain-specific knowledge is used to guide the models toward better performance. More specifically, many of the models introduced here will be based on the Deep Belief Network, but with the inclusion of additional latent variables. The chapters of the thesis share one major theme: an existing model is modified leading to a novel framework consisting of more structured latent variables. We take care to ensure that the novel frameworks have a good balance between human engineering and learning entirely from data.

## 1.1 Contributions of This Thesis

The most significant research contributions in this thesis are the following:

- Two methods for improved learning of binary Restricted Boltzmann Machines and Gaussian Restricted Boltzmann Machines. For binary Restricted Boltzmann Machines, we demonstrate that the non-intuitive idea of data normalization for binary data works well in improving the learning process. For Gaussian Restricted Boltzmann Machines, we analyze the inductive bias of the model in preferring components whose centers have large coordinate values. Using this insight, we propose a contrast normalization step in the training of Gaussian Restricted Boltzmann Machines, leading to improved training for images of objects.

- We show how to make Restricted Boltzmann Machines robust to noise and occlusion by introducing multiplicative gating and additional "clean" data variables. The structure of the occluder can also be learned after pre-training of the object model.

- We demonstrate how to incorporate surface normal, albedo, and lighting variables into a hidden layer of the Deep Belief Network. The resulting novel model can use posterior inference to separate illumination from shape in a single image, leading to better recognition.

- We show how to stack multiple layers of Mixture of Factor Analyzers on top of each other in the same way that stacking Restricted Boltzmann Machines forms a Deep Belief Network. This gives a consistent gain in model performance on a wide range of data.

- We propose a model for learning conditional distributions which can be multimodal. The proposed model is based on a standard feedforward neural network but with some hidden units being stochastic instead of deterministic. A learning algorithm based on Importance Sampling that optimizes the variational lower-bound is also provided.

- We extend Factor Analysis to the multilinear setting, where latent factors form groups and interact with each other multiplicatively. This model contains parameters which form a tensor and is capable of separating style from content during posterior inference.

- Finally, we develop a probabilistic framework with visual attention to learn generative models of objects within large images. Selective visual attention shifts objects of interest from anywhere in the image onto a smaller canonical representation, where a Deep Belief Network is used for generative modeling. Efficient bottom-up inference is performed by a Convolutional Neural Network in order to quickly explore the posterior distribution.

## 1.2 Summary of Remaining Chapters

**Chapter 2 Building Blocks**. In this chapter we provide the technical overview of the basic building blocks of deep generative models. The models include binary Restricted Boltzmann Machines and Gaussian Restricted Boltzmann Machines. We also analyze normalization techniques that improve the learning of those two models. We discuss how to combine building blocks into deeper models via layer-wise stacking. This results in deep models such as the Deep Belief Network and the Deep Boltzmann

Machine. There is also a brief discussion of other generative building blocks such as Factor Analyzers and denoising and contractive Autoencoders.

**Chapter 3 Generative models with structure for vision**. In this chapter we will detail two models related to computer vision. The first model is the Robust Boltzmann Machine, which allows Boltzmann Machines to be more robust to pixel noise and corruptions. Latent variables represent the spatial structure of the occluders and interact with the visible units using multiplicative gating. The second model is the Lambertian Deep Belief Network. It contains hidden variables corresponding to the surface normals and albedo of objects. By transferring learned knowledge from similar objects, albedo and surface normal estimation from a single image is possible in this model, leading to good one-shot face recognition performance when facing severe illumination variations. This chapter is based on the published papers of Tang et al. (2012b,c).

**Chapter 4 Density learning with structure**. In this chapter we discuss how generative density models can be improved by adding structure to the latent variables. The first model is called the Deep Mixture of Factor Analysers, where the aggregate posteriors of the latent factors of Factor Analysis are modeled by a second level Mixture of Factor Analysers. The second model consists of introducing stochastic hidden nodes into a standard feedforward neural network. Importance sampling in conjunction with error backpropagation can then be used to maximize the variational lower bound. This chapter is based on the published papers of Tang et al. (2012d); Tang and Salakhutdinov (2013).

**Chapter 5 A higher-order generative model: The Tensor Analyzer**. This chapter proposes a novel tensor based model which extends Factor Analysis to the multilinear setting. In a Tensor Analyzer, the latent factors form groups and higher-order multiplicative interactions occur between the groups during data generation. An efficient alternating Gibbs sampling algorithm is provided and we demonstrate its ability to perform style/content separation. The model can be learned using unsupervised, supervised and semi-supervised approaches. Its applications range from density modeling to separation of style and content (e.g. one-shot learning of face images under illumination). This chapter is based on the published paper of Tang et al. (2013).

**Chapter 6 Learning generative models with visual attention**. Building generative models of full images is challenging. This chapter shows how to leverage visual attention to learn generative models of objects of interest within high resolution images. A Deep Belief Network is employed as the deep generative model of small canonical views of objects. A geometric Similarity transformation routes input to the canonical frame, providing invariance to translation, rotation and scaling. The novel model introduced is fully probabilistic and is capable of learning objects of interest from large images after the pre-training stage. This chapter is based on the published paper of Tang et al. (2014).

**Chapter 7: Conclusions**. In this chapter we provide a brief summary of the thesis, summarizes our contributions, and provide a discussion for future work.

# Chapter 2

# Building Blocks

Most unsupervised learning models seek to model the structure or the probability density of data, without using high-level categorical labels. Algorithms vary in complexity, from simple ones like K-means to complicated ones such as Boltzmann machines and Bayesian nonparametric models. "Deep Learning" derives its name from forming a multilayer model where the basic building block of each layer is an unsupervised model. An example of this is the stacking of Restricted Boltzmann Machines on top of each other, leading to a Deep Belief Network. Within all unsupervised models, those with latent variables are particularly interesting. Latent variables can be learned to transform the data from its input representation to a more useful latent representation.

In this chapter we discuss several popular latent-variable, unsupervised models, which will form the basic building blocks of the remaining chapters in this thesis. Section 2.1 reviews the Restricted Boltzmann Machine, which will serve as the building block of both Deep Belief Networks and Deep Boltzmann Machines. Both the model equations and some novel insights into improved training will be discussed. Section 2.2.2 reviews an alternate model known as the Autoencoder. The main difference between Autoencoders and Restricted Boltzmann Machines is that the latter are probabilistic. Autoencoders have the advantage of easier training but they do not assign a probability density to data, so handling missing or noisy data cases can be a problem. Section 2.2 covers other popular models such as Mixture of Factor Analyzers and sparse models. Section 2.3 reviews the formulation of the Deep Belief Network and the Deep Boltzmann Machine.

## 2.1 The Restricted Boltzmann Machine Family

### 2.1.1 Binary Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is a type of Markov Random Field, or an undirected graphical model that has a bipartite structure with two sets of binary stochastic nodes: the visible $\mathbf{v} \in \{0,1\}^{N_v}$ and hidden $\mathbf{h} \in \{0,1\}^{N_h}$ layer nodes (Smolensky, 1986; Welling et al., 2005). $N_v$ is the number of visible nodes and $N_h$ is the number of hidden nodes. The RBM has visible to hidden connections but no intra-layer connections. For any configuration of the nodes, we can define an energy function as:

$$E_{RBM}(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i}^{N_v} b_i v_i - \sum_{j}^{N_h} c_j h_j - \sum_{i,j}^{N_v, N_h} W_{ij} v_i h_j \tag{2.1}$$

where $\theta = \{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$ are the model parameters. The probability distribution over configurations $\{\mathbf{v}, \mathbf{h}\}$ is:

$$p(\mathbf{v}, \mathbf{h}; \theta) = \frac{p^*(\mathbf{v}, \mathbf{h})}{\mathcal{Z}(\theta)} = \frac{\exp^{-E(\mathbf{v},\mathbf{h})}}{\mathcal{Z}(\theta)} \tag{2.2}$$

where we have used $p^*(\cdot)$ to represent the unnormalized probability distribution and $\mathcal{Z}(\theta) = \sum_{\mathbf{v},\mathbf{h}} \exp^{-E(\mathbf{v},\mathbf{h})}$ is the normalization constant. One reason for the popularity of the RBM is that it is easy to calculate the conditional distribution over either layer given the other.

$$p(\mathbf{v} = \mathbf{1}|\mathbf{h}; \theta) = \prod_i^{N_v} p(v_i = 1|\mathbf{h}) = \sigma\big(\mathbf{W}^\mathsf{T}\mathbf{v} + \mathbf{c}\big) \tag{2.3}$$

$$p(\mathbf{h} = \mathbf{1}|\mathbf{v}; \theta) = \prod_j^{N_h} p(h_j = 1|\mathbf{v}) = \sigma\big(\mathbf{W}\mathbf{h} + \mathbf{b}\big) \tag{2.4}$$

Where the $\sigma(x) = 1/(1 + \exp(-x))$ is the logistic sigmoid function. Unlike directed models, an RBM's conditional distribution over hidden nodes is factorial and very easy to compute. In addition, RBMs are trained with unsupervised Hebbian learning which makes them more biologically plausible than some of the other models.

### Learning

In practice, learning follows not the gradient of the log-likelihood but an approximate objective function, known as "Contrastive Divergence" or (CD) (Hinton, 2002). The idea of CD is to approximate the $\mathbb{E}_{model}[\cdot]$ by running a Gibbs chain for only 1 iteration, instead of $\infty$ as required for exact maximum likelihood learning. Specifically, the weights are updated as

$$\Delta\mathbf{W} = \alpha(\mathbb{E}_{data}[\mathbf{v}\mathbf{h}^\mathsf{T}] - \mathbb{E}_{recons}[\mathbf{v}\mathbf{h}^\mathsf{T}]) \tag{2.5}$$

where $\mathbb{E}_{recons}[\cdot]$ is found by starting a Gibbs chain with the data, and running one iteration by sampling the hidden given the visible, then reconstructing the visible given the hidden activations.

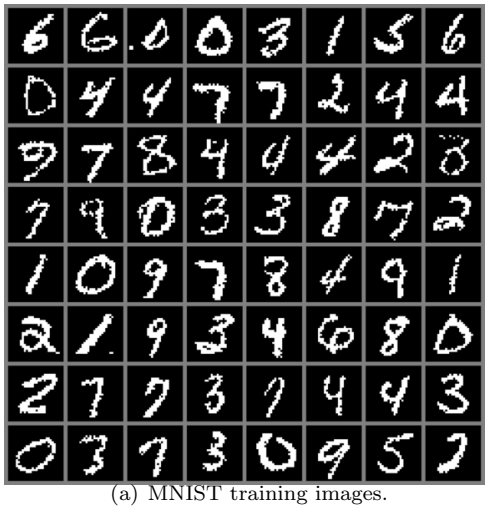
While CD can work well for certain problems (Hinton, 2002; Hinton et al., 2006; Larochelle et al., 2007), it is in general suboptimal and can leave undesirable modes in state space (Salakhutdinov and Murray, 2008). This is because by starting the Gibbs chain at the data and running for a short while, it often fails to explore other low energy parts of the state space. This led to the proposal of more advanced algorithms called Persistent Contrastive Divergence (Tieleman, 2008), Fast Persistent Contrastive Divergence (Tieleman and Hinton, 2009), and Coupled Adaptive Simulated Tempering (Salakhutdinov, 2010).

### Data Normalization

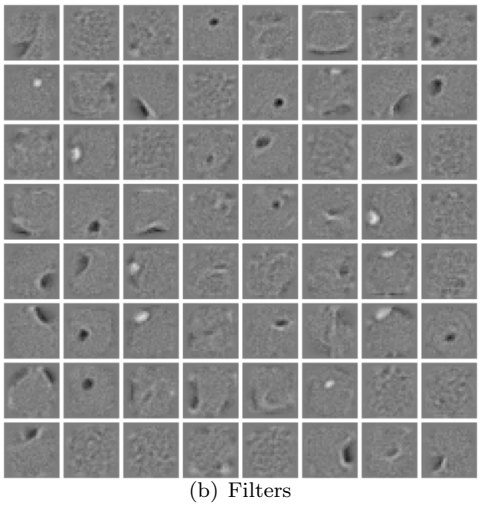
Using the aforementioned training algorithms, learning works well when input data vectors are sparse:

$$\Big(\frac{1}{N}\sum_n^N v_i^n\Big) \ll 1.0$$

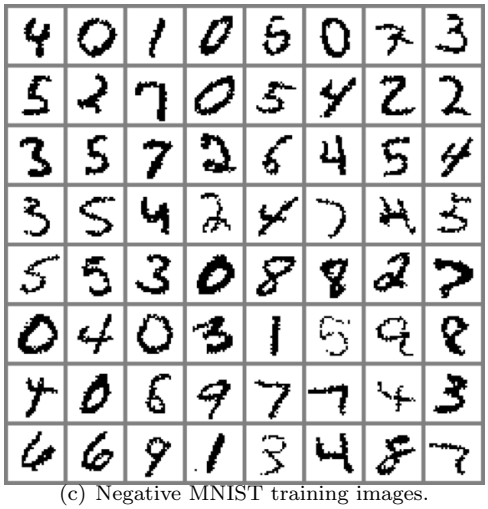
The standard MNIST digits are an example since they are white digits on a black background. We demonstrate this with the MNIST digits. In Fig. 2.1 top row, we learned a 500 hidden nodes RBM on the MNIST digits.



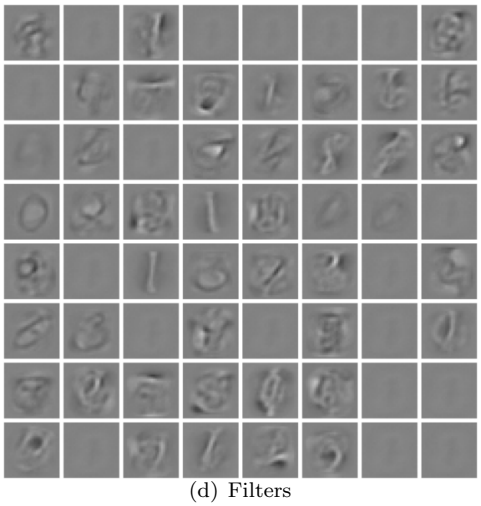
(a) MNIST training images.



(b) Filters



(c) Negative MNIST training images.



(d) Filters

Figure 2.1: Top: MNIST digits and learned filters from a 500 hidden nodes RBM using Fast Persistent Contrastive Divergence (FPCD) for 30 epochs. Bottom: Negative MNIST digits and learned filters from a 500 hidden nodes RBM using FPCD for 30 epochs. Notice some filters have not been used by the model.

However, when the input data mean is near 1.0, learning is considerably worse. On the bottom row of Fig. 2.1 we can create a new set of input data by using the negative images of MNIST. This transformation preserves structure in the underlying data density. Therefore, we expect the RBMs to learn equally well on the new dataset. Using the same training hyper-parameters as before, we obtain the filters which have many dead units.

Quantitatively, for the RBM trained on the standard MNIST, its average log-probability estimated using Annealed Importance Sampling (AIS) are -96 nats for both the training and test set.

However, for the RBM trained on Negative MNIST, its average log-probability estimated using AIS are -110 nats for both the training and test sets. The difference of 14 nats shows that the RBM learning

using approximate maximum likelihood is *prejudiced* against input data with large means.

We can find a solution to this asymmetric problem by normalizing the data. In most machine learning algorithms, it is considered to be standard protocol to remove the mean of data before training:

$$\begin{aligned} z_i &= v_i - \frac{1}{N}\sum_n^N v_i^n \\ &= v_i - \mu_i \end{aligned} \tag{2.6}$$

A ensuing whitening step can also be applied. Removing the mean usually allows the optimization to be easier. Data distribution with a non-zero mean would increase the eigenvalue of the Hessian of the objective function, affecting optimization detrimentally. While this can be done easily for any feedforward model, in the binary RBM case, we show that having real-valued inputs can still be used to effectively train.

Following this principle, we wish to learn an RBM on data with zero-mean, namely the variables $\mathbf{z} = \mathbf{v} - \boldsymbol{\mu}$. Note that our original visibles are $\mathbf{v} \in \{0,1\}^{N_v}$ and hidden $\mathbf{h} \in \{0,1\}^{N_h}$. The new zero-meaned variable $\mathbf{z}$ is still discrete, but its domain have been shifted by $\boldsymbol{\mu}$. For example, the domain of $v_i$ is $\{0,1\}$, while the domain of $z_i$ is $\{-0.3, 0.7\}$, if $\mu_i = 0.3$.

In this case, the energy function is:

$$E_{RBM}(\mathbf{z},\mathbf{h};\theta) = -\sum_i^{N_v} b_i z_i - \sum_j^{N_h} c_j h_j - \sum_{i,j}^{N_v,N_h} W_{ij} z_i h_j \tag{2.7}$$

$$= -\sum_i^{N_v} b_i (v_i - \mu_i) - \sum_j^{N_h} c_j h_j - \sum_{i,j}^{N_v,N_h} W_{ij}(v_i - \mu_i) h_j \tag{2.8}$$

The posterior distribution $p(h_j = 1|\mathbf{z})$ has the same form as the posterior in a standard RBM:

$$p(h_j = 1|\mathbf{z}) = \frac{1}{1 + \exp(-\mathbf{z}^\mathsf{T}\mathbf{W}_{(:,j)} - c_j)} \tag{2.9}$$

The conditional distribution of the visibles $z_j$ given the hidden states is:

$$p(z_i = 1 - \mu_i|\mathbf{h}) = \frac{\exp((1-\mu_i)\phi_i)}{\exp((1-\mu_i)\phi_i) + \exp(-\mu_i\phi_i)}, \quad where\ \phi_i = \mathbf{W}_{(i,:)}\mathbf{h} + b_i \tag{2.10}$$

The probability of the other state of $z_i$ is $p(z_i = 0 - \mu_i|\mathbf{h}) = 1 - p(z_i = 1 - \mu_i|\mathbf{h})$. We can simplify the above conditional distribution further by multiplying by $\frac{\exp((1-\mu_i)\phi_i)}{\exp((1-\mu_i)\phi_i)}$:

$$p(z_i = 1 - \mu_i|\mathbf{h}) = \frac{1}{1 + \exp(-\phi_i)} \tag{2.11}$$

$$p(z_i = 1 - \mu_i|\mathbf{h}) \equiv p(v_i = 1|\mathbf{h}) \tag{2.12}$$

The equivalence of the two conditional distributions $p(z_i = 1 - \mu_i|\mathbf{h})$ and $p(v_i = 1|\mathbf{h})$ means that we can implement this solution by easily modifying existing algorithms with minimal intrusion. Specifically, when sampling $z_i$ given $\mathbf{h}$, we first sample $v_i$ just like in the standard RBM (using Eq. 2.11). We can then obtain the value of $z_i$ by subtracting the mean from $v_i$: $z_i = v_i - \mu_i$. We stress that this operation is proper due to the equivalence of the two probability distributions in Eq. 2.12.

After learning the parameters $\{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$ of the RBM defined over $\mathbf{z}$ and $\mathbf{h}$, we can obtain an RBM over the original visibles $\mathbf{v}$:

$$E_{RBM}(\mathbf{z}, \mathbf{h}; \theta) = -\sum_{i}^{N_v} b_i(v_i - \mu_i) - \sum_{j}^{N_h} c_j h_j - \sum_{i,j}^{N_v, N_h} W_{ij}(v_i - \mu_i)h_j \tag{2.13}$$

$$= -\sum_{i} b_i v_i - \sum_{j} c_j h_j - \sum_{i,j} W_{ij} v_i h_j + \sum_{i,j} W_{ij}\mu_i h_j + \sum_{i} b_i \mu_i \tag{2.14}$$

$$= -\sum_{i} b_i v_i - \sum_{i,j} W_{ij} v_i h_j - \sum_{j}(c_j - \sum_{i} W_{ij}\mu_i)h_j + const. \tag{2.15}$$

The energy in Eq. 2.15 is equal (up to an additive constant) to the energy of an RBM over $\mathbf{v}$ with weights: $\mathbf{W}$, visible biases: $\mathbf{b}$, and hidden biases: $\mathbf{c} - \boldsymbol{\mu}^\top \mathbf{W}$.

We give a summarization of the Contrastive Divergence algorithm to learn parameters on zero-meaned data in Alg. 8. The three additions needed are in **bold**.

**Algorithm 1** Contrastive Divergence Training of RBM on zero-meaned data.

0: Randomly initialize the weights and biases to be small.
1: **Subtract data mean from all training data vectors: $\mathbf{z} = \mathbf{v} - \boldsymbol{\mu}$**
**for** $t = 1 : NumberEpochs$ **do**
**for** $n = 1 : NumberDataSamples$ **do**
*Positive Phase*:
2: Compute hidden activations and sample using Eq. 3.16.
3: Calculate the MLE gradient of the positive phase.
*Negative Phase*:
4: Compute reconstructions $\hat{v}_i$ and sample using Eq. 2.11.
5: **Compute reconstructions $\hat{\mathbf{z}}$: $\hat{\mathbf{z}} = \hat{\mathbf{v}} - \boldsymbol{\mu}$**
6: Compute hidden activations and sample using Eq. 3.16.
7: Calculate the MLE gradient of the negative phase.
8: Approx. gradient = positive phase gradient − negative phase gradient.
9: Update the parameters.
**end for**
**end for**
10: **Modify hidden biases: $\mathbf{c}_{new} \leftarrow \mathbf{c} - \boldsymbol{\mu}^\top \mathbf{W}$**

To demonstrate the advantage of RBM learning using zero-meaned data, we trained a 500 hidden nodes RBM with FPCD as well as using the zero-mean (ZM) normalization technique. Learning rate is fixed at 0.01, the weights are initialized from a Gaussian distribution with a standard deviation of 0.01, the visible and hidden biases are initialized to be 0.0. A weight penalty of 0.0002 is used for the visible to hidden weights. Training was run for 30 and 1000 epochs with 600 mini-batches per epoch. We presented the AIS estimated log-probability (nats) in Table 2.1.

| Dataset | FPCD-30 | ZM-30 | FPCD-1000 | ZM-1000 |
|---|---|---|---|---|
| MNIST | -96 (-96) | -94 (-94) | -84(-81) | -84 (-81) |
| Neg. MNIST | -110(-110) | -96 (-96) | -87(-85) | -84 (-81) |

Table 2.1: AIS est. log-probability in nats. Average training log-probs are in parenthesis. FPCD-30 means the model was trained for 30 epochs using Fast Persistent Contrastive Divergence.

For Negative MNIST, the model learned by FPCD was significantly worse than the equivalent model from MNIST. However, we see that it makes no difference for the ZM algorithm. For learning using 30 epochs, ZM is better than the standard FPCD on the original MNIST data. When training for 1000 epochs (probably achieving convergence), ZM training achieves the same log-prob for both the MNIST and Negative MNIST. In Fig. 2.2, we plot the learned filters of Negative MNIST using the ZM algorithm.

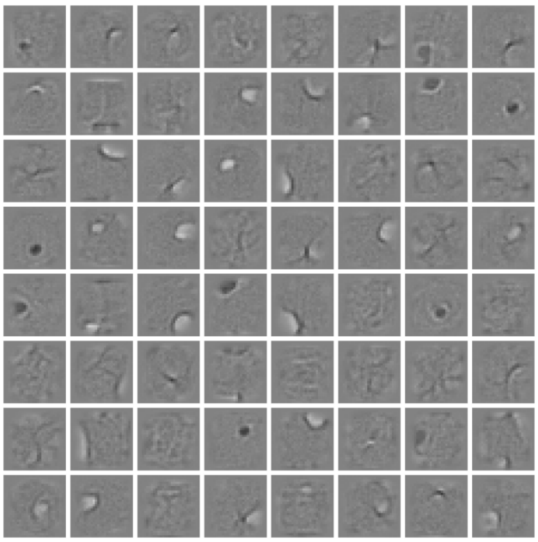


Figure 2.2: Filters learned by FPCD on zero-meaned Negative MNIST.

We have shown here that by simply subtracting the mean of the data prior to learning the parameters, we can achieve similar boost in learning performance. In addition, the extra incurred computation cost is one matrix subtraction for every step of Contrastive Divergence, which is negligible compared to the other matrix-matrix multiplications that are required.

### 2.1.2 Gaussian Restricted Boltzmann Machines

Binary Restricted Boltzmann Machines are designed for modeling binary data, as their visible units are Bernoulli random variables. For continuous and real-valued data such as images and speech, we can specify the visible units to be Gaussian. This leads to Gaussian Restricted Boltzmann Machines (GRBMs) (Hinton and Salakhutdinov, 2006). They have been successfully applied to tasks including image classification, video action recognition, and speech recognition (Lee et al., 2009; Taylor et al., 2010; Mohamed et al., 2011). GRBMs can be viewed as a mixture of diagonal Gaussians with shared parameters, where the number of mixture components is exponential in the number of hidden nodes. While there are more complicated models for density learning, GRBMs still remain very popular due to their ease of implementation. This is especially true if backpropagation will be used to fine-tune the

resulting deep network for a discriminative task. GRBMs have been used as the first layer of Deep Belief Nets for image category classification (Krizhevsky, 2009), and 3D object Recognition NORB (Nair and Hinton, 2010).

Formally, the GRBM is a bipartite Markov Random Field over the $N_v$ visible nodes $V \in \mathbb{R}^{N_v}$ and $N_h$ hidden nodes $H \in \{0,1\}^{N_h}$, defined by an energy function:

$$E(\mathbf{v}, \mathbf{h}; \theta) = \frac{1}{2}\mathbf{v}^\mathsf{T}\Lambda\mathbf{v} - \mathbf{v}^\mathsf{T}\Lambda\mathbf{b} - \mathbf{h}^\mathsf{T}\mathbf{c} - \mathbf{v}^\mathsf{T}\Lambda^{\frac{1}{2}}\mathbf{W}\mathbf{h} \tag{2.16}$$

$\theta = \{\mathbf{W}, \mathbf{b}, \mathbf{c}, \Lambda\}$ are the model parameters. $\Lambda$ is the diagonal precision matrix of $\mathbf{v}$, and ${\Lambda_{ii}}^{1/2} = \lambda_i = 1/\sigma_i$. Using the Gibbs distribution, we can exponentiate the energy function to obtain the probability density over the variables:

$$p(\{v_i, h_j\}) \propto \exp\big[-\frac{1}{2}\sum_i^{N_v}\frac{v_i^2}{\sigma_i^2} + \sum_i^{N_v}\frac{b_i v_i}{\sigma_i^2} + \sum_j^{N_h} c_j h_j + \sum_{ij}^{N_v, N_h}\frac{W_{ij}}{\sigma_i}v_i h_j\big] \tag{2.17}$$

For a given $V = \mathbf{v}$, marginalizing over all possible hidden configurations, we have

$$\log p^*(\mathbf{v}) = -\frac{1}{2}\mathbf{v}^\mathsf{T}\Lambda\mathbf{v} + \mathbf{v}^\mathsf{T}\Lambda\mathbf{b} + \sum_j^{N_h}\log\Big(1 + \exp\{\frac{1}{\sigma}\mathbf{v}^\mathsf{T}\mathbf{W}_{(:,j)} + c_j\}\Big) \tag{2.18}$$

$$\log p^*(\mathbf{h}) = \mathbf{c}^\mathsf{T}\mathbf{h} + \frac{1}{2}\hat{\mathbf{b}}^\mathsf{T}\Lambda\hat{\mathbf{b}}, \qquad \hat{b_i}(\mathbf{h}) = b_i + \sigma_i\sum_j W_{ij}h_j \tag{2.19}$$

Here we use $p^*(\cdot)$ to denote the unnormalized probability density function and $\mathbf{W}_{(:,j)}$ to mean the $j$-th column of matrix $\mathbf{W}$. Conditional distributions required for inference are:

$$p(h_j = 1|\mathbf{v}) = sigmoid\big(c_j + \frac{\mathbf{v}}{\boldsymbol{\sigma}}^\mathsf{T} W_{(:,j)}\big) \tag{2.20}$$

$$p(v_i|\mathbf{h}) \sim \mathcal{N}(b_i + \sigma_i\sum_j W_{ij}h_j, \sigma_i^2) \equiv \mathcal{N}(\mu_i(\mathbf{h}), \sigma_i^2) \tag{2.21}$$

Gradients for maximum likelihood learning are as follows:

$$\frac{\partial \log p(\mathbf{v})}{\partial \mathbf{W}} = \mathbb{E}_{data}[\frac{1}{\sigma}\mathbf{v}\mathbf{h}^\mathsf{T}] - \mathbb{E}_{model}[\frac{1}{\sigma}\mathbf{v}\mathbf{h}^\mathsf{T}] \tag{2.22}$$

$$\frac{\partial \log p(\mathbf{v})}{\partial \mathbf{b}} = \mathbb{E}_{data}[\frac{1}{\sigma^2}\mathbf{v}] - \mathbb{E}_{model}[\frac{1}{\sigma^2}\mathbf{v}] \tag{2.23}$$

$$\frac{\partial \log p(\mathbf{v})}{\partial \mathbf{c}} = \mathbb{E}_{data}[\mathbf{h}] - \mathbb{E}_{model}[\mathbf{h}] \tag{2.24}$$

Note that $\lambda_i = 1/\sigma_i$ is the inverse standard deviation of the model. We can also learn $\lambda_i$ by using the gradient:

$$\frac{\partial \log p^*(\mathbf{v})}{\partial \lambda_i} = -\lambda_i v_i^2 + 2\lambda_i v_i b_i + \sum_j^{N_h} p(h_j|\mathbf{v})v_i W_{ij} \tag{2.25}$$

Just like binary RBMs in Section 2.1.1, we can use CD, PCD and FPCD for learning the parameters of the GRBM. The only modification is the inclusion of the residual variance parameters as specified by the above equations. Additional layers of binary RBMs are often stacked on top of a GRBM to form

a Deep Belief Net (DBN) (Hinton et al., 2006). A DBN can be shown to increase the variational lower bound of the model's data log-likelihood. Inference in a DBN is approximate but efficient, where the probability of the higher layer states is a function of the lower layer states (see Equation 2.20).

### AIS for GRBM

The partition function for GRBMs is intractable, as in the case of binary RBMs. However, we have an algorithm to find an approximation of the partition function. In Salakhutdinov and Murray (2008), Annealed Importance Sampling (AIS) was applied to quantitatively evaluate binary RBMs. We now extend the AIS algorithm for estimating the log-partition function of a GRBM.

For any GRBM which defines an unnormalized probability $p^*_B(\mathbf{v})$ over the visible vector $\mathbf{v}$, we would like to estimate its log partition function $\log Z_B = \log \int_{\mathbf{v}} d\mathbf{v} p^*_B(\mathbf{v})$. Here we use $p^*(\cdot)$ to represent the unnormalized probability distribution. We require a simpler distribution $p^*_A(\mathbf{v})$ with a tractable log partition function $\log Z_A$ as well as a set of intermediate distributions:

$$p_k(\mathbf{v}) \propto p^*_A(\mathbf{v})^{1-\beta_k} p^*_B(\mathbf{v})^{\beta_k} \tag{2.26}$$

where $k = 1 \dots K-1$ and $0.0 = \beta_0 < \beta_1 < ... < \beta_K = 1.0$.

$p^*_B(\mathbf{v})$ is the GRBM distribution of interest. For ease of notation, we assume a spherical noise model by using $\sigma^2$, noting that it is trivial to use AIS with a diagonal noise model instead.

$$\log p^*_B(\mathbf{v}) = -\frac{1}{2\sigma^2}\mathbf{v}^\mathsf{T}\mathbf{v} + \frac{1}{\sigma^2}\mathbf{v}^\mathsf{T}\mathbf{b} + \sum_j^{N_h} \log\Big(1 + \exp\{\frac{1}{\sigma}\mathbf{v}^\mathsf{T}\mathbf{W}_{(:,j)} + c_j\}\Big) \tag{2.27}$$

For the tractable distribution $p^*_A(\mathbf{v})$, we use a spherical Gaussian with a mean at $\mathbf{b_A}$:

$$\begin{aligned}\log p_A(\mathbf{v}) = &- N_v \log(\sqrt{2\pi}) - N_v \log(\sigma) - \frac{1}{2\sigma^2}\mathbf{b_A}^\mathsf{T}\mathbf{b_A} \\ &- \frac{1}{2\sigma^2}\mathbf{v}^\mathsf{T}\mathbf{v} + \frac{1}{\sigma^2}\mathbf{v}^\mathsf{T}\mathbf{b_A} \qquad (2.28)\\ \log p^*_A(\mathbf{v}) = &- \frac{1}{2\sigma^2}\mathbf{v}^\mathsf{T}\mathbf{v} + \frac{1}{\sigma^2}\mathbf{v}^\mathsf{T}\mathbf{b_A} \qquad (2.29)\end{aligned}$$

Applying Eq. 2.26 to the joint density over $\mathbf{v}$ and $\mathbf{h}$, the unnormalized log-probability of an intermediate GRBM is:

$$\log p^*_k(\mathbf{v}, \mathbf{h}) = (1 - \beta_k) \log p^*_A(\mathbf{v}, \mathbf{h}) + \beta_k \log p^*_B(\mathbf{v}, \mathbf{h}) \tag{2.30}$$

Marginalizing out the hiddens:

$$\log p^*_k(\mathbf{v}) = -\frac{1}{2\sigma^2}\mathbf{v}^\mathsf{T}\mathbf{v} + \frac{1}{\sigma^2}\mathbf{v}^\mathsf{T}\tilde{\mathbf{b}} + \sum_j^{N_h} \log\Big(1 + \exp\{\beta_k(\frac{1}{\sigma}\mathbf{v}^\mathsf{T}\mathbf{W}_{(:,j)} + c_j)\}\Big) \tag{2.31}$$

where $\tilde{\mathbf{b}} = (1 - \beta_k)\mathbf{b_A} + \beta_k \mathbf{b}$. The conditional distributions needed for the Markov Chain transition operators are obtained from the intermediate GRBM's joint density:

$$p^*_k(\mathbf{v}, \mathbf{h}) = \exp\Big(-\frac{1}{2\sigma^2}\mathbf{v}^\mathsf{T}\mathbf{v} + \frac{1}{\sigma^2}\mathbf{v}^\mathsf{T}\tilde{\mathbf{b}} + \beta_k \mathbf{h}^\mathsf{T}\mathbf{c} + \frac{\beta_k}{\sigma}\mathbf{v}^\mathsf{T}\mathbf{W}\mathbf{h}\Big) \tag{2.32}$$

$$p_k(\mathbf{h}|\mathbf{v}) = \sigma\Big(\beta_k(\frac{1}{\sigma}\mathbf{v}^\mathsf{T}\mathbf{W} + \mathbf{c})\Big) \tag{2.33}$$

$$p_k(\mathbf{v}|\mathbf{h}) \sim \mathcal{N}(\tilde{\mathbf{b}} + \beta_k\sigma\mathbf{W}\mathbf{h}, \sigma^2) \tag{2.34}$$

Using these the modified MCMC transition operator associated to Equation 2.32, we can use the AIS algorithm outlined in Salakhutdinov and Murray (2008) to estimate the partition function. To see how accurate is the above procedure at estimating the log partition function of GRBMs, we trained two GRBMs with a small number of hidden nodes so that their log partition function can be calculated exactly. The first GRBM is trained on 2D synthetic data points with 6 hidden nodes. The second GRBM has 15 hidden nodes and is trained on a downsampled version of NORB which is $16 \times 16$ pixels. For the first GRBM, exact integration gave the log partition function of -43.11 nats, while the AIS estimation with 20,000 intermediate distributions and 100 chains gave the log partition function of -43.018 $\pm$ 0.096 nats (3 standard deviation). For the $16 \times 16$ NORB GRBM with 15 hidden nodes, the exact log partition function was -197.854 nats while AIS estimation gave -197.89 $\pm$ 0.077 nats.

### Learning

When building models such as deep autoencoders or deep neural networks for classification, GRBM training acts as a pretraining step. The goal of pretraining is to use unsupervised learning to find good weights to initialize a fully discriminative network (Hinton and Salakhutdinov, 2006). In these tasks, the ultimate goal is feedfoward inference where the residual variances are not involved, therefore, learning proper values are not very important. This led to many papers that used a fixed residual variance (Hinton and Salakhutdinov, 2006; Lee et al., 2007) e.g. $\sigma_i = 1$. This would be a gross approximation since while the marginal distribution over the entire dataset maybe 1.0, the residual variance conditioned on the hidden states should be much smaller than that. For density modeling and tasks such as denoising involving feedback processing, the residual variance is very important. This can be seen by looking at the log-probability equation that GRBM uses to assign log-prob to a test case:

$$\log p^*(\mathbf{v}) = -\frac{1}{2}\mathbf{v}^\mathsf{T}\Lambda\mathbf{v} + \mathbf{v}^\mathsf{T}\Lambda\mathbf{b} + \sum_j^{N_h}\log\Big(1 + \exp\{\mathbf{v}^\mathsf{T}\Lambda^{\frac{1}{2}}\mathbf{W}_{(:,j)} + c_j\}\Big) \tag{2.35}$$

Residual variance has a big effect on the log-probability assigned to a data vector. For example, in a Gaussian, the partition function involves the log determinant of the covariance matrix $\log p(\mathbf{v}) = const - \sum_i \log\sigma_i - \frac{1}{2}\sum_i \frac{(v_i-\mu_i)^2}{\sigma_i^2}$. A small log-determinant, or equivalently small residual variance $\sigma_i^2$, is desired for a high log-probability.

Therefore, we seek to learn a much smaller $\sigma_i$ from data, using Eq. 2.25. However, in practice, learning proper residual variance parameters $\sigma_i^2$ has been observed to be difficult using CD, PCD, or even FPCD algorithms (Hinton, 2010), requiring a much smaller learning rate and longer training time (Hinton and Salakhutdinov, 2006). A trick which is often employed is to not add any noise during reconstruction (Hinton, 2010). Even with many tricks, the learned $\sigma_i$ is often too large, about the same as the marginal variance of a pixel across the training set. Some papers have tried to use different types of RBM altogether (Le Roux et al., 2011). At the heart of the problem is that learning a small value for $\sigma_i$, or equivalently a big value for $\lambda_i$ is difficult even if it is warranted by the data. Without learning precise noise parameters for the visible nodes, the sampling required during the negative phase of learning would be very blurry and add too much noise to the gradients.

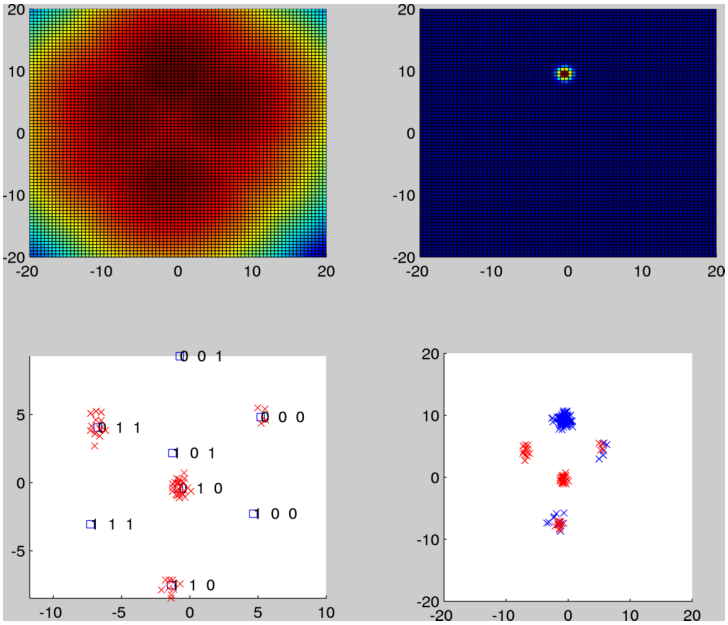



Figure 2.3: Examples of GRBM trained on 2D synthetic data. The $\lambda$ is fixed to be 0.5 for both the x and y dimensions, and CD-1 is used for training. Best viewed in color.

Here we show an example of how learning GRBMs fails for simple 2D synthetic data consisting of 4 clusters. Figure 2.3 shows a synthetic dataset where the the data points are red 'x' (bottom-left panel). A GRBM with 3 hidden units is trained on this dataset with CD-1. The top-left panel is the unnormalized log-probability over the 2D visible space as specified by the learned GRBM. The top-right panel is the normalized probability density. In the bottom-left panel, the binary hidden configurations of the 3 hidden nodes (8 configurations in total: $8 = 2^3$) are displayed in black. Each hidden configuration is associated with a conditional Gaussian mean in the visible space which are shown in blue squares, $\boldsymbol{\mu}(\mathbf{h})$, see Equation 2.21. The bottom-right panel plots the random samples (blue 'x') of the GRBM using MCMC with simulated annealing. Note that the GRBM places all of its mass around a *single* hidden configuration whose $\boldsymbol{\mu}(\mathbf{h})$ is *farthest* from the origin. It is also of interest that while CD learning can accurately place 4 component means on modes of the data, the GRBM does not assign any significant probability mass to the mixing proportion of those components.

The issue is **not** due to the learning algorithm (CD, PCD or FPCD), but rather a property of the GRBM model itself. The inability to learn a large $\lambda_i$ is related to the fact that posterior inference in the GRBM is designed to be factorial for computation efficiency purposes. To justify this argument, we first view the GRBM as an exponential (in $N_h$) mixture of diagonal Gaussians:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} \mathcal{N}(\mathbf{b} + \Lambda^{\frac{1}{2}}\mathbf{W}\mathbf{h}, \Lambda^{-1}) p(\mathbf{h}) \tag{2.36}$$

where the center of each diagonal Gaussian is $\boldsymbol{\mu}(\mathbf{h}) = \mathbf{b} + \Lambda^{\frac{1}{2}}\mathbf{W}\mathbf{h}$. Integrating out the visible nodes from Eq. 2.16, we find that the prior on each components of the mixture is:

$$p(\mathbf{h}) \propto \exp\{\mathbf{c}^\mathsf{T}\mathbf{h} + \frac{1}{2}\boldsymbol{\mu}(\mathbf{h})^\mathsf{T}\Lambda\boldsymbol{\mu}(\mathbf{h})\} \tag{2.37}$$

This prior will tend to be very big for components of the mixture with a big $\boldsymbol{\mu}(\mathbf{h})$[1]. In other words, the component with its center farthest from the origin will tend to be favored. This is exactly the reason why the GRBM in Figure 2.3 places all of its mass around coordinate (0,10). This affinity for components with large means is not a problem if we just want to learn useful hidden features. However, when it comes to density modeling, a model with small precision is still much better then a precise model with biased hidden component mixing proportions.

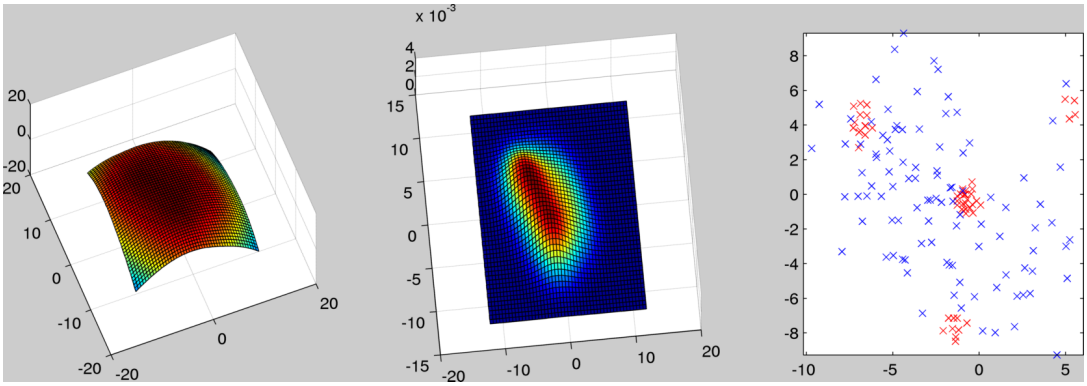



Figure 2.4: A very imprecise density learned by the GRBM due to its affinity for hidden configurations with large means.

This flaw of the GRBM as an (exponential) mixture density model means that when trained using PCD (in order to learn a better density model), a large $\boldsymbol{\lambda}$ for the visible nodes will mean that other than the favored component, none of the data "belonging" to other components will have any meaningful probability at all. Therefore, it is much better if the GRBM learned a small $\boldsymbol{\lambda}$ so that all data can be assigned a moderate density from the favored component. Figure 2.4 illustrates the density that is typically learned using PCD.

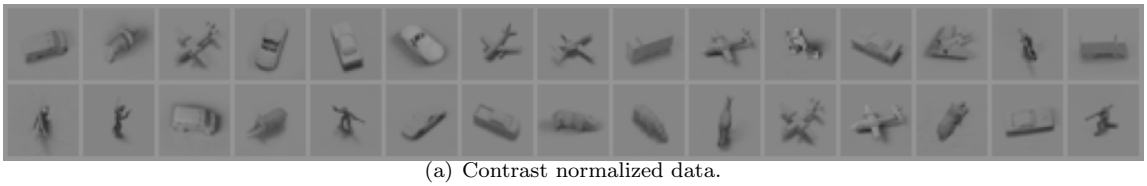
Since the culprit seems to be $\frac{1}{2}\boldsymbol{\mu}(\mathbf{h})^{\mathsf{T}}\Lambda\boldsymbol{\mu}(\mathbf{h})$ from Equation. 2.37, we might want to add a negative term in the GRBM's energy function to cancel it out. However, any term we want to add will have to be quadratic in $\mathbf{h}$ and will reintroduce the Explaining-Away problem during inference. In fact, adding such terms will lead us back to a directed graphical model without any of the fast inference advantages of a GRBM. Therefore, the "bad prior" property of GRBM is an inductive bias of the model and a side effect of having a factorial posterior.

### A Remedy

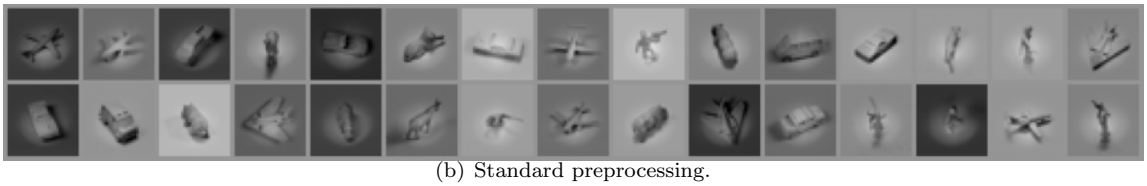
Instead of modifying the model, we will add a data preprocessing step to ameliorate the above mentioned "bad prior" problem. The standard preprocessing for training a GRBM (on images) consist of removing the mean of each pixel across training samples and also dividing each pixel by its standard deviation over training samples. Instead of the standard procedure, we advocate the alternative method of subtracting from each pixel the mean of all pixels within a single image and then dividing all pixels by a value such that the norm of the image vector is a constant. In computer vision, this method is known as contrast normalization and is often included as a preprocessing step to account for affine transformations of image intensity. Our reason for using this normalization technique is to keep the inductive bias of the probability assigned to the hidden configurations more-or-less the same, irrespective of the data cluster

[1]The bias $\mathbf{c}$ typically only has a small influence.

location. This is a sensible method to use since in high dimensional space, contrast normalization simply removes image intensity variations and does not lose any critical information of the object. As a concrete

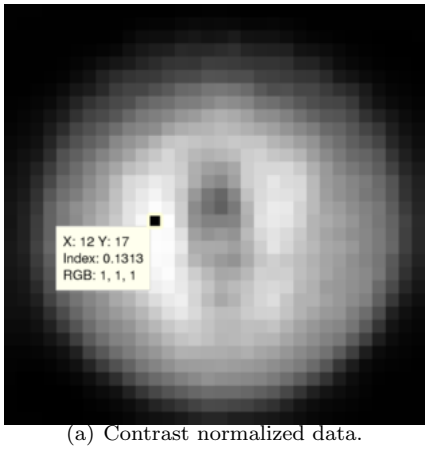


(a) Contrast normalized data.

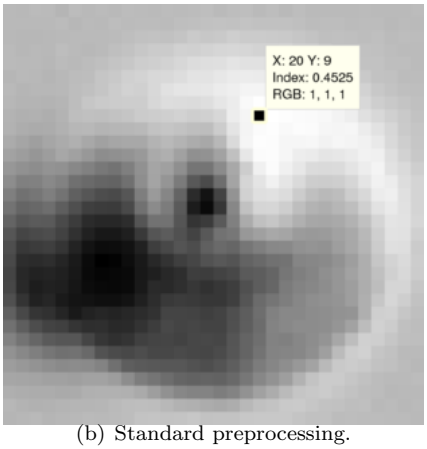


(b) Standard preprocessing.

Figure 2.5: Preprocessing methods. Contrast normalization gets rid of background intensity variations, but does not lose any significant information for building a generative of objects.



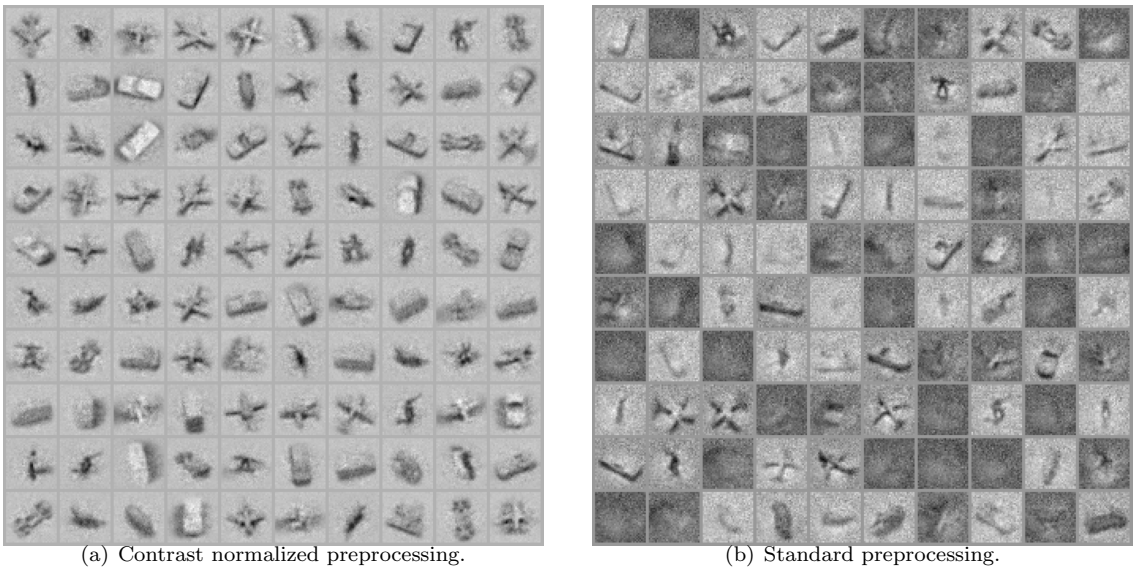

(a) Contrast normalized data.




(b) Standard preprocessing.

Figure 2.6: Contrast normalization and its effect on learning precision. Note that for clarity in presentation, the display intensity range of the two plots are set differently. The "index" value are the standard deviation learnt for each pixel.

example, we trained a GRBM on the NORB dataset (LeCun et al., 2004). Figure 2.5 shows the input images based on (a) contrast normalization and (b) standard mean and standard deviation normalization. Visually, all the important information of the objects is preserved by contrast normalization. We trained a GRBM with 1000 hidden nodes using CD-1 for 100 epochs. We then looked at the residual standard deviation learned for every pixel. In Figure 2.6, we show the learnt standard deviation of the visible nodes. Note the different intensity scaling of each subfigure. The contrast normalized GRBM is able to learn a much more appealing variance of the pixels with much smaller values on average. Residual variances are bigger in the middle where different object parts appearances creates more variance. Note that highest standard deviation learned from contrast normalized data is 0.13 while for the normal preprocessed data it is much higher at 0.45. We also show the negative particles of each model during learning in Figure 2.7. Comparing them to the training data, we can see that the negative samples for



(a) Contrast normalized preprocessing. (b) Standard preprocessing.

Figure 2.7: Samples of the model from the negative phase of learning, using contrast normalized data vs. standard data. Samples from the model trained on contrast normalized data (a) are clearly better and sharper compared to (b).

the contrast normalized version are much sharper. In particular, there are a lot of blob-like clouds in Figure 2.7 (b).

**Stepped Sigmoid Units**

A simple method to increase the dynamic range of the hidden units of the GRBM is proposed by Nair and Hinton (2010), called Stepped Sigmoid Units (SSUs). For every hidden unit in the original GRBM, $K$ copies of it with the same weights and bias can be instantiated. Each copy has a different but fixed offset: $-0.5, -1.5, -2.5, \ldots$. This gives hidden units more dynamic range in response to input $x$ without adding any additional parameters. During learning, instead of sampling $K$ times more hidden units as before, we can simply compute the expected value of the sum of all copies. The expected value of the sum copies is approximately $\log(1 + e^x)$ as $K$ approaches infinity (Nair and Hinton, 2010). After training, we can convert a SSU back into a standard GRBM by replicating the hidden nodes $K$ times with the fixed bias offsets. These units have been shown to be better at classification and this helped to popularize Rectified Linear Units (ReLUs) in deep learning.

In addition, SSUs are also better generative models than GRBMs. While a GRBM is an exponential mixture of diagonal Gaussians, a SSU is an exponential mixture of Gaussians with non-diagonal covariances. A hidden unit $h_j$ of a SSU can be either off ($h_j = 0$) or activate approximately linearly with respect to the incoming signal. There are $2^{N_h}$ hidden configurations for a SSU with $N_h$ hidden units, where each hidden configuration gives a different linear model. Hence, a SSU is an exponential mixture of linear models[2]. Therefore, SSUs can model correlation in the data more efficiently than GRBMs. However, the "bad prior" problem as discussed in the above subsection is still present for the SSUs, since every SSU has an equivalent GRBM.

We compared GRBMs and SSUs as density models on two image databases and one audio database.

[2]Although most of the configurations are only assigned a very small prior probability.

Annealed Importance Sampling is used to estimate the log partition functions. For the GRBM, the Fast PCD algorithm (Tieleman and Hinton, 2009) was used where the fantasy particles were run for 20 Gibbs iterations between parameter updates. The learning rate is linearly decayed to 0 and its initial value was picked using a validation set. For the SSU, Fast PCD was also used. The number of replicated hidden nodes $K$ was picked using the validation set, typically the best $K$ was between 4 and 6.

| Dataset | GRBM (nats) | SSU (nats) |
|---|---|---|
| $12 \times 12$ pixels Toronto Face Database | $\mathbf{-19}$ | $\mathbf{-27}$ |
| $48 \times 48$ pixels Toronto Face Database | **2835** | **3345** |
| $24 \times 24$ TFD with translation | **32.2** | **36.7** |
| $24 \times 24$ TFD with rotation | **296** | **352.1** |
| TIMIT acoustic spectrograms | **1175** | **1268** |
| Berkeley $8 \times 8$ image patches | $\mathbf{-98}$ | $\mathbf{-105}$ |

Table 2.2: Toronto Face Database (TFD) (Susskind, 2011) is a dataset of 100,000 cropped face images. Berkeley database (Martin et al., 2001) contains natural image patches. TIMIT[1] is a database for speech recognition. TFD with translation is created by randomly translating a face within a noise background image. TFD with rotation is created by using random rotations. Higher numbers correspond to better models.

Table 2.2 summarizes the results. The values are the test set log-probability in nats. For the data with higher dimensionality, the SSU performed better than the GRBM. However, for the data with lower dimensionality (e.g. $12 \times 12$ and $8 \times 8$ images), GRBMs perform better.

SSUs are very similar to a later proposed model called the Spike and Slab RBM (ssRBM) (Courville et al., 2011). Motivated by the inability of Gaussian RBMs to model non-diagonal input correlations efficiently, a ssRBM contains spike (the original binary hidden variables) as well as slab (continuous) hidden variables. The idea is that for every particular binary hidden configuration, the continuous slab variables result in a non-diagonal Gaussian model of the input data. Learning requires alternating sampling between the spike and slab variables.

To see the similarity, we note that the energy function of the ssRBM can be written as:

$$-E(\mathbf{v}, \mathbf{s}, \mathbf{h}) = -\frac{1}{2}\mathbf{v}^\mathsf{T}\Lambda\mathbf{v} + \sum_{i=1}^{N}\left(\mathbf{v}^\mathsf{T}W_i s_i h_i - \frac{1}{2}s_i^2 + c_i h_i\right) \tag{2.38}$$

where an isotropic Gaussian prior is placed on $\mathbf{s}$ (slab variables). Conditioned on $\mathbf{h}$ (spike variables), the model becomes an undirected Factor Analysis model.

Instead of having both the slab and spike variables, hidden units of an SSU correspond to a new variable $g_i$ which is obtained by multiplying the spike and the slab: $g_i = h_i s_i$. Let us also define a useful function that enforces $s$ to be positive:

$$H(s) = \begin{cases} s & \text{if } s \geq 0 \\ -\infty & \text{else} \end{cases} \tag{2.39}$$

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

If we modify the ssRBM energy function by enforcing $s$ to only be positive,

$$-E(\mathbf{v},\mathbf{s},\mathbf{h}) = -\frac{1}{2}\mathbf{v}^\mathsf{T}\Lambda\mathbf{v} + \sum_{i=1}^{N}\left(\mathbf{v}^\mathsf{T}W_i s_i h_i - \frac{1}{2}H(s_i)^2 + c_i h_i\right), \tag{2.40}$$

then the latent variable of interest is $g_i$, and we are interested in the expected value

$$\begin{aligned}\mathbb{E}_{p(h_i,s_i|\mathbf{v})}[g_i] &= \int_{h_i,s_i} h_i s_i p(h_i,s_i|\mathbf{v}) dh_i ds_i \\ &= \int_{h_i,s_i} g_i \frac{\exp(\mathbf{v}^\mathsf{T}W_i s_i h_i - \frac{1}{2}H(s_i)^2 + c_i h_i)}{\int_{h_i,s_i}\exp(\mathbf{v}^\mathsf{T}W_i s_i h_i - \frac{1}{2}H(s_i)^2 + c_i h_i)} dh_i ds_i \\ &= \frac{\int_{h_i,s_i} g_i \exp(\mathbf{v}^\mathsf{T}W_i s_i h_i - \frac{1}{2}H(s_i)^2 + c_i h_i) dh_i ds_i}{\int_{h_i,s_i}\exp(\mathbf{v}^\mathsf{T}W_i s_i h_i - \frac{1}{2}H(s_i)^2 + c_i h_i) dh_i ds_i}\end{aligned} \tag{2.41}$$

denoting $\phi_i = \mathbf{v}^\mathsf{T}W_i$ and because $h_i$ is binary, the numerator is expanded into integrals with $h_i = 0$ and $h_i = 1$:

$$0 + \int_0^\infty s_i \exp(\phi_i s_i - \frac{1}{2}s_i^2 + c_i) ds_i \tag{2.42}$$

$$= \sqrt{2\pi}e^{c_i + \frac{1}{2}\phi_i^2}\int_0^\infty s_i \mathcal{N}(\phi_i, 1) ds_i \tag{2.43}$$

and the denominator is

$$\begin{aligned}&\sqrt{2\pi}\int_0^\infty (2\pi)^{-\frac{1}{2}}\exp(-\frac{1}{2}s_i^2) ds_i + \sqrt{2\pi}e^{c_i+\frac{1}{2}\phi_i^2}\int_0^\infty (2\pi)^{-\frac{1}{2}}\exp^{-\frac{1}{2}(s_i-\phi_i)^2} ds_i \\ =&\sqrt{2\pi}(\frac{1}{2} + e^{c_i+\frac{1}{2}\phi_i^2}\Phi(\phi_i))\end{aligned} \tag{2.44}$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal. Therefore,

$$\begin{aligned}\mathbb{E}_{p(h_i,s_i|\mathbf{v})}[g_i] &= \frac{\sqrt{2\pi}e^{c_i+\frac{1}{2}\phi_i^2}\int_0^\infty s_i\mathcal{N}(\phi_i,1)ds_i}{\sqrt{2\pi}(\frac{1}{2} + e^{c_i+\frac{1}{2}\phi_i^2}\Phi(\phi_i))} \\ &= \frac{\int_0^\infty s_i\mathcal{N}(\phi_i,1)ds_i}{(\frac{1}{2}e^{-c_i-\frac{1}{2}\phi_i^2} + \Phi(\phi_i))}\end{aligned} \tag{2.45}$$

We plot the expected value of $g_i$ in blue as a function of $\phi_i$ (if we assume $c_i = 1.0$):

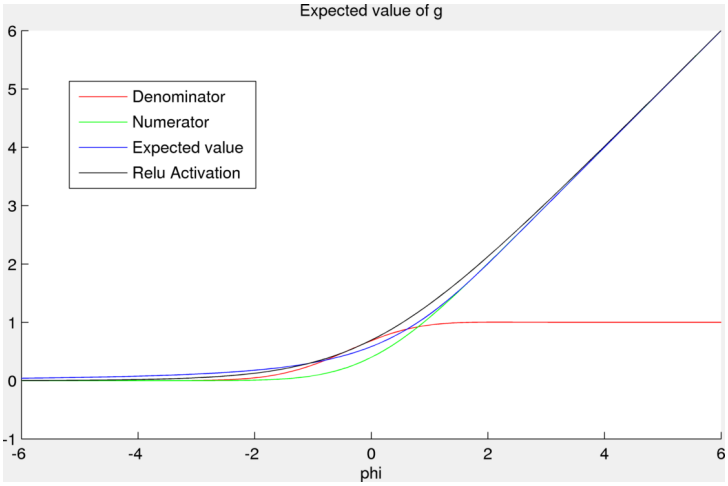



Figure 2.8: Activation function of the ssRBM (blue) is very similar to that of SSU's (Relu in black). Best viewed in color.

As $\phi_i >> 0$, $\mathbb{E}[g_i] = \phi_i$: the numerator of Eq. 2.45 approaches the expected value of a Gaussian centered at $\phi_i$, while the denominator approaches 1.0. The above analysis shows that the expected value of SSU's hidden units is very similar to the expected value of the ssRBM's spike $h_j$ times the slab variable $s_j$.

In conclusion, we find that using contrast normalization as the preprocessing step led to better generative models learned by both the GRBM and the SSU. Learning a very precise noise model has a tremendous effect on the log-probability assigned to the data by the generative model. SSUs are typically better than GRBM for data with larger dimensionality, however their performance is sensitive to the number of replicated hidden units $K$.

## 2.2 Other Models

### 2.2.1 Higher-order Boltzmann Machines

We briefly discuss some of the other types of Boltzmann Machines that have been used as building blocks for deeper models. The hidden units of a single RBM still model only linear correlations among the visible units. In other words, there are no visible-visible interaction or multiplications in an RBM. Therefore, an RBM is not suitable for modeling higher-order correlations among the visible units.

Multiplications among the visible units are introduced in (Memisevic and Hinton, 2010; Ranzato and Hinton, 2010; Memisevic and Hinton, 2010), leading to a model known as covariance RBMs and the mean and covariance RBM (mcRBM). Instead of having 1 visible layer and 1 hidden layer, the visible layer is duplicated, resulting in 2 visible (although identical) layer and 1 hidden layer. A factorized tensor is used to model the interaction between the 3 groups of variables. Results on modeling natural images have shown this type of model is effective at learning Gabor-like colorful filters from real life datasets.

Third-order RBMs have been used as powerful classifiers for object recognition. In an implicit mixture of RBMs (Nair and Hinton, 2009b), one-of-K class codes gate the slices of a tensor which leads to a different RBM per class. Training can proceed using the Contrastive Divergence algorithm, resulting

in a more flexible and higher-order RBM mixture model.

### 2.2.2 Autoencoders

Feedforward neural networks are usually used to transform input data (images or other modalities) into output class labels. In the absence of labels, the target output could be assigned to be the original input data, where the objective is to reconstruct the data as well as possible. Since the model seeks to transform or encode the data into a code space before decoding again, this type of model is known as an autoencoder. In order to prevent the network from learning the identity function, regularization of the parameters of autoencoders by restricting the dimensionality of the codes are needed. Autoencoders are useful for performing nonlinear compression of data and learning efficient representation or codes useful for additional tasks. Autoencoders are a type of unsupervised learner, not requiring any labels for training.

A better way to train multilayer or deep autoencoders was proposed by Hinton and Salakhutdinov (2006), where already trained Restricted Boltzmann Machines were stacked on top of one another in order to help optimization. That paper is arguably the seminal paper of deep learning, as it popularized the idea of "pre-training".

Instead of mapping the data back to itself, we can first corrupt the data randomly and try to denoise. This is the central aim of Denoising autoencoders (Vincent et al., 2010; Bengio et al., 2013). One way to corrupt the input is to randomly select a subset of the features and set their value to zero. During training, different features are set to zero during every weight update. The idea of denoising autoencoders is to learn the low-dimensional manifold of the data, since projecting a noisy data vector back to its clean version will create an energy ravine around the data manifold in data space.

Another recent model in the Autoencoder family is the Contractive Autoencoder (Rifai et al., 2011b,a). Contractive Autoencoders differ from standard Autoencoders in the addition of a regularization term which penalizes the Jacobian of the hidden nodes with respect to the input. In other words, the objective encourages the encoder to not change much around training data points. However, the overall objective also includes the reconstruction error of the input. Learning balances these two objectives and will end up learning an encoder which changes only along the data manifold.

Sparse versions of Autoencoders have also been proposed (Ranzato et al., 2007; Henaff et al., 2011). In sparse Autoencoders, the activities of the hidden layers are encouraged to be sparse. A simple criterion which restricts the information content of the code is proposed, thereby leading to the learning of informative features.

Central to all of the aforementioned autoencoders is the data reconstruction cost. A particularly novel feature learning method known as Sparse Filtering proposes the radical idea of not caring about reconstruction cost (Ngiam et al., 2011). Arguing against the reconstruction cost in terms of the added computation cost, Sparse Filtering simply tries to find projections of the data which lead to both population sparsity and lifetime sparsity of the hidden features. Empirical results showed Sparse Filtering to be an effective way of unsupervised pre-training before supervised fine-tuning.

### 2.2.3 Factor Analysis

Factor analysis was first introduced in psychology as a latent variable model to find the "underlying factors" behind covariates. The latent variables are called factors and are of lower dimension than the

covariates. Factor analyzers are linear models as the factor loadings span a linear subspace within the vector space of the covariates. To deal with non-linear data distributions, Mixtures of Factor Analyzers (MFA) (Ghahramani and Hinton, 1996) can be used. MFAs approximate nonlinear manifolds by making local linear assumptions.

Let $\mathbf{x} \in \mathbb{R}^D$ denote the $D$-dimensional data, $\{\mathbf{z} \in \mathbb{R}^d : d \leq D\}$ denote the $d$-dimensional latent variable, and $c \in \{1, \ldots, C\}$ denote the component indicator variable of $C$ total components. The MFA is a directed generative model, defined as follows:

$$p(c) = \pi_c, \quad \sum_{c=1}^{C} \pi_c = 1, \tag{2.46}$$

$$p(\mathbf{z}|c) = p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), \tag{2.47}$$

$$p(\mathbf{x}|\mathbf{z}, c) = \mathcal{N}(\mathbf{x}; \mathbf{W}_c\mathbf{z} + \boldsymbol{\mu}_c, \boldsymbol{\Psi}_c), \tag{2.48}$$

where $\mathbf{I}$ is the $d \times d$ identity matrix. The parameters of the $c$-th component include a mixing proportion $\pi_c$, a factor loading matrix $\mathbf{W}_c \in \mathbb{R}^{D \times d}$, mean $\boldsymbol{\mu}_c$, and a diagonal matrix $\boldsymbol{\Psi}_c \in \mathbb{R}^{D \times D}$, which represents the independent noise variances for each of the variables.

By integrating out the latent variable $\mathbf{z}$, a MFA model becomes a mixture of Gaussians with constrained covariance:

$$p(\mathbf{x}|c) = \int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z}, c)p(\mathbf{z}|c)d\mathbf{z} = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \Gamma_c) \tag{2.49}$$

$$\Gamma_c = \mathbf{W}_c\mathbf{W}_c^{\mathsf{T}} + \boldsymbol{\Psi}_c$$

$$p(\mathbf{x}) = \sum_{c=1}^{C} \pi_c \, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \Gamma_c). \tag{2.50}$$

For inference, we are interested in the posterior:

$$p(\mathbf{z}, c|\mathbf{x}) = p(\mathbf{z}|\mathbf{x}, c)p(c|\mathbf{x}) \tag{2.51}$$

The posterior over the components can be found using Bayes rule:

$$p(c|\mathbf{x}) = \frac{p(\mathbf{x}|c)p(c)}{\sum_{\gamma=1}^{C} p(\mathbf{x}|\gamma)p(\gamma)} \tag{2.52}$$

Given component $c$, the posterior over the latent factors is also a multivariate Gaussian:

$$p(\mathbf{z}|\mathbf{x}, c) = \mathcal{N}(\mathbf{z}; \mathbf{m}_c, \mathbf{V}_c^{-1}), \tag{2.53}$$

where

$$\mathbf{V}_c = \mathbf{I} + \mathbf{W}_c^{\mathsf{T}}\boldsymbol{\Psi}_c^{-1}\mathbf{W}_c,$$
$$\mathbf{m}_c = \mathbf{V}_c^{-1}\mathbf{W}_c^{\mathsf{T}}\boldsymbol{\Psi}_c^{-1}(\mathbf{x} - \boldsymbol{\mu}_c).$$

Maximum likelihood learning of a MFA model is straightforward using the EM algorithm (Rubin and Thayer, 1982). During the E-step, Eqs. 2.52, 5.3 are used to compute the posterior over the latent

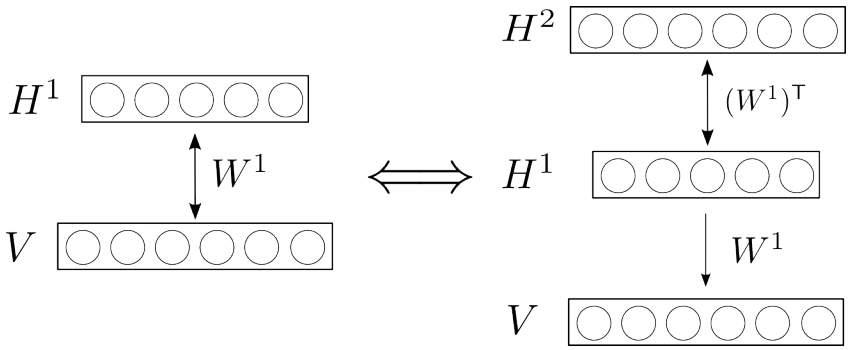



Figure 2.9: The RBM on the left has an equivalent DBN on the right.

variables given the current setting of the model parameters. During the M-step, the expected complete-data log-likelihood is maximized with respect to the model parameters $\theta = \{\pi_c, \mathbf{W}_c, \boldsymbol{\mu}_c, \boldsymbol{\Psi}_c\}_{c=1}^{C}$:

$$\mathbb{E}_{p(\mathbf{z},c|\mathbf{x};\theta_{old})}[\log p(\mathbf{x}, \mathbf{z}, c; \theta)]$$

## 2.3 Deep generative models

We briefly review Deep Belief Networks (DBNs) and Deep Boltzmann Machines (DBMs).

### 2.3.1 Deep Belief Network

A Deep Belief Network is a hybrid of directed and undirected graphical model where the top layer is an RBM and subsequent lower layers form a directed graphical model known as a Sigmoid Belief Network (Neal, 1992). DBNs are attractive due to their capabilities as a generative model as well as the greedy layer-wise training algorithm. DBNs are built from stacking together RBMs and have been successfully adapted to handwritten digit recognition, dimensionality reduction, natural image modeling, and speech recognition (Hinton et al., 2006; Hinton and Salakhutdinov, 2006; Osindero and Hinton, 2007; Mohamed et al., 2011).

An RBM defines a joint distribution which can be factored into a prior and a conditional:

$$p(\mathbf{v}, \mathbf{h}) = p(\mathbf{v}|\mathbf{h})p(\mathbf{h}) \tag{2.54}$$

Using this definition, the original RBM has an equivalent 2 layer network shown in Fig. 2.9. The network on the right is a DBN composed by stacking 2 RBMs on top of each other. The weight for the top RBM $(W^1)^\mathsf{T}$ is the transpose of the bottom RBM $W^1$. Note that the top RBM has undirected connections while the bottom RBMs function as a directed network.

For any distribution approximating the true posterior $q(\mathbf{h}^1|\mathbf{v}) \approx p(\mathbf{h}^1|\mathbf{v})$, we have a variational lower bound on the log probability of the data

$$\log p(\mathbf{v};\theta) = \textstyle\sum_{\mathbf{h}^1} q(\mathbf{h}^1|\mathbf{v})\Big[\log p(\mathbf{v}, \mathbf{h}^1; \theta)\Big] + \mathcal{H}(q(\mathbf{h}^1|\mathbf{v})) + KL(q(\mathbf{h}^1|\mathbf{v})||p(\mathbf{h}^1|\mathbf{v})) \tag{2.55}$$

$$\geq \textstyle\sum_{\mathbf{h}^1} q(\mathbf{h}^1|\mathbf{v})\Big[\log p(\mathbf{v}|\mathbf{h}^1; W^1) + \log p(\mathbf{h}^1; W^2)\Big] + \mathcal{H}(q(\mathbf{h}^1|\mathbf{v})) \tag{2.56}$$

$\mathcal{H}(\cdot)$ is the entropy functional and $KL(\cdot)$ is the Kullback-Leibler divergence. When the RBM is formu-

lated as a 2 layer DBN as in Fig. 2.9, the bound is tight.

Greedy learning of a DBN requires the freezing of $W^1$ and learning $W^2$ to optimize this lower bound. This is essentially maximizing

$$\sum_{\mathbf{h}^1} q(\mathbf{h}^1|\mathbf{v}) \log p(\mathbf{h}^1; W^2) \tag{2.57}$$

At first, when $W^2 \equiv (W^1)^\mathsf{T}$, any increase in the lower bound will increase the log-likelihood, since the bound is tight. However, when the bound is not tight ($KL(q||p) > 0$), changing $W^2$ to increase the bound might actually decrease the actual log-likelihood, since $KL(q||p)$ might decrease more. Such could be the situation when greedily learning deeper layers. Despite a lack of guarantee, in practice the CD algorithm tends to find good weights to model $p(\mathbf{v}|\mathbf{h}^1; W^1)$, and learning a $W^2$ will better model $p(\mathbf{h}^1; W^2)$, thus improving the DBN.

Algorithm 2 summarizes the greedy learning of a DBN. For a $L$ layer DBN, its joint probability can be defined as

$$p(\mathbf{v}, \mathbf{h}^1, \ldots, \mathbf{h}^{L-1}, \mathbf{h}^L) = p(\mathbf{v}|\mathbf{h}^1) \ldots p(\mathbf{h}^{L-1}, \mathbf{h}^L) \tag{2.58}$$

**Algorithm 2** Greedy Learning DBN

1: Learn $W^1$ for the 1st layer RBM.
2: Freeze $W^1$, learn 2nd layer RBM with $q(\mathbf{h}|\mathbf{v})$ as its visible layer data.
3: Freeze $W^2$, learn 3rd layer RBM with $q(\mathbf{h}^2|\mathbf{h}^1)$ as its visible layer data.
4: Recursively learn as many layers as desired.

To generate data from a DBN, the standard way is to run Gibbs sampling for the top RBM, then propagate down the stochastic activation to the visible layer. For posterior inference, $q(\cdot)$ is used to approximate the true posterior which is in general intractable to compute.

### 2.3.2 Deep Boltzmann Machines

While the DBN is an hybrid generative and discriminative network, the DBM is a multilayer network which is entirely undirected and defined by a single energy function. In this section, we briefly discuss the DBM as an alternative deep generative model to the DBN.

A 3 layer DBM is defined by the energy function

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3) = -\mathbf{v}^\mathsf{T} W^1 \mathbf{h}^1 - (\mathbf{h}^1)^\mathsf{T} W^2 \mathbf{h}^2 - (\mathbf{h}^2)^\mathsf{T} W^2 \mathbf{h}^3 \tag{2.59}$$

$$p(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3) = \frac{1}{Z} \exp^{-E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3)} \tag{2.60}$$

the conditional distributions over each of the layers are given by[3]

$$p(v_i = 1|\mathbf{h}^1) = \sigma\Big(\sum_j W^1_{ij} h^1_j\Big) \tag{2.61}$$

$$p(h^1_j = 1|\mathbf{h}^2, \mathbf{v}) = \sigma\Big(\sum_i W^1_{ij} v_i + \sum_k W^2_{jk} h^2_k\Big) \tag{2.62}$$

[3]We ommitted the biases for clarity of presentation.

$$p(h_k^2 = 1|\mathbf{h}^3, \mathbf{h}^1) = \sigma\Big(\sum_j W_{jk}^2 h_j^1 + \sum_l W_{kl}^3 h_l^3\Big) \tag{2.63}$$

$$p(h_l^3 = 1|\mathbf{h}^2) = \sigma\Big(\sum_k W_{kl}^3 h_k^2\Big) \tag{2.64}$$

Approximate maximum likelihood learning can be used to learn the model. However, unlike in the learning of RBMs, the first term in the objective involves the calculation of data-dependent expectations, is no longer independent given the visible nodes due to the multiple hidden layers. For DBMs, we can use a variational approach to estimate the data-dependent expectations. For any value of the parameters $\theta$, we can decompose the log-likelihood of the data as

$$\log p(\mathbf{v};\theta) = \sum_{\mathbf{h}} q(\mathbf{h}|\mathbf{v};\mu)\Big[\log p(\mathbf{v},\mathbf{h};\theta)\Big] + \mathcal{H}(q) + KL(q(\mathbf{h}|\mathbf{v};\mu)||p(\mathbf{h}|\mathbf{v};\theta)) \tag{2.65}$$

$$\geq \sum_{\mathbf{h}} q(\mathbf{h}|\mathbf{v};\mu)\Big[\log p(\mathbf{v},\mathbf{h};\theta)\Big] + \mathcal{H}(q) \tag{2.66}$$

where we have used $\mathbf{h}$ above to include all hidden nodes. In variational learning, we first optimize $\mu$ to tighten the lower bound of $\log p(\mathbf{v};\theta)$. As a consequence of the decomposition, it also means we minimize the Kullback-Leibler divergence between the approximating and true posteriors. For simplicity, we take the approximating posteriors to be $q(\mathbf{h}|\mathbf{v};\mu) = \prod_{j=1}^{N_h} q(h_j)$, where $\mu_j \equiv q(h_j = 1)$. We can solve for $\mu_j$ by running mean-field fixed-point equations to convergence

$$\mu_j^1 \leftarrow \sigma\Big(\sum_i W_{ij} v_i + \sum_k W_{jk}\mu_k^2\Big) \tag{2.67}$$

$$\mu_k^2 \leftarrow \sigma\Big(\sum_j W_{jk}\mu_j^1 + \sum_l W_{kl}\mu_l^3\Big) \tag{2.68}$$

$$\mu_l^3 \leftarrow \sigma\Big(\sum_k W_{kl}\mu_k^2\Big) \tag{2.69}$$

where the superscript of the $\mu$'s indicate the layer of the hidden nodes. Empirically, 25 iterations of this mean-field updates guarantees convergence. Additionally, damping by using a momentum term can also help convergence. After finding variational parameters $\mu_j$, learning proceeds by optimizing $\theta$ while fixing $\mu$.

By using mean-field for approximating $p(\mathbf{h}|\mathbf{v})$, the assumption is that the true conditional distribution is unimodal. This assumption is often not a bad one to make when the task at hand is to interpret visual or auditory data. It is also advantageous for $\mathbf{h}$ to be unimodal if the system requires further processing (Salakhutdinov and Hinton, 2009; Hinton et al., 2006).

In order for the DBM to work in practice, a pre-training step is also required similar to DBNs. The pre-training is also greedy and layer-wise and will initialize the weights at a better location before learning. In addition, sparsity constraints[4] on the hidden layers are also found to be critical for DBM learning.

The main advantage of the DBM is that the conditional distribution of any intermediate layer is a function of both the bottom layer and the top layer. In contrast, the approximate inference procedure for DBN is only a function of the bottom layer. This fact allows for the top-down influence to alter

[4]Typically the hidden layers nodes are encouraged to have an average activation between 0.1 and 0.2.

the purely bottom-up inputs and allows for the posterior inference to take into account the correlation between hidden nodes. Empirically, Salakhutdinov and Hinton (2009) has demonstrated that DBMs is slightly better than DBNs on MNIST and NORB, both for generative modeling and discriminative performance.

The main disadvantage of the DBM is the computational resources used during learning and influence. Let $n$ be the number of mean-field iterations needed during variational influence, that's $n$ times more expensive than the approximate inference computation needed for a DBN. This problem can be significantly alleviated by using separate weights to "predict" the variational parameters $\mu$ (Salakhutdinov and Larochelle, 2010).

In the remaining chapters of this thesis, the generative model of the novel frameworks we propose is the Deep Belief Network. The fundamental ideas and general tenants of our approach will not change had we used the Deep Boltzmann Machine instead. However, we chose to use Deep Belief Networks because they are easier to train and the results are comparable to Deep Boltzmann Machines for many tasks explored in this thesis.

# Chapter 3

# Generative models with structure for vision

Deep learning based probabilistic generative models include the Deep Belief Network (Hinton et al., 2006), the Deep Boltzmann Machine (Salakhutdinov and Hinton, 2009), and Sigmoid Belief Networks (Neal, 1992). These models are very generic regarding the types of data they are capable of modeling. They are all very flexible, each one of them being a jack-of-all-trades model for any data, from images to text to acoustic signals. However, this flexibility means that some simple and very useful domain-specific knowledge is not being employed to increase accuracy and improve generalization. Specifically, domain-specific knowledge from vision can easily be incorporated into a model to allow for better generalization. At the same time, the new model can still learn statistically from the data and learn useful latent representations.

In this chapter we look at two glaring shortcomings of deep generative models and look toward computer vision for simple ideas to address these concerns. The first model allows us to better deal with occlusion and noise of the input. Occlusion and noise affecting one part of the input image should not affect all of the hidden latent variables. This can be achieved with multiplicative interactions at the pixel level which is not part of the standard models. Section 3.1 introduces a simple way to make Boltzmann machines more robust to noise and corruptions, the Robust Boltzmann Machine. It contains additional binary noise indicators which allow Boltzmann machines to be robust to corruptions. In the domain of visual recognition, the Robust Boltzmann Machine is able to accurately deal with occlusions and noise by using multiplicative gating to induce a scale mixture of Gaussians over pixels. Image denoising and inpainting correspond to posterior inference in the Robust Boltzmann Machine. This model is trained in an unsupervised fashion with unlabeled noisy data and can learn the spatial structure of the occluders. Compared to standard algorithms, the Robust Boltzmann Machine is significantly better at recognition and denoising on several face databases.

The second model allows us to better deal with lighting or illumination variations for improved recognition. Section 3.2 leverages the Lambertian image formation process and combines it with a Deep Belief Network to provide better modeling of illumination variations. The idea is to first estimate an illumination invariant representation before using it for recognition. The object albedo and surface normals are examples of such representations. In this section, we introduce a multilayer generative model where the latent variables include the albedo, surface normals, and the light source. Combining

Deep Belief Nets with the Lambertian reflectance assumption, our model can learn good priors over the albedo from 2D images. Illumination variations can be explained by changing only the lighting latent variable in our model. By transferring learned knowledge from similar objects, albedo and surface normal estimation from a *single* image is possible in our model. Experiments demonstrate that our model is able to generalize as well as improve over standard baselines in *one-shot* face recognition.

In this chapter, we tried very hard to strike a balance between using too much laborious hand engineering on the one hand and the laissez-faire connectionist learn-everything approach on the other. By adding more modeling capability in these higher-order multiplicative operations, we nudge the model to *learn* to use these new features to improve generalization. Both proposed models show dramatic performance gain by using minimal hand engineering.

## 3.1 Robust Boltzmann Machines

While Boltzmann Machines have been successful at unsupervised learning and density modeling of images and speech data, they can be very sensitive to noise and occlusion. Occlusion in particular can be devastating to test recognition performance as the appearance of the occluded object can vary dramatically. Examples include trying to recognize the face of a person who is drinking from a red coffee mug or trying to find an object partially occluded by a stack of papers. In both cases, the appearance of the occluders should not affect the recognition of the objects of interest, yet many algorithms are significantly influenced by their appearance.

Typical approaches for dealing with occluders are to use an architecture which is engineered to be robust against occlusion and/or to augment the training set with noisy examples. Local descriptors, such as SIFT (Lowe, 2004) and Convolutional Neural Nets (LeCun et al., 1998) are examples of such engineered architectures. There are, however, some drawbacks to these approaches. For SIFT and Convolutional Nets, hyper-parameters such as the descriptor window size and local filter size need to be specified. Augmenting the training set requires the ability to synthetically generate corruptions, which is challenging for shadows, specular reflections and occlusion by unknown objects.

This section describes an alternative unsupervised approach that learns to distinguish between corrupted and uncorrupted pixels and to find useful latent representations of both that lead to improved object discrimination. The family of Boltzmann Machine models have been shown to give good results on facial expression (Ranzato et al., 2011) and speech recognition tasks (Mohamed et al., 2011). We present a novel model that allows Boltzmann Machines to be robust to corruptions in the data. Building on a similar model for binary data (Tang, 2010), our model uses multiplicative gating to induce a scale mixture of two Gaussian distributions over the data variables. Furthermore, our framework can successfully learn the *statistical structure* of the noise and occluders without explicit supervision. Our model has several key advantages:

- Multiplicative gating allows for the presence of novel occluders with exotic appearances.
- The structure of the occluders and noise statistics can be learned from the data in an unsupervised fashion.
- Completely automated image inpainting and denoising correspond to posterior inference in the model.

Generative image models with occlusion have been well studied in the vision and machine learning literature (Kannan et al., 2005; Williams and Titsias, 2004). Recently, models involving Restricted Boltzmann Machines have also been applied to image segmentation (Roux et al., 2011) and foreground-background modeling (Heess et al., 2011). Compared to the above work, the fully undirected nature of our model facilitates efficient inference. Face recognition under occlusion has also been explored in (Wright et al., 2009; Zhou et al., 2009; Jia and Martinez, 2008). Zhou et al. (2009) used an MRF to model contiguous occlusion. However, their model is not as flexible since its parameters are not learned from data.

### 3.1.1 The Model

The Robust Boltzmann Machine (RoBM) is an undirected graphical model with three components. The first is a Gaussian Restricted Boltzmann Machine (GRBM) (See Section 2.1.2) modeling the density of the noise-free or "clean" data. The second is a Restricted Boltzmann Machine (RBM) modeling the structure of the occluder/noise. The RoBM also contains a multiplicative gating mechanism which allows it to be robust to unexpected corruptions of the observed variables. Section 2.1 gives a indepth discussions on RBMs and GRBMs.

The GRBM is not robust to noise as it assumes a diagonal Gaussian as its conditional distribution over the visible nodes. This means that the log probability assigned to a noisy outlier would be very low and classification accuracy tends to be poor for noisy, out-of-sample test cases. The RoBM solves this problem by using gating at each visible node, inducing a scale mixture of two Gaussians.

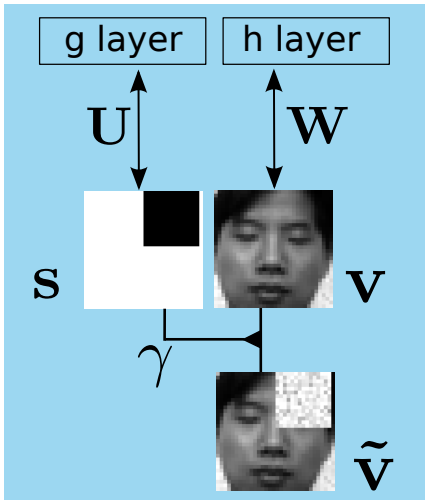



Figure 3.1: The Robust Boltzmann Machine with real images demonstrating its latent representations. $\tilde{\mathbf{v}}$ is observed, $\mathbf{s}$ and $\mathbf{v}$ are inferred. $\mathbf{g}$ denotes the binary hidden variables of the RBM modeling the structure of the occluder, while $\mathbf{h}$ denotes the binary hidden variables of the RBM modeling the "clean" face. The model uses the higher layer RBMs to separate out the clean face and the occluder/noise. Best viewed in color.

Fig. 3.1 shows how an RoBM model should decompose an occluded face. Note that only $\tilde{\mathbf{v}}$ is observed and the RoBM model uses its prior over face images to infer the unoccluded face and the occluding shape. Fig. 3.2 shows the graphical model of the RoBM model. Filled triangles emphasize that $s_i$ can dynamically change the weight between $v_i$ and $\tilde{v}_i$.

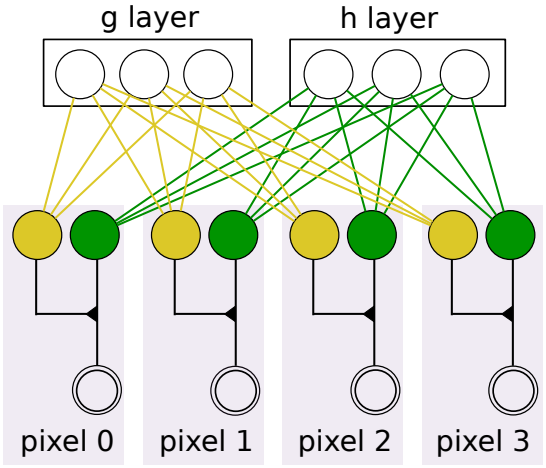



Figure 3.2: Graphical model of the Robust Boltzmann Machine. Filled triangles indicate gating of the connection between $v_i$ and $\tilde{v}_i$ by $s_i$. The yellow connections are the weights of the RBM while the green connections are the weights of the GRBM. Each pixel is modeled by three random variables: $v_i$, $\tilde{v}_i$ and $s_i$. Best viewed in color.

Its energy is obtained by combining gating terms involving an RBM of binary indicator variables $s_i$, a GRBM with real-valued variables $v_i$, and a Gaussian noise model of $\tilde{v}_i$:

$$\begin{aligned} E_{RoBM}(\mathbf{v}, \tilde{\mathbf{v}}, \mathbf{s}, \mathbf{h}, \mathbf{g}) = & \frac{1}{2} \sum_i \frac{\gamma_i^2}{\sigma_i^2} s_i (v_i - \tilde{v}_i)^2 \\ & - \sum_i d_i s_i - \sum_k e_k g_k - \sum_{i,k} U_{ik} s_i g_k \\ & + \frac{1}{2} \sum_i \frac{(v_i - b_i)^2}{\sigma_i^2} - \sum_j c_j h_j - \sum_{ij} W_{ij} v_i h_j \\ & + \frac{1}{2} \sum_i \frac{(\tilde{v}_i - \tilde{b}_i)^2}{\tilde{\sigma}_i^2} \end{aligned} \tag{3.1}$$

In the above energy function, the first line is the gating interaction term involving $s_i$, $v_i$, and $\tilde{v}_i$. It allows $\tilde{v}_i$ to be very different from $v_i$ when $s_i = 0$. $\gamma_i^2$ regulates the coupling between $v_i$ and $\tilde{v}_i$ when $s_i = 1$. The second line is the energy function of the RBM modeling the structure/correlations of the noise indicators $\mathbf{s}$. The third line is the energy function of the GRBM modeling "clean" data $\mathbf{v}$. The last line in the above energy function specifies the noise distribution: $\tilde{b}_i$ is the mean of the noise and $\tilde{\sigma}_i^2$ is the variance of the noise. In particular, if the model estimates that the $i$-th node is corrupted with noise ($s_i = 0$), then $\tilde{v}_i \sim \mathcal{N}(\tilde{v}_i | \tilde{b}_i; \tilde{\sigma}_i^2)$.

### 3.1.2 Properties of the model

The motivation for using the RoBM is to achieve better generalization by eliminating the influence of corrupted pixels. The gating serves as a buffer between what is observed ($\tilde{v}_i$) and what is preferred by the GRBM ($v_i$). When $\tilde{v}_i$ is corrupted, RoBM can still set $v_i$ to the noiseless value while turning off $s_i$. If the RBM model of $\mathbf{s}$ assigns equal energies for both states of $s_i$, then no data penalty costs would be

incurred by the corruption to $\tilde{v}_i$.[1]

Robust Statistics, such as the M-estimator (Huber, 1981), use loss functions which do not increase super-linearly. For fitting parametric mixture models, robustness is provided by using a heavy-tailed distribution for the likelihood function of each component (Svensén and Bishop, 2005). The RoBM model is also a robust mixture model with a scale mixture of two Gaussians over the observed $\tilde{\mathbf{v}}$. To see this, we can formulate RoBM as a mixture model with $2^{N_h+N_g}$ components: $p(\tilde{\mathbf{v}}) = \sum_{\mathbf{h},\mathbf{g}} p(\tilde{\mathbf{v}}|\mathbf{h},\mathbf{g})p(\mathbf{h},\mathbf{g})$, where each component's likelihood function is factorial, $p(\tilde{\mathbf{v}}|\mathbf{h},\mathbf{g}) = \prod_i p(\tilde{v}_i|\mathbf{h},\mathbf{g})$. It can be shown that

$$p(\tilde{v}_i|\mathbf{h},\mathbf{g}) = \prod_i \left\{ \pi_i \mathcal{N}(\tilde{v}_i|\tilde{b}_i;\tilde{\sigma}_i^2) + (1-\pi_i)\mathcal{N}(\tilde{v}_i|\mu_i^{new};\frac{\sigma_i^2\tilde{\sigma}_i^2}{\tilde{\sigma}_i^2+\sigma_i^2}) \right\}, \tag{3.2}$$

where $\pi_i$ is a function of $\mathbf{g}$ and $\mathbf{h}$, and $\mu_i^{new}$ is a linear combination of $\tilde{b}_i$ and $\mu_i$ (Eq. 2.21). This means that $p(\tilde{v}_i|\mathbf{h},\mathbf{g})$ is a mixture of two Gaussians with different variances: one large ($\tilde{\sigma}_i^2$) and one much smaller, since $\tilde{\sigma} >> \sigma$. The mixing proportions are not fixed but rather *depend* on $\mathbf{g}$ and $\mathbf{h}$, which can learn the spatial structures (if any) of the corruptions.

The RoBM model is also a generalization of the common MRF framework used for image restoration and denoising (Geman and Geman, 1984; Roth and Black, 2005). Setting $s_i = 1, \forall_i$, and $\frac{\gamma_i^2}{\sigma_i^2}$ to the noise variance of the data penalty, we recover an MRF model with the GRBM specifying its image prior instead of the usual local smoothness potentials. Whereas the parameters of the data penalty in standard MRFs are usually manually specified, the equivalent parameters in the RoBM, $s_i\gamma_i^2$, are actually random variables. We will show in section 3.1.4 that the distribution of $s_i\gamma_i^2$ can be learned from noisy data in an unsupervised fashion.

### 3.1.3 Inference

Inference in the RoBM consists of finding the posterior distribution of the latent variables conditioned on the observed variables: $p(\mathbf{v},\mathbf{s},\mathbf{g},\mathbf{h}|\tilde{\mathbf{v}})$. This distribution is complicated but we can use the alternating Gibbs operator to sample from this posterior. Alternating Gibbs is much more efficient than standard Gibbs as we have two alternating conditional distributions which are easy to sample from.

Conditional 1: $p(\mathbf{v},\mathbf{s}|\mathbf{g},\mathbf{h},\tilde{\mathbf{v}})$

Conditional 2: $p(\mathbf{g},\mathbf{h}|\mathbf{v},\mathbf{s},\tilde{\mathbf{v}})$

**Conditional 1:** we can efficiently draw samples by first sampling $p(\mathbf{s}|\mathbf{g},\mathbf{h},\tilde{\mathbf{v}})$, then $p(\mathbf{v}|\mathbf{s},\mathbf{h},\tilde{\mathbf{v}})$, since:

$$p(\mathbf{v},\mathbf{s}|\mathbf{g},\mathbf{h},\tilde{\mathbf{v}}) = p(\mathbf{v}|\mathbf{s},\mathbf{h},\tilde{\mathbf{v}})p(\mathbf{s}|\mathbf{g},\mathbf{h},\tilde{\mathbf{v}}). \tag{3.3}$$

In addition, due to the form of Eq. 3.1, when given $\mathbf{g}$ and $\mathbf{h}$, the distribution over $\mathbf{v}$ and $\mathbf{s}$ is factorial:

$$\begin{aligned} p(\mathbf{v},\mathbf{s}|\mathbf{g},\mathbf{h},\tilde{\mathbf{v}}) &= \prod_i p(v_i,s_i|\mathbf{g},\mathbf{h},\tilde{\mathbf{v}}) \\ &= \prod_i p(v_i|s_i,\mathbf{g},\mathbf{h},\tilde{\mathbf{v}})p(s_i|\mathbf{g},\mathbf{h},\tilde{\mathbf{v}}). \end{aligned} \tag{3.4}$$

[1] There will still be a small penalty from the noise model.

**Algorithm 3** Inference in the RoBM: $p(\mathbf{v}, \mathbf{s}, \mathbf{h}, \mathbf{g}|\tilde{\mathbf{v}})$

1: Randomly initialize the layers of $\mathbf{h}$, $\mathbf{g}$.
**for** $t = 1 : NumberGibbsSteps$ **do**
2: Sample from $p(\mathbf{s}|\mathbf{g}, \mathbf{h}, \tilde{\mathbf{v}})$, using Eq. 3.5.
3: Sample from $p(\mathbf{v}|\mathbf{s}, \mathbf{h}, \tilde{\mathbf{v}})$, using Eq. 3.8.
4: Sample from $p(\mathbf{g}, \mathbf{h}|\mathbf{v}, \mathbf{s}, \tilde{\mathbf{v}})$, using Eq. 3.9.
**end for**

Moreover, it can be shown by integrating out $v_i$ that

$$p(s_i|\mathbf{g}, \mathbf{h}, \tilde{\mathbf{v}}) = \frac{\alpha^{s_i}\beta^{1-s_i}}{\alpha + \beta} \tag{3.5}$$

$$\alpha = \hat{\sigma}_i \exp\Big\{(d_i + U_i\mathbf{g}) - \frac{1}{2}\frac{\gamma_i^2}{\sigma_i^2}\tilde{v}_i^2 + \frac{1}{2}\frac{\hat{\mu}_i^2}{\hat{\sigma}_i^2}\Big\} \tag{3.6}$$

$$\beta = \sigma_i \exp\Big\{\frac{1}{2}\frac{\mu_i^2}{\sigma_i^2}\Big\}, \tag{3.7}$$

where $\mu_i$ is defined by Eq. 2.21, $\hat{\mu}_i = \frac{\mu_i + \gamma_i^2\tilde{v}_i}{\gamma_i^2+1}$, and $\hat{\sigma}_i = \frac{\sigma_i}{\sqrt{\gamma_i^2+1}}$. Note that $d_i + U_i\mathbf{g}$ is the total input coming from the $\mathbf{g}$ layer, and $\boldsymbol{\mu}$ is the total input coming from the $\mathbf{h}$ layer. After sampling $s_i$, the conditional distribution over $v_i$ is:

$$p(v_i|s_i, \mathbf{h}, \tilde{\mathbf{v}}) \sim \mathcal{N}\Big(\frac{s_i\gamma_i^2}{s_i\gamma_i^2 + 1}\tilde{v}_i + \frac{\mu_i}{s_i\gamma_i^2 + 1}, \frac{\sigma_i^2}{s_i\gamma_i^2 + 1}\Big) \tag{3.8}$$

The above equation has a very intuitive interpretation. When $s_i = 0$, node $i$ is corrupted, $v_i$ is distributed according to $v_i \sim \mathcal{N}(\mu_i; \sigma_i^2)$, where $\mu_i$ is determined by the hidden nodes of the GRBM. However, when $s_i = 1$, node $i$ is not corrupted, and its mean is a weighted average of $\mu_i$ and the observed input $\tilde{v}_i$. The weighting is determined by the parameter $\gamma_i^2$, which acts as the precision of the sensor noise. When it is large, $v_i$ will be very similar to $\tilde{v}_i$. When it is small, $v_i$ is allowed to be different from $\tilde{v}_i$ since its deviation can be explained by the observation noise.

**Conditional 2:** The 2nd conditional is efficient to compute as it can be factored into a product of the RBM and GRBM posteriors:

$$p(\mathbf{g}, \mathbf{h}|\mathbf{v}, \mathbf{s}, \tilde{\mathbf{v}}) = p(\mathbf{g}, \mathbf{h}|\mathbf{v}, \mathbf{s}) = p(\mathbf{h}|\mathbf{v})p(\mathbf{g}|\mathbf{s}), \tag{3.9}$$

where $p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v})$ (see Equation 2.4). Similarly, we also have $p(\mathbf{g}|\mathbf{s}) = \prod_k p(g_k|\mathbf{s})$. The algorithm for performing posterior inference is shown in Algorithm 3.

### 3.1.4 Learning

The parameters of the RoBM can be learned by maximizing the log-likelihood over the observed noisy images $\tilde{\mathbf{v}}$:

$$\hat{\theta} = \arg\max_{\theta} \quad \log p(\tilde{\mathbf{v}}; \theta), \tag{3.10}$$

where $\theta$ is the collection of all parameters of the RoBM in Eq. 3.1. In an undirected graphical model, such as a Boltzmann Machine, maximum likelihood learning can be accomplished by gradient ascent,

where gradients with respect to the parameters are given by the difference of two expectations:

$$\frac{\partial}{\partial\theta}\mathbb{E}[\log p(\tilde{\mathbf{v}}_n;\theta)] = \mathbb{E}_{model}\Big[\frac{\partial E_{RoBM}}{\partial\theta}\Big] - \mathbb{E}_{data}\Big[\frac{\partial E_{RoBM}}{\partial\theta}\Big]. \tag{3.11}$$

$\mathbb{E}_{model}[\cdot]$ denotes the expectation with respect to the distribution defined by the RoBM model (Eq. 3.1), while $\mathbb{E}_{data}[\cdot]$ denotes the empirical expectation with respect to the data distribution $p_{data}(\tilde{\mathbf{v}},\mathbf{v},\mathbf{s},\mathbf{h},\mathbf{g}) = p(\mathbf{v},\mathbf{s},\mathbf{h},\mathbf{g}|\tilde{\mathbf{v}})p_{data}(\tilde{\mathbf{v}})$, where $p_{data}(\tilde{\mathbf{v}}) = \frac{1}{N}\sum_n \delta(\tilde{\mathbf{v}} - \tilde{\mathbf{v}}_n)$.

Exact maximum likelihood learning in this model is intractable, but efficient approximate learning can be done as follows. We first approximate $\mathbb{E}_{data}[\cdot]$ by sampling from the posterior $p(\mathbf{v},\mathbf{s},\mathbf{h},\mathbf{g}|\tilde{\mathbf{v}})$ using a small number of alternating Gibbs updates (see Alg. 3). To approximate $\mathbb{E}_{model}[\cdot]$, we need to sample $\tilde{\mathbf{v}}$ as specified by the RoBM parameters. To sample from $\tilde{\mathbf{v}}$ given $\mathbf{v}$ and $\mathbf{s}$, we can sample each $\tilde{v}_i$ independently since $p(\tilde{\mathbf{v}}|\mathbf{v},\mathbf{s}) = \prod_i p(\tilde{v}_i|\mathbf{v},\mathbf{s})$. The conditional distribution over $\tilde{v}_i$ is a Gaussian distribution:

$$p(\tilde{v}_i|\mathbf{v},\mathbf{s}) \sim \mathcal{N}\Big(\tilde{v}_i\Big|\alpha v_i + \beta\tilde{b}_i, \frac{\sigma_i^2\tilde{\sigma}_i^2}{\sigma_i^2 + s_i\gamma_i^2\tilde{\sigma}_i^2}\Big),$$
$$\alpha = \frac{s_i\gamma_i^2\tilde{\sigma}_i^2}{\sigma_i^2 + s_i\gamma_i^2\tilde{\sigma}_i^2}, \beta = \frac{\sigma_i^2}{\sigma_i^2 + s_i\gamma_i^2\tilde{\sigma}_i^2}.$$

The mean of this distribution is a linear combination of what the GRBM expects and what the noise term expects. In addition, the coefficients $\alpha$ and $\beta$ depend on the random variable $s_i$. When $s_i = 0$, indicating that noise is present, $\tilde{v}_i$ is correctly sampled from the noise model with mean $\tilde{b}_i$ and variance $\tilde{\sigma}_i^2$.

During learning, we use a type of Stochastic Approximation of the Robbins-Monro type also known as Persistent Contrastive Divergence (Tieleman, 2008) to compute the model's expectation. Using PCD, we only need to run the Gibbs chain for a small number of iterations after each update of the parameters. With some mild conditions on the learning rates (Younes, 1998), we are guaranteed to converge to a locally optimal solution.

While it is possible to learn to maximize the objective function in Eq. 3.10 starting with random weights, it is much faster and easier if we first *pretrain* the parameters of the GRBM on "clean" data. It is not unreasonable for a model to have seen many noise-free examples of face images before learning on faces disguised with sunglasses. Learning is still unsupervised as *no* corresponding pairs of images of the same person, one with sunglasses and one without, are used during learning. The algorithm for RoBM learning is outlined in Alg. 4.

### 3.1.5 Experiments

We demonstrate the effectiveness of the RoBM on several standard face databases. Since the novelty of our model is in its ability to learn the structure and statistics from noisy data, we will first demonstrate it by using the Yale Face Database (Georghiades et al., 2001). We will then show that denoising with the RoBM is significantly better than standard algorithm on the large Toronto Face Database (Susskind, 2011). Finally, we investigate the RoBM's recognition performance when test images contain noise or occlusions as in the Yale Database or contain disguise as in the AR Face Database (Martínez and Benavente, 1998).

**Algorithm 4** Parameter Estimation for the RoBM

Pretrain the $\{\mathbf{v}, \mathbf{h}\}$ GRBM with "clean" data and initialize $\{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$ of the RoBM with the pretrained parameters. Initialize other parameters randomly.
Initialize randomly the state of negative fantasy particles $\{\mathbf{v}^{fp}, \tilde{\mathbf{v}}^{fp}, \mathbf{s}^{fp}, \mathbf{g}^{fp}, \mathbf{h}^{fp}\}$ needed by PCD.
Initialize learning rate $\eta_0 \leftarrow 0.001$
**for** $m = 1$ : number learning epochs **do**
  **for** $n = 1$ : number of training cases **do**
    Use Alg. 3 to sample from $p(\mathbf{v}, \mathbf{s}, \mathbf{h}, \mathbf{g}|\tilde{\mathbf{v}}_n)$
    Calculate $\mathbb{E}_{data}\left[\frac{\partial E_{RoBM}}{\partial \theta}\right]$ using the samples of $\{\mathbf{v}, \mathbf{s}, \mathbf{h}, \mathbf{g}, \text{and } \tilde{\mathbf{v}}_n\}$.
    Use Alg. 3 sample from $p(\mathbf{v}^{fp}, \mathbf{s}^{fp}, \mathbf{h}^{fp}, \mathbf{g}^{fp}|\tilde{\mathbf{v}}^{fp})$
    Calculate $\mathbb{E}_{model}\left[\frac{\partial E_{RoBM}}{\partial \theta}\right]$ using the fantasy particles $\{\mathbf{v}^{fp}, \mathbf{s}^{fp}, \mathbf{h}^{fp}, \mathbf{g}^{fp}, \text{and } \tilde{\mathbf{v}}^{fp}\}$.
    Update: $\theta_{t+1} \leftarrow \theta_t + \frac{\partial \log p(\tilde{\mathbf{v}}_n)}{\partial \theta}$ (see Eq. 3.10).
  **end for**
  Decrease learning rate: $\eta_{t+1} = \eta_0/m$
**end for**

### Effects of Learning

We first demonstrate that RoBM's learning algorithm described in Sec. 3.1.4 can be successfully applied to learn directly from noisy data, without any knowledge of a clean image and its noisy version. We use the Yale Database for this experiment. The Yale Face Database contains 15 subjects with 11 images per subject. The face images are frontal but vary in illumination and expression. Following the standard protocol, we randomly select 8 images per subject as training and 3 for testing. We cropped images to the resolution of $32 \times 32$ and trained a GRBM model with visible nodes $\mathbf{v}$ and hidden nodes $\mathbf{h}$ on the "clean" faces. The training used Persistent Contrastive Divergence for a total of 50 epochs. We then initialized the RoBM's parameters $\{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$ with the pretrained GRBM and applied the learning algorithm in Algorithm 4 to learn the parameters of the RoBM model. In all of our experiments, $U_{ik}, e_k, \tilde{b}_i$ are initialized to 0.0, $d_i$ to 4.0, $\gamma_i$ to 20.0, and $\tilde{\sigma}_i^2$ is initialized to 1.0.

Fig. 3.3 shows the learning process of the RoBM. The columns represent the internal activation of the RoBM during learning from epoch 1 to epoch 50. The top row displays the training examples. The top panel shows an example that has synthetic grid-patterned noise, while the bottom panel shows an example that has an occlusion by sunglasses. The second and third rows display the inferred faces $\mathbf{v}$ and the structure of the occluder/noise $\mathbf{s}$.

During the first learning epoch, the $\mathbf{U}$ matrix was initialized to zero. Therefore, no structure in $\mathbf{s}$ is modeled initially. This is confirmed by the fact that the inferred $\mathbf{s}$ are very noisy. As learning proceeds, we observe the trend that the actual shapes of the occluders are cleanly detected[2] and are modeled by the $\{\mathbf{s}, \mathbf{g}\}$ RBM. This demonstrates that we can in fact learn the noise structure in an unsupervised manner, when given a pretrained face density model.

To isolate the effect of having a model of the noise/occluder, we compare an RoBM model with hand-tuned parameters with an RoBM model trained on the noisy data. For the hand-tuned RoBM, we set its biases $d_i$ such that the sigmoid of $d_i$ would give the probability of each pixel being corrupted. Table 3.1 shows the PSNR of denoised Yale faces using an hand-tuned RoBM vs. an learned RoBM. For random noise, 40% of the pixels were corrupted by random noise with a standard deviation of 0.4. For block occlusion, $12 \times 12$ blocks were superimposed on a random part of the $32 \times 32$ faces.

[2]Some speckle will remain since we are viewing random samples from the posterior.

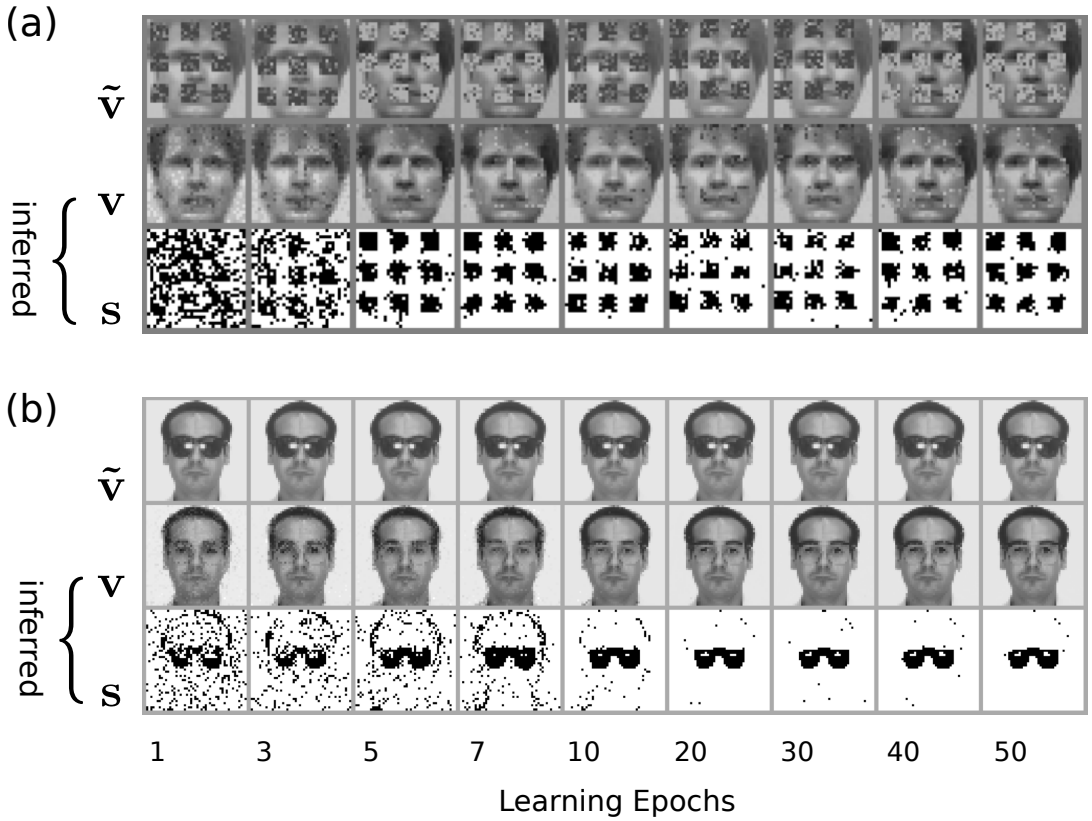



Figure 3.3: Internal states of the RoBM during learning: columns from left to right represent epochs 1 to 50. The first row is the training data $\tilde{\mathbf{v}}$, the second row is the inferred $\mathbf{v}$, and the third row is the inferred $\mathbf{s}$. 20 Gibbs iterations were run to sample from the posterior.

| RoBMs Parameters | hand-tuned | learned |
|---|---|---|
| Random noise | 30.0 ± 0.77 | 30.4 ± 0.88 |
| Block occlusion | 26.7 ± 0.85 | 28.6 ± 0.82 |

Table 3.1: Peak Signal to Noise Ratio (PSNR) in dB for denoising on Yale faces for a hand-tuned and learned RoBM. The numbers are averages over 40 trials ± the standard error of the mean.

For random noise, learning the structure of the noise does not add any value, thus similar results are expected. However, for block occlusions, structure learning helps denoising dramatically, resulting in an increase of 2 dB in the average denoised image.

**Denoising**

We next experimented on the large-scale Toronto Face Database (TFD) (Susskind, 2011). The TFD is a collection of (mostly) publicly available aligned face images. We used 60,000 training and 2,000 test $24 \times 24$ images. All test images are different from the training images by a Euclidean distance of at least 5.0. This eliminates cases where a test image is very similar to a training image, which is a possibility as the TFD faces were aggregated from a large collection of databases without separation by identity.

We first pretrained a GRBM model with 2,000 hidden nodes using FPCD (Tieleman and Hinton, 2009) on the 60,000 training images for 500 epochs. The RoBM model was initialized exactly as described in the previous subsection. We then learned the joint model using Algorithm 4. For block noise, we

trained the RoBM model on data occluded by blocks at random positions. For random noise, we trained the RoBM model on data corrupted by random noise. After learning, we used 50 Gibbs iterations to sample from the posterior distribution. The denoised image of the RoBM model is taken to be the exponentially weighted average of the posterior samples with a weight of 0.9.

In all of our experiments, we compare performance of the RoBM model to the following four baseline models. Our first denoising algorithm, called RBM, consists of taking the pretrained GRBM model and initializing it with a noisy data. We run a few alternating Gibbs updates and take the exponentially weighted average as the denoised output. The second model, called PCA denoising algorithm, projects a noisy image onto a 75 dimensional subspace. The PCA reconstruction is then taken to be the denoised image. Our third algorithm performs Wiener filtering using MATLAB's *wiener2* function and a window size of 5. Our final baseline model finds the closest Euclidean nearest neighbor of the noisy test image in the training set.

Fig. 3.4 shows the denoising results for one face.

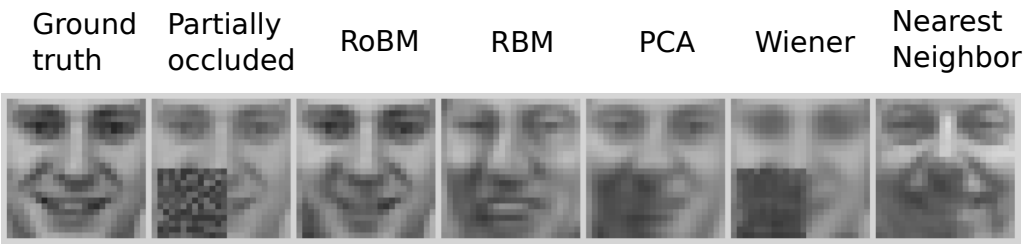



Figure 3.4: Difference between various denoising algorithms for block occlusion.

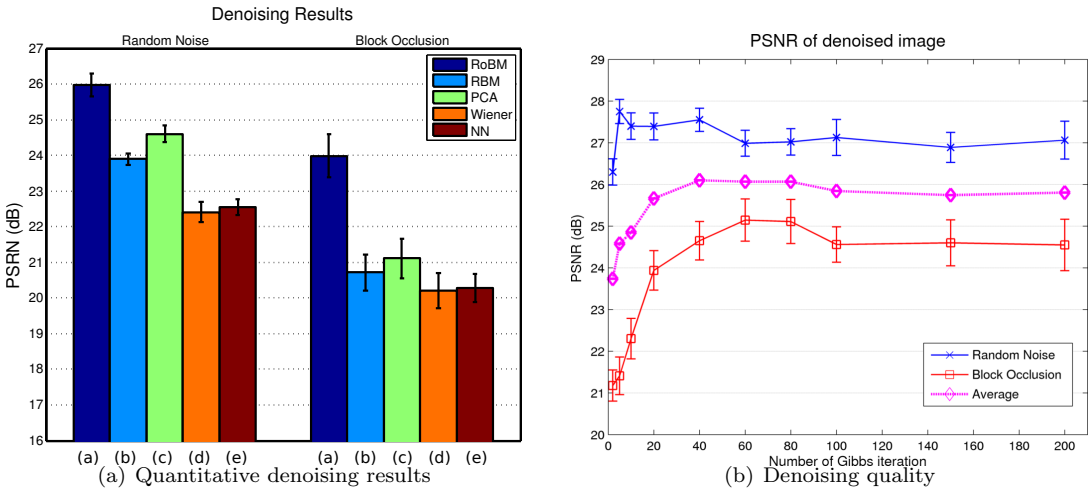
The RoBM model performs significantly better than other methods. Since there is a dark occluder in the bottom left of the image, nearest neighbor found a different face with a shadow on the bottom left. While Wiener filtering works well for Gaussian noise, it is not suitable for occlusions. PCA and RBM are unable to fully restore the occluded area, whereas RoBM is able to properly denoise due to its ability to gate off the occluded area and use its face prior to infer what is behind the occluder. We present similar qualitative results for random noise and occlusion in Fig. 3.5. Quantitatively, RoBM performs better than other models in terms of peak signal to noise ratio of the denoised results. Fig. 3.6(a) shows the results for both random noise and block occlusion.

We also investigated how sensitive our denoising results are to the hyper-parameter that specifies how many Gibbs iterations to run during inference. In Fig. 3.6(b), we plot the PSNR vs. the number of Gibbs iterations for both random noise and occlusion.


(a) Random Noise


(b) Block Occlusions

Figure 3.5: Qualitative comparison of various denoising algorithms for random noise and occlusion. The first row has the original faces and the second row is corrupted with noise. Starting from the third row, we have denoising results from RoBM, RBM, PCA, Wiener filtering, and Nearest Neighbor, respectively.




(a) Quantitative denoising results (b) Denoising quality

Figure 3.6: Left: Methods are (a) RoBM, (b) RBM, (c) PCA, (d) Wiener, (e) Nearest Neighbor. Right: PSNR versus the number of Gibbs iterations used for sampling from the posterior during inference.

From this plot, we see that 40 to 60 iterations tend to give the best *average* performance.

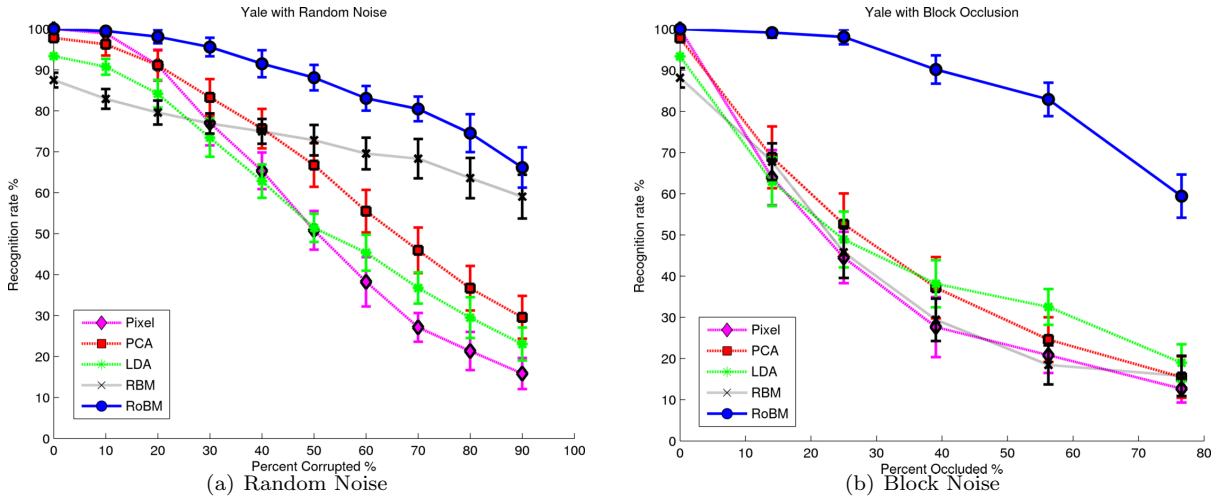
### Recognition

In this section we test the ability of the RoBM to accurately recognize faces in the presence of noise and occlusion. We first add synthetic noise and occlusions to the faces in the Yale Face database and plot classification accuracy as a function of the degree of noise/occlusion. We then test recognition performance with natural disguises (sunglasses and scarf) from the AR Face Database.

The classifier is a multi-class linear SVM trained on different feature representations of the faces. Recognition using the RoBM consists of first running 30 Gibbs iterations for denoising followed by classification using its hidden outputs before the sigmoid nonlinearity (Eq. 2.4). We provide comparisons to other benchmark models: pixels, LDA (Lu et al., 2003), Eigenfaces (Turk and Pentland, 1991), and the standard GRBM. For the GRBM model, we first pretrain it and then run a few iterations of alternating Gibbs updates before classification.

### Yale Face Database

As in Sec. 3.1.5, we used 8 images per subject for training and 3 for testing, and trained the RoBM model as specified in Sec. 3.1.5. During testing, for each noisy image, we ran 30 iterations of Gibbs sampling to arrive at a clean face. For classification, we feed the $\mathbf{h}$ layer activations (before the sigmoid nonlinearity) into the linear SVM. Fig. 3.7(a) shows that RoBM performs better than other benchmark models, particularly when the amount of noise increases. The RBM method was run for 5 iterations of alternating Gibbs starting from the initial noisy image, where 5 iterations were chosen as it gave the best results on the test set. PCA used 50 eigenfaces and LDA was learned using the code provided by (Lu et al., 2003). Fig. 3.7(b) displays the recognition accuracy as a function of the percentage of the image that is blocked.




(a) Random Noise (b) Block Noise

Figure 3.7: Recognition rates on the Yale Database as a function of the percentage of pixels corrupted by noise. Random noise with standard deviation of 0.5 was added to the corrupted pixels. Recognition rates on the Yale Database as a function of the percentage of pixels occluded. Block occlusion was applied as in Fig. 3.5(b).


Figure 3.8: Intermediate results during RoBM inference. The leftmost images are the test samples.

### AR Face Database

Table 3.2 further shows that the RoBM model significantly outperforms all other models on the AR face recognition task.

The AR database contains faces with real-life disguises using sunglasses and a scarf. We used a subset of 114 people each with 8 images for a total of 912 training images. For every person, there are two additional images with sunglasses and two with scarf occlusions, which we used as our test set. We first cropped and downsized the images to a resolution of $32 \times 32$ and pretrained a GRBM with 2000 hidden nodes. Initializing the RoBM model with weights from the pretrained GRBM, we learned one RoBM model on sunglasses and one RoBM model on the scarf images. After learning for 50 epochs, Fig. 3.8 displays the inferred "clean" face.

| Algorithms | Sunglasses | Scarf |
|---|---|---|
| RoBM | 84.5 % | 80.7 % |
| RBM | 61.7 % | 32.9 % |
| Eigenfaces | 66.9 % | 38.6 % |
| LDA | 56.1 % | 27.0 % |
| Pixel | 51.3 % | 17.5 % |

Table 3.2: Recognition results on the AR Face Database.

### 3.1.6 Shortcomings

The Robust Boltzmann Machine has a couple of shortcomings which we leave to future work. The first is that the model needs to first be trained on the "clean" faces. It is better to train the entire model jointly from scratch on clean as well as occluded faces. The second is that the model only assumes one foreground object and an occluder. This assumption is not correct if there are multiple objects in the scene occluding each other. This problem can be corrected by introducing one mask latent image per object and modifying the energy function accordingly.

## 3.2 Deep Lambertian Neural Networks

In this section, we discuss the second model in the chapter which takes inspiration from computer vision and combines it to form a more robust deep generative model for modeling faces under illumination variations. Deep models have achieved excellent recognition accuracy by first learning generatively from data before using the learned latent variables (features) for discriminative tasks. The advantage of using this indirect approach for discrimination is that it is possible to learn meaningful latent variables that achieve strong generalization. In vision, illumination is a major cause of variation. When the light source direction and intensity changes in a scene, dramatic changes in image intensity occur. This is detrimental to recognition performance as most algorithms use image intensities as inputs.

The potential problem for using the latent representation of a standard Deep Belief network for modeling face images under illumination variations can be seen in Figure 3.9. Note that the hidden representation is more similar for images with similar illumination, not similar for images with the same identity.

A natural way of attacking this problem is to learn a model where the albedo, surface normals, and the lighting are explicitly represented as the latent variables. Since the albedo and surface normals are physical properties of an object, they are features which are invariant with respect to illumination.

Separating the surface normals and the albedo of objects using multiple images obtained under different lighting conditions is known as photometric stereo (Woodham, 1980). Hayakawa (1994) described a method for photometric stereo using SVD, which estimated the shape and albedo up to a linear transformation. Using integrability constraints, Yuille et al. (1999) proposed a similar method to reduce the ambiguities to a *generalized bas relief* ambiguity. A related problem is the estimation of intrinsic images (Barrow and Tenenbaum, 1978; Gehler et al., 2011). However, in those works, the shading (inner product of the lighting vector and the surface normal vector) instead of the surface normals is estimated. In addition, the use of three color channels simplifies that task.

In the domain of face recognition, Belhumeur and Kriegman (1996) showed that the set of images of an object under varying lighting conditions lie on a polyhedral cone (illumination cone), assuming a

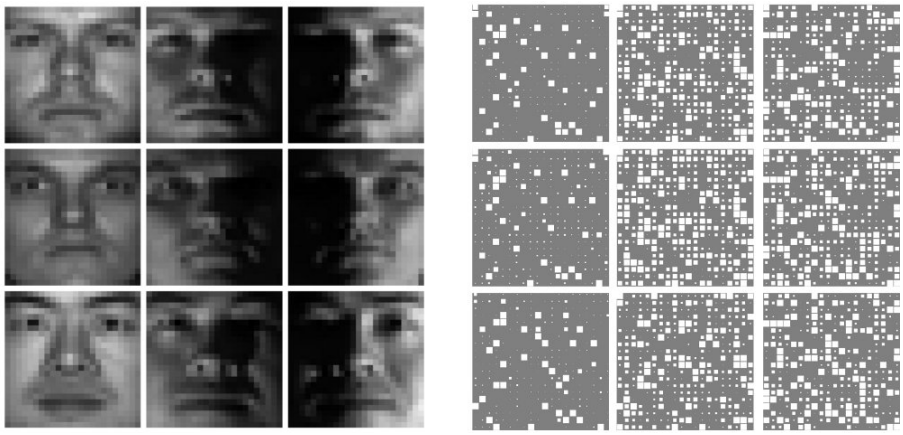


Figure 3.9: Deep Belief Networks pay more attention to illumination variations rather than identity. Left: three person under three different lighting variations. Right: hidden layer activation clusters along the columns (lighting variations).

Lambertian reflectance and a fixed object pose. Recognition algorithms were developed based on the estimation of the illumination cone (Georghiades et al., 2001; Lee et al., 2005). The main drawback of these models is that they require multiple images of an object under varying lighting conditions for estimation. While Zhang and Samaras (2006) and Wang et al. (2009) present algorithms that only use a single training image, their algorithms require bootstrapping with a 3D morphable face model. For every generic object class, building a 3D morphable model would be labor intensive.

In this section, we introduce a generative model which **(a)** incorporates albedo, surface normals, and the lighting as latent variables; **(b)** uses multiplicative interaction to approximate the Lambertian reflectance model; **(c)** learns from sets of 2D images the distributions over the 3D object shapes; and **(d)** is capable of one-shot recognition from a single training example.

The Deep Lambertian Network (DLN) is a hybrid undirected-directed model with Gaussian Restricted Boltzmann Machines (and potentially Deep Belief Networks) modeling the prior over the albedo and surface normals. Good priors over the albedo and normals are necessary since for inference with a single image, the number of latent variables is 4 times the number of observed pixels. Estimation is an ill-posed problem and requires priors to find a unique solution. A density model of the albedo and the normals also allows for parameter sharing across individual objects that belong to the same class. The conditional distribution for image generation follows from the Lambertian reflectance model. Estimating the albedo and surface normals amounts to performing posterior inference in the DLN model with no requirements on the number of observed images. Inference is efficient as we can use alternating Gibbs sampling to approximately sample latent variables in the higher layers. The DLN is a *permutation invariant* model which can learn from any object class and strikes a balance between laborious approaches in vision (which require 3D scanning (Blanz and Vetter, 1999)) and the generic unsupervised deep learning approaches.

A popular generative model of objects and faces is the Gaussian Restricted Boltzmann Machines (GRBMs). See Section 2.1 for a detailed discussion on GRBMs. GRBMs use Eq. 2.21 to generate the intensity of a particular pixel $v_i$. This generative model is inefficient when dealing with illumination variations in $\mathbf{v}$. Specifically, the hidden activations needed to generate a bright image of an object are very different from the activations needed to generate a dark image of the *same* object.

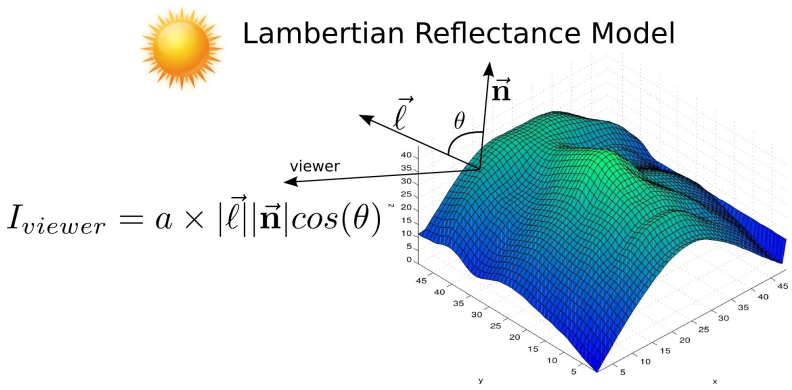



Figure 3.10: Diagram of the Lambertian Reflectance model. $\boldsymbol{\ell} \in \mathbb{R}^3$ points to the light source. $\vec{\mathbf{n}}_i \in \mathbb{R}^3$ is the surface normal, which is perpendicular to the tangent plane at a point on the surface.

The Lambertian reflectance model is widely used for modeling illumination variations and is a good approximation for diffuse object surfaces (those without any specular highlights). Under the Lambertian model, illustrated in Fig. 3.10, the $i$-th pixel intensity is modelled as $v_i = a_i \times max(\vec{\mathbf{n}}_i^{\mathsf{T}}\vec{\boldsymbol{\ell}}, 0)$. The albedo $a_i$, also known as the reflection coefficient, is the diffuse reflectivity of a surface at pixel $i$, which is material dependent but illumination invariant. In contrast to the generative process of the GRBM, the image of an object under different lighting conditions can be generated without changing the albedo and the surface normals. Multiplications *within* hidden variables in the Lambertian model give rise to this nice property.

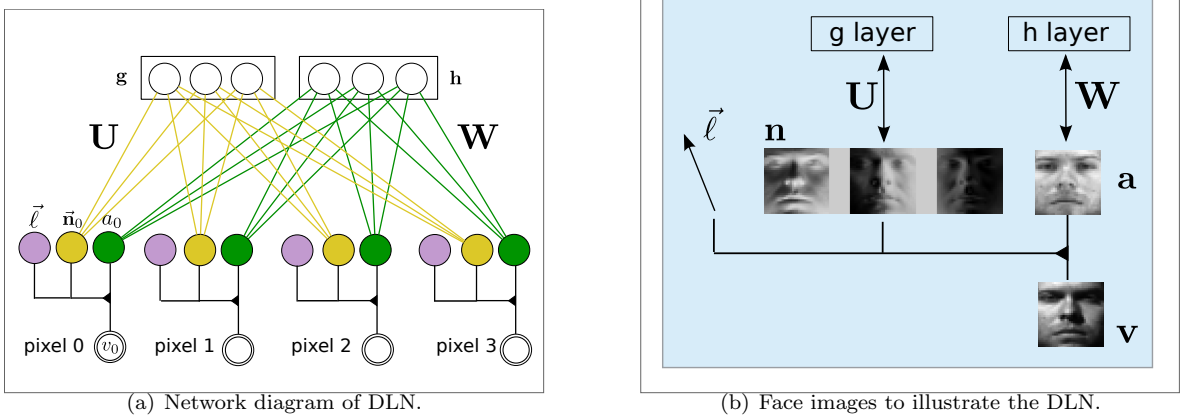
### 3.2.1 The Model

The DLN is a hybrid undirected-directed generative model that combines Deep Belief Networks (DBNs) with the Lambertian reflectance model. In the DLN, the visible layer consists of image pixel intensities $\mathbf{v} \in \mathbb{R}^{N_v}$, where $N_v$ is the number of pixels in the image. The first layer hidden variables are the albedo, surface normals, and a light source vector. Specifically, for every pixel $i$, there are two corresponding latent random variables: the albedo $a_i \in \mathbb{R}^1$ and surface normal $\mathbf{n}_i \in \mathbb{R}^3$. Over an image, $\mathbf{a} \in \mathbb{R}^{N_v}$ is the image albedo, $\mathbf{N}$ is the surface normals matrix of dimension $N_v \times 3$, where $\mathbf{n}_i$ denotes the $i$-th row of $\mathbf{N}$. The light source variable $\boldsymbol{\ell} \in \mathbb{R}^3$ points in the direction of the light source in the scene. We use GRBMs to model the albedo and surface normals, and a Gaussian prior to model $\boldsymbol{\ell}$. It is important to use GRBMs since we expect the distribution over albedo and surface normals to be multi-modal (see Fig. 3.13).

Fig. 3.11 shows the architecture of the DLN model: Panel (a) displays a standard network where filled triangles denote multiplicative gating between pixels and the first hidden layer. Panel (b) demonstrates the desired latent representations inferred by our model given input $\mathbf{v}$. While we use GRBMs as the prior models on albedo and surface normals, Deep Belief Network priors can be obtained by stacking additional binary RBM layers on top of the $g$ and $h$ layers. For clarity of presentation, in this section we use GRBM priors[3].

The DLN combines the elegant properties of the Lambertian model with the GRBMs, resulting in a deep model capable of learning albedo and surface normal statistics from images in a weakly-supervised

[3]Extending our model to more flexible DBN priors is trivial.




(a) Network diagram of DLN.

(b) Face images to illustrate the DLN.

Figure 3.11: Graphical model of the Deep Lambertian Network. The yellow weights model the surface normals while the green weights model the albedo. The arrow in the left figure is the light source direction vector, pointing towards the light source. Note that the light vector is shared for all pixels in the image. Best viewed in color.

fashion. The DLN has the following generative process:

$$p(\mathbf{v}, \mathbf{a}, \mathbf{N}, \boldsymbol{\ell}) = p(\mathbf{a})p(\mathbf{N})p(\boldsymbol{\ell})p(\mathbf{v}|\mathbf{a}, \mathbf{N}, \boldsymbol{\ell}) \tag{3.12}$$

$$p(\mathbf{a}) \sim GRBM(\mathbf{a}) \tag{3.13}$$

$$p(\mathbf{N}) \approx GRBM(vec(\mathbf{N})) \tag{3.14}$$

$$p(\boldsymbol{\ell}) \sim \mathcal{N}(\boldsymbol{\ell}|\boldsymbol{\mu}_{\boldsymbol{\ell}}, \boldsymbol{\Lambda}) \tag{3.15}$$

$$p(\mathbf{v}|\mathbf{a}, \mathbf{N}, \boldsymbol{\ell}) = \prod_{i}^{N_v} \mathcal{N}(v_i|a_i(\mathbf{n}_i^{\mathsf{T}}\boldsymbol{\ell}); \sigma_{v_i}^2), \tag{3.16}$$

where $vec(\mathbf{N})$ denotes the vectorization of matrix $\mathbf{N}$.

The GRBM prior in Eq. 3.14 is only approximate since we enforce the soft constraint that the norm of $\mathbf{n}_i$ is equal to 1.0. We achieve this via an extra energy term in Eq. 3.17. Eq. 3.16 represents the probabilistic version of the Lambertian reflectance model. We have dropped "*max*" for convenience. "*max*" is not critical in our model as maximum likelihood learning regulates the generation process. In addition, a prior on lighting direction fits well with the psychophysical observations that human perception of shape relies on the assumption that light originates from above (Kleffner and Ramachandran, 1992).

DLNs can also handle multiple images of the same object under varying lighting conditions. Let $P$ be the number of images of the same object. We use $\mathbf{L} \in \mathbb{R}^{3\times P}$ to represent the lighting matrix with columns $\{\boldsymbol{\ell}_p : p = 1, 2, \ldots, P\}$, and $\mathbf{V} \in \mathbb{R}^{N_v \times P}$ to represent the matrix of corresponding images. The

DLN energy function is defined as:

$$\begin{aligned} E_{DLN}(\mathbf{V},\mathbf{a},\mathbf{N},\mathbf{L},\mathbf{g},\mathbf{h}) &= \frac{1}{2}\sum_{p}^{P}\sum_{i}^{N_v}\frac{(v_{ip}-a_i(\mathbf{n}_i^\mathsf{T}\boldsymbol{\ell}_p))^2}{\sigma_{v_i}^2} \\ &+ \frac{1}{2}\sum_{p}^{P}(\boldsymbol{\ell}_p-\boldsymbol{\mu}_l)^\mathsf{T}\boldsymbol{\Lambda}(\boldsymbol{\ell}_p-\boldsymbol{\mu}_l) + \frac{\eta}{2}\sum_{i}^{N_v}(\mathbf{n}_i^\mathsf{T}\mathbf{n}_i-1.0)^2 \\ &+ E_{GRBM}(\mathbf{a},\mathbf{h}) + E_{GRBM}(vec(\mathbf{N}),\mathbf{g}) \end{aligned} \tag{3.17}$$

The first line in the energy function is proportional to $\log p(\mathbf{v}|\mathbf{a},\mathbf{N},\boldsymbol{\ell})$, the multiplicative interaction term from the Lambertian model. The second line corresponds to the quadratic energy of $\log p(\boldsymbol{\ell})$ and the soft norm constraint on $\mathbf{n}_i$. This constraint is critical for the correct estimation of the albedo, since we can interpret the albedo at each pixel as the $L_2$ norm of the pixel surface normal. The third line contains the two GRBM energies: $\mathbf{h}\in\mathbb{R}^{N_h}$ represents the binary hidden variables of the albedo GRBM and $\mathbf{g}\in\mathbb{R}^{N_g}$ represents the hiddens of the surface normal GRBM:

$$E_{GRBM}(\mathbf{a},\mathbf{h}) = \frac{1}{2}\sum_{i}^{N_v}\frac{(a_i-b_i)^2}{\sigma_{a_i}^2} - \sum_{j}^{N_h}c_jh_j - \sum_{i,j}^{N_v,N_h}W_{ij}a_ih_j \tag{3.18}$$

$$\begin{aligned} E_{GRBM}(vec(\mathbf{N}),\mathbf{g}) =& \frac{1}{2}\sum_{i,m=1}^{N_v,3}\frac{n_{im}^2}{\sigma_{n_{im}}^2} - \sum_{i,m=1}^{N_v,3}\frac{d_{im}n_{im}}{\sigma_{n_{im}}^2} \\ &- \sum_{k}^{N_g}e_kg_k - \sum_{i,m=1,k}^{N_v,3,N_g}U_{imk}n_{im}g_k \end{aligned} \tag{3.19}$$

### 3.2.2 Inference

Given images of the same object under one or more lighting conditions, we want to infer the posterior distribution over the latent variables (including albedo, surface normals and light source): $p(\mathbf{a},\mathbf{N},\mathbf{L},\mathbf{g},\mathbf{h}|\mathbf{V})$. With GRBMs modeling the albedo $\mathbf{a}$ and surface normals $\mathbf{N}$, the posterior is complicated with no closed form solution. However, we can resort to Gibbs sampling using 4 sets of conditional distributions:

- *Conditional 1*: $p(\mathbf{g},\mathbf{h}|\mathbf{a},\mathbf{N},\mathbf{L},\mathbf{V})$
- *Conditional 2*: $p(\mathbf{a}|\mathbf{N},\mathbf{L},\mathbf{h},\mathbf{V})$
- *Conditional 3*: $p(\mathbf{L}|\mathbf{N},\mathbf{a},\mathbf{V})$
- *Conditional 4*: $p(\mathbf{N}|\mathbf{a},\mathbf{L},\mathbf{g},\mathbf{V})$

*Conditional 1* is easy to compute as it factorizes over $\mathbf{g}$, and $\mathbf{h}$: $p(\mathbf{g},\mathbf{h}|\mathbf{a},\mathbf{N},\mathbf{L},\mathbf{v}) = p(\mathbf{h}|\mathbf{a})p(\mathbf{g}|\mathbf{N})$. Since Gaussian RBMs model the albedo $\mathbf{a}$ and the surface normals $\mathbf{N}$, the two factorized conditional distributions have the same form as Eq. 2.20.

*Conditional 2* factorizes into a product of Gaussian distributions over $N_v$ pixel-specific albedo variables:

$$p(\mathbf{a}|\mathbf{N},\mathbf{L},\mathbf{h},\mathbf{V}) = \prod_i^{N_v} p(a_i|\mathbf{N},\mathbf{L},\mathbf{h},\mathbf{V}) \sim$$
$$\prod_i^{N_v} \mathcal{N}\Big(a_i \Big| \frac{\sigma_{a_i}^2 \sum_p s_{ip} v_{ip} + \phi_i^h \sigma_{v_i}^2}{\sigma_{a_i}^2 \sum_p s_{ip}^2 + \sigma_{v_i}^2}; \frac{\sigma_{a_i}^2 \sigma_{v_i}^2}{\sigma_{a_i}^2 \sum_p s_{ip}^2 + \sigma_{v_i}^2}\Big),$$

where $s_{ip} = \mathbf{n}_i^\mathsf{T} \boldsymbol{\ell}_p$ is the illumination shading at pixel $i$ and $\phi_i^h = b_i + \sigma_{a_i}^2 \sum_j W_{ij} h_j$ is the top-down influence of the albedo GRBM.

This conditional distribution has a very intuitive interpretation. When a light source has zero strength, ($\boldsymbol{\ell}_p = 0 \rightarrow s_{ip} = 0$), then $p(a_i|\mathbf{n}_i, \boldsymbol{\ell}_p, \mathbf{h}, v_i)$ has mean at $\phi_i^h$, which is purely the top-down activation.

*Conditional 3* factorizes into a product distribution over $P$ separate light variables: $p(\mathbf{L}|\mathbf{N},\mathbf{a},\mathbf{V}) = \prod_{p=1}^P p(\boldsymbol{\ell}_p|\mathbf{N},\mathbf{a},\mathbf{v}_p)$, where $p(\boldsymbol{\ell}_p|\mathbf{N},\mathbf{a},\mathbf{v}_p)$ is defined by a quadratic energy function:

$$\begin{aligned} E(\boldsymbol{\ell}_p|\mathbf{N},\mathbf{a},\mathbf{v}) \;=\; & \frac{1}{2}\boldsymbol{\ell}_p^\mathsf{T}\Big(\boldsymbol{\Lambda} + \sum_i \frac{\mathbf{m}_i \mathbf{m}_i^\mathsf{T}}{\sigma_{v_i}^2}\Big)\boldsymbol{\ell}_p \\ & -\Big(\boldsymbol{\mu}_l^\mathsf{T}\boldsymbol{\Lambda} + \sum_i \frac{v_{ip}\mathbf{m}_i}{\sigma_{v_i}^2}\Big)^\mathsf{T}\boldsymbol{\ell}_p. \end{aligned}$$

Hence the conditional distribution over $\boldsymbol{\ell}_p$ is a multivariate Gaussian of the form:

$$p(\boldsymbol{\ell}_p|\mathbf{N},\mathbf{a},\mathbf{v}) \sim \mathcal{N}(\boldsymbol{\ell}_p|\tilde{\boldsymbol{\Lambda}}^{-1}\Big(\boldsymbol{\mu}_l^\mathsf{T}\boldsymbol{\Lambda} + \sum_i \frac{v_{ip}\mathbf{m}_i}{\sigma_{v_i}^2}\Big); \tilde{\boldsymbol{\Lambda}}^{-1}),$$

where $\tilde{\boldsymbol{\Lambda}} = \boldsymbol{\Lambda} + \sum_i \frac{\mathbf{m}_i \mathbf{m}_i^\mathsf{T}}{\sigma_{v_i}^2}$, and $\mathbf{m}_i = a_i \mathbf{n}_i$.

*Conditional 4* can be decomposed into a product of distributions over the surface normals of each pixel:

$$p(\mathbf{N}|\mathbf{L},\mathbf{a},\mathbf{g},\mathbf{V}) = \prod_i p(\mathbf{n}_i|\mathbf{L},\mathbf{g},a_i,\mathbf{v}_i)$$

Since in our model we have the soft norm constraint on $\mathbf{n}_i$ ( $\frac{\eta}{2}\sum_i^{N_v}(\mathbf{n}_i^\mathsf{T}\mathbf{n}_i - 1.0)^2$), there is no simple closed form for $p(\mathbf{n}_i|\mathbf{L},\mathbf{g},a_i,\mathbf{v}_i)$. We use the Hamiltonian Monte Carlo (HMC) algorithm for sampling.

HMC (Duane et al., 1987; Neal, 2010) is an auxiliary variable MCMC method which combines Hamiltonian dynamics with the Metropolis algorithm to sample continuous random variables. In order to use HMC, we must have a differentiable energy function over the variables. In this case, the energy of conditional 4 takes form:

$$\begin{aligned} E(\mathbf{n}_i) \;=\; & \frac{1}{2}\mathbf{n}_i{}^\mathsf{T}\left(\frac{\sum_p (a_i\boldsymbol{\ell}_p)(a_i\boldsymbol{\ell}_p)^\mathsf{T}}{\sigma_{v_i}^2} + \mathbf{D}_i\right)\mathbf{n}_i \\ & -\left(\frac{a_i\sum_p v_{ip}\boldsymbol{\ell}_p}{\sigma_{v_i}^2} + \boldsymbol{\phi}_i^{g\mathsf{T}}\mathbf{D}_i\right)\mathbf{n}_i + \frac{\eta}{2}(\mathbf{n}_i{}^\mathsf{T}\mathbf{n}_i - 1)^2, \end{aligned}$$

where $\boldsymbol{\phi}_i^g$ is the top-down mean of $\mathbf{n}_i$ from the g-layer, and $\mathbf{D}_i = diag(\sigma_{n_{i1}}^{-2}, \sigma_{n_{i2}}^{-2}, \sigma_{n_{i3}}^{-2})$ is the $3 \times 3$ diagonal matrix.

We note that there is a linear ambiguity when we estimate the normals and lighting direction. In Eq. 3.16, $\mathbf{n}_i^\intercal \boldsymbol{\ell}_p = \mathbf{n}_i^\intercal \mathbf{R}\mathbf{R}^{-1}\boldsymbol{\ell}_p$. This means that we can only estimate $\mathbf{n}_i$ and $\boldsymbol{\ell}_p$ up to a linear transformation. Fortunately, while $\mathbf{R}$ is unknown, it is *constant* across $\{\mathbf{v}_i\}_{i=1}^P$ due to the learned priors over $\mathbf{N}$, $\mathbf{a}$ and $\boldsymbol{\ell}$. Therefore, recognition and image relighting tasks (Sec. 3.2.4) are not affected.

### 3.2.3 Learning

Learning is accomplished using a variant of the EM algorithm. In the E-step, MCMC samples are drawn from the approximate posterior distribution (Neal and Hinton, 1998). We first sample from the conditional distributions in Sec. 3.2.2 to approximate the posterior $p(\mathbf{a}, \mathbf{N}, \mathbf{L}, \mathbf{h}, \mathbf{g}|\mathbf{V}; \theta^{old})$. We then optimize the joint log-likelihood function w.r.t. the model parameters. Specifically,

$$\triangle\theta = -\alpha \int p(\mathbf{a}, \mathbf{N}, \mathbf{L}, \mathbf{h}, \mathbf{g}|\mathbf{V}; \theta^{old}) \frac{\partial}{\partial\theta}\big\{E(\mathbf{V}, \mathbf{a}, \mathbf{N}, \mathbf{L}, \mathbf{h}, \mathbf{g}; \theta)\big\} \tag{3.20}$$

where $\alpha$ is the learning rate. We approximate the integral using:

$$\frac{1}{N}\sum_i^N \frac{\partial}{\partial\theta}\big\{E(\mathbf{V}, \mathbf{a}^{(i)}, \mathbf{N}^{(i)}, \mathbf{L}^{(i)}, \mathbf{h}^{(i)}, \mathbf{g}^{(i)}; \theta)\big\},$$

where the samples $\{\mathbf{a}^{(i)}, \mathbf{N}^{(i)}, \mathbf{L}^{(i)}, \mathbf{h}^{(i)}, \mathbf{g}^{(i)}\}$ are approximately drawn from the posterior distribution $p(\mathbf{a}, \mathbf{N}, \mathbf{L}, \mathbf{h}, \mathbf{g}|\mathbf{V}; \theta^{old})$ in the E-step. Maximum likelihood learning of GRBMs (and DBNs) is intractable. We therefore turn to Contrastive Divergence (CD) (Hinton, 2002) to compute an approximate gradient during learning. The complete training algorithm for the DLN in presented in Alg. 5.

Rather than starting with randomly initialized weights, we can achieve better convergence by first training the albedo GRBM on a separate face database. We can then transfer the learned weights before learning the complete DLN.

### 3.2.4 Experiments

We experiment with the Yale B and the Extended Yale B face databases. Combined, the two databases contain 64 frontal images of 38 different subjects. 45 images for each subject are further divided into 4 subsets of increasing illumination variations. Fig. 3.12 shows samples from the Yale B and Extended Yale B database.

For each subject, we used approximately 45 frontal images for our experiments[4]. We separated 28 subjects from the Extended Yale B database for training and held-out all 10 subjects from the original Yale B database for testing[5]. The preprocessing step involved downsizing the face images to the resolution of $24 \times 24$. Using equations of Sec. 3.2.2, we can infer one albedo image and one set of surface normals from each of the 28 subjects. These 28 training albedo and surface normal samples are insufficient for multilayer generative models with millions of parameters. Therefore, we leverage a large set of the face images from the Toronto Face Database (TFD) (Susskind, 2011). The TFD is a collection of 100,000 face images from a variety of other datasets. To create more training data for the surface normals, we randomly translated all 28 sets of them by $\pm 2$ pixels.

[4] A few of the images are corrupted.

[5] We used the cropped images provided by the Yale B Extended database website: http://goo.gl/LKwtX.

**Algorithm 5** Learning Deep Lambertian Networks

1: Pretrain the $\{\mathbf{a}, \mathbf{h}\}$ albedo GRBM with faces images and initialize $\{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$ of the DLN with the GRBM's parameters.
2: Initialize other weights $\sim \mathcal{N}(0, 0.01^2)$, $\sigma^2 \leftarrow 1.0$.
**repeat**
*//Approximate E-step*:
**for** $n = 1$ **to** #training subjects **do**
3: Given $\mathbf{V}^n$, sample $p(\mathbf{a}, \mathbf{N}, \mathbf{L}, \mathbf{h}, \mathbf{g}|\mathbf{V^n}; \theta^{old})$ using the conditionals defined in Sec. 3.2.2, obtaining samples of $\{\mathbf{a}^{(i)}, \mathbf{N}^{(i)}, \mathbf{L}^{(i)}\}$.
**end for**

*//Approximate M-step*:
4: Treating $\{\mathbf{a}^{(i)}\}$ as training data, CD is used to learn the weights of the albedo GRBM.
5: Treating $\{\mathbf{N}^{(i)}\}$ as training data, CD is used to learn the weights of the surface normal GRBM.
6: Maximum likelihood estimations of the parameters $\sigma_{v_i}^2$ , $\boldsymbol{\mu_\ell}$, and $\boldsymbol{\Lambda}$ are computed.
**until** convergence


Figure 3.12: Examples from the Yale B Extended face database. Each row contains samples from an illumination subset.

The DLN used 2 layer DBNs (instead of single layer GRBMs) to model the priors over $\mathbf{a}$ and $\mathbf{N}$. The albedo DBN had 800 $\mathbf{h}^1$ nodes and 200 $\mathbf{h}^2$ nodes. The normals DBN had 1000 $\mathbf{g}^1$ nodes and 100 $\mathbf{g}^2$ nodes. To see what the DLN's prior on the albedo looks like, we show samples generated by the albedo DBN in Fig. 3.13. Learning the multi-modal albedo prior is made possible by the use of unsupervised TFD data.

### 3.2.5 Inference

After learning, we investigated the inference process in the DLN. Although the DLN can use multiple images of the same object during inference, it is important to investigate how well it performs with a *single* test image. We are also interested in the number of iterations that sampling would take to find the posterior modes.

In our first experiment, we presented the model with a single Yale B face image from a *held-out* test subject, as shown in Fig. 3.14. The light source illuminates the subject from the bottom right, causing a significant shadow across the top left of the subject's face. Since the albedo captures a

Figure 3.13: Random samples after 50,000 Gibbs iterations of the Deep Belief Network modeling the learned albedo prior.

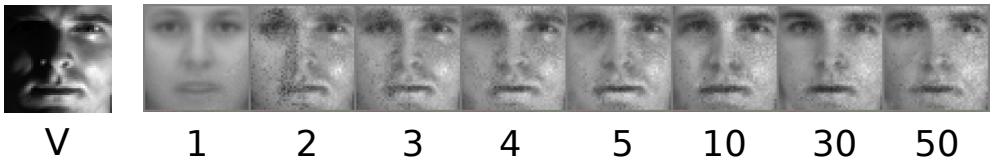



Figure 3.14: **Left**: A single input test image. **Right**: Intermediate samples of estimated albedo during alternating Gibbs sampling: iterations 1 to 50. The albedo was initialized with the visible biases.

lighting invariant representation of the face, correct posterior distribution should automatically perform illumination normalization. Using the algorithm described in Sec. 3.2.2, we clamp the visible nodes to the test face image and sample from the 4 conditionals in an alternating fashion. HMC was used to sample from $\mathbf{N}$. In total, we perform 50 iterations of alternating Gibbs sampling. During each iteration, the $\mathbf{N}$ variables are sampled using HMC with 20 leap-frog iterations and 10 HMC steps. The step size was set to 0.01 with a mass of 2.0. The acceptance rate was around 0.7.

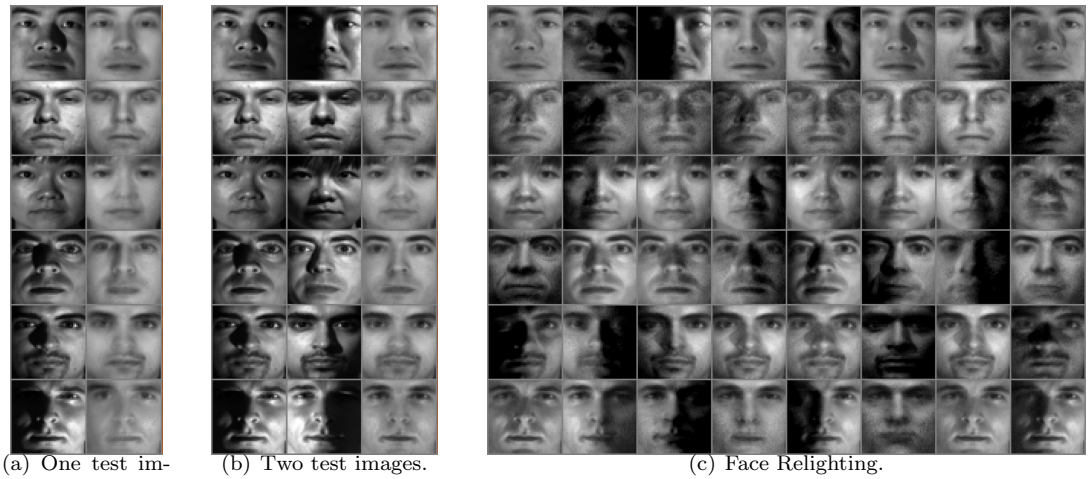
We plot the intermediate samples of the inferred albedo $\mathbf{a}$ from iterations 1 to 50 in Fig. 3.14. Note that the Gibbs chain quickly jumps (at iteration 5) into the correct mode. Good results are obtained due to the knowledge transfer of the albedo and surface normals learned from other subjects. We next randomly selected single test images from the 10 Yale B test subjects. Using exactly the same sampling algorithm, Fig. 3.15(a) shows the inferred albedos. The left column displays the test image, while the right column contains the estimated albedo. We also found that using two test images per subject improves performance. Specifically, we sampled from $p(\mathbf{a}, \mathbf{N}|\mathbf{V} \in \mathbb{R}^{N_v \times 2})$ instead of $p(\mathbf{a}, \mathbf{N}|\mathbf{v} \in \mathbb{R}^{N_v})$. The results are displayed in Fig. 3.15(b).

### 3.2.6 Relighting

The task of face relighting is useful to demonstrate *strong generalization* capabilities of the model. The goal is to generate face images of a particular person under never-before seen lighting conditions. Realistic images can only be generated if the albedo and surface normals of that particular person were correctly inferred. We first sample the lighting variable $\boldsymbol{\ell}$ from its Gaussian prior defined by $\{\boldsymbol{\mu}, \boldsymbol{\Lambda}\}$. Conditioned on the inferred $\mathbf{a}$ and $\mathbf{N}$ (see Fig. 3.15(b)), we use Eq. 3.16 to draw samples of $\mathbf{v}$. Fig. 3.15(c) shows relighted face images of *held-out* test subjects.

### 3.2.7 Recognition

We next test the performance of DLN at the task of face recognition. For the 10 test subjects of Yale B, only image(s) from subset 1 (with 7 images) are used for training. Images from subsets 2-4 are used for testing. In order to use DLN for recognition, we first infer the albedo ($a_i$) and surface normals



(a) One test image. (b) Two test images. (c) Face Relighting.

Figure 3.15: **Left**: Inference results when using only a single test image. The 1st column contains the test images and the 2nd column contains the albedos. **Middle**: Results improve slightly when using an additional test image with a different illumination. **Right**: Using the estimated albedo and surface normals, we show synthesized images under novel lighting conditions.

($\mathbf{n}_i$) conditioned on the provided training image(s) of test subjects. For every subject, a 3 dimensional linear subspace is spanned by the inferred albedo and surface normals. In particular, we consider the matrix $\mathbf{M}$ of dimensions $N_v \times 3$, with the $i$-th row set to $\mathbf{m}_i = a_i\mathbf{n}_i$. The columns of $\mathbf{M}$ span the 3 dimensional linear subspace. Test images of the test subjects are compared to all 10 subspaces and are labeled according to the label of its nearest subspace.

Fig. 3.16 plots the recognition errors as a function of number of training images used. DBN results are obtained by training a 2 layer DBN directly on the training images, and a linear SVM was trained on the top-most hidden activations of the DBN. That standard DBN can not handle lighting variations very accurately. Another approach, called Normalized Correlation, first normalizes images to unit norm. For each test image, its cosine similarity to all training images is computed. The test image takes on the label of the closest training image. Normalized Correlation performs significantly better than Nearest Neighbor due to its normalization, which removes some of the lighting variations. Finally, the SVD method finds a 3 dimensional linear subspace (with the largest singular values) spanned by the training images of each of the test subjects. A test image is assigned to the closest subspace.

We note that for the important task of *one-shot* recognition, DLN significantly outperforms many other methods. In the computer vision literature, (Zhang and Samaras, 2006; Wang et al., 2009) report lower error rates on the Yale B dataset. However, their algorithms make use of pre-existing 3D morphable models, whereas the DLN learns the 3D information automatically from 2D images.

### 3.2.8 Generic Objects

The DLN is applicable not only on face images but also images of generic objects. We used 50 objects from the Amsterdam Library of Images (ALOI) database (Geusebroek et al., 2005). For every object, 15 images of varying lighting were divided into 10 for training and 5 for testing. Using the provided masks for each object, images are cropped and rescaled to the resolution of $48 \times 48$. We used a DLN

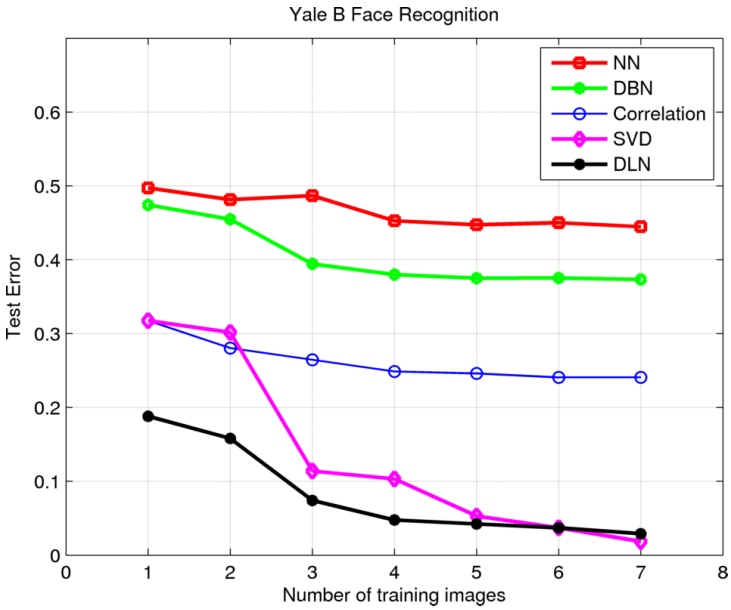



Figure 3.16: Recognition results on the Yale B face database. **NN**: nearest neighbor. **DBN**: Deep Belief Network. **Correlation**: normalized cross correlation. **SVD**: singular value decomposition. **DLN**: Deep Lambertian Network.

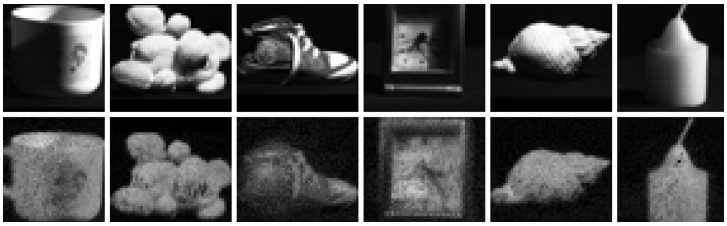


Figure 3.17: Inference conditioned on test objects, using 50 Gibbs iterations. **Top**: Images of objects under new illumination. **Bottom**: Inferred albedo. It is interesting how the model can estimate the albedo of the dark side of the cup.

with $N_h = 1000$ and $N_g = 1500$. A 500 $\mathbf{h}^2$ layer and 500 $\mathbf{g}^2$ layer were also added. After training, we performed posterior inference using one of the held-out image. Fig. 3.17 shows results. The top row contains test images and the bottom row displays the inferred albedo images after 50 alternating Gibbs iterations.

### 3.2.9 Shortcomings

There are several short comings with the model. With respect to modeling, cast shadows and specular reflections will not be handled accurately by the Deep Lambertian Network. In addition, we only assume one light source in this work. Multiple light sources can be handled in a straight forward way, but the number of light sources are needed. A possible extension is to use the Phong reflection model instead of the Lambertian reflection model. Another failure case is when the the training images are not aligned properly. This is due to the fact that the albedo and surface normals are sensitive to the underlying object position.

## 3.3 Conclusions and Future Work

In this chapter, we have looked at two weaknesses of RBM-based generative models: robustness to corruptions and illumination variations. For both cases, we have added higher-order multiplicative interactions to the original model in order to address these concerns. We first described the novel Robust Boltzmann Machine which allows Boltzmann Machines to be robust to noise and occlusions. By first training on noise-free images followed by unsupervised learning on noisy images, our model can learn the *structure* of the noise which allows it to perform much better on face denoising and recognition tasks. We then introduced a generative model with meaningful latent variables and multiplicative interactions simulating the Lambertian Reflectance model. We have shown that by learning priors on these illumination-invariant variables directly from data, we can improve on one-shot recognition tasks as well as generate images under novel illuminations.

For future work, we can follow along the same lines of thought and seek to explicitly add additional variations of natural images and objects in the real world. Geometric transformation such as viewpoint changes is a possibility. In this case, a full 3D model would need to be represented by the latent variables of the generative model. A first question that must be addressed is what kind of 3D representation should we employ? Possible candidates include point clouds, surface meshes, or voxel grids.

# Chapter 4

# Density learning with structure

Unsupervised learning of generative models is important for revealing structure in the data and for discovering features that can be used for subsequent discriminative learning. It is also useful for creating a good prior that can be used for tasks such as image denoising and inpainting or tracking animate motion. While the most popular deep learning models have focused on the discriminative performance of deep learning models, the basic task of learning the probability density of the data using latent variable generative models is still very important. In this chapter we propose two ways to improve density models with a ideas from deep learning.

Section 4.1 discusses how a multilayer mixture of factor analyzers (MFA) can be learned using the stacking technique similar to how Deep Belief Nets are formed out of Restricted Boltzmann Machines. An efficient way to learn deep density models that have many layers of latent variables is to learn one layer at a time using a model that has only one layer of latent variables. After learning each layer, samples from the posterior distributions for that layer are used as training data for learning the next layer. This approach is commonly used with Restricted Boltzmann Machines, which are *undirected* graphical models with a single hidden layer, but it can also be used with Mixtures of Factor Analysers (MFAs) which are *directed* graphical models. In this section, we present a greedy layer-wise learning algorithm for Deep Mixtures of Factor Analysers (DMFAs). Even though a DMFA can be converted to an equivalent shallow MFA by multiplying together the factor loading matrices at different levels, learning and inference are much more efficient in a DMFA and the sharing of each lower-level factor loading matrix by many different higher level MFAs prevents overfitting. We demonstrate empirically that DMFAs learn better density models than both MFAs and two types of Restricted Boltzmann Machine on a wide variety of datasets.

Section 4.2 proposes a new model and learning algorithm for learning conditional distributions. Multilayer perceptrons (MLPs) or neural networks are popular models used for nonlinear regression and classification tasks. As regressors, MLPs model the conditional distribution of the predictor variables $Y$ given the input variables $X$. However, this predictive distribution is assumed to be unimodal (e.g. Gaussian). For tasks involving structured prediction, the conditional distribution should be multi-modal, resulting in one-to-many mappings. By using stochastic hidden variables rather than deterministic ones, Sigmoid Belief Nets (SBNs) can induce a rich multimodal distribution in the output space. However, previously proposed learning algorithms for SBNs are not efficient and unsuitable for modeling real-valued data. In this section, we propose a stochastic feedforward network with hidden layers composed

of *both deterministic and stochastic* variables. A new Generalized EM training procedure using importance sampling allows us to efficiently learn complicated conditional distributions. Our model achieves superior performance on synthetic and facial expressions datasets compared to conditional Restricted Boltzmann Machines and Mixture Density Networks. In addition, the latent features of our model improves classification and can learn to generate colorful textures of objects.

## 4.1 Deep Mixture of Factor Analyzers

A recent latent variable density model based on Markov Random Fields is the Gaussian Restricted Boltzmann Machine (GRBM) (Hinton and Salakhutdinov, 2006). A GRBM can be viewed as a mixture of diagonal Gaussians with the number of components exponential in the number of hidden variables, but with a lot of parameter sharing between the exponentially many Gaussians. See Section 2.1.2 for a full discussion on GRBMs. In Hinton et al. (2006), it was shown that a trained RBM model can be improved by using a second RBM to create a model of the "aggregated posterior" (Eq. 4.9) of the first RBM, where the aggregated posterior is the equally weighted average of the posterior distributions over the hidden units of the first RBM for each training case. The second RBM is then used to replace the prior over the hidden units of the first RBM that is implicitly defined by the weights and biases of the first RBM. With mild assumptions on how training is performed at the higher layer, it was proven that a variational lower bound on the log-likelihood is guaranteed to improve. The second level RBM can do a better job of modeling the first RBM's aggregated posterior than the first level RBM because its parameters are not also being used to model the conditional distribution of the data given the states of the units in the first hidden layer.

A rival model for real-valued high-dimensional data is the Mixture of Factor Analyzers (MFA) (Ghahramani and Hinton, 1996). MFAs simultaneously perform clustering and dimensionality reduction of the data by making locally linear assumptions (Verbeek, 2006). Unlike RBMs, MFAs are directed graphical models where a multivariate standard normal prior is specified for the latent factors for all components. Learning typically uses the EM algorithm to maximize the data log-likelihood. Each FA in the mixture has an isotropic Gaussian prior over its factors and a Gaussian posterior for each training case, but when the posterior is aggregated over many training cases it will typically be non-Gaussian. We can, therefore, improve a variational lower bound on the log probability of the training data by replacing the prior of each FA by a separate, second-level MFA that learns to model the aggregated posterior of that FA better than it is modeled by an isotropic Gaussian. Empirically, the average test log-likelihood also increases for models of both low and high-dimensional data.

While it is true that a two layer MFA can be collapsed back into a standard one layer MFA, learning the two models is nevertheless quite different due to the sharing of factor loadings among the second layer components of the Deep MFA. Parameter sharing helps to reduce overfitting and greatly reduces the computational cost of learning. The EM algorithm also benefits from an easier objective function due to the greedy layer-wise learning, so it is less likely to get stuck in poor local optima.

In related works, multilayer factor analysis was also part of the model in (Chen et al., 2011). However, that work mainly focused on learning convolutional features with non-parametric Bayesian priors on the *parameters*. By using max-pooling and decimation of the first layer factors, their model was designed to learn discriminative features, rather than a top-down generative model of pixel values.

### 4.1.1 Mixture of Factor Analysers

The basic building blocks of Deep Mixture of Factor Analysers are Factor Analysers. Factor analysis was first introduced in psychology as a latent variable model to find the "underlying factor" behind covariates. The latent variables are called factors and are of lower dimension than the covariates. Factor analyzers are linear models as the factor loadings span a linear subspace within the vector space of the covariates. To deal with non-linear data distributions, Mixtures of Factor Analyzers (MFA) (Ghahramani and Hinton, 1996) can be used. MFAs approximate nonlinear manifolds by making local linear assumptions.

Let $\mathbf{x} \in \mathbb{R}^D$ denote the $D$-dimensional data, $\{\mathbf{z} \in \mathbb{R}^d : d \leq D\}$ denote the $d$-dimensional latent variable, and $c \in \{1, \ldots, C\}$ denote the component indicator variable of $C$ total components. The MFA is a directed generative model, defined as follows:

$$p(c) = \pi_c, \quad \sum_{c=1}^{C} \pi_c = 1, \tag{4.1}$$

$$p(\mathbf{z}|c) = p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), \tag{4.2}$$

$$p(\mathbf{x}|\mathbf{z}, c) = \mathcal{N}(\mathbf{x}; \mathbf{W}_c\mathbf{z} + \boldsymbol{\mu}_c, \boldsymbol{\Psi}_c), \tag{4.3}$$

where $\mathbf{I}$ is the $d \times d$ identity matrix. The parameters of the $c$-th component include a mixing proportion $\pi_c$, a factor loading matrix $\mathbf{W}_c \in \mathbb{R}^{D \times d}$, mean $\boldsymbol{\mu}_c$, and a diagonal matrix $\boldsymbol{\Psi}_c \in \mathbb{R}^{D \times D}$, which represents the independent noise variances for each of the variables.

By integrating out the latent variable $\mathbf{z}$, a MFA model becomes a mixture of Gaussians with constrained covariance:

$$p(\mathbf{x}|c) = \int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z}, c)p(\mathbf{z}|c)d\mathbf{z} = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \Gamma_c) \tag{4.4}$$

$$\Gamma_c = \mathbf{W}_c\mathbf{W}_c^{\mathsf{T}} + \boldsymbol{\Psi}_c$$

$$p(\mathbf{x}) = \sum_{c=1}^{C} \pi_c \, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \Gamma_c). \tag{4.5}$$

## Inference

For inference, we are interested in the posterior:

$$p(\mathbf{z}, c|\mathbf{x}) = p(\mathbf{z}|\mathbf{x}, c)p(c|\mathbf{x}) \tag{4.6}$$

The posterior over the components can be found using Bayes rule:

$$p(c|\mathbf{x}) = \frac{p(\mathbf{x}|c)p(c)}{\sum_{\gamma=1}^{C} p(\mathbf{x}|\gamma)p(\gamma)} \tag{4.7}$$

Given component $c$, the posterior over the latent factors is also a multivariate Gaussian:

$$p(\mathbf{z}|\mathbf{x}, c) = \mathcal{N}(\mathbf{z}; \mathbf{m}_c, \mathbf{V}_c^{-1}), \tag{4.8}$$

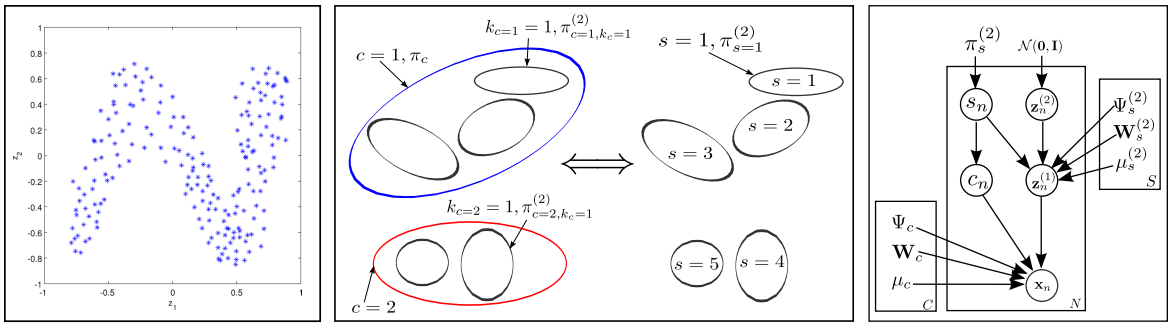



Figure 4.1: **Left**: Hypothetical distribution of the latent factors of a *single* component. In this case the aggregated posterior is not Gaussian distributed. **Middle**: Illustration of our model for 2D data with each ellipse representing a Gaussian component. The first layer MFA has two components colored blue ($c = 1$) and red ($c = 2$). Their mixing proportions are given by $\pi_c$. For the blue component, we further learn a second layer MFA with three components. For the red component, we learn a separate second layer MFA with two components. We also introduce the second layer component indicator variable $k_c = 1, \ldots, K_c$, where $K_c$ is the total number of the second layer components associated with the first layer component $c$. $K_c$ is specific to the first layer component and need not be same for all $c$. In our example, $K_1 = 3$ and $K_2 = 2$. **Right**: Graphical model of a DMFA. Best viewed in color.

where

$$\mathbf{V}_c = \mathbf{I} + \mathbf{W}_c^\mathsf{T} \mathbf{\Psi}_c^{-1} \mathbf{W}_c,$$
$$\mathbf{m}_c = \mathbf{V}_c^{-1} \mathbf{W}_c^\mathsf{T} \mathbf{\Psi}_c^{-1} (\mathbf{x} - \boldsymbol{\mu}_c).$$

Maximum likelihood learning of a MFA model is straightforward using the EM algorithm. During the E-step, Eqs. 4.7, 4.8 are used to compute the posterior over the latent variables given the current setting of the model parameters. During the M-step, the expected complete-data log-likelihood is maximized with respect to the model parameters $\theta = \{\pi_c, \mathbf{W}_c, \boldsymbol{\mu}_c, \mathbf{\Psi}_c\}_{c=1}^C$:

$$\mathbb{E}_{p(\mathbf{z},c|\mathbf{x};\theta_{old})}[\log p(\mathbf{x}, \mathbf{z}, c; \theta)]$$

### 4.1.2 Deep Mixtures of Factor Analysers

After MFA training reaches convergence, the model can be improved by increasing the number $C$ of mixture components or the dimensionality $d$ of the latent factors per component. This amounts to adjusting the conditional distributions $p(\mathbf{x}|\mathbf{z}, c)$. However, as we demonstrate in our experimental results, this approach quickly leads to overfitting, particularly when modeling high-dimensional data.

An alternative is to replace the standard multivariate normal prior on the latent factors: $p(\mathbf{z}|c) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. The "aggregated posterior" is the empirical average over the data of the posteriors over the factors: $\frac{1}{N} \sum_{n=1}^N \sum_{c=1}^C p(\mathbf{z}_n, c|\mathbf{x}_n)$ and a component-specific aggregated posterior is:

$$\frac{1}{N} \sum_{n=1}^N p(\mathbf{z}_n, c_n = c|\mathbf{x}_n) \tag{4.9}$$

If each factor analyser in the mixture was a perfect model of the data assigned to it, the component-specific aggregated posterior would be distributed according to an isotropic Gaussian, but in practice, it

is non-Gaussian. Figure 4.1 (left panel) shows a component-specific aggregated posterior (with $d = 2$), which is highly non-Gaussian. In this case, we wish to replace a simple standard normal prior by a more powerful MFA prior:

$$p(\mathbf{z}|c) = \text{MFA}(\boldsymbol{\theta}_c^{(2)}) \tag{4.10}$$

Here, $\boldsymbol{\theta}_c^{(2)}$ emphasizes that the new MFA's parameters are at the second layer and are specific to component $c$ of the first layer MFA.

More concretely, the variational lower bound on the log-likelihood of the model given data $\mathbf{x}$ is:

$$\begin{aligned}\mathcal{L}(\mathbf{x};\boldsymbol{\theta}) &= \sum_{c=1}^{C}\int_{\mathbf{z}} q(\mathbf{z},c|\mathbf{x};\boldsymbol{\theta})\log p(\mathbf{x},\mathbf{z},c;\boldsymbol{\theta})d\mathbf{z} + \mathcal{H}(q)\\ &= \sum_{c=1}^{C}\int_{\mathbf{z}} q(\mathbf{z},c|\mathbf{x};\boldsymbol{\theta})\Big\{\log p(\mathbf{x}|\mathbf{z},c;\boldsymbol{\theta})\\ &\qquad\qquad + \log p(\mathbf{z}|c) + \log \pi_c\Big\}d\mathbf{z} + \mathcal{H}(q),\end{aligned} \tag{4.11}$$

where $\mathcal{H}(\cdot)$ is the entropy of the posterior distribution $q$ and $\boldsymbol{\theta}$ represent the first layer MFA parameters. The DMFA formulation seeks to find a better prior $\log p(\mathbf{z}|c)$ (using Eq. 4.10), while holding the first layer parameters fixed. Initially, when $q(\mathbf{z},c|\mathbf{x};\boldsymbol{\theta}) \equiv p(\mathbf{z},c|\mathbf{x};\boldsymbol{\theta})$, the bound is tight. Therefore, any increase in the bound will lead to an increase in the true likelihood of the model. Maximizing the bound of Eq. 4.11 with respect to $\boldsymbol{\theta}^{(2)}$ is equivalent to maximizing:

$$\sum_{c=1}^{C}\int_{\mathbf{z}} q(\mathbf{z},c|\mathbf{x};\boldsymbol{\theta})\log p(\mathbf{z}|c;\boldsymbol{\theta}^{(2)}) \tag{4.12}$$

averaged over the training data vectors. This is equivalent to fitting component-specific second-layer MFAs with vectors drawn from $q(\mathbf{z},c|\mathbf{x};\boldsymbol{\theta})$ as data. The same scheme can be extended to training third-layer MFAs. With proper initialization, we are guaranteed to improve the lower bound on the log-likelihood, but the log-likelihood itself can fall (Hinton et al., 2006).

Fig. 4.1 (middle panel) shows a schematic representation of our model. Using $\pi_{k_c}^{(2)}$ to denote the second layer mixing proportion of component $k_c$, we have:

$$\forall\ c : \sum_{k_c=1}^{K_c}\pi_{k_c}^{(2)} = 1 \tag{4.13}$$

A DMFA replaces the old MFA prior $p_{MFA}(\mathbf{z},c) = p(c)p(\mathbf{z}|c)$ with a better prior:

$$p_{DMFA}(\mathbf{z},c) = p(c)p(k_c|c)p(\mathbf{z}|k_c) \tag{4.14}$$

Therefore, when sampling from a DMFA, we first sample $c$ using $\pi_c$, followed by sampling the second layer component $k_c$ using $\pi_{k_c}^{(2)}$. Finally, we can sample $\mathbf{z}$ using the Gaussian of component $k_c$, as in Eq. 5.2.

A simpler, but completely equivalent DMFA formulation is to enumerate over all possible second layer components $k_c$. We use a new component indicator variable $s = 1,\ldots,S$ to denote a specific second layer component, where $S = \sum_{c=1}^{C} K_c$. The mixing proportions are defined as $\pi_s^{(2)} = p(c(s))p(k_c(s)|c(s))$, where $c(s)$ and $k_c(s)$ denotes the first and second layer components $c$ and $k_c$ to which $s$ corresponds.

For example $c(2) = 1$ and $c(5) = 2$. We note that the size of $S$ is exponential in the number of DMFA layers. The generative process of this formulation is very intuitive and we shall use it throughout the remaining sections.

Fig. 4.1 (right panel) shows the graphical model for a 2 layer DMFA. Specifically,

$$p(s) = \pi_s^{(2)} \tag{4.15}$$

$$p(\mathbf{z}^{(2)}|s) = \mathcal{N}(\mathbf{z}^{(2)}; 0, \mathbf{I}) \tag{4.16}$$

$$p(\mathbf{z}^{(1)}|\mathbf{z}^{(2)}, s) = \mathcal{N}\big(\mathbf{z}^{(1)}; \mathbf{W}_s^{(2)}\mathbf{z}^{(2)} + \boldsymbol{\mu}_s^{(2)}, \Psi_s^{(2)}\big) \tag{4.17}$$

$$c \leftarrow c(s), \quad \text{(deterministic)} \tag{4.18}$$

$$p(\mathbf{x}|\mathbf{z}^{(1)}, c) = \mathcal{N}\big(\mathbf{x}; \mathbf{W}_c^{(1)}\mathbf{z}^{(1)} + \boldsymbol{\mu}_c^{(1)}, \Psi_c^{(1)}\big) \tag{4.19}$$

Eq. 4.18 is fully deterministic as every $s$ belongs to one and only one $c$. $\mathbf{z}^{(1)} \in \mathbb{R}^{d^{(1)}}$, $\mathbf{z}^{(2)} \in \mathbb{R}^{d^{(2)}}$, $\mathbf{W}_c^{(1)} \in \mathbb{R}^{D \times d^{(1)}}$, $\mathbf{W}_s^{(2)} \in \mathbb{R}^{d^{(1)} \times d^{(2)}}$, $\boldsymbol{\mu}_c^{(1)} \in \mathbb{R}^{d^{(1)}}$, and $\boldsymbol{\mu}_s^{(2)} \in \mathbb{R}^{d^{(2)}}$. Finally, $\boldsymbol{\Psi}_c^{(1)}$ and $\boldsymbol{\Psi}_s^{(2)}$ are $d^{(1)} \times d^{(1)}$ and $d^{(2)} \times d^{(2)}$ diagonal matrices of the first and second layers respectively.

DMFA has an equivalent *shallow* form, which is obtained by integrating out the latent factors. If we integrate out the first layer factors $\mathbf{z}^{(1)}$, we obtain:

$$\begin{aligned} p(\mathbf{x}|\mathbf{z}^{(2)}, s) = \mathcal{N}\big(\mathbf{x}; \mathbf{W}_c^{(1)}(\mathbf{W}_s^{(2)}\mathbf{z}^{(2)} + \boldsymbol{\mu}_s^{(2)}) + \boldsymbol{\mu}_c^{(1)}, \\ \Psi_c^{(1)} + \mathbf{W}_c^{(1)}\Psi_s^{(2)}{\mathbf{W}_c^{(1)}}^{\mathsf{T}}\big) \end{aligned} \tag{4.20}$$

By further integrating out $\mathbf{z}^{(2)}$:

$$\begin{aligned} p(\mathbf{x}|s) = \mathcal{N}(\mathbf{x}; \mathbf{W}_c^{(1)}\boldsymbol{\mu}_s^{(2)} + \boldsymbol{\mu}_c^{(1)}, \\ \Psi_c + \mathbf{W}_c^{(1)}(\Psi_s^{(2)} + \mathbf{W}_s^{(2)}{\mathbf{W}_s^{(2)}}^{\mathsf{T}}){\mathbf{W}_c^{(1)}}^{\mathsf{T}}\big) \end{aligned} \tag{4.21}$$

From Eq. 4.20, we can see that a DMFA can be reduced to a standard MFA where $\mathbf{z}^{(2)}$ are the factors and $s$ indicates the mixture component. This "collapsed" MFA is regularized due to its parameter sharing. In particular, the means of the components $s$ with the same first layer component $c$ all must lie on a hyperplane spanned by $\mathbf{W}_c^{(1)}$. The covariance of these components all share the same outer product factorization ($\mathbf{W}_c^{(1)}{\mathbf{W}_c^{(1)}}^{\mathsf{T}}$) but with different "core matrices" ($\Psi_s^{(2)} + \mathbf{W}_s^{(2)}{\mathbf{W}_s^{(2)}}^{\mathsf{T}}$).

Assuming that the number of the second layer components are equal, i.e. $\forall c : K_c = K$, a standard shallow MFA with $S = C \times K$ mixture components and $d^{(1)}$ factors per component would require $O(DKd^{(1)}C)$ parameters. A DMFA with two layers, on the other hand, would require $O(Dd^{(1)}C + d^{(1)}d^{(2)}CK) = O((D + d^{(2)}K)d^{(1)}C)$ parameters. Note that a DMFA requires a much smaller number of effective parameters than an equivalent shallow MFA, since $d^{(2)} << D$. As we shall see in Sec. 4.1.6, this sharing of parameters is critical for preventing overfitting.

### 4.1.3 Inference

Exact inference in a collapsed DMFA model is of order $O(CK)$ since the data likelihood must be computed for each mixture component. We can incur a lower cost by using an approximate inference, which is $O(C + K)$. First, we compute the posterior $p(\mathbf{z}^{(1)}, c|\mathbf{x}) = p(\mathbf{z}^{(1)}|\mathbf{x}, c)p(c|\mathbf{x})$ using Eq. 4.7. This posterior is exact if we had a standard normal prior over $\mathbf{z}^{(1)}$, but it is an approximation of the exact

**Algorithm 6** Learning DMFAs

Given data: $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$.
*//Layer 1 training*
Train 1st layer MFA on $X$ with $C$ components and $d$ factors using EM $\rightarrow$ MFA1.

*//Layer 2 training*
Create dataset $Y_c$ for each of the $C$ components.
$Y_c \leftarrow \emptyset$
**for** $i = 1$ to $N$ **do**
  **for** $c = 1$ to $C$ **do**
    compute $p(c|\mathbf{x}_i)$ and $p(\mathbf{z}^{(1)}|\mathbf{x}_i, c)$, Eqs. 4.7 & 4.8.
  **end for**
  Find $\hat{c} = \arg\max_c p(c|\mathbf{x}_i)$.
  Sample $\mathbf{z}_i^{(1)}$ from $\mathcal{N}(\mathbf{z}^{(1)}; \mathbf{m}_{\hat{c}}, \mathbf{V}_{\hat{c}}^{-1})$.
  Add $\mathbf{z}_i^{(1)}$ to dataset $Y_{\hat{c}}$: $Y_{\hat{c}} = Y_{\hat{c}} \cup \{\mathbf{z}_i^{(1)}\}$.
**end for**

$d^{(2)}$ and $K_c$: # of 2nd layer factors and components.
**for** $c = 1$ to $C$ **do**
  Train a separate 2nd layer MFA on $Y_c$ with $d^{(2)}$ factors and $K_c$ components using EM $\rightarrow$ MFA2{c}.
**end for**

posterior of the DMFA model. The entropy of the posterior $p(c|\mathbf{x})$ is likely to be very low in high dimensional spaces. We therefore make a point estimate by selecting the component $c$ with maximum posterior probability:

$$\hat{c} = \arg\max_c p(c)p(c|\mathbf{x}) \tag{4.22}$$

$$\begin{aligned} p(\mathbf{z}^{(1)}|\mathbf{x}) &= \sum_c p(\mathbf{z}^{(1)}|\mathbf{x}, c)p(c|\mathbf{x})dc \\ &\approx p(\mathbf{z}^{(1)}|\mathbf{x}, \hat{c}) \end{aligned} \tag{4.23}$$

For the second layer, we treat $\hat{c}$ and $\mathbf{z}^{(1)}$ as data, and compute the posterior distribution $p(\mathbf{z}^{(2)}, s|\mathbf{z}^{(1)}, \hat{c})$ in a similar fashion.

### 4.1.4 Learning

A DMFA can be trained efficiently using a greedy layer-wise algorithm. The first layer MFA is trained in a standard way. We then use Eq. 4.23 to infer the component $\hat{c}$ and the factors associated with that component for each training case $\{\mathbf{x}_n\}$. We then freeze the first layer parameters and treat the sampled first layer factor values for every component $\left\{\{\mathbf{z}_n^{(1)}\}_c\right\}$ as training data for the second layer MFAs. Algorithm 6 details this layer-wise training algorithm. After greedy learning, "backfitting" by collapsing a DMFA and running additional EM steps is also possible. However, more care is needed to prevent overfitting.

### 4.1.5 Experiments

We demonstrate the advantages of learning DMFAs on both low dimensional and high dimensional datasets, including face images, natural image patches, and speech acoustic data.

**Toronto Face Database** (TFD): The Toronto Face Database is a collection of aligned faces from a variety of (mostly) publicly available face image databases (Susskind, 2011). From the original resolution of $108 \times 108$, we downsampled to resolutions of $48 \times 48$ or $24 \times 24$. We then randomly selected 30,000 images for training, 10,000 for validation, and 10,000 for testing.

**CIFAR-10**: The CIFAR-10 dataset (Krizhevsky, 2009) consists of 60,000 $32 \times 32 \times 3$ color images of 10 object classes. There are 50,000 training images and 10,000 test images. Out of 50,000 training images, 10,000 were set aside for validation.

**TIMIT Speech**: TIMIT is a corpus of phonemically and lexically transcribed speech of American English speakers of different sexes and dialects[1]. The corpus contains a 462-speaker training set, a 50-speaker validation set, and a 24-speaker core test set. For our purposes, we extracted data vectors every 10-ms from the continuous speech data. Each frame analyzes a 25-ms Hamming window using a set of filter banks based on the Fast Fourier Transform. Concatenating 11 frames, we obtain 1353 dimensional input vectors. We randomly selected 30,000 vectors for training, 10,000 for validation, and 10,000 for testing.

**Berkeley Natural Images**: The Berkeley segmentation database (Martin et al., 2001) contain 300 images from natural scenes. We randomly extracted 2 million $8 \times 8$ image patches for training, 50,000 patches for validation, and 50,000 for testing.

**UCI**: We used 4 datasets from the UCI repository (Murphy and Aha, 1995). These are low dimensional datasets and have relatively few training examples. These were the only UCI datasets we tried.

For all image datasets, the DC component of each image was removed: $\mathbf{x} \leftarrow \mathbf{x} - mean(\mathbf{x})$. This removes the huge illumination variations across data samples. No other preprocessing steps were used. For the TIMIT and UCI datasets, we normalize input vectors to zero mean and scale the entire input by a single number to make the average standard deviation be one. For evaluating the log probabilities of DMFAs, we always first collapsed it to a shallow MFA in order to obtain the exact data log-likelihood.

### 4.1.6 Overfitting

We first trained a 20 component MFA on $24 \times 24$ faces until convergence[2], which took 33 iterations. The number of factors was set to half of the input dimensionality, $d^{(1)} = D/2 = 288$. Fig. 4.2 shows the corresponding training and validation log-likelihoods[3]. We next stacked a second MFA layer with five second layer components ($K_c = 5$) for each of the first layer components and $d^{(2)} = 50$ second layer factors. The DMFA (MFA2) model improved as learning continued for an additional 20 iterations (see red and blue lines in Fig. 4.2). As a comparison, immediately after we initially formed the two-layer MFA, we collapsed it into its equivalent shallow representation and performed additional training (magenta and black lines in Fig. 4.2). Observe that the shallow MFA starts overfitting due to its extra capacity (5 times more parameters). MFA2, on the other hand, shows improvements on both the training and validation data. We note that training a shallow MFA with 100 components from random initialization is significantly worse (see Table. 4.1).

To give a sense of the computation costs, training the first layer MFA took 1600 seconds on a multi-core Xeon machine. The second layer MFA training took an additional 580 seconds.

[1] www.ldc.upen.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1

[2] Convergence is achieved when the log-likelihood changed by less than 0.01% from the previous EM iteration.

[3] Similar results were obtained for different numbers of components and factors.

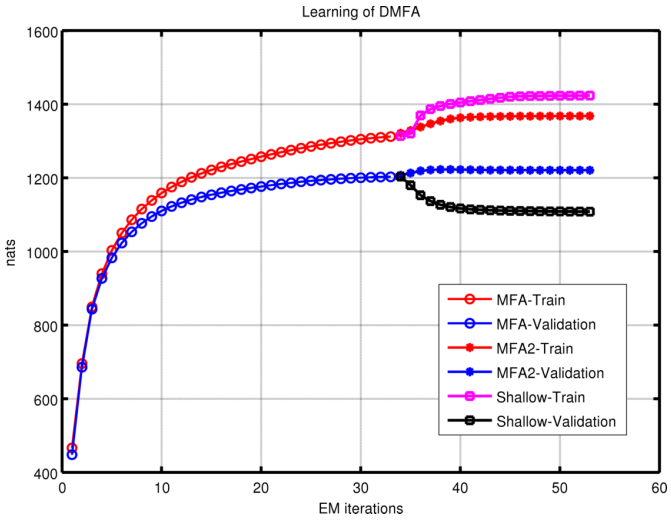



Figure 4.2: DMFA improves over MFA. Overfitting occurs during further training of a shallow MFA with increased capacity. Best viewed in color.

### 4.1.7 Qualitative Results

We next demonstrate qualitative improvements of the samples from a DMFA over a standard MFA model. As the baseline, we first trained a MFA model on 30,000 $24 \times 24$ face images from the TFD, with 288 factors and 100 components. We then trained a DMFA with 20 first layer components and 5 second layer components for each of the 20 first layer components. The DMFA has the same number of parameters as the baseline MFA. The two-layer MFA (MFA2) performs better compared to the standard MFA by around 20 nats on the test set. Fig 4.3 further shows samples from the two models. Qualitatively, the DMFA appears to generate better samples compared to the shallow MFA model.

### 4.1.8 High Dimensional Data

Next, we explore the benefits of DMFAs on the high dimensional CIFAR and TIMIT datasets. We first trained a MFA model with the number of factors equal to half of the input dimensionality. The number of mixture components was set to 20. For MFA2, 5 components with 50 latent factors were used. For the 3rd layer MFA (MFA3) 3 factors with 30 latent factors were used.

Table 4.1 shows the average training and test log-likelihood. In addition, we provide results for two types of RBM models that are commonly used when modeling high-dimensional real-valued data, including image patches and speech. The SSU model is a type of RBM with Gaussian visible and stepped sigmoid hidden units (Nair and Hinton, 2010). By using rectified linear activations, SSU can be viewed as a mixture of linear models with the number of components exponential in the number of hidden variables. A simpler Gaussian RBM (GRBM) model uses Gaussian visible and binary hidden units. It can also be viewed as a mixture of diagonal Gaussians with exponential number of components. For both the GRBM and SSU, we used Fast Persistent Contrastive Divergence (Tieleman and Hinton, 2009) for learning and AIS (Salakhutdinov and Murray, 2008) to estimate their log-partition functions (See Section 2.1.2). The AIS estimators have standard errors of around 5 nats, which are too small to affect the conclusions we can draw from Table 4.1.

The number of parameters for the GRBMs and SSU are matched to the MFA model, which means

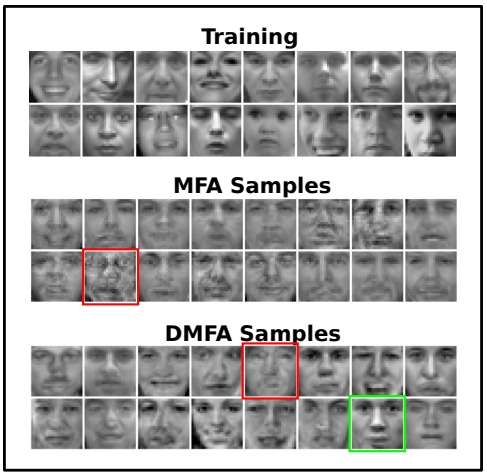


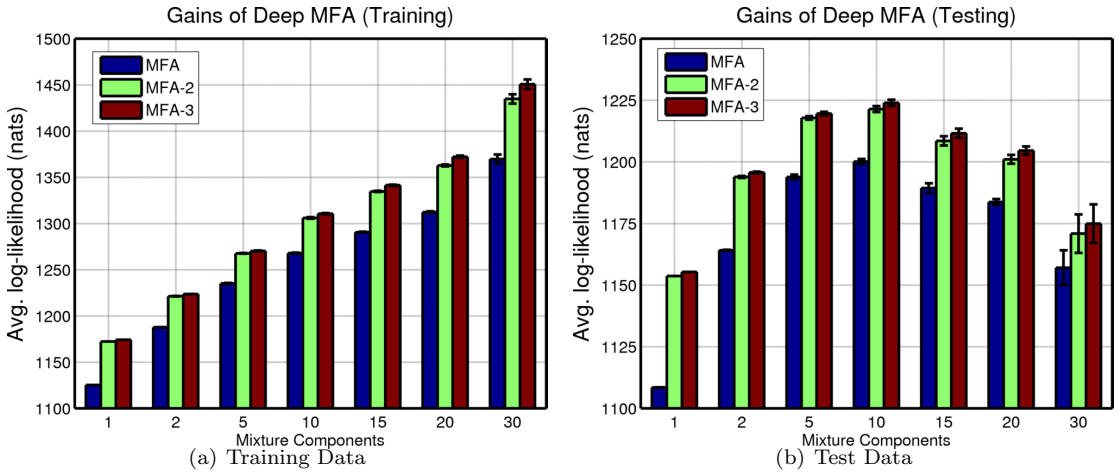

Figure 4.3: **Top**: training images. **Middle**: samples from MFA. **Bottom**: samples from DMFA. The red box highlights the worst looking samples of the two models while the green box highlights a very realistic looking generated sample.

| Dataset | GRBM | SSU | MFA | MFA-2 | MFA-3 | Shallow1 | Shallow2 | Diff-2 | Diff-3 |
|---|---|---|---|---|---|---|---|---|---|
| TFD-24 | 766 | 859 | 1312 | 1368 | 1380 | 1325 | 1506 | 57.1 ± 0.1 | 12± 0.2 |
| | **758** | **841** | **1185** | **1202** | **1207** | **1184** | **1039** | **18.7** ± 0.2 | **4.1** ± 0.08 |
| TFD-24-Rot | 843 | 950 | 1412 | 1469 | 1477 | 1428 | 1505 | 56.9 ± 0.1 | 8.5 ± 0.13 |
| | **822** | **929** | **1283** | **1305** | **1306** | **1284** | **1125** | **21.6** ± 0.2 | **1.4** ± 0.04 |
| TFD-48 | 2426 | 3675 | 6020 | 6141 | 6151 | 6036 | 6461 | 119.2 ± 0.4 | 11.7 ± 0.1 |
| | **2413** | **3557** | **5159** | **5242** | **5250** | **5161** | **4299** | **85.3** ± 0.5 | **5.6** ± 0.1 |
| CIFAR: | 2725 | 2818 | 4486 | 4573 | 4583 | 4565 | 4214 | 86.8 ± 0.4 | 10.6 ± 0.2 |
| | **2365** | **2494** | **3587** | **3621** | **3622** | **3592** | **2873** | **33.2** ± 0.4 | **0.5** ± 0.06 |
| TIMIT: | 1244 | 1316 | 2662 | 2802 | 2804 | 2707 | 3219 | 133.5 ± 0.2 | 1.4 ± 0.05 |
| | **1175** | **1268** | **2298** | **2450** | **2451** | **2305** | **1169** | **147.2** ± 0.5 | **0.4** ± 0.07 |

Table 4.1: Model performance on various high dimensional datasets (nats). TFD-24-Rot is generated by randomly rotating $24 \times 24$ face images in the range of $\pm 45$ deg. Diff-2 and Diff-3 are the gains from going from 1 to 2 layers and from 2 to 3 layers, respectively. For all datasets, Diff-2 and Diff-3 are statistically significant at $p = 0.01$.

that approximately 6,000 hidden nodes are used. Increasing the number of hidden units did not result in any significant improvements of GRBM and SSU models. Hyperparameters are selected using the validation set. After MFA learning converged, a MFA2 model is initialized. The means of the MFA-2 components were slightly perturbed from zero so as to break symmetry. Shallow1 results were obtained by collapsing these newly initialized MFA2 models and further training using EM with early stopping. Shallow2 results were obtained by starting at random initialization (with multiple restarts) with the equivalent number of parameters as the corresponding Shallow1 models. We note the significant gains by DMFAs for the TIMIT and TFD-48 datasets.

Fig. 4.4 displays gains of 2 and 3 layer MFA as we vary the number of the first layer mixture components. It is interesting to observe that MFA and DMFA significantly outperformed various RBM models. This result suggests that it may be possible to improve many of the existing deep networks for modeling real-valued data that use GRBMs for the first hidden layer, though better density models do not necessarily learn features that are better for discrimination.




(a) Training Data (b) Test Data

Figure 4.4: Improvements of DMFA over standard MFA on $24 \times 24$ face images vs. the number of first layer components. Gains are observed across different numbers of first layer components. Surprisingly, while the dataset contains thousands of different people, more than 10 mixture components results in overfitting. Best viewed in color.

| Dataset | dim. | size | Gaussian | Cop. MCDN | MFA | MFA-2 | DMFA gain |
|---|---|---|---|---|---|---|---|
| Parkinsons | 15 | 5875 | -11.65 | -3.48 | -0.63 | -0.33 | $0.296 \pm 0.024$ |
| Ionosphere | 32 | 351 | -41.10 | -27.45 | -20.10 | -18.53 | $1.565 \pm 0.252$ |
| Wine(red) | 11 | 1599 | -13.72 | -11.25 | -10.22 | -10.07 | $0.143 \pm 0.015$ |
| Wine(white) | 11 | 4898 | -13.76 | -12.11 | -11.02 | -10.89 | $0.121 \pm 0.036$ |

Table 4.2: Test set predictive log-likelihood on 4 UCI datasets (nats). Reported results are from 10-fold cross validation on each dataset. MFA results are from our experiments. Other results are from Silva et al. (2011).

### 4.1.9 Low Dimensional Data

DMFAs can also be used with low dimensional data. Following Silva et al. (2011), we used 4 continuous datasets from the UCI repository. We removed the discrete variables from all datasets. For the Parkinsons dataset, one variable from any pair whose Pearson correlation coefficient is greater than 0.98 was also removed (for details see Silva et al. (2011)). Table 4.2 reports the averaged test results using 10-fold cross validation. Compared to the recently introduced Copula Networks, MFAs give much better test predictive performance. Adding a second layer produced significant gains in model performance. The improvements from adding a second layer on all datasets were statistically significant using the paired t-test at $p = 0.01$.

### 4.1.10 Natural Images

One important application of generative models is in the task of image restoration which can be formulated as a MAP estimation problem. As confirmed by Zoran and Weiss (2011), a better prior almost certainly leads to a better signal to noise ratio of the restored image. In addition, Zoran and Weiss (2011) have shown that combining a mixture of Gaussians model trained on $8 \times 8$ patches of natural images with a patch-based denoising algorithm, allowed them to achieve state-of-the-art results. Following their work, we trained a two-layer MFA on $8 \times 8$ patches from the Berkeley database. Two million training

| PIX | PCA | ICA | GMM | MFA | MFA-2 |
|---|---|---|---|---|---|
| 78.3 | 114.2 | 115.9 | 167.2* | 166.5 | 169.3 |

Table 4.3: Average test log-likelihood (in nats) of various models learned on 50,000 $8 \times 8$ test patches. **PIX**: independent pixels. **PCA**: Principle Component Analysis. **ICA**: Independent Component Analysis. **GMM**: Mixture of Gaussians with 200 components. **MFA** Mixture of Factor Analysers with 200 components. **MFA-2** Two layer DMFA. MFA and MFA-2 results are from our experiments, other numbers are taken from Zoran and Weiss (2011). GMM's $167.2^*$ is different from the previously reported 164.5 due to the random extraction of test patches. 167.2 was obtained by evaluating the downloaded model of Zoran and Weiss (2011) on our own test patches.

and 50,000 test patches were extracted from the 200 training and 100 test images, respectively. Table 4.3 shows results. Note that the DMFA improves upon the current state-of-the-art GMMs model of Zoran and Weiss (2011) by about 2 nats, while substantially outperforming other commonly used models including PCA and ICA. Finally, we trained a shallow equivalent to MFA-2 (5 times more parameters than MFA) from random initialization and achieved only 164.9 nats, thereby demonstrating that DMFAs are necessary in order to achieve the extra gain.

### 4.1.11 Allocating more components to more popular Factor Analysers

Until now, we have given every higher level MFA the same number of components to model the aggregated posterior of its lower level Factor Analyser ($\forall c : K_c = K$). While simple to implement, this is not optimal. An alternative is to use more second layer components for the first layer components with bigger mixing proportions. We tested this hypothesis by first training a MFA model on $48 \times 48$ TFD faces, which achieved an average test log-likelihood of 5159 nats. For the two-layer MFA, instead of assigning 5 components to each of the first layer components, we let $K_c \propto \pi_c$, with $min(K_c) = 2$ and $\sum_c^C K_c = 5 \times C$. With all other learning hyper-parameters held constant, the resulting DMFA achieved 5246 nats on the test set. Compared to 5242 nats of our previous model (c.f. Table 4.1), the new method accounted for a gain of 4 nats. As another alternative, a measure of Gaussianity of the aggregated posterior could be used to determine $K_c$.

## 4.2 Learning Stochastic Feedforward Neural Networks

In the second section of the chapter, we propose a model and a learning algorithm for modeling multimodal conditional distributions. This algorithm simply sprinkles in some stochastic hidden nodes to the standard feedforward neural network. The stochastic nodes give the model the ability to be multimodal.

Multilayer perceptrons (MLPs) are general purpose function approximators. The outputs of a MLP can be interpreted as the sufficient statistics of a member of the exponential family (conditioned on the input $X$), thereby inducing a distribution over the output space $Y$. Since the nonlinear activations are all *deterministic*, MLPs model the conditional distribution $p(Y|X)$ with a unimodal assumption (e.g. an isotropic Gaussian)[4].

For many structured prediction problems, we are interested in a conditional distribution $p(Y|X)$ that

[4]For example, in a MLP with one input, one output and one hidden layer: $p(y|\mathbf{x}) \sim \mathcal{N}(y|\mu_y, \sigma_y^2)$, $\mu_y = \sigma\big(W_2\sigma(W_1\mathbf{x})\big)$, $\sigma(a) = 1/(1+\exp(-a))$ is the sigmoid function. Note that the Mixture of Density Network is an exception to the unimodal assumption (Bishop, 1994).

is multimodal and may have complicated structure[5]. One way to model the multi-modality is to make the hidden variables stochastic. Conditioned on a particular input $X$, different hidden configurations lead to different $Y$. Sigmoid Belief Nets (SBNs) (Neal, 1990, 1992) are models capable of satisfying the multi-modality requirement. With binary input, hidden, and output variables, they can be viewed as directed graphical models where the sigmoid function is used to compute the degrees of "belief" of a child variable given the parent nodes. Inference in such models is generally intractable. The original paper by Neal (Neal, 1992) proposed a Gibbs sampler which cycles through the hidden nodes one at a time. This is problematic as Gibbs sampling can be very slow when learning large models or fitting moderately-sized datasets. In addition, slow mixing of the Gibbs chain would typically lead to a biased estimation of gradients during learning. A variational learning algorithm based on the mean-field approximation was proposed in Saul et al. (1996) to improve the learning of SBNs. A drawback of the variational approach is that, similar to Gibbs, it has to cycle through the hidden nodes one at a time. Moreover, beside the standard mean-field variational parameters, additional parameters must be introduced to lower-bound an intractable term that shows up in the expected free energy, making the lower-bound looser. Gaussian fields are used in Barber and Sollich (1999) for inference by making Gaussian approximations to units' input, but there is no longer a lower bound on the likelihood.

Here, we introduce the Stochastic Feedforward Neural Network (SFNN) for modeling conditional distributions $p(\mathbf{y}|\mathbf{x})$ over *continuous* real-valued $Y$ output space. Unlike SBNs, to better model continuous data, SFNNs have hidden layers with *both* stochastic *and* deterministic units. The left panel of Fig. 4.5 shows a diagram of SFNNs with multiple hidden layers. Given an input vector $\mathbf{x}$, different states of the stochastic units can generate different modes in $Y$. For learning, we present a novel Monte Carlo variant of the Generalized Expectation Maximization algorithm. Importance sampling is used for the E-step for inference, while error backpropagation is used by the M-step to improve a variational lower bound on the data log-likelihood. SFNNs have several attractive properties, including:

- We can draw samples from the exact model distribution without resorting to MCMC.
- Stochastic units form a distributed code to represent an exponential number of mixture components in output space.
- As a directed model, learning does not need to deal with a global partition function.
- A Combination of stochastic and deterministic hidden units can be jointly trained using the backpropagation algorithm, as in standard feed-forward neural networks.

The two main alternative models are Conditional Gaussian Restricted Boltzmann Machines (C-GRBMs) (Taylor et al., 2006) and Mixture Density Networks (MDNs) (Bishop, 1994). Note that Gaussian Processes (Rasmussen and Williams, 2006) and Gaussian Random Fields (Rue and Held, 2005) are unimodal and therefore incapable of modeling a multimodal $Y$. Conditional Random Fields (Lafferty, 2001) are widely used in NLP and vision, but often assume $Y$ to be discrete rather than continuous. C-GRBMs are popular models used for human motion modeling (Taylor et al., 2006), structured prediction (Mnih et al., 2011), and as a higher-order potential in image segmentation (Li et al., 2013). While C-GRBMs have the advantage of exact inference, they are energy based models that define *different* partition functions for different input $X$. Learning also requires Gibbs sampling which is prone to poor mixing. MDNs use a mixture of Gaussians to represent the output $Y$. The components' means, mixing proportions, and the output variances are all predicted by a MLP conditioned on $X$. As with SFNNs,

[5] An equivalent problem is learning one-to-many functions from $X \mapsto Y$.

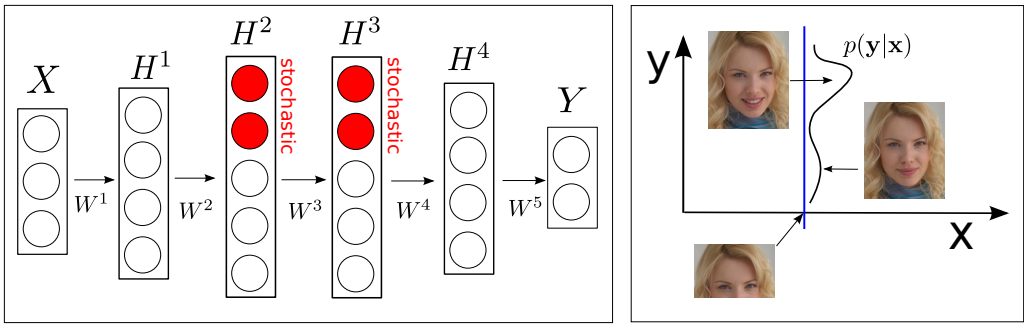



Figure 4.5: *Stochastic Feedforward Neural Networks.* **Left***: Network diagram. Red nodes are stochastic and binary, while the rest of the hiddens are deterministic sigmoid nodes.* **Right***: motivation as to why multimodal outputs are needed. Given the top half of the face* $\mathbf{x}$*, the mouth in* $\mathbf{y}$ *can be different, leading to different expressions.*

the backpropagation algorithm can be used to train MDNs efficiently. However, the number of mixture components in the output $Y$ space must be pre-specified and the number of parameters is linear in the number of mixture components. In contrast, with $N_h$ stochastic hidden nodes, a SFNN can use its distributed representation to model up to $2^{N_h}$ mixture components in the output $Y$.

### 4.2.1 Stochastic Feedforward Neural Networks

SFNNs contain binary stochastic hidden variables $\mathbf{h} \in \{0,1\}^{N_h}$, where $N_h$ is the number of hidden nodes. For clarity of presentation, we construct a SFNN from a one-hidden-layer MLP by replacing the sigmoid nodes with stochastic binary ones. Note that other types stochastic units can also be used. The conditional distribution of interest, $p(y|\mathbf{x})$, is obtained by marginalizing out the latent stochastic hidden variables: $p(y|\mathbf{x}) = \sum_{\mathbf{h}} p(y, \mathbf{h}|\mathbf{x})$. SFNNs are directed graphical models where the generative process starts from $\mathbf{x}$, flows through $\mathbf{h}$, and then generates output $y$. Thus, we can factorize the joint distribution as: $p(y, \mathbf{h}|\mathbf{x}) = p(y|\mathbf{h})p(\mathbf{h}|\mathbf{x})$. To model real-valued $y$, we have $p(y|\mathbf{h}) = \mathcal{N}(y|W_2\mathbf{h} + b_2, \sigma_y^2)$ and $p(\mathbf{h}|\mathbf{x}) = \sigma(W_1\mathbf{x} + b_1)$, where $b$ is the bias. Since $\mathbf{h} \in \{0,1\}^{N_h}$ is a vector of Bernoulli random variables, $p(y|\mathbf{x})$ has *potentially* $2^{N_h}$ different modes[6], one for every possible binary configurations of $\mathbf{h}$. The fact that $\mathbf{h}$ can take on different states in a SFNN is the reason why we can learn one-to-many mappings, which would be impossible with standard MLPs.

The modeling flexibility of SFNN comes with computational costs. Since we have a mixture model with potentially $2^{N_h}$ components conditioned on any $\mathbf{x}$, $p(y|\mathbf{x})$ does not have a closed-form expression. We can use Monte Carlo approximation with $M$ samples for its estimation:

$$p(y|\mathbf{x}) \simeq \frac{1}{M} \sum_{m=1}^{M} p(y|\mathbf{h}^{(m)}), \qquad \mathbf{h}^{(m)} \sim p(\mathbf{h}|\mathbf{x}). \tag{4.24}$$

This estimator is unbiased and has relatively low variance. This is because the accuracy of the estimator does not depend on the dimensionality of $\mathbf{h}$ and that $p(\mathbf{h}|\mathbf{x})$ is factorial, meaning that we can draw samples from the *exact* distribution.

If $\mathbf{y}$ is discrete, it is sufficient for all of the hiddens to be discrete. However, using only discrete hiddens is suboptimal when modeling real-valued output $Y$. This is due to the fact that while $y$ is continuous,

[6]In practice, due to weight sharing, we will not be able to have close to that many modes for a large $N_h$.

there are only a finite number of discrete hidden states, each one (e.g. $\mathbf{h}'$) leads to a component which is a Gaussian: $p(y|\mathbf{h}') = \mathcal{N}(y|\mu(\mathbf{h}'), \sigma_y^2)$. The mean of a Gaussian component is a function of the hidden state: $\mu(\mathbf{h}') = W_2^{\mathsf{T}}\mathbf{h}' + b_2$. When $\mathbf{x}$ varies, only the probability of *choosing* a specific hidden state $\mathbf{h}'$ changes via $p(\mathbf{h}'|\mathbf{x})$, not $\mu(\mathbf{h}')$. However, if we allow $\mu(\mathbf{h}')$ to be a deterministic function of $\mathbf{x}$ as well, we can learn a smoother $p(y|\mathbf{x})$, even when it is desirable to learn small residual variances $\sigma_y^2$. This can be accomplished by allowing for both stochastic *and* deterministic units in a single SFNN hidden layer, allowing the mean $\mu(\mathbf{h}', \mathbf{x})$ to have contributions from two components, one from the hidden state $\mathbf{h}'$, and another one from defining a deterministic mapping from $\mathbf{x}$. As we demonstrate in our experimental results, this is crucial for learning good density models of the real-valued $Y$.

In SFNNs with only one hidden layer, $p(\mathbf{h}|\mathbf{x})$ is a factorial Bernoulli distribution. If $p(\mathbf{h}|\mathbf{x})$ has low entropy, only a few discrete $\mathbf{h}$ states out of the $2^{N_h}$ total states would have any significant probability mass. We can increase the entropy over the stochastic hidden variables by adding a second hidden layer. The second hidden layer takes the stochastic and any deterministic hidden nodes of the first layer as its input. This leads to our proposed SFNN model, shown in Fig. 4.5.

In our SFNNs, we assume a conditional diagonal Gaussian distribution for the output $Y$: $\log p(\mathbf{y}|\mathbf{h}, \mathbf{x}) \propto -\frac{1}{2}\sum_i \log \sigma_i^2 - \frac{1}{2}\sum_i (y_i - \mu(\mathbf{h}, \mathbf{x}))^2/\sigma_i^2$. We note that we can also use any other parameterized distribution (e.g. Student's t) for the output variables. This is a win compared to the Boltzmann Machine family of models, which require the output distribution to be from the exponential family.

### 4.2.2 Learning

We present a Monte Carlo variant of the Generalized EM algorithm (Neal and Hinton, 1998) for learning SFNNs. Specifically, importance sampling is used during the E-step to approximate the posterior $p(\mathbf{h}|y, \mathbf{x})$, while the Backprop algorithm is used during the M-step to calculate the derivatives of the parameters of both the stochastic and deterministic nodes. Gradient ascent using the derivatives will guarantee that the variational lower bound of the model log-likelihood will be improved. The drawback of our learning algorithm is the requirement of sampling the stochastic nodes $M$ times for every weight update. However, as we will show in the experimental results, 20 samples is sufficient for learning good SFNNs.

The requirement of sampling is typical for models capable of structured learning. As a comparison, energy based models, such as conditional Restricted Boltzmann Machines, require MCMC sampling per weight update to estimate the gradient of the log-partition function. These MCMC samples do not converge to the true distribution, resulting in a biased estimate of the gradient.

For clarity, we provide the following derivations for SFNNs with one hidden layer containing only stochastic nodes[7]. For any approximating distribution $q(\mathbf{h})$, we can write down the following variational lower-bound on the data log-likelihood:

$$\log p(y|\mathbf{x}) = \log \sum_{\mathbf{h}} p(y, \mathbf{h}|\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{h}|y, \mathbf{x}) \log \frac{p(y, \mathbf{h}|\mathbf{x})}{p(\mathbf{h}|y, \mathbf{x})} \geq \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(y, \mathbf{h}|\mathbf{x}; \theta)}{q(\mathbf{h})}, \tag{4.25}$$

where $q(\mathbf{h})$ can be any arbitrary distribution. For the tightest lower-bound, $q(\mathbf{h})$ need to be the exact posterior $p(\mathbf{h}|y, \mathbf{x})$. While the posterior $p(\mathbf{h}|y, \mathbf{x})$ is hard to compute, the "conditional prior" $p(\mathbf{h}|\mathbf{x})$ is easy (corresponds to a simple feedforward pass). We can therefore set $q(\mathbf{h}) \triangleq p(\mathbf{h}|\mathbf{x})$. However, this

[7] It is straightforward to extend the model to multiple and hybid hidden layered SFNNs.

would be a very bad approximation as learning proceeds, since the learning of the likelihood $p(y|\mathbf{h},\mathbf{x})$ will increase the KL divergence between the conditional prior and the posterior. Instead, it is critical to use importance sampling with the conditional prior as the proposal distribution.

Let $Q$ be the expected complete data log-likelihood, which is a lower bound on the log-likelihood that we wish to maximize:

$$Q(\theta,\theta_{old}) = \sum_{\mathbf{h}} \frac{p(\mathbf{h}|y,\mathbf{x};\theta_{old})}{p(\mathbf{h}|\mathbf{x};\theta_{old})} p(\mathbf{h}|\mathbf{x};\theta_{old}) \log p(y,\mathbf{h}|\mathbf{x};\theta) \simeq \frac{1}{M}\sum_{m=1}^{M} w^{(m)} \log p(y,\mathbf{h}^{(m)}|\mathbf{x};\theta), \tag{4.26}$$

where $\mathbf{h}^{(m)} \sim p(\mathbf{h}|\mathbf{x};\theta_{old})$ and $w^{(m)}$ is the importance weight of the $m$-th sample from the proposal distribution $p(\mathbf{h}|\mathbf{x};\theta_{old})$. Using Bayes Theorem, we have

$$w^{(m)} = \frac{p(\mathbf{h}^{(m)}|y,\mathbf{x};\theta_{old})}{p(\mathbf{h}^{(m)}|\mathbf{x};\theta_{old})} = \frac{p(y|\mathbf{h}^{(m)},\mathbf{x};\theta_{old})}{p(y|\mathbf{x};\theta_{old})} \simeq \frac{p(y|\mathbf{h}^{(m)};\theta_{old})}{\frac{1}{M}\sum_{m=1}^{M} p(y|\mathbf{h}^{(m)};\theta_{old})}. \tag{4.27}$$

Eq. 4.24 is used to approximate $p(y|\mathbf{x};\theta_{old})$. For convenience, we define the partial objective of the $m$-th sample as $Q^{(m)} \triangleq w^{(m)}\big(\log p(y|\mathbf{h}^{(m)};\theta) + \log p(\mathbf{h}^{(m)}|\mathbf{x};\theta)\big)$. We can then approximate our objective function $Q(\theta,\theta_{old})$ with $M$ samples from the proposal: $Q(\theta,\theta_{old}) \simeq \frac{1}{M}\sum_{m=1}^{M} Q^{(m)}(\theta,\theta_{old})$. For our generalized M-step, we seek to perform gradient ascent on $Q$:

$$\frac{\partial Q}{\partial \theta} \simeq \frac{1}{M}\sum_{m=1}^{M} \frac{\partial Q^{(m)}(\theta,\theta_{old})}{\partial \theta} = \frac{1}{M}\sum_{m=1}^{M} w^{(m)} \frac{\partial}{\partial \theta}\Big\{ \log p(y|\mathbf{h}^{(m)};\theta) + \log p(\mathbf{h}^{(m)}|\mathbf{x};\theta)\Big\}. \tag{4.28}$$

The gradient term $\frac{\partial}{\partial\theta}\{\,\cdot\,\}$ is computed using error backpropagation of two sub-terms. The first part, $\frac{\partial}{\partial\theta}\{\log p(y|\mathbf{h}^{(m)};\theta)\}$, treats $y$ as the targets and $\mathbf{h}^{(m)}$ as the input data, while the second part, $\frac{\partial}{\partial\theta}\{\log p(\mathbf{h}^{(m)}|\mathbf{x};\theta)\}$, treats $\mathbf{h}^{(m)}$ as the targets and $\mathbf{x}$ as the input data. In SFNNs with a mixture of deterministic and stochastic units, backprop will additionally propagate error information from the first part to the second part.

The full gradient is a weighted summation of the $M$ partial derivatives, where the weighting comes from how well a particular state $\mathbf{h}^{(m)}$ can generate the data $y$. This is intuitively appealing, since learning adjusts both the "preferred" states' abilities to generate the data (first part in the braces), as well as increase their probability of being picked conditioning on $\mathbf{x}$ (second part in the braces). The detailed EM learning algorithm for SFNNs is listed in Algorithm 7.

### 4.2.3 Cooperation during learning

We note that for importance sampling to work well in general, a key requirement is that the proposal distribution is not small where the true distribution has significant mass. However, things are slightly different when using importance sampling during learning. Our proposal distribution $p(\mathbf{h}|\mathbf{x})$ and the posterior $p(\mathbf{h}|y,\mathbf{x})$ are not fixed but rather governed by the model parameters. Learning adapts these distributions in a synergistic and cooperative fashion.

Let us hypothesize that at a particular learning iteration, the conditional prior $p(\mathbf{h}|\mathbf{x})$ is small in certain regions where $p(\mathbf{h}|y,\mathbf{x})$ is large, which is undesirable for importance sampling. The E-step will draw $M$ samples and weight them according to Eq. 4.27. While all samples $\mathbf{h}^{(m)}$ will have very low log-likelihood due to the bad conditional prior, there will be a certain preferred state $\hat{\mathbf{h}}$ with the largest

**Algorithm 7** EM learning algorithm

Given training $D$ dimensional data pairs: $\{\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\}$, $n = 1 \dots N$. Hidden layers $\mathbf{h}^1$&$\mathbf{h}^4$ are deterministic, $\mathbf{h}^2$&$\mathbf{h}^3$ are hybrid. $\theta = \{W^{1,2,3,4,5}, bias, \sigma_y^2\}$
**repeat**
*//Approximate E-step*:
Compute $p(\mathbf{h}^2|\mathbf{x}^{(n)}) = Bernoulli\big(\sigma(W^2\sigma(W^1\mathbf{x}^{(n)}))\big)$
$\mathbf{h}^2_{determ} \leftarrow p(\mathbf{h}^2_{determ}|\mathbf{x}^{(n)})$
**for** $m = 1$ **to** $M$ (importance samples) **do**
Sample: $\mathbf{h}^2_{stoch} \sim p(\mathbf{h}^2_{stoch}|\mathbf{x}^{(n)})$.
let $\mathbf{h}^2$ be the concatenation of $\mathbf{h}^2_{stoch}$ and $\mathbf{h}^2_{determ}$.
$p(\mathbf{h}^3|\mathbf{x}^{(n)}) = Bernoulli\big(\sigma(W^3\mathbf{h}^2)\big)$
$\mathbf{h}^3_{determ} \leftarrow p(\mathbf{h}^3_{determ}|\mathbf{x}^{(n)})$
Sample: $\mathbf{h}^3_{stoch} \sim p(\mathbf{h}^3_{stoch}|\mathbf{x}^{(n)})$
let $\mathbf{h}^3$ be the concatenation of $\mathbf{h}^3_{stoch}$ and $\mathbf{h}^3_{determ}$.
Compute $p(\mathbf{y}|\mathbf{x}^{(n)}) = \mathcal{N}\big(\sigma(W^5\sigma(W^4\mathbf{h}^3)); \sigma_y^2\big)$
**end for**
Compute $w^{(m)}$ for all $m$, using Eq. 4.24.

*//M-step*:
$\triangle\theta \leftarrow 0$
**for** $m = 1$ **to** $M$ **do**
Compute $\frac{\partial Q^{(m)}(\theta,\theta_{old})}{\partial\theta}$ by Backprop.
$\triangle\theta = \triangle\theta + \partial Q^{(m)}/\partial\theta$
**end for**
$\theta_{new} = \theta_{old} + \frac{\alpha}{M}\triangle\theta$, *//*$\alpha$ is the learning rate.
**until** convergence

weight. Learning using Eq. 4.28 will accomplish two things: (1) it will adjust the generative weights to allow preferred states to better generate the observed $y$; (2) it will make the conditional prior better by making it more likely to predict $\hat{\mathbf{h}}$ given $\mathbf{x}$. Since the generative weights are shared, the fact that $\hat{\mathbf{h}}$ generates $y$ accurately will probably reduce the likelihood of $y$ under another state $\tilde{\mathbf{h}}$. The updated conditional prior tends to be a better proposal distribution for the updated model. The cooperative interaction between the conditional prior and posterior during learning provides some robustness to the importance sampler.

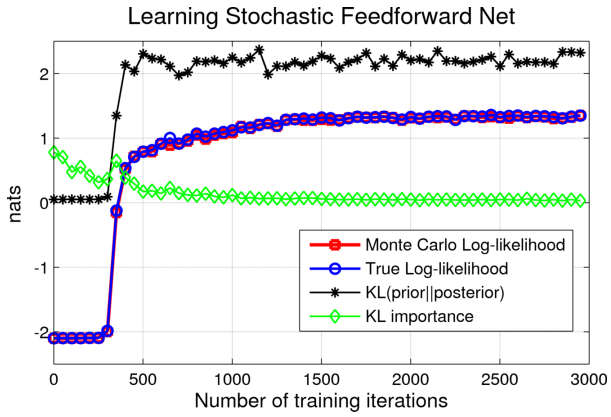
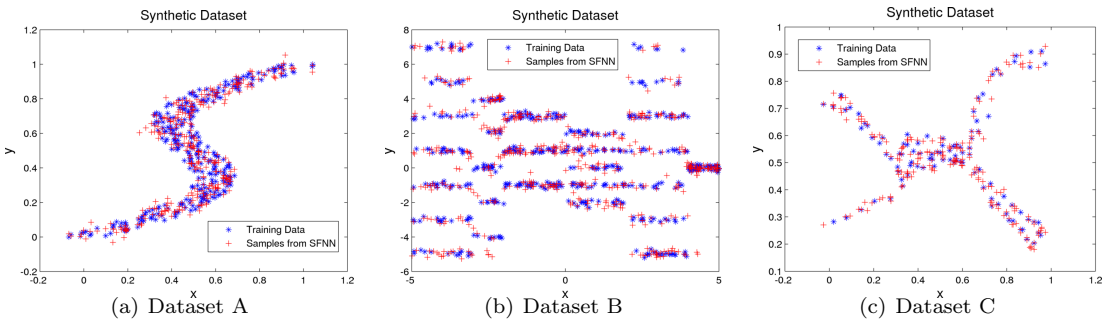



Figure 4.6: KL divergence and log-likelihoods. Best viewed in color.

Empirically, we can see this effect as learning progress on Dataset A of Sec. 4.2.5 in Fig. 4.6. The




(a) Dataset A (b) Dataset B (c) Dataset C

Figure 4.7: *Three synthetic datasets of 1-dimensional one-to-many mappings. For any given $x$, multiple modes in $y$ exist. Blue stars are the training data, red pluses are* exact *samples from SFNNs. Best viewed in color.*

plot shows the model log-likelihood given the training data as learning progresses until 3000 weight updates. 30 importance samples are used during learning with 2 hidden layers of 5 stochastic nodes. We chose 5 nodes because it is small enough that the true log-likelihood can be computed using brute-force integration. As learning progresses, the Monte Carlo approximation is very close to the true log-likelihood using only 30 samples. As expected, the KL from the posterior and prior diverges as the generative weights better model the multiple modes around $x = 0.5$. We also compared the KL divergence between our empirical weighted importance sampled distribution and true posterior, which converges toward zero. This demonstrate that the prior distribution has learned to *not* be small in regions of large posterior. In other words, this shows that the E-step in the learning of SFNNs is close to exact for this dataset and model.

### 4.2.4 Experiments

We first demonstrate the effectiveness of SFNN on synthetic one dimensional one-to-many mapping data. We then use SFNNs to model face images with varying facial expressions and emotions. SFNNs outperform other competing density models by a large margin. We also demonstrate the usefulness of latent features learned by SFNNs for expression classification. Finally, we train SFNNs on a dataset with in-depth head rotations, a database with colored objects, and a image segmentation database. By drawing samples from these trained SFNNs, we obtain qualitative results and insights into the modeling capacity of SFNNs. We provide computation times for learning in Section 4.2.8.

### 4.2.5 Synthetic datasets

As a proof of concept, we used three one dimensional one-to-many mapping datasets, shown in Fig. 4.7. Our goal is to model $p(y|x)$. Dataset A was used by Bishop (1994) to evaluate the performance of the Mixture Density Networks (MDNs). Dataset B has a large number of tight modes conditioned on any given x, which is useful for testing a model's ability to learn many modes and a small residual variance. Dataset C is used for testing whether a model can learn modes that are far apart from each other.

We randomly split the data into a training, validation, and a test set. We report the average test set log-probability averaged over 5 folds for different models in Table 4.4. The method called 'Gaussian' is a 2D Gaussian estimated on $(x, y)$ jointly, and we report $\log p(y|x)$ which can be obtained easily in closed-form. For Conditional Gaussian Restricted Boltzmann Machine (C-GRBM) we used 25-step

| | Gaussian | MDN | C-GRBM | SBN | SFNN |
|---|---|---|---|---|---|
| A | 0.078 ±0.02 | 1.05 ±0.02 | 0.57 ±0.01 | 0.79 ±0.03 | 1.04 ±0.03 |
| B | -2.40 ±0.07 | -1.58 ±0.11 | -2.14 ±0.04 | -1.33 ±0.10 | -0.98 ±0.06 |
| C | 0.37 ±0.07 | 2.03 ±0.05 | 1.36 ±0.05 | 1.74 ±0.08 | 2.21 ±0.16 |

Table 4.4: Average test log-probability density on synthetic 1D datasets.

Contrastive Divergence (Hinton, 2002) (CD-25) to estimate the gradient of the log partition function. We used Annealed Importance Sampling (Neal, 2001; Salakhutdinov and Murray, 2008) with 50,000 intermediate temperatures to estimate the partition function. SBN is a Sigmoid Belief Net with three hidden stochastic binary layers between the input and the output layer. It is trained in the same way as SFNN, but there are no deterministic units. Finally, SFNN has four hidden layers with the inner two being hybrid stochastic/deterministic layers (See Fig. 4.5). We used 30 importance samples to approximate the posterior during the E-step. All other hyper-parameters for all of the models were chosen to maximize the validation performance.

Table 4.4 reveals that SFNNs consistently outperform all other methods. Fig. 4.7 further shows samples drawn from SFNNs as red 'pluses'. Note that SFNNs can learn small residual variances to accurately model Dataset B. Comparing SBNs to SFNNs, it is clear that having deterministic hidden nodes is a big win for modeling continuous $y$.

### 4.2.6 Modeling Facial Expression

Conditioned on a subject's face with neutral expression, the distribution of all possible emotions or expressions of this particular individual is multimodal in pixel space. We learn SFNNs to model facial expressions in the Toronto Face Database (Susskind, 2011). The Toronto Face Database consists of 4000 images of 900 individuals with 7 different expressions. Of the 900 subjects, there are 124 with 10 or more images per subject, which we used as our data. We randomly selected 100 subjects with 1385 total images for training, while 24 subjects with a total of 344 images were selected as the test set.

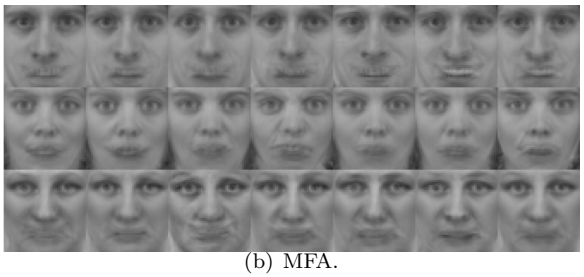
For each subject, we take the average of their face images as $\mathbf{x}$ (mean face), and learn to model this subject's varying expressions $\mathbf{y}$. Both $\mathbf{x}$ and $\mathbf{y}$ are grayscale and downsampled to a resolution of $48 \times 48$. We trained a SFNN with 4 hidden layers of size 128 on these facial expression images. The second and third "hybrid" hidden layers contained 32 stochastic binary and 96 deterministic hidden nodes, while the first and the fourth hidden layers consisted of only deterministic sigmoids. We refer to this model as SFNN2. We also tested the same model but with only one hybrid hidden layer, that we call SFNN1. We used mini-batches of size 100 and and 30 importance samples for the E-step. A total of 2500 weight updates were performed. Weights were randomly initialized with standard deviation of 0.1, and the residual variance $\sigma_y^2$ was initialized to the variance of $\mathbf{y}$.

For comparisons with other models, we trained a Mixture of Factor Analyzers (MFA) (Ghahramani and Hinton, 1996), Mixture Density Networks (MDN), and Conditional Gaussian Restricted Boltzmann Machines (C-GRBM) on this task. For the Mixture of Factor Analyzers model, we trained a mixture with 100 components, one for each training individual. Given a new test face $\mathbf{x}_{test}$, we first find the training $\hat{\mathbf{x}}$ which is closest in Euclidean distance. We then take the parameters of the $\hat{\mathbf{x}}$'s FA component, while replacing the FA's mean with $\mathbf{x}_{test}$. Mixture Density Networks are trained using code provided by the NETLAB package (Nabney, 2002). The number of Gaussian mixture components and the number of

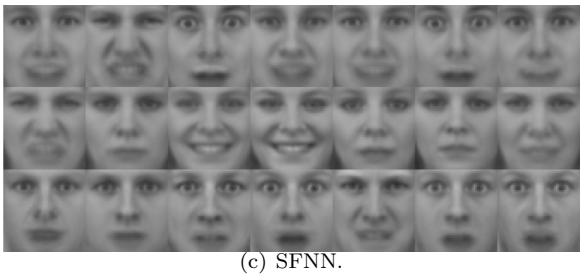
| | MFA | MDN | C-GRBM | SFNN1 | SFNN2 |
|---|---|---|---|---|---|
| Nats | 1406 ±52 | 1321 ±16 | 1146 ±113 | 1488 ±18 | 1534 ±27 |
| Time | 10 secs. | 6 mins. | 158 mins. | 112 secs. | 113 secs. |

Table 4.5: Average test log-probability and total training time on facial expression images. Note that for continuous data, these are probability densities and can be positive.

(a) Conditional Gaussian RBM.



(b) MFA.



(c) SFNN.

Figure 4.8: Samples generated from various models. SFNN samples are superior since they capture various expressions while preserving the identity of the conditioning face.

hidden nodes were selected using a validation set. Optimization is performed using the scaled conjugate gradient algorithm until convergence. For C-GRBMs, we used CD-25 for training. The optimal number of hidden units, selected via validation, was 1024. A population sparsity objective on the hidden activations was also part of the objective (Nair and Hinton, 2009a). The residual diagonal covariance matrix is also learned. Optimization used stochastic gradient descent with mini-batches of 100 samples each.

Table 4.5 displays the average log-probabilities along with standard errors of the 344 test images. We also recorded the total training time of each algorithm, although this depends on the number of weight updates and whether or not GPUs are used. For MFA and MDN, the log-probabilities were computed exactly. For SFNNs, we used Eq. 4.24 with 1000 samples. We can see that SFNNs substantially outperform all other models. Having two hybrid hidden layers (SFNN2) improves model performance over SFNN1, which has only one hybrid hidden layer.

Qualitatively, Fig. 4.8 shows samples drawn from the trained models. The leftmost column shows

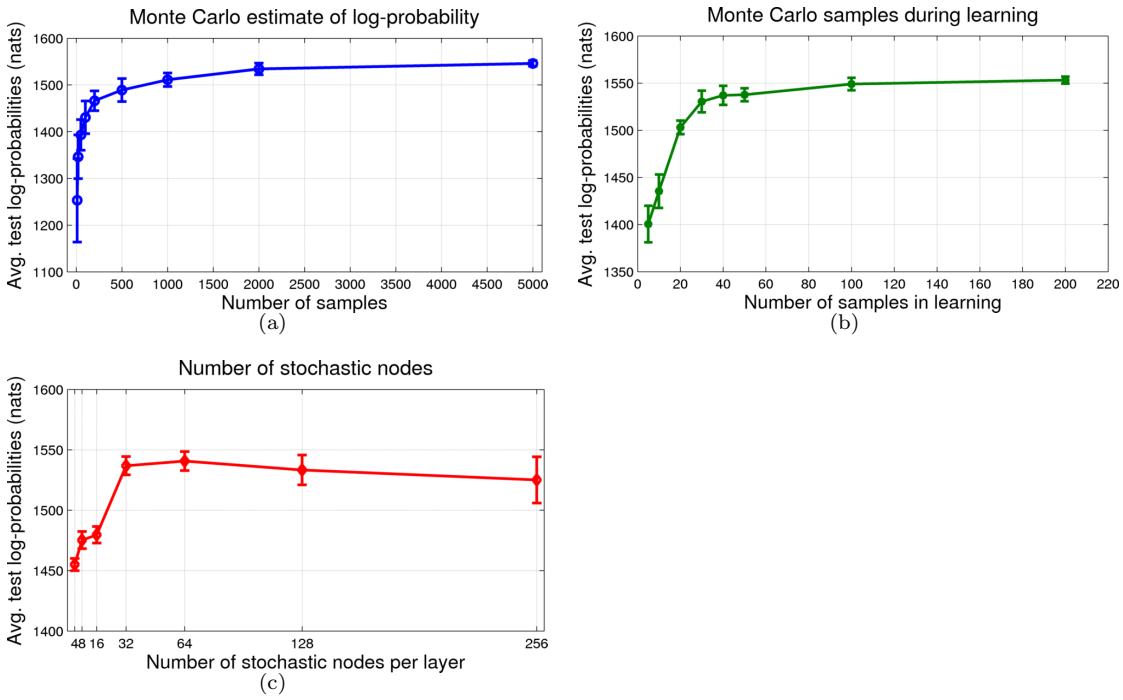


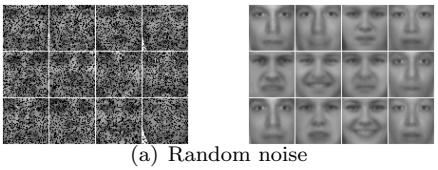

Figure 4.9: Plots demonstrate how hyperparameters affect the evaluation and learning of SFNNs.

the mean faces of 3 test subjects, followed by 7 samples from the distribution $p(\mathbf{y}|\mathbf{x})$. For C-GRBM, samples are generated from a Gibbs chain, where each successive image is taken after 1000 steps. For the other 2 models, displayed samples are *exact*. MFAs overfit on the training set, generating samples with significant artifacts. Samples produced by C-GRBMs suffer from poor mixing and get stuck at a local mode. SFNN samples show that the model was able to capture a combination of mutli-modality and preserved much of the identity of the test subjects. We also note that SFNN generated faces are not simple memorization of the training data. This is validated by its superior performance on the *test* set in Table 4.5.

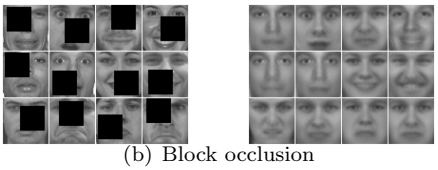
We further explored how different hyperparameters (e.g. # of stochastic layers, # of Monte Carlo samples) can affect the learning and evaluation of SFNNs. We used face images and SFNN2 for these experiments. First, we wanted to know the number of $M$ in Eq. 4.24 needed to give a reasonable estimate of the log-probabilities. Fig. 4.9(a) shows the estimates of the log-probability as a function of the number of samples. We can see that having about 500 samples is reasonable, but more samples provides a slightly better estimate. The general shape of the plot is similar for all other datasets and SFNN models. When $M$ is small, we typically underestimate the true log-probabilities. While 500 or more samples are needed for accurate model *evaluation*, only 20 or 30 samples are sufficient for learning good models (as shown in Fig. 4.9(b). This is because while $M = 20$ gives suboptimal approximation to the true posterior, learning still improves the variational lower-bound. In fact, we can see that the difference between using 30 and 200 samples during learning results in only about 20 nats of the final average test log-probability. In Fig. 4.9(c), we varied the number of binary stochastic hidden variables in the 2 inner hybrid layers. We did not observe significant improvements beyond more than 32 nodes. With more hidden nodes, over-fitting can also be a problem.

**Expression Classification**

The internal hidden representations learned by SFNNs are also useful for classification of facial expressions. For each $\{\mathbf{x}, \mathbf{y}\}$ image pair, there are 7 possible expression types: neutral, angry, happy, sad, surprised, fearful, and disgusted. As baselines, we used regularized linear softmax classifiers and multilayer perceptron classifier taking pixels as input. The mean of every pixel across all cases was set to 0 and standard deviation was set to 1.0. We then append the learned hidden features of SFNNs and C-GRBMs to the image pixels and re-train the same classifiers. The results are shown in the first row of Table 4.6. Adding hidden features from the SFNN trained in an unsupervised manner (without expression labels) improves accuracy for both linear and nonlinear classifiers.



(a) Random noise



(b) Block occlusion

Figure 4.10: **Left**: Noisy test images $\mathbf{y}$. Posterior inference in SFNN finds $E_{p(\mathbf{h}|\mathbf{x},\mathbf{y})}[\mathbf{h}]$. **Right**: generated $\mathbf{y}$ images from the expected hidden activations.

| | Linear | C-GRBM +Linear | SFNN +Linear | MLP | SFNN +MLP |
|---|---|---|---|---|---|
| clean | 80.0% | 81.4% | **82.4%** | 83.2% | 83.8 % |
| 10% noise | 78.9% | 79.7% | 80.8% | 82.0% | 81.7 % |
| 50% noise | 72.4% | **74.3%** | 71.8% | 79.1% | 78.5% |
| 75% noise | 52.6% | 58.1% | **59.8%** | 71.9% | **73.1**% |
| 10% occl. | 76.2% | 79.5% | 80.1% | 80.3% | **81.5**% |
| 50% occl. | 54.1% | 59.9% | **62.5%** | 58.5% | **63.4**% |
| 75% occl. | 28.2% | 33.9% | **37.5%** | 33.2% | **39.2**% |

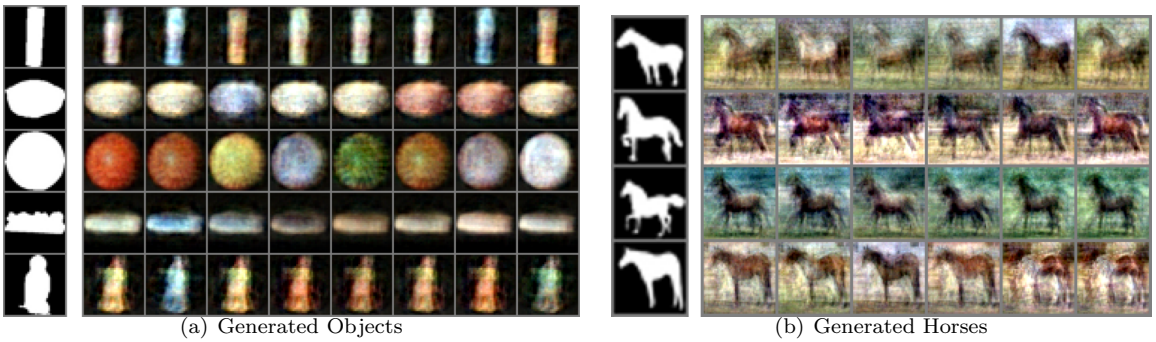
Table 4.6: Recognition accuracy over 5 folds. Bold numbers indicate that the difference in accuracy is statistically significant than the competitor models, for both linear and nonlinear classifiers.

SFNNs are also useful when dealing with noise. As a generative model of $\mathbf{y}$, it is somewhat robust to noisy and occluded pixels. For example, the left panels of Fig. 4.10, show corrupted test images $\mathbf{y}$. Using the importance sampler described in Sec. 4.2.2, we can compute the expected values of the binary stochastic hidden variables given the corrupted test $\mathbf{y}$ images[8]. In the right panels of Fig. 4.10, we show the corresponding generated $\mathbf{y}$ from the inferred average hidden states. After this denoising process, we can then feed the denoised $\mathbf{y}$ and $E[\mathbf{h}]$ to the classifiers. This compares favorably to simply filling in the missing pixels with the average of that pixel from the training set. Classification accuracies under noise are also presented in Table 4.6. For example 10% noise means that 10 percent of the pixels of *both* $\mathbf{x}$ and $\mathbf{y}$ are corrupted, selected randomly. 50% occlusion means that a square block with 50% of the original area is randomly positioned in both $\mathbf{x}$ and $\mathbf{y}$. Gains in recognition performance from using SFNN are particularly pronounced when dealing with large amounts of random noise and occlusions.

### 4.2.7 Additional Qualitative Experiments

Not only are SFNNs capable of modeling facial expressions of aligned face images, they can also model complex real-valued conditional distributions. Here, we present some qualitative samples drawn from

[8]For this task we assume that we have knowledge of which pixels is corrupted.



(a) Generated Objects (b) Generated Horses

Figure 4.11: Samples generated from a SFNN after training on object and horse databases. Conditioned on a given foreground mask, the appearance is multimodal (different color and texture). Best viewed in color.

SFNNs trained on more complicated distributions. We trained SFNNs to generate colorful images of common objects from the Amsterdam Library of Objects database (Geusebroek et al., 2005), conditioned on the foreground masks. This is a database of 1000 everyday objects under various lighting, rotations, and viewpoints. Every object also comes with a foreground segmentation mask. For every object, we selected the image under frontal lighting without any rotations, and trained a SFNN conditioned on the foreground mask. Our goal is to model the appearance (color and texture) of these objects. Of the 1000 objects, there are many objects with similar foreground masks (e.g. round or rectangular). Conditioned on the test foreground masks, Fig. 4.11(a) shows random samples from the learned SFNN model. We also tested on the Weizmann segmentation database (Borenstein and Ullman, 2002) of horses, learning a conditional distribution of horse appearances conditioned on the segmentation mask. The results are shown in Fig. 4.11(b).

Figure 4.12: Samples from SFNN trained on rotated faces.

We also tested on the UMIST faces database (Graham and Allinson, 1998), which contains in-depth 3D rotation of heads. We trained the same SFNNs as for facial expression on the 16 training subjects, using 4 subjects for testing. Conditioned on the profile view, we are modeling the distribution of rotations up to 90 degrees. Fig. 4.12 displays 3 test subjects' profile view along with seven exact samples drawn from the model (plotted on the right hand side). This is a particularly difficult task, as the model must learn to generate face parts, such as eyes and nose.

### 4.2.8 Computation Time

Despite having to draw $M$ samples during learning, Fig. 4.9 empirically demonstrated that 20 samples is often sufficient[9]. This is in part due to the fact that samples from the conditional prior are exact and in part due to the cooperation that occurs during learning. Regarding hardware, our experiments are performed on nVidia GTX580 GPUs. This gives us over 10x speedup over CPUs. For example, a 4 hidden layer SFNN with 2304 input and output dimensions, 128 stochastic hidden nodes, and 50 samples per E-step, can update its parameters in 0.15 secs on a minibatch of 100 cases.

In Table 4.5, C-GRBM is also trained on the GPU, but is much slower due to its use of a large hidden layer and 25 CD steps. For example, the C-GRBM requires 1.16 secs per parameter update. MFA and MDNs are run on CPUs and we can also expect 10x speedup from moving to GPUs.

## 4.3 Discussion

In this chapter we discussed two novel generative models for improving density learning. The first model is based on the Mixture of Factor Analyzers, where we have shown that we get a consistent and substantial performance gain by training a second layer MFA on the aggregated posterior of the first layer MFAs. We have also shown that as density models, MFAs significantly outperform undirected RBM models for real-valued data. Higher input dimensionality leads to bigger gains from learning DMFAs. However, adding a third MFA layer appears to be of little value. A possible extension of our work is to train a mixture of linear dynamical systems and then to train a higher-level mixture of linear dynamical systems to model the aggregated posterior of each component of the first level mixture.

The second model is a feedforward neural network with a hybrid of stochastic and deterministic hidden nodes. We have proposed an efficient learning algorithm that allows us to learn rich multi-modal conditional distributions, supported by quantitative and qualitative empirical results. The major drawbacks of SFNNs are that inference is not trivial and $M$ samples are needed for the importance sampler. While this is sufficiently fast for our experiments we can potentially accelerate inference by learning a separate recognition network to perform inference in one feedforward pass. These techniques have previously been used by Hinton et al. (1995); Salakhutdinov and Larochelle (2010) with success.

[9] We note that this is still $M$ times more expensive than standard backprop.

# Chapter 5

# A higher-order generative model: The Tensor Analyzer

Chapter 3 proposed two new models by adding meaningful latent variables which were domain-specific to vision. However, it is more desirable to have a flexible higher-order latent representation which can *learn* meaningful representation without any domain knowledge (such as the way light interacts with surface normals of objects).

In this chapter we introduce the Tensor Analyzer, which is a multilinear generalization of Factor Analyzers. Factor Analysis is a statistical method that seeks to explain linear variations in data by using unobserved latent variables. Due to its *additive* nature, it is not suitable for modeling data that is generated by multiple groups of latent factors which interact *multiplicatively*. We describe an efficient way of sampling from the posterior distribution over factor values and we demonstrate that these samples can be used in the EM algorithm for learning interesting mixture models of natural image patches. Tensor Analyzers can also accurately recognize a face under significant pose and illumination variations when given only one previous image of that face. We also show that Tensor Analyzers can be trained in unsupervised, semi-supervised, or fully supervised settings.

Exploratory Factor Analysis is widely used in statistics to identify underlying linear factors. Mixtures of Factor Analyzers have been used successfully for unsupervised learning (Yang et al., 1999; Verbeek, 2006). Factor Analyzers (FAs) model each observation vector as a weighted linear combination of the unobserved factor values plus additive uncorrelated noise. For many types of data, this additive generative process is less suitable than a generative process that also contains multiplicative interactions between latent factors.

An example of multiplicative interactions is the set of face images under varying illuminations. It is known that these images of a particular person approximately lie on a 3 dimensional linear subspace, making a FA with 3 factors a good model (Belhumeur and Kriegman, 1996). The linear subspace (and thus the factor loadings) will be *person-specific* due to the way facial structures interact with the light to create a face image. As a result of this person-specific property, the factor loadings need to be a *function* of the person identity when modeling face images of *multiple* people. Instead of modeling 100 individuals with 100 separate FAs, it is desirable to use person-identity variables to linearly combine a "basis" (a set of) factor loadings to compactly model all 100 faces, drastically reducing the number of parameters. This factorial representation, as shown in Fig. 5.1, naturally allows for generalization

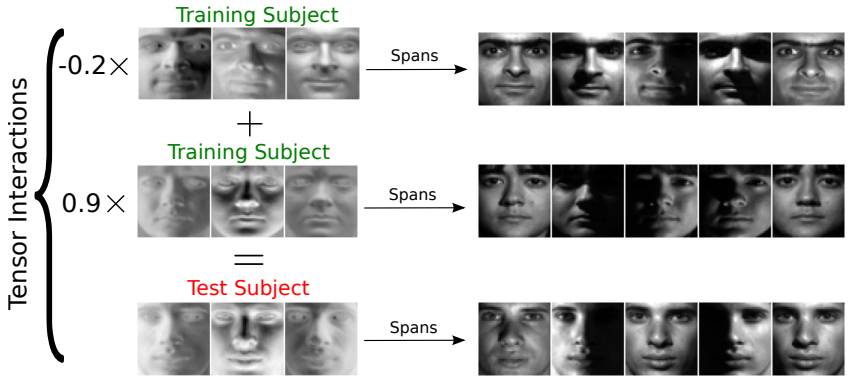



Figure 5.1: Illustration of why TA is needed. Three dimensional bases (left) span the pixel space of specific individuals. In order to properly generalize to a novel test subject, tensor interactions of the learned training bases are needed to form a new basis for modeling a test subject.

to new people required for one-shot face recognition. Fig. 5.1 provides an illustration and Sec. 5.6.3 provides experimental validations.

To this end, we introduce Tensor Analyzers (TAs), which generalize FAs to the multilinear setting by introducing a factor loading tensor and multiple groups of latent factors. Utilizing the loading tensor, a group can change how another group's factors interact with the observed variables. This allows latent factor groups in a TA to learn highly interpretable representations. In the faces example, one group could represent lighting direction while the other could represent the identity.

In the special case of a TA with only one group of factors, its loading tensor reduces to a loading matrix, and the model is exactly the same as an ordinary FA. In a TA, when conditioned on all but one group of factors, the model effectively becomes a FA where the factor loadings are a function of the factor values in the groups we are conditioning on. The posterior distribution of the factor values in a FA can be computed analytically, so by cycling through each group of factors, efficient alternating Gibbs sampling is therefore possible in the TA. A TA is a proper density model so the extension to a mixture of TAs (MTA) is straightforward. When performing inference or learning in a TA, it is easy to make use of a supervisory signal that specifies the fact that 2 or more different observations are generated from the same factor values in some of the factor groups. This allows for TAs to seamlessly transition from an unsupervised density model to a semi and fully supervised model.

## 5.1 Related Works

Bilinear models with priors on the latent variables have been previously studied in the machine learning, statistics, and computer vision literatures. In Grimes and Rao (2005), sparsity is induced on the codes of a bilinear model to learn translational invariant representation from video. However, their model does not try to maximize $\log p(\mathbf{x})$, but instead finds the MAP estimate of the code activations, à la sparse coding. Culpepper et al. (2011) describe an outer-product factorization of the bilinear model that is trained as a density model using EM, but expensive Hamiltonian dynamics are required for sampling from the posterior. In addition, their model only admits an approximate M-step and does not make it easy to incorporate label information. Wang et al. (2007) proposed a multifactor GPLVM extension for modeling human motion (henceforth referred to as GPSC). In their model, the parameters are integrated

out, and factors are kernalized. Optimization is needed to find the latent coordinates. Computationally, as in GPLVM, GPSC scales cubicly in the size of the training data.

While TAs and the above models take as input *i.i.d.* data vectors, there exists a plethora of tensor decomposition (TD) methods when the data comes in the form of **N-way tensors** (Tucker, 1963; Carroll and Chang, 1970; Lathauwer and Vandewalle, 2004; Wang and Ahuja, 2003; Sun et al., 2006). The SVD algorithm was used to learn a bilinear model to separate style and content (Tenenbaum and Freeman, 2000), which we will refer to as the S&C model. Tucker decomposition was applied to a 5-mode array of face images in Vasilescu and Terzopoulos (2002), finding multilinear bases called TensorFaces. Shashua and Hazan (2005) enforced non-negative constraints to PARAFAC decomposition. Chu and Ghahramani (2009) introduced a Bayesian probabilistic version of Tucker decomposition, while Xu et al. (2012) provided a nonparametric Bayesian extension. The main disadvantages of the tensor decomposition methods are that data must be arranged in a tensor and that inference given a *single* new test case is ill-posed and can be ad hoc[1].

In contrast, TAs do not have any of the above deficiencies. Our main contribution is in the introduction of standard Gaussian priors on each latent group, which allows us to utilize the efficient inference procedure of Factor Analysis as part of TA's inference procedure. We also provide the EM algorithm that allows TAs to learn directly from data vectors in an entirely unsupervised manner. It can also make use of supervision in the form of equality constraints that specify that one group of factors should have the same vector of values for a subset of the training cases (Sec. 5.5.3). As an extension to FA, TA inherits an efficient inference algorithm that is used in each step of alternating Gibbs sampling and a closed-form M-step during learning. Unlike bilinear models, it can handle multilinear cases with 3 or more groups of latent factors. It can also be easily extended to a mixture model, provided we are willing to compute approximate densities, as described in (Sec. 5.5.1). In addition, posterior inference for a *single* test case is simple and accurate, as demonstrated by our one-shot face recognition experiments of Sec. 5.6.3.

## 5.2 Preliminaries

Following Kolda and Bader (2009), we refer to the number of dimensions of the tensor as its *order* (also known as modes). We will use bold lowercase letters to denote vectors (tensors of order one), e.g. $\mathbf{x}$; bold uppercase letters for matrices (tensors of order two), e.g. $\mathbf{W}$. We use the notation $\mathbf{w}_{(i,:)}$ to denote the $i$-th row of matrix $\mathbf{W}$. Higher order tensors are denoted by Euler script letters, e.g. a third-order tensor with dimensions of $I$, $J$, and $K$: $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$.

**Fibers**: Fibers are higher-order generalization of row/column vectors. Elements of a tensor fiber are found by fixing all but one index. Specifically, $\boldsymbol{t}_{(:,j,k)}$ is the mode-1 fiber of the tensor $\mathcal{T}$. Row and column vectors are the mode-2 and mode-1 fiber of a 2nd-order tensor, respectively.

**Matricization**: Matricization is the process of "flattening" a tensor into a matrix, by reordering the elements of the tensor. It is denoted by $\mathbf{T}_{(n)}$, where the mode-$n$ fibers of $\mathcal{T}$ are placed in the columns of the resulting matrix $\mathbf{T}_{(n)}$. For example, given $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$, $\mathbf{T}_{(1)} \in \mathbb{R}^{I \times JK}$.

**n-mode vector product**: By multiplying a vector $\mathbf{y} \in \mathbb{R}^{D_n}$ with a tensor $\mathcal{T} \in \mathbb{R}^{D_1 \times D_2 \times \cdots \times D_N}$ along the mode-$n$, the *n-mode (vector) product* is denoted by $\mathcal{T} \bar{\times}_n \mathbf{y}$. The resulting tensor is of size $D_1 \times$

[1] E.g., the asymmetric model in (S&C) requires EM learning of a separate model during test time. Probabilistic TD methods require new test cases to come with labels.

$\cdots \times D_{n-1} \times D_{n+1} \times \cdots \times D_N$.

### 5.2.1 Factor Analyzers

Let $\mathbf{x} \in \mathbb{R}^D$ denote the $D$-dimensional data, let $\{\mathbf{z} \in \mathbb{R}^d : d \leq D\}$ denote $d$-dimensional latent factors. FA is defined by a prior and likelihood:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), \quad p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{\Lambda}\mathbf{z} + \boldsymbol{\mu}, \mathbf{\Psi}), \tag{5.1}$$

where $\mathbf{I}$ is the $d \times d$ identity matrix; $\mathbf{\Lambda} \in \mathbb{R}^{D \times d}$ is the factor loading matrix, $\boldsymbol{\mu}$ is the mean. A diagonal $\mathbf{\Psi} \in \mathbb{R}^{D \times D}$ represents the variance of the observation noise. By integrating out the latent variable $\mathbf{z}$, a FA model becomes a Gaussian with constrained covariance:

$$p(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z} = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \Gamma), \tag{5.2}$$

where $\Gamma = \mathbf{\Lambda}\mathbf{\Lambda}^\mathsf{T} + \mathbf{\Psi}$. For inference, we are interested in the posterior, which is also a multivariate Gaussian:

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mathbf{m}, \mathbf{V}^{-1}), \tag{5.3}$$

where $\mathbf{V} = \mathbf{I} + \mathbf{\Lambda}^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{\Lambda}$, and $\mathbf{m} = \mathbf{V}^{-1}\mathbf{\Lambda}^\mathsf{T}\mathbf{\Psi}^{-1}(\mathbf{x} - \boldsymbol{\mu})$. Maximum likelihood estimation of the parameters is straightforward using the EM algorithm (Rubin and Thayer, 1982). During the E-step, Eq. 5.3 is used to compute the posterior sufficient statistics. During the M-step, the expected complete-data log-likelihood $\mathbb{E}_{p(\mathbf{z}|\mathbf{x};\theta_{old})}[\log p(\mathbf{x}, \mathbf{z}; \theta)]$ is maximized with respect to the model parameters $\theta = \{\mathbf{\Lambda}, \boldsymbol{\mu}, \mathbf{\Psi}\}$.

## 5.3 Tensor Analyzers

TA replaces FA's factor loading $\mathbf{\Lambda} \in \mathbb{R}^{D \times d}$ with a factor "loading" tensor $\mathcal{T} \in \mathbb{R}^{D \times d_1 \times \cdots \times d_J}$. In addition, a TA has $J$ groups of factors: $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_J\}$: $j = 1, \ldots, J$, $\mathbf{z}_j \in \mathbb{R}^{d_j}$. The key property of TAs is that the interactions between a particular factor group and the data is modified by the factor values in other groups. Given $\{\mathbf{z}_2, \ldots, \mathbf{z}_J\}$, $\mathbf{\Lambda}_{new}$, which is the factor loading matrix between $\mathbf{z}_1$ and $\mathbf{x}$, is given by:

$$\mathbf{\Lambda}_{new} = (((\mathcal{T} \;\bar{\times}_2\; \mathbf{z}_2) \;\bar{\times}_3\; \mathbf{z}_3) \bar{\times} \cdots) \;\bar{\times}_J\; \mathbf{z}_J).$$

We will use the notation TA$\{D, d_1, d_2, \ldots, d_J\}$ to denote the aforementioned TA. By using a $(J+1)$-order tensor $\mathcal{T}$, a TA can model multiplicative interactions among its latent factors $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_J\}$. In contrast, FAs do not model multiplicative interactions involving terms such as $z_i z_j : \; i \neq j$.

Each group of factors has a standard Normal prior:

$$p(\mathbf{z}_j) = \mathcal{N}(\mathbf{z}_j|0, \mathbf{I}), \quad j = 1, 2, \ldots, J. \tag{5.4}$$

For clarity of presentation, we assume $J = 3$ for the following equations. The likelihood $p(\mathbf{x}|\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3)$ is:

$$\mathcal{N}(\mathbf{x}|\mathbf{m} + \sum_j^3 \mathbf{W}_j \mathbf{z}_j + \mathbf{T}_{(1)}(\mathbf{z}_3 \otimes \mathbf{z}_2 \otimes \mathbf{z}_1), \mathbf{\Psi}), \tag{5.5}$$

where $\mathbf{x} \in \mathbb{R}^D$, $\mathbf{m}$, and $\mathbf{\Psi}$ are same as in FA. $\mathbf{W}_j \in \mathbb{R}^{D \times d_j}$ are the "biases" factor loadings, $\mathbf{T}_{(1)} \in$

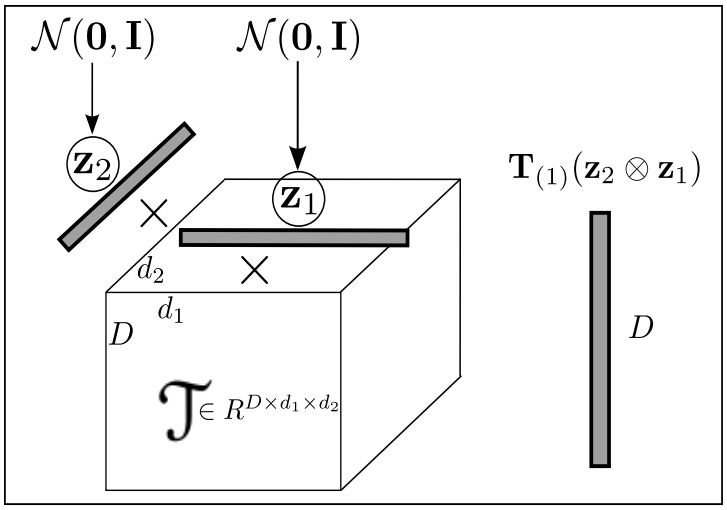



Figure 5.2: Diagram of TA's ($J = 2$) generative process. $\mathbf{\Lambda}_{new} = \mathcal{T} \bar{\times}_2 \mathbf{z}_2$ gives a new factor loading matrix for $\mathbf{z}_1$. $\mathbf{\Lambda}_{new}\mathbf{z}_1 \triangleq \mathbf{T}_{(1)}(\mathbf{z}_2 \otimes \mathbf{z}_1)$ determines the mean of $p(\mathbf{x}|\mathbf{z}_1, \mathbf{z}_2)$.

$\mathbb{R}^{D \times (d_1 d_2 d_3)}$ is the matricization of the tensor $\mathcal{T}$, and "$\otimes$" is the Kronecker product operator. Multiplicative interactions are due to the term: $\mathbf{z}_3 \otimes \mathbf{z}_2 \otimes \mathbf{z}_1$, which is a vector with dimensionality of $d_1 d_2 d_3$. We note that

$$\mathbf{T}_{(1)}(\mathbf{z}_3 \otimes \mathbf{z}_2 \otimes \mathbf{z}_1) = \sum_{i,j,k} \mathbf{t}_{(:,i,j,k)} \mathbf{z}_1(i) \mathbf{z}_2(j) \mathbf{z}_3(k),$$

where $\mathbf{z}_1(i)$ is the $i$-th element of vector $\mathbf{z}_1$, and $\mathbf{t}$ is the mode-1 fiber of $\mathcal{T}$. $\mathbf{T}_{(1)}(\mathbf{z}_3 \otimes \mathbf{z}_2 \otimes \mathbf{z}_1)$ is also equivalent to $\mathbf{\Lambda}_{new}\mathbf{z}_1$.

For clarity, we can concatenate the factors and loading matrices: let $\mathbf{y} \in \mathbb{R}^{d_1+d_2+d_3+1} \triangleq [\mathbf{z}_1; \mathbf{z}_2; \mathbf{z}_3; 1]$; $\mathbf{W} \in \mathbb{R}^{D \times (d_1+d_2+d_3+1)} \triangleq [\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{m}]$; and $\mathbf{u} \in \mathbb{R}^{d_1 d_2 d_3} = \mathbf{z}_3 \otimes \mathbf{z}_2 \otimes \mathbf{z}_1$. The joint log-likelihood of the TA is:

$$\begin{aligned} \log\, p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) = \sum_{j=1}^{3} \Big( -\frac{d_j}{2} \log(2\pi) - \frac{1}{2} \mathbf{z}_j^\mathsf{T} \mathbf{z}_j \Big) \\ - \frac{D}{2} \log(2\pi) - \frac{1}{2} \log|\mathbf{\Psi}| - \frac{1}{2} (\mathbf{x} - \mathbf{e})^\mathsf{T} \mathbf{\Psi}^{-1} (\mathbf{x} - \mathbf{e}), \end{aligned}$$

where $\mathbf{e} = \mathbf{W}\mathbf{y} + \mathbf{T}_{(1)}\mathbf{u}$. The last term of the above equation indicates that the TA models contain higher-order interactions (outer products of all factors). In comparison, FAs have only 2nd-order interactions among their latent factors. Fig. 5.2 displays a visual diagram of the TA's generative process.

Conditioned on any two of the three groups of factors, e.g. $\mathbf{z}_2$ and $\mathbf{z}_3$, the log-likelihood of $\mathbf{x}$ and $\mathbf{z}_1$ becomes:

$$\begin{aligned} \log\ p(\mathbf{x}, \mathbf{z}_1 | \mathbf{z}_2, \mathbf{z}_3) = -\frac{1}{2} \big( (d_1 + D) \log(2\pi) + \mathbf{z}_1^\mathsf{T} \mathbf{z}_1 \\ + \log|\mathbf{\Psi}| + (\mathbf{x} - \mathbf{e})^\mathsf{T} \mathbf{\Psi}^{-1} (\mathbf{x} - \mathbf{e}) \big). \end{aligned} \tag{5.6}$$

Here, $\mathbf{e}$ can be re-written as $(\mathbf{m} + \mathbf{W}_2\mathbf{z}_2 + \mathbf{W}_3\mathbf{z}_3) + (\mathbf{W}_1 + \mathcal{T} \bar{\times}_3 \mathbf{z}_3 \bar{\times}_2 \mathbf{z}_2)\mathbf{z}_1$. We can see that conditioned

on $\mathbf{z}_2$ and $\mathbf{z}_3$, we have a FA with parameters (c.f. Eq. 5.1):

$$\begin{aligned}\boldsymbol{\mu} &= \mathbf{m} + \mathbf{W}_2\mathbf{z}_2 + \mathbf{W}_3\mathbf{z}_3, \\ \boldsymbol{\Lambda} &= \mathbf{W}_1 + \mathcal{T} \;\bar{\times}_3\; \mathbf{z}_3 \;\bar{\times}_2\; \mathbf{z}_2.\end{aligned} \tag{5.7}$$

The marginal probability density function is a Gaussian: $p(\mathbf{x}|\mathbf{z}_2, \mathbf{z}_3) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}\boldsymbol{\Lambda}^\mathsf{T} + \boldsymbol{\Psi})$.

## 5.4 Inference

Higher order interaction in the TA means that inference is more complicated, since the joint posterior $p(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_3|\mathbf{x})$ has no closed-form solution. We resort to alternating Gibbs sampling by cycling through $p(\mathbf{z}_1|\mathbf{x}, \mathbf{z}_2, \mathbf{z}_3)$; $p(\mathbf{z}_2|\mathbf{x}, \mathbf{z}_1, \mathbf{z}_3)$; $p(\mathbf{z}_3|\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2)$.

Conditioned on two groups of factors, the posterior of the third is simple as the model reduces to a FA:

$$p(\mathbf{z}_1|\mathbf{x}, \mathbf{z}_2, \mathbf{z}_3) = \mathcal{N}(\mathbf{z}_1|\mathbf{V}^{-1}\boldsymbol{\Lambda}^\mathsf{T}\boldsymbol{\Psi}^{-1}(\mathbf{x} - \boldsymbol{\mu}), \mathbf{V}^{-1}), \tag{5.8}$$

where $\mathbf{V} = \mathbf{I} + \boldsymbol{\Lambda}^\mathsf{T}\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}$. $\boldsymbol{\mu}$, $\boldsymbol{\Lambda}$ are defined by Eq. 5.7.

Although inference involves a matrix inverse, it only has cost of $O(d^3)$, where $d << D$, is the dimension of a latent factor group. $d$ can be small since the data is assumed to be explained by a low dimensional manifold. We provide detailed timing evaluations in Sec. 5.6.5. It is important to note that the ability to perform alternating Gibbs sampling is critical to the efficiently performing inference. If we had to resort to Gibbs sampling of each latent factor variables separately or use Hybrid Monte Carlo, convergence would be too slow for learning on large data (Sec. 5.6.2).

## 5.5 Learning

Maximum likelihood learning of a TA is similar to FA and is straightforward using a stochastic variant of the EM algorithm (Levine and Casella, 2001). During the E-step, MCMC samples are drawn from the posterior distribution using alternating Gibbs sampling. In the M-step, the samples are used to approximate the sufficient statistics involving $\mathbf{u}$ and $\mathbf{y}$, followed by closed-form updates of the model parameters, $\theta = \{\mathbf{W}, \mathbf{T}_{(1)}, \boldsymbol{\Psi}\}$.

The objective function $Q$ of the EM algorithm is the expected complete log-likelihood, taken over

the posterior distribution of the latent factors, and summed over $N$ training cases:

$$Q = E\Big[\log \prod_i^N (2\pi)^{-D/2} |\mathbf{\Psi}|^{-1/2} \exp\{-\frac{1}{2}(\mathbf{x}_i - \mathbf{e}_i)^\mathsf{T} \mathbf{\Psi}^{-1} (\mathbf{x}_i - \mathbf{e}_i)\}\Big] \tag{5.9}$$

$$= const - \frac{N}{2}\log|\mathbf{\Psi}| - \sum_i^N E\Big[\frac{1}{2}\mathbf{x}_i^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{x}_i - \mathbf{x}_i^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{e}_i + \frac{1}{2}\mathbf{e}_i^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{e}_i\Big] \tag{5.10}$$

$$= const - \frac{N}{2}\log|\mathbf{\Psi}| - \sum_i^N E\Big[\frac{1}{2}\mathbf{x}_i^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{x}_i - \mathbf{x}_i^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{y}_i - \mathbf{x}_i^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{T}_{(1)}\mathbf{u}_i + \mathbf{y}_i^\mathsf{T}\mathbf{W}^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{T}_{(1)}\mathbf{u}_i + \frac{1}{2}\mathbf{y}_i^\mathsf{T}\mathbf{W}^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{y}_i + \frac{1}{2}\mathbf{u}_i^\mathsf{T}\mathbf{T}_{(1)}^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{T}_{(1)}\mathbf{u}_i\Big] \tag{5.11}$$

$$= const - \frac{N}{2}\log|\mathbf{\Psi}| - \sum_i^N \Big(\frac{1}{2}\mathbf{x}_i^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{x}_i - \mathbf{x}_i^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{W}E[\mathbf{y}_i] - \mathbf{x}_i^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{T}_{(1)}E[\mathbf{u}_i] + E[\mathbf{u}_i^\mathsf{T}\mathbf{T}_{(1)}^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{y}_i] + \frac{1}{2}E[\mathbf{y}_i^\mathsf{T}\mathbf{W}^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{y}_i] + \frac{1}{2}E[\mathbf{u}_i^\mathsf{T}\mathbf{T}_{(1)}^\mathsf{T}\mathbf{\Psi}^{-1}\mathbf{T}_{(1)}\mathbf{u}_i]\Big) \tag{5.12}$$

The closed-form M-step update equations and the solutions for setting $\frac{\partial Q}{\partial \theta} = 0$ are given below:

$$\frac{\partial Q}{\partial \mathbf{W}} = -\sum_i^N \Big( - \mathbf{\Psi}^{-1}\mathbf{x}_i E[\mathbf{y}_i]^\mathsf{T} + \mathbf{\Psi}^{-1}\mathbf{T}_{(1)}E[\mathbf{u}_i\mathbf{y}_i^\mathsf{T}] + \mathbf{\Psi}^{-1}\mathbf{W}E[\mathbf{y}_i\mathbf{y}_i^\mathsf{T}]\Big) = 0 \tag{5.13}$$

$$\mathbf{W} = \Big(\sum_i^N \mathbf{x}_i E[\mathbf{y}_i^\mathsf{T}] - \mathbf{T}_{(1)}\sum_i^N E[\mathbf{u}_i\mathbf{y}_i^\mathsf{T}]\Big)\Big(\sum_i^N E[\mathbf{y}_i\mathbf{y}_i^\mathsf{T}]\Big)^{-1} \tag{5.14}$$

$$\frac{\partial Q}{\partial \mathbf{T}_{(1)}} = -\sum_i^N \Big( - \mathbf{\Psi}^{-1}\mathbf{x}_i E[\mathbf{u}_i^\mathsf{T}] + \mathbf{\Psi}^{-1}\mathbf{W}E[\mathbf{y}_i\mathbf{u}_i^\mathsf{T}] + \frac{1}{2}\mathbf{\Psi}^{-1}\mathbf{T}_{(1)}(2)E[\mathbf{u}_i\mathbf{u}_i^\mathsf{T}]\Big) = 0 \tag{5.15}$$

$$\mathbf{T}_{(1)} = \Big(\sum_i^N \mathbf{x}_i E[\mathbf{u}_i^\mathsf{T}] - \mathbf{W}\sum_i^N E[\mathbf{y}_i\mathbf{u}_i^\mathsf{T}]\Big)\Big(\sum_i^N E[\mathbf{u}_i\mathbf{u}_i^\mathsf{T}]\Big)^{-1} \tag{5.16}$$

$$\frac{\partial Q}{\partial \mathbf{\Psi}^{-1}} = \frac{N}{2}\mathbf{\Psi} - \sum_i^N \Big(\frac{1}{2}\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - \mathbf{W}E[\mathbf{y}_i]\mathbf{x}_i^\mathsf{T} - \mathbf{T}_{(1)}E[\mathbf{u}_i]\mathbf{x}_i^\mathsf{T} + \mathbf{T}_{(1)}E[\mathbf{u}_i\mathbf{y}_i^\mathsf{T}]\mathbf{W}^\mathsf{T} + \frac{1}{2}\mathbf{W}E[\mathbf{y}_i\mathbf{y}_i^\mathsf{T}]\mathbf{W}^\mathsf{T} + \frac{1}{2}\mathbf{T}_{(1)}E[\mathbf{u}\mathbf{u}^\mathsf{T}]\mathbf{T}_{(1)}^\mathsf{T}\Big) = 0 \tag{5.17}$$

$$\mathbf{\Psi} = \frac{1}{N} diag\Big\{ \sum_i^N \Big(\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - 2\mathbf{T}_{(1)}\big(E[\mathbf{u}_i]\mathbf{x}_i^\mathsf{T} - E[\mathbf{u}_i\mathbf{y}_i^\mathsf{T}]\mathbf{W}^\mathsf{T} - \frac{1}{2}E[\mathbf{u}\mathbf{u}^\mathsf{T}]\mathbf{T}_{(1)}^\mathsf{T}\big) - 2\mathbf{W}\big(E[\mathbf{y}_i]\mathbf{x}_i^\mathsf{T} - \frac{1}{2}E[\mathbf{y}_i\mathbf{y}_i^\mathsf{T}]\mathbf{W}^\mathsf{T}\big)\Big)\Big\} \tag{5.18}$$

During learning, we have found that the Gibbs sampler mixes very fast and that a relatively small number of steps (20 to 100) are needed to achieve good performance. It is important to stress that Gibbs sampling is efficient in TAs because the posterior for each group is exact when conditioned on all other groups. It also mixes quickly because even though the variables of the posterior are dependent, the posterior itself is likely to be unimodal: an image of the face is explained by 1 lighting code and

**Algorithm 8** EM Learning for TA

1: Given training data with $N$ samples: $\mathbf{X} \in \mathbb{R}^{D \times N}$
2: Initialize $\theta$: $\{\mathbf{W}, \mathbf{T}_{(1)}\} \sim \mathcal{N}(0, .01^2)$, $\mathbf{\Psi} \leftarrow 10 * \text{std}(\mathbf{X})$.
**repeat**
*//Approximate E-step*:
**for** $n = 1$ **to** $N$ **do**
3: Sample $\{\mathbf{z}_1^{(n)}, \mathbf{z}_2^{(n)}, \mathbf{z}_3^{(n)}\}$ from $p(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 | \mathbf{x}^{(n)})$ using Eq. 5.8, and alternating between $\mathbf{z}_1, \mathbf{z}_2, \& \ \mathbf{z}_3$.
**end for**

*//M-step*:
4: Concatenate samples $\{\mathbf{z}_1^{(n)}, \mathbf{z}_2^{(n)}, \mathbf{z}_3^{(n)}\}$ into $\{\mathbf{y}^n, \mathbf{u}^n\}$
5: Approximate posterior expectations using samples: $E[\mathbf{y}_n] \simeq \mathbf{y}^n$, $E[\mathbf{u}_n \mathbf{y}_n{}^\mathsf{T}] \simeq \mathbf{u}^n \mathbf{y}^{n\mathsf{T}}$, etc.
6: Update $\{\mathbf{W}, \mathbf{T}_{(1)}, \mathbf{\Psi}\}$ using Eqs. 5.14, 5.16, and 5.18.
**until** convergence

1 subject code (See Sec. 5.6.3). We show trace plots of the latent variables to demonstrate the fast convergence property.

Posterior inference in a TA using alternating Gibbs sampling is efficient. We present trace plots of the 6 random latent factors of two TAs in Fig. 5.3. Left panel is from a TA learned on 2D synthetic datasets of Sec. 5.6.1, while the right panel is from a TA modeling high dimensional face images under illumination variations. The plots demonstrate that samples mix very quickly in around 20 Gibbs iterations.

We also looked at how posterior inference converges in a 200 component MTA trained on 8 x 8 natural image patches. The results are in Fig. 5.4.

### 5.5.1 Likelihood Computation

For model comparison, we are interested in evaluating the data log-likelihood $\log p(\mathbf{x}|\theta)$. As noted in Sec. 5.3, a TA with $J$ groups of factors reduces to a FA when conditioned on $J-1$ factor groups. Utilizing the fact that $\log p(\mathbf{x}|\theta)$ can be easily computed (Eq. 5.2), a Monte Carlo estimation of data log-likelihood in TA can be performed by sampling from the prior of the $J-1$ groups of factors. For example, in a model TA$\{D, d_1, d_2\}$, $J = 2$:

$$\begin{aligned} \log p(\mathbf{x}) &= \log \int_{\mathbf{z}_2} p(\mathbf{x}|\mathbf{z}_2) p(\mathbf{z}_2) d\mathbf{z}_2 \\ &\simeq \log \frac{1}{K} \sum_{k=1}^{K} p(\mathbf{x}|\mathbf{z}_2^{(k)}), \quad \mathbf{z}_2^{(k)} \sim \mathcal{N}(0, I). \end{aligned} \tag{5.19}$$

This simple estimator is asymptotically unbiased but has high variance unless the dimensionality of $\mathbf{z}_2$, or $d_2$, is very small. Since $\mathbf{z}_1$ can be analytically integrated out, the Monte Carlo technique can be accurate when only one factor group has large dimensionality.

### 5.5.2 Annealed Importance Sampling for TA

For large $d_j$, however, simple Monte Carlo estimation is very inefficient, giving an estimator with large variance. In this situation, Annealed Importance Sampling (Neal, 2001) is needed.

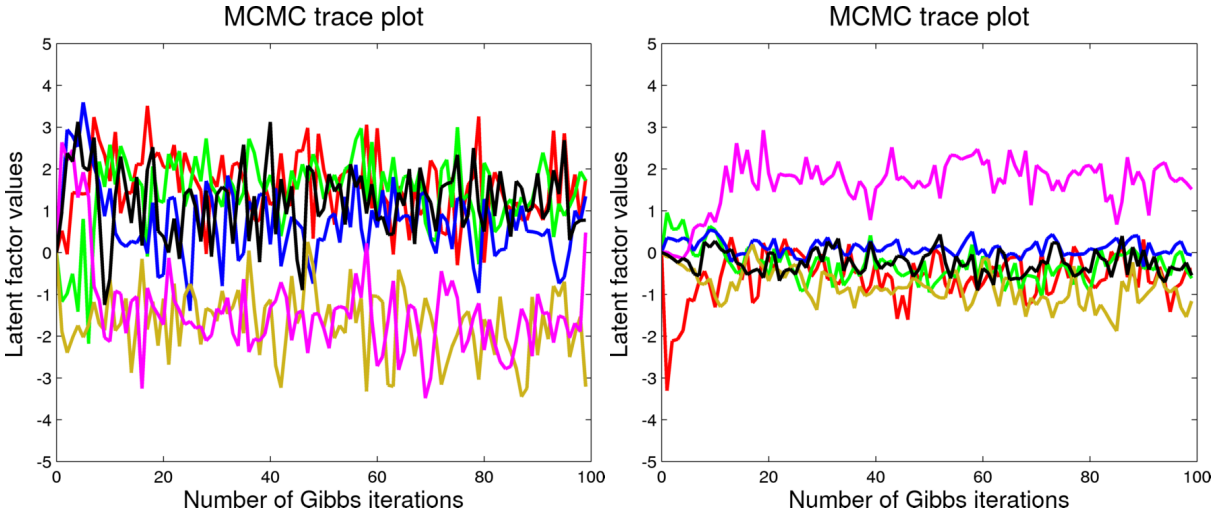



Figure 5.3: **left**: MCMC trace plot for TA learning on synthetic data. **right**:MCMC trace plot for TA learning on high dimensional face images.

We can treat the problem of estimating $\log p(\mathbf{x})$ as calculating the partition function of the unnormalized posterior distribution $p^*(\mathbf{z}|\mathbf{x}) \triangleq p(\mathbf{x}, \mathbf{z})$, where $p^*(\cdot)$ denotes an unnormalized distribution. The basic Importance Sampling gives:

$$p(\mathbf{x}) = \int_{\mathbf{z}} d\mathbf{z}\ p(\mathbf{x}, \mathbf{z}) \tag{5.20}$$

$$p(\mathbf{x}) = \int_{\mathbf{z}} d\mathbf{z}\ \frac{p^*(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})} q(\mathbf{z}) \tag{5.21}$$

$$p(\mathbf{x}) = \frac{Z_p}{1} \simeq \frac{1}{M} \sum_i^M w^{(i)}, \quad w^{(i)} = \frac{p^*(\mathbf{z}^{(i)}|\mathbf{x})}{q(\mathbf{z}^{(i)})}, \quad z^{(i)} \sim q(\mathbf{z}) \tag{5.22}$$

Annealed Importance Sampling specifies a set of intermediate distributions, where $\beta$ varies from 0.0 to 1.0.

$$p_\beta(\mathbf{z}) \propto q(\mathbf{z})^{1-\beta} p^*(\mathbf{z}|\mathbf{x})^\beta \tag{5.23}$$

For TAs, the log of the tractable base distribution $q(\mathbf{z})$ is:

$$\log q(\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_J\}) = \sum_{j=1}^J \Big( -\frac{d_j}{2} \log(2\pi) - \frac{1}{2} \mathbf{z}_j^\mathsf{T} \mathbf{z}_j \Big), \tag{5.24}$$

which is simply the prior distribution over the latent factors.

Since we are using the prior as the base distribution and the unnormalized posterior distribution is the distribution of interest, we can write the intermediate distribution as:

$$p_\beta(\{\mathbf{z}_j\}) \propto q(\{\mathbf{z}_j\})^{1-\beta} p(\mathbf{x}, \{\mathbf{z}_j\})^\beta = p(\{\mathbf{z}_j\}) p^\beta(\mathbf{x}|\{\mathbf{z}_j\}) \tag{5.25}$$

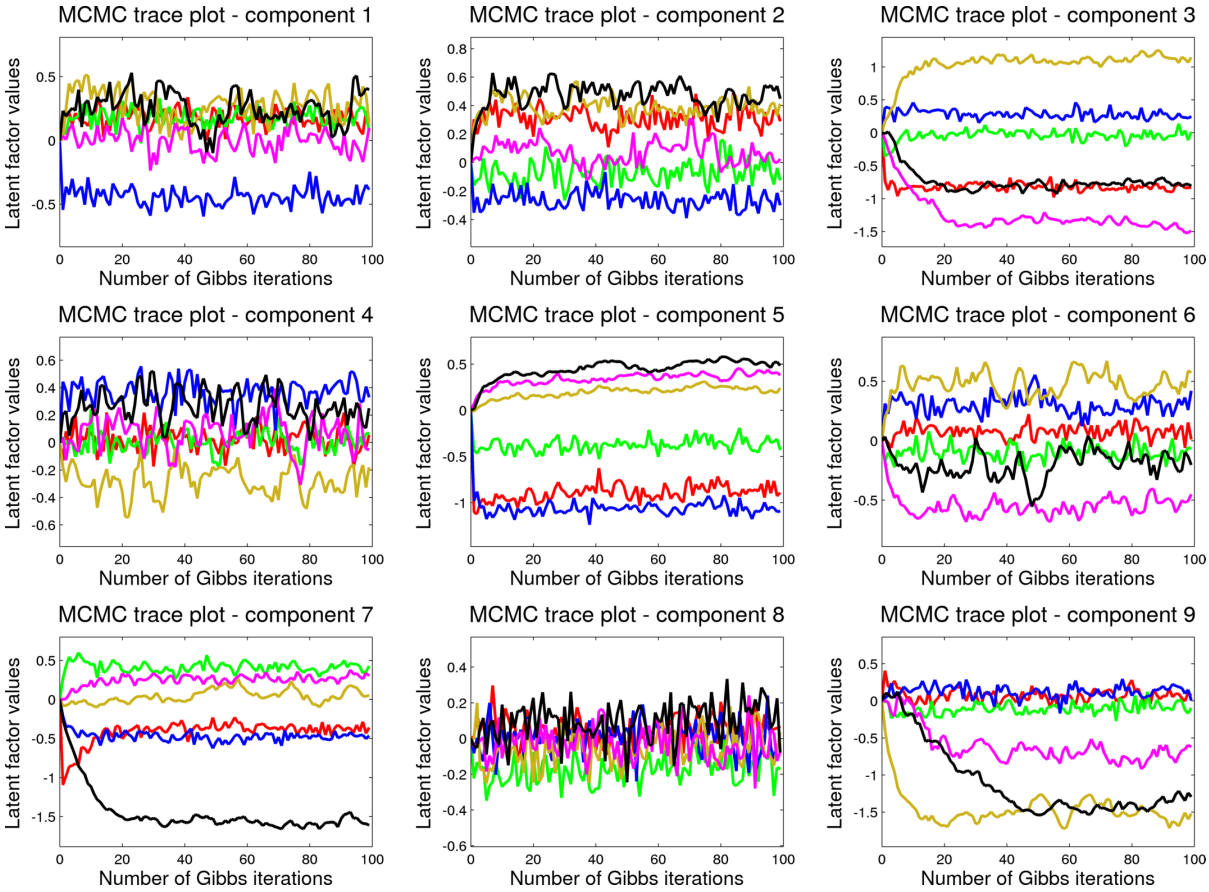



Figure 5.4: Gibbs sampling quickly converges around 20 iterations. 9 components were randomly picked from a MTA with 200 components trained on natural image patches. For each component, we randomly selected 6 latent factors (one for every color).

Therefore,

$$\log p_\beta(\{\mathbf{z}_j\}) = \sum_j^J \Big( -\frac{d_j}{2}\log(2\pi) - \frac{1}{2}\mathbf{z}_j^\mathsf{T}\mathbf{z}_j \Big) - \frac{\beta D}{2}\log(2\pi) - \frac{\beta}{2}\log|\mathbf{\Psi}| - \frac{\beta}{2}(\mathbf{x}-\mathbf{e})^\mathsf{T}\mathbf{\Psi}^{-1}(\mathbf{x}-\mathbf{e}) + C(\beta) \tag{5.26}$$

To ensure that $p_\beta(\{\mathbf{z}_j\})$ sums to 1.0, we find that:

$$C(\beta) = \frac{\beta}{2}\log|\mathbf{\Psi}| + \frac{\beta D}{2}\log(2\pi) - \frac{1}{2}\log|\beta\mathbf{\Psi}| - \frac{D}{2}\log(2\pi) \tag{5.27}$$

Therefore,

$$\log p_\beta(\{\mathbf{z}_j\}) = \sum_j^J \Big( -\frac{d_j}{2}\log(2\pi) - \frac{1}{2}\mathbf{z}_j^\mathsf{T}\mathbf{z}_j \Big) - \frac{D}{2}\log(2\pi) - \frac{1}{2}\log|\beta\mathbf{\Psi}| - \frac{1}{2}(\mathbf{x}-\mathbf{e})^\mathsf{T}(\beta\mathbf{\Psi}^{-1})(\mathbf{x}-\mathbf{e}) \tag{5.28}$$

For AIS, we also need a MCMC operator which leaves $p_\beta(\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_J\})$ invariant. We use a Gibbs sampler, which simply performs alternating Gibbs sampling of the TA's posterior where the original diagonal noise $\mathbf{\Psi}$ is modified to be $\beta\mathbf{\Psi}$. We denote this operator as $T_\beta(\mathbf{z}' \leftarrow \mathbf{z})$.

In Fig. 5.5 (left), We present the AIS algorithm. For clarity, we use $\mathcal{Z}_k$ to denote the set of all latent

**Algorithm 9** AIS for TA

let $k = 1, 2, \ldots, K$, $\beta_{k=1} = 0$, $\beta_{k=K} = 1$.
**for** $i = 1$ to $M$ **do**
  Draw $\mathcal{Z}_2$ from $q(\mathcal{Z})$
  $w^{(i)} = \frac{p_{\beta_2}(\mathcal{Z}_2)}{p_{\beta_1}(\mathcal{Z}_2)}$
  **for** $k = 2$ to $K-1$ **do**
    Sample $\mathcal{Z}_{k+1}$ from $T_{\beta_k}(\mathcal{Z}' \leftarrow \mathcal{Z}_k)$
    $w^{(i)} = w^{(i)} \times \frac{p_{\beta_{k+1}}(\mathcal{Z}_{k+1})}{p_{\beta_k}(\mathcal{Z}_{k+1})}$
  **end for**
**end for**
$p(\mathbf{x}) \simeq \frac{1}{M} \sum_i^M w^{(i)}$

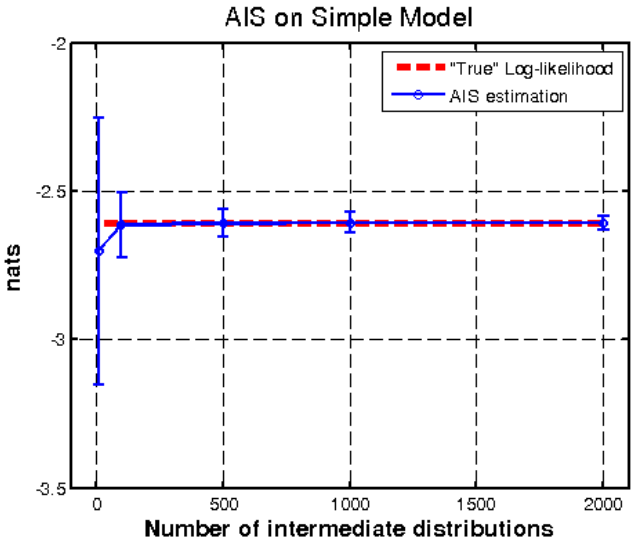



Figure 5.5: **left**: AIS algorithm for TA. **right**: Experimental validation of AIS on a small model, where 100,000 Monte Carlo samples estimate the "true" log-likelihood.

factors $\mathcal{Z} = \{\mathbf{z}_j\}$ of the $k$-th intermediate distribution, $k = 1, 2, \ldots, K$. $M$ is the number of independent AIS chains.

On the right panel of Fig. 5.5, we experimented with the variance of the AIS estimator. Using a small model of TA{2,2,2} trained on a 2D dataset of Fig. 5.6, we ran AIS algorithm with varying number of intermediate distributions to estimate the average data log-likelihood. $M$ is set to 10 in all experiments. In the plot, we can see that the variance of the estimator quickly shrinks to less than 0.1 nats as we use 500 or more intermediate distributions. The dashed line represents the estimated log-likelihood by sampling from the prior, using 100,000 samples. Since we are sampling from only 2 dimensions, this Monte Carlo estimator has very low variance and its value is taken to be the true data log-likelihood.

### 5.5.3 Equality Constraints

An equality constraint indicates that a subset $\{\mathbf{x}^{(k)}\}_{k=1}^K$ of the training data have the same factor values for the $j$-th factor group $\mathbf{z}_j$. For example, if group $j = 1$ represents the identity of a person and group $j = 2$ represents lighting directions, an equality constraint on group 1 indicates that all images of the subset are from the same person, while an equality constraint on group 2 indicates that images of this subset are from the same lighting conditions.

During learning, the availability of equality constraints will only change the inference step. Assuming we have constraints for the factor group $j = 1$, the posterior for $\mathbf{z}_j$ (Eq. 5.8) will be modified as follows:

$$p(\mathbf{z}_1 | \{\mathbf{x}^{(k)}\}, \{\mathbf{z}_2^{(k)}\}, \{\mathbf{z}_3^{(k)}\}) = \mathcal{N}(\mathbf{z}_1 | \tilde{\mathbf{V}}^{-1} \sum_{k=1}^{K} \{\mathbf{\Lambda}^{(k)\mathsf{T}} \mathbf{\Psi}^{-1} (\mathbf{x}^{(k)} - \boldsymbol{\mu}^{(k)})\}, \tilde{\mathbf{V}}^{-1}), \tag{5.29}$$

where $\tilde{\mathbf{V}} = \mathbf{I} + \sum_{k=1}^K \mathbf{\Lambda}^{(k)\mathsf{T}} \mathbf{\Psi}^{-1} \mathbf{\Lambda}^{(k)}$; $\mathbf{\Lambda}^{(k)} = \mathbf{W}_1 + \mathcal{T} \bar{\times}_3 \mathbf{z}_3^{(k)} \bar{\times}_2 \mathbf{z}_2^{(k)}$; and $\boldsymbol{\mu}^{(k)} = \mathbf{m} + \mathbf{W}_2 \mathbf{z}_2^{(k)} + \mathbf{W}_3 \mathbf{z}_3^{(k)}$.

The M-step is not affected by the presence of equality constraints, so TAs can learn when equality constraints are provided for arbitrary subsets of the data.

### 5.5.4 Mixture of Tensor Analyzers

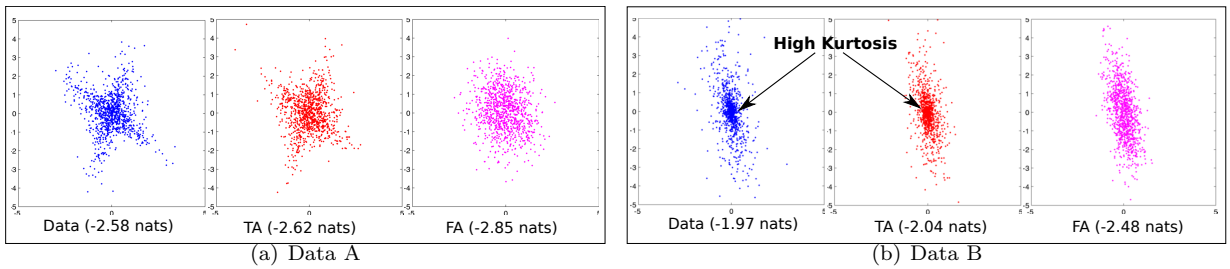
Extending TAs to Mixture of Tensor Analyzers (MTAs) is straightforward, as Sec. 5.5.1 showed how $p(\mathbf{x}|c)$ can be efficiently approximated. Each component $c$ will have its own parameters $\theta_c = \{\mathbf{W}_c, \mathbf{T}_{(1),c}, \mathbf{\Psi}_c\}$. Posterior distribution over the factors and components can be decomposed as: $p(\{\mathbf{z}_j\}, c|\mathbf{x}) = p(\{\mathbf{z}_j\}|\mathbf{x}, c)p(c|\mathbf{x})$, where $p(\{\mathbf{z}_j\}|\mathbf{x}, c)$ can be sampled using Eq. 5.8.

MTAs should be used instead of TAs when modeling highly multimodal data with multiplicative interactions such as multiple types of objects under varying illumination.

For our experiments with MTAs, $p(c|\mathbf{x})$ is approximated with Eq. 5.19 using 1000 samples per mixture component. For our natural images experiment, it means only 180 ms per mixture component is required, see Sec. 5.6.2.

## 5.6 Experiments

### 5.6.1 Synthetic Data



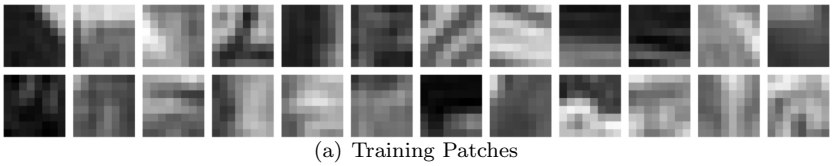

(a) Data A (b) Data B

Figure 5.6: TA vs. FA on 2D synthetic datasets. TAs can better model complex densities.

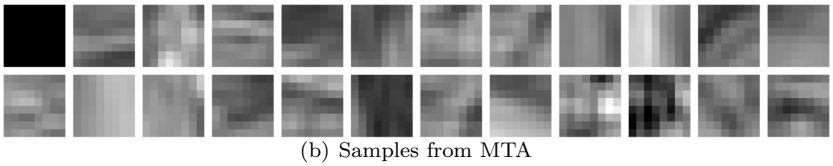
As a proof of concept, we compared TA to FA on two synthetic datasets (Fig. 5.6 A & B). Data A is highly structured and is generated using a TA with random parameters. Data B has high kurtosis, with density concentrated at the origin. For both datasets, we learned using a TA$\{D = 2, d_1 = 2, d_2 = 2\}$ and a FA with the same number of parameters as the TA. The TAs performed model recovery nicely. The left panel of Fig. 5.6(a) displays training points. The data log-likelihood of the true model is -2.58. The middle panel plots the samples of a TA, which achieved the log-probability of -2.62 on the training data. The right panel plots samples drawn from a FA. Likewise in Fig. 5.6(b), TA is a significantly better model than the FA: $-2.04 \pm 0.05$ to $-2.48 \pm 0.09$. We also tested mixtures of TAs vs mixtures of FA (MFA) on data generated by randomly initialized MFA models. The performance of MTA and MFA were very similar, demonstrating that (M)TAs can also efficiently emulate (M)FAs when necessary.

### 5.6.2 Natural Images

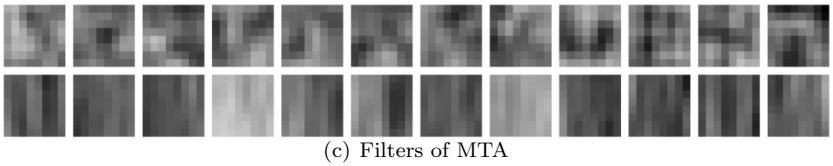
Learning a good density model of natural images is useful for image denoising and inpainting. Following Zoran and Weiss (2011), we compared MTAs to MFAs and other models on modeling image patches. Two million $8 \times 8$ patches were extracted from the training set of the Berkeley Natural Images



(a) Training Patches



(b) Samples from MTA



(c) Filters of MTA

| Independent Pixel | ICA | GRBM | DBN | Factored Bilinear |
|---|---|---|---|---|
| 78.3 | 135.7 ±1.2 | 137.8 ±1.0 | 144.4 ±1.1 | 145.9 ±2.7 |

| Single FA | MFA | Deep MFA | GMM | MTA |
|---|---|---|---|---|
| 130.1 ±0.6 | 166.5 ±1.8 | 169.3 ±1.5 | 167.2 | 158.2 ±3.3 |

(d) Average test log-probability (measured in nats)

Figure 5.7: Natural image patches. (a) Training data. (b) Samples from MTA. (c) Each row contains filters from a different MTA component. (d) Average test log-probability comparisons. ICA: Independent Component Analysis. GRBM: Gaussian Restricted Boltzmann Machine (Hinton and Salakhutdinov, 2006). DBN: Deep Belief Nets (Hinton et al., 2006). Factored Bilinear: (Culpepper et al., 2011).

database (Martin et al., 2001) for training, while 50,000 patches from the test set were extracted for testing. The DC component of each image patch was set to 0 by subtracting the patch mean from every pixel. For MTAs and MFAs[2], we used 200 components and selected the number of latent factors based on cross validation. For MTAs, 64 factors were used while each component of the MTA is a TA{64, 64, 5}. For the Factorized Bilinear model, we used code from the authors[3]. Independent Pixel and GMM results are from Zoran and Weiss (2011), while the Deep MFA result is from Tang et al. (2012d).

After training, Fig. 5.7(b) shows that samples of a MTA closely match the training patches in Fig. 5.7(a). Each row of Fig. 5.7(c) shows filters (fibers of the tensor) of one of the MTA components. Components of MTA specialize to model patches of different spatial frequencies and orientations. Fig. 5.7(d) lists quantitative evaluations of the different models. Although MTAs achieve good results, they do not outperform MFAs or GMMs with 200 mixture components[4].

We also compared the performance of MFAs vs. MTAs when the number of mixture components is smaller. Fig. 5.8 shows the average log-likelihoods on the test set. Due to the fact that MTAs can model higher-order interaction, they are able to outperform MFAs with 50 or less mixture components. However, as the number of components increases, the advantage of MTAs disappears.

[2]MFA code (Verbeek, 2006) is downloaded from http://lear.inrialpes.fr/∼verbeek/software.php

[3]https://github.com/jackculpepper/dir-linear.

[4]The gap (less than 10 nats) in performance can be attributed to the fact that the E-step in MTA is not exact and requires sampling, while MFA has a closed-form posterior. Sampling introduces noise which adversely affects the M-step updates.

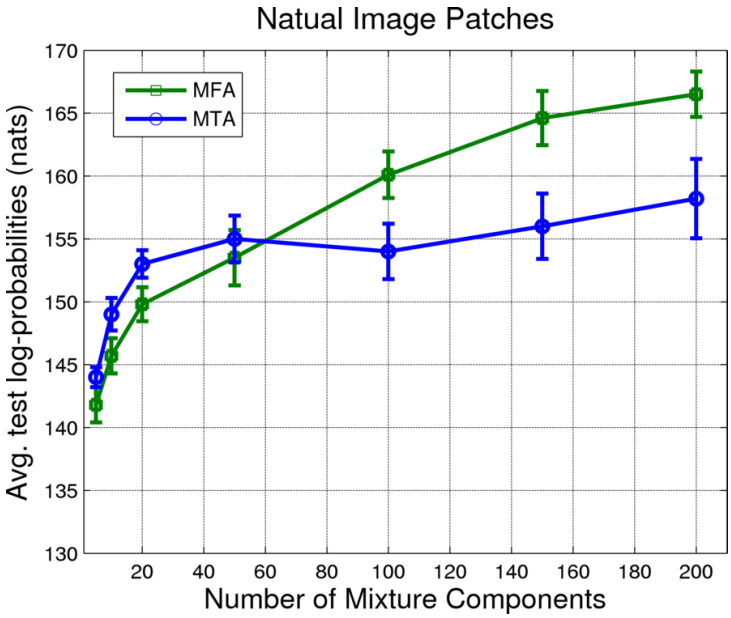



Figure 5.8: Performance of MTA vs. MFA with different numbers of mixture components.

### 5.6.3 Face Recognition with equality constraints

In a one-shot learning setting, classifiers must be trained from only one example per class. For face recognition, only one example per test subject is used for training. We use the Yale B database and its Extended version (Lee et al., 2005). The database contains 38 subjects under 45 different lighting conditions. We use 28 subjects for training and test on the 10 subjects from the original Yale B database. The images are first downsampled to $24 \times 24$, and we used a TA$\{576, 80, 4\}$, which contains 2 groups of factors.

The learned TA allows for strong generalization to new people under new lighting conditions. It achieves an average test log-probability of $\mathbf{836 \pm 7}$ on the images of the 10 held-out subjects. As a comparison, the best MFA model achieved only $791 \pm 10$. The number of components and factors of the MFA are selected using grid search. The gain of 45 nats demonstrates a significant win for the TA.

Qualitatively, to see how well the TA is able to factor out identity from lighting, we first sample from the posterior distribution conditioned on a single test image using step 3 of Alg. 8. We then fix a factor group's sampled values and sample the factors of the other group using its Normal prior. Results are shown in Fig. 5.9. A row in panel (b) shows the same person under different sampled lighting conditions. A row in panel (c) shows sampled people, but under the same lighting condition. We emphasize that only a *single* test image from *novel* subjects is used for inference (panel (a)).

The one-shot recognition task consists of using a *single* image of each of the 10 test subjects for training during the *testing* phase. The rest of the images of the test subjects are then classified into 1 of 10 classes. Labels indicating lighting directions of the test images can be optionally provided. TAs can operate with or without the labels, while tensor decomposition methods require these labels. We first use the 28 training subjects to learn the parameters for the TA{576,80,4} in the training phase. Equality constraints specifying which images have the same identity or lighting type are used during training. During the testing phase, a single image for each of the 10 test subjects is used to compute

(a) Test (b) Fixing Identity (c) Fixing Lighting

| Tasks | Human | NN | XCORR | SVM | FA | DLN | CP | NTF | Tucker | pTucker | S&C | GPSC | TA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| One-shot recognition | 9.3 | 51.1 | 33.6 | 35.5 | 19.6 | 19 | N/A | N/A | N/A | N/A | 24.0 | N/A | **6.4** |
| Using lighting labels | | | | | | | 18.3 | 20.4 | 37.3 | 23.6 | 21.7 | 14.5 | |

(d) Recognition Error in %

Figure 5.9: Tensor Analyzer is able to simultaneously decompose a test image into separate identity and lighting factors. (a) 4 test images with frontal, left, right and top light sources. (b) Random samples with identity factor fixed to the inferred values from the test faces in (a). (c) Random samples with lighting factor fixed to the inferred values. (d) Comparison to other methods. Tensor decomposition methods require additional labels of lighting direction during the test phase.

the mean of the posterior $p(\mathbf{z}_{identity}, \mathbf{z}_{light}|\mathbf{x})$, using 200 Gibbs steps. For each one of the 10 sampled $\mathbf{z}_{identity}$, TA{576,80,4} is transformed into a person-specific FA using Eqs. 5.6, 5.7. The resulting 10 FAs define the class-conditionals of a generative classifier, which we use to classify test images of the 10 test subjects.

We compared TA to standard classifiers and tensor decomposition methods. Human error is the average of testing several human subjects on the same one-shot recognition task. NN is the nearest neighbour classifier, and XCORR represents the normalized Cross Correlation. Multiclass linear SVM from the LIBLINEAR package (Fan et al., 2008) is used, where the hyperparameter $C$ was chosen using validation. The Factor Analyzers method requires the transfer of the factor loadings from the training phase. We first learned 28 FAs, one for each training subject. During the testing phase, each one of the 10 training images of the test subject is matched with the FA which gives it the largest likelihood. A new FA is created, centered at the training image. The rest of the parameters are transferred from the matched FA. In essence, the transferred loadings model the lighting variations of the training subjects (1 of 28), which is most similar to the test phase training image. After the creation of these 10 new FAs, classification is same as in the TA method.

We also compared TA to various Tensor Decomposition methods, including CP: CANDECOMP / PARAFAC (Carroll and Chang, 1970); NTF: Nonnegative Tensor Factorization (Shashua and Hazan, 2005); Tucker (Tucker, 1963); and Probabilistic Tucker decomposition (Chu and Ghahramani, 2009). For CP, Tucker, and NTF, we used the N-way toolbox (Bro, 1998). For the S&C bilinear method, we implemented the exact algorithm as stated in Sec. 3.2 of Tenenbaum and Freeman (2000). The style and content codes are adapted for new test images during one-shot learning. For GPSC, the code provided by the authors was used. In all experiments, hyper-parameters were selected by cross-validation. Recognition errors are shown in Fig. 5.9(d). Observe that the TA not only outperforms these methods by a significant margin, but it even achieves better than human performance.

### 5.6.4 Learning with incomplete equality constraints

We now demonstrate the advantage of the TA in a semi-supervised setting on the UMIST face database (Graham and Allinson, 1998). It contains 20 subjects with 20 to 40 training images per subject. The variation consists of in-depth head rotations. We compared the TA to S&C model and NN on the one-shot face recognition task. Out of the 20 subjects, 15 were used for training and 5 for testing. The split was randomized over 10 different trials. The images are downsampled to the resolution of $24 \times 24$. We experimented with equality constraints, using 3, 4, or 5 images per training subject. During the training phase, a TA{576,3,5} model was trained using 30 EM iterations. The algorithms for classification are exactly the same as in Sec. 5.6.3. Fig. 5.10 plots recognition errors as a function of the number of

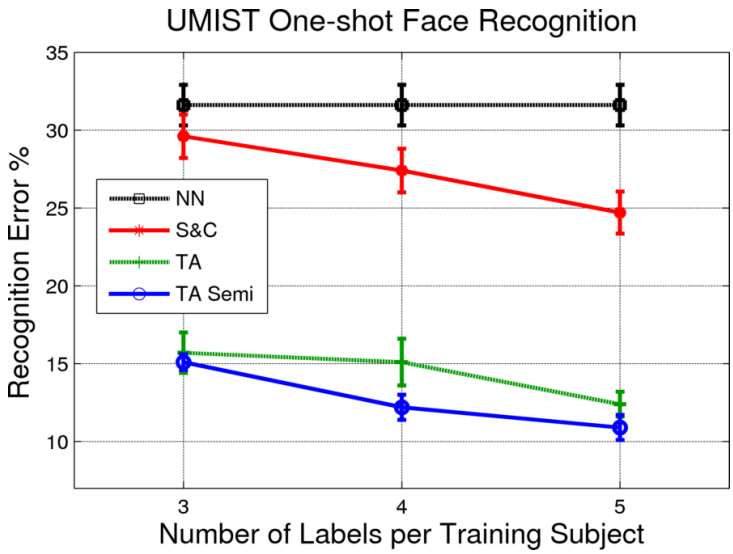



Figure 5.10: UMIST Recognition errors.

images per subject used during the *training phase*. For the case where 5 images per training subject are available, TA achieves significantly lower errors at 12.4% compared to 24.9% for S&C. If we add an equal number of images without equality constraints during training, the error is further reduced to **10.9%**.

### 5.6.5 Computation Time

Our experiments used Matlab on a standard multicore workstation, we report the computation time required for inference and learning. MTA on face images in Sec. 5.6.3: inference per image per Gibbs update takes 14 ms (milliseconds), while learning took a total of 587 secs. MFA learning took around 108 secs. In practice, matrix inversion is not very expensive in our model. For the face recognition model TA{576,80,4}, while having the same number of parameters as a FA with 320 latent factors (which requires inverting 320x320 matrix), only require inversion of 80x80 and 4x4 matrices, which takes 0.4 ms. MTA on natural images: inference per image per Gibbs update takes 47 ms, while the entire learning algorithm took 33 hours. In comparison, inference per image in Mixture of Factor Analyzers (MFA) takes 18 ms, while learning took 9.5 hours.

## 5.7 Shortcomings

Tensor Analyzers are excellent for face recognition under heavy illumination variations. However, one disadvantage of the model is that it requires labels in order to learn to accurately separate the factors of variation. In addition, while the model works well when the input data contains multilinear variations, it is not natural for modeling non-linear variations. Examples of non-linear variations include in-depth rotations of an object or articulated human pose variations.

## 5.8 Conclusions

We have introduced a new density model which extends Factor Analysis to modeling multilinear interactions. Using efficient alternating sampling and the EM algorithm, we have shown that (M)TAs can learn more complex densities and separate factors of variation, leading to the learning of simple concepts. Moreover, at the important task of one-shot face recognition, TAs outperform a variety of other models.

Tensor Analyzers assume that the data is generated from a multilinear interaction of the hidden factors. While this assumption holds true for faces under varying illumination conditions, other forms of variations do not necessarily follow these interactions. In particular, 2D transformations such as translations and rotations are not multilinear in pixel intensity space. While we can still use Tensor Analyzers to model these multi-nonlinear interactions, it is not a good model and the priors of the latent codes will not be anywhere close to Gaussian. The ideal way to approach this problem is to first transform the input data from pixel space to some higher-level feature space, where style and content variations with respect to these features are close to multilinear. In this scenario, a deep network first transforms the pixels into a more salient feature space. Tensor Analyzers would then be used to model the multilinear interaction in feature space.

# Chapter 6

# Learning generative models with visual attention

Attention has long been proposed by psychologists to be important for efficiently dealing with the massive amounts of sensory stimulus in the neocortex. In this chapter, inspired by the attention models in visual neuroscience and the need for object-centered data for generative models, we propose a deep-learning based generative framework using attention. The attentional mechanism propagates signals from the region of interest in a scene to an aligned canonical representation for generative modeling. By ignoring scene background clutter, the generative model can concentrate its resources on the object of interest. A convolutional neural net is employed to provide good initializations during posterior inference which uses Hamiltonian Monte Carlo. Upon learning images of faces, our model can robustly attend to the face region of novel test subjects. More importantly, our model can learn generative models of new faces from a novel dataset of large images where the face locations are not known.

Existing generative models can synthesize realistic looking digits, faces and (to a lesser degree) objects. However, the training data for these models are often cropped to be low-resolution (64 pixels by 64 pixels or less). There are two main reasons for doing this. The first is to reduce the dimensionality of the input so that the parameters of the model is greatly reduced. Note that for models such as the de-convolutional neural nets this might not be such a big problem. The second reason is that by choosing to focus on a specific object, the manifold of variation is much easier to model. For example, it is much easier to model cropped faces rather than the entire head, which includes the neck and hair. Some papers have tried to model the big picture or full sized in-the-wild images by using tiled convolutions. However, due to the complexity of the objects and backgrounds, learning generative model of the entire scene has achieved limited success. Nevertheless, for recognition tasks we may not need to model the entire scene. This is because the object of interest typically only occupies a small region within the image. In this chapter, we propose a model where attention is used to route pertinent stimuli within the large image for generative modeling.

## 6.1 Motivation

Building rich generative models that are capable of extracting useful, high-level latent representations from high-dimensional sensory input lies at the core of solving many AI-related tasks, including object

recognition, speech perception and language understanding. These models capture underlying structure in data by defining flexible probability distributions over high-dimensional data as part of a complex, partially observed system. Some of the successful generative models that are able to discover meaningful high-level latent representations include the Boltzmann Machine family of models: Restricted Boltzmann Machines, Deep Belief Nets (Hinton et al., 2006), and Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009). Mixture models, such as Mixtures of Factor Analyzers (Ghahramani and Hinton, 1996) and Mixtures of Gaussians, have also been used for modeling natural image patches (Zoran and Weiss, 2011). Energy-based models have also been proposed for learning a sparse and over-complete representation of the data (Teh et al., 2003). More recently, denoising auto-encoders have been proposed as a way to model the transition operator that has the same invariant distribution as the data generating distribution (Bengio et al., 2013).

Generative models have an advantage over discriminative models when parts of the images are occluded or missing. Occlusions are very common in realistic settings and have been largely ignored in recent literature on deep learning. In addition, prior knowledge can be easily incorporated in generative models in the form of structured latent variables, such as lighting and deformable parts. However, the enormous amount of content in high-resolution images makes generative learning difficult (Lee et al., 2009; Ranzato et al., 2011). Therefore, generative models have found most success in learning to model small patches of natural images and objects: Zoran and Weiss (2011) learned a mixture of Gaussians model over $8 \times 8$ image patches; Salakhutdinov and Hinton (2009) used $64 \times 64$ centered and uncluttered stereo images of toy objects on a clear background; Tang et al. (2012a) used $24 \times 24$ images of centered and cropped faces. The fact that these models require curated training data limits their applicability to completely unlabeled data, which is virtually unlimited.

In this chapter, we propose a framework to infer the region of interest in a big image for generative modeling. This will allow us to learn a generative model of faces on a very large dataset of (unlabeled) images containing faces. Our framework is able to dynamically route the relevant information to the generative model and can ignore the background clutter. The need to dynamically and selectively route information is also present in the biological brain. A plethora of evidence points to the presence of attention in the visual cortex (Posner and Gilbert, 1999; Buffalo et al., 2010). Recently, in visual neuroscience, attention has been shown to exist not only in extrastriate areas, but also all the way down to V1 (Kanwisher and Wojciulik, 2000).

Hinton (1981) first proposed a way to dynamically route the stimulus to a canonical object-based frame of reference. To recognize familiar object in an unusual orientation, a canonical frame of reference can be imposed on the object, leading to a viewpoint-independent description. Retina-based units represent object parts in any orientation, position and size. Dynamic routing by mapping units activates object-based units leading to recognition. This form of routing is viewed as a form of attention in neuroscience. A computational model was proposed by Anderson and Van Essen (Anderson and Essen, 1987) and then extended by Olshausen et al. (1993). Dynamic routing has been hypothesized as providing a way for achieving shift and size invariance in the visual cortex (Wiskott, 2004; Chikkerur et al., 2010). Tsotsos et al. (1995) proposed a model combining search and attention called the Selective Tuning model. Larochelle and Hinton (2010) proposed a way of using third-order Boltzmann Machines to combine information gathered from many foveal glimpses. Their model chooses where to look next to find locations that are most informative of the object class. Reichert et al. (2011) proposed a hierarchical model to show that certain aspects of *covert* object-based attention can be modeled by Deep

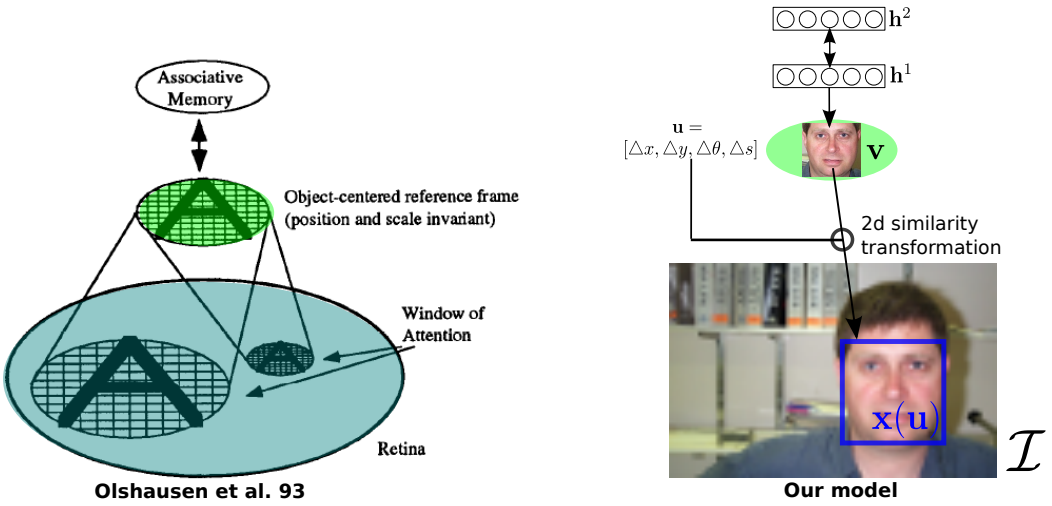



Figure 6.1: **Left:** The Shifter Circuit, a well-known neuroscience model for visual attention (Olshausen et al., 1993); **Right**: The proposed model uses 2D similarity transformations from geometry and a Gaussian DBN to model canonical face images. Associative memory corresponds to the DBN, object-centered frame correspond to the visible layer and the attentional mechanism is modeled by 2D similarity transformations.

Boltzmann Machines. Several other related models attempt to learn where to look for objects (Alexe et al., 2012; Ranzato, 2014) and for video based tracking (Denil et al., 2012). Inspired by Hinton (1981) and Olshausen et al. (1993), we use 2D similarity transformations to implement the scaling, rotation, and shift operation required for routing. Our main motivation is to enable the learning of generative models in big images where the location of the object of interest is unknown a-priori. Frey and Jojic (1999) proposed a similar framework for learning a mixture of Gaussians from images and videos by estimating spatial transformations. However, their framework represents transformations by a set of discrete matrices, which limits (due to computation costs) their model to only a relative coarse set of translations, scales and rotations.

## 6.2 The Model

Let $\mathcal{I}$ be a high resolution image of a scene, e.g. a $256 \times 256$ image. We want to use attention to propagate regions of interest from $\mathcal{I}$ up to a canonical representation. For example, in order to learn a model of faces, the canonical representation could be a $24 \times 24$ aligned and cropped frontal face image. Let $\mathbf{v} \in \mathbb{R}^D$ represent this low resolution canonical image. In this work, we focus on a Deep Belief Network[1] to model $\mathbf{v}$.

This is illustrated in the diagrams of Fig. 6.1. The left panel displays the model of Olshausen et al. (1993), whereas the right panel shows a graphical diagram of our proposed generative model with an attentional mechanism. Here, $\mathbf{h}^1$ and $\mathbf{h}^2$ represent the latent hidden variables of the DBN model, and $\triangle x, \triangle y, \triangle\theta, \triangle s$ (position, rotation, and scale) are the parameters of the 2D similarity transformation.

The 2D similarity transformation is used to rotate, scale, and translate the canonical image $\mathbf{v}$ onto the canvas that we denote by $\mathcal{I}$. Let $\mathbf{p} = [x\ y]^\mathsf{T}$ be a pixel coordinate (e.g. $[0, 0]$ or $[0, 1]$) of the canonical image $\mathbf{v}$. Let $\{\mathbf{p}\}$ be the set of all coordinates of $\mathbf{v}$. For example, if $\mathbf{v}$ is $24 \times 24$, then $\{\mathbf{p}\}$ ranges

[1] Other generative models can also be used with our attention framework.

from $[0, 0]$ to $[23, 23]$. Let the "gaze" variables $\mathbf{u} \in \mathbb{R}^4 \equiv [\triangle x, \triangle y, \triangle\theta, \triangle s]$ be the parameter of the Similarity transformation. In order to simplify derivations and to make transformations be linear w.r.t. the transformation parameters, we can equivalently redefine $\mathbf{u} = [a,\ b,\ \triangle x,\ \triangle y]$, where $a = s\sin(\theta) - 1$ and $b = s\cos(\theta)$ (see Szeliski (2011) for details). We further define a function $\mathsf{w} := \mathsf{w}(\mathbf{p}, \mathbf{u}) \rightarrow \mathbf{p}'$ as the transformation function to *warp* points $\mathbf{p}$ to $\mathbf{p}'$:

$$\mathbf{p}' \triangleq \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1+a & -b \\ b & 1+a \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \triangle x \\ \triangle y \end{bmatrix}. \tag{6.1}$$

We use the notation $\mathcal{I}(\{\mathbf{p}\})$ to denote the bilinear interpolation of $\mathcal{I}$ at coordinates $\{\mathbf{p}\}$ with anti-aliasing. Let $\mathbf{x}(\mathbf{u})$ be the extracted low-resolution image at warped locations $\mathbf{p}'$:

$$\mathbf{x}(\mathbf{u}) \triangleq \mathcal{I}(\mathsf{w}(\{\mathbf{p}\}, \mathbf{u})). \tag{6.2}$$

Intuitively, $\mathbf{x}(\mathbf{u})$ is a patch extracted from $\mathcal{I}$ according to the shift, rotation and scale parameters of $\mathbf{u}$, as shown in Fig. 6.1, right panel. It is this patch of data that we seek to model generatively. Note that the dimensionality of $\mathbf{x}(\mathbf{u})$ is equal to the cardinality of $\{\mathbf{p}\}$, where $\{\mathbf{p}\}$ denotes the set of pixel coordinates of the canonical image $\mathbf{v}$. Unlike standard generative learning tasks, the data $\mathbf{x}(\mathbf{u})$ is not static but changes with the latent variables $\mathbf{u}$. Given $\mathbf{v}$ and $\mathbf{u}$, we model the top-down generative process over[2] $\mathbf{x}$ with a Gaussian distribution having a diagonal covariance matrix $\sigma^2\mathrm{I}$:

$$p(\mathbf{x}|\mathbf{v}, \mathbf{u}, \mathcal{I}) \propto \exp\left( -\frac{1}{2} \sum_i \frac{(x_i(\mathbf{u}) - v_i)^2}{\sigma_i^2} \right). \tag{6.3}$$

The fact that we do not seek to model the rest of the regions/pixels of $\mathcal{I}$ is by design. By using a cropped 2D similarity transformation to mimic attention, we can discard the complex background of the scene and let the generative model focus on the object of interest. The proposed generative model takes the following form:

$$p(\mathbf{x}, \mathbf{v}, \mathbf{u}|\mathcal{I}) = p(\mathbf{x}|\mathbf{v}, \mathbf{u}, \mathcal{I})p(\mathbf{v})p(\mathbf{u}), \tag{6.4}$$

where for $p(\mathbf{u})$ we use a flat prior that is constant for all $\mathbf{u}$, and $p(\mathbf{v})$ is defined by a 2-layer Gaussian Deep Belief Network. The conditional $p(\mathbf{x}|\mathbf{v}, \mathbf{u}, \mathcal{I})$ is given by a Gaussian distribution as in Eq. 6.3. To simplify the inference procedure, $p(\mathbf{x}|\mathbf{v}, \mathbf{u}, \mathcal{I})$ and the GDBN model of $\mathbf{v}$, $p(\mathbf{v})$, will share the same noise parameters $\sigma_i$.

## 6.3 Inference

While the generative equations in the last section are straightforward and intuitive, inference in these models is typically intractable due to the complicated energy landscape of the posterior. During inference, we wish to compute the distribution over the gaze variables $\mathbf{u}$ and canonical object $\mathbf{v}$ given the big image $\mathcal{I}$. Unlike in standard RBMs and DBNs, there are no simplifying factorial assumptions about the conditional distribution of the latent variable $\mathbf{u}$. Having a 2D similarity transformation is reminiscent of third-order Boltzmann machines with $\mathbf{u}$ performing top-down multiplicative gating of the connections

[2]We will often omit dependence of $\mathbf{x}$ on $\mathbf{u}$ for clarity of presentation.

between $\mathbf{v}$ and $\mathcal{I}$. It is well known that inference in these higher-order models is rather complicated.

One way to perform inference in our model is to resort to Gibbs sampling by computing the set of alternating conditional posteriors: The conditional distribution over the canonical image $\mathbf{v}$ takes the following form:

$$p(\mathbf{v}|\mathbf{u},\mathbf{h}^1,\mathcal{I}) = \mathcal{N}\Big(\frac{\boldsymbol{\mu}+\mathbf{x}(\mathbf{u})}{2};\boldsymbol{\sigma}^2\Big), \tag{6.5}$$

where $\mu_i = b_i + \sigma_i^2 \sum_j W_{ij} h_j^1$ is the top-down influence of the DBN. Note that if we know the gaze variable $\mathbf{u}$ and the first layer of hidden variables $\mathbf{h}^1$, then $\mathbf{v}$ is simply defined by a Gaussian distribution, where the mean is given by the average of the top-down influence and bottom-up information from $\mathbf{x}$. The conditional distributions over $\mathbf{h}^1$ and $\mathbf{h}^2$ given $\mathbf{v}$ are given by the standard DBN inference equations (Hinton et al., 2006). The conditional posterior over the gaze variables $\mathbf{u}$ is given by:

$$\begin{aligned} p(\mathbf{u}|\mathbf{x},\mathbf{v}) &= \frac{p(\mathbf{x}|\mathbf{u},\mathbf{v})p(\mathbf{u})}{p(\mathbf{x}|\mathbf{v})}, \\ \log p(\mathbf{u}|\mathbf{x},\mathbf{v}) \propto \log p(\mathbf{x}|\mathbf{u},\mathbf{v}) + \log p(\mathbf{u}) &= \frac{1}{2}\sum_i \frac{(x_i(\mathbf{u})-v_i)^2}{\sigma_i^2} + const. \end{aligned} \tag{6.6}$$

Using Bayes' rule, the unnormalized log probability of $p(\mathbf{u}|\mathbf{x},\mathbf{v})$ is defined in Eq. 6.6. We stress that this equation is atypical in that the random variable of interest $\mathbf{u}$ actually affects the conditioning variable $\mathbf{x}$ (see Eq. 6.2) We can explore the gaze variables using Hamiltonian Monte Carlo (HMC) algorithm (Duane et al., 1987; Neal, 2010). Intuitively, conditioned on the canonical object $\mathbf{v}$ that our model has in "mind", HMC searches over the entire image $\mathcal{I}$ to find a region $\mathbf{x}$ with a good match to $\mathbf{v}$.

If the goal is only to find the MAP estimate of $p(\mathbf{u}|\mathbf{x},\mathbf{v})$, then we may want to use second-order methods for optimizing $\mathbf{u}$. This would be equivalent to the Lucas-Kanade framework in computer vision, developed for image alignment (Baker and Matthews, 2002). However, HMC has the advantage of being a proper MCMC sampler that satisfies detailed balance and fits nicely with our probabilistic framework.

The HMC algorithm first specifies the Hamiltonian over the position variables $\mathbf{u}$ and auxiliary momentum variables $\mathbf{r}$: $\mathcal{H}(\mathbf{u},\mathbf{r}) = U(\mathbf{u}) + K(\mathbf{r})$, where the potential function is defined by $U(\mathbf{u}) = \frac{1}{2}\sum_i \frac{(x_i(\mathbf{u})-v_i)^2}{\sigma_i^2}$ and the kinetic energy function is given by $K(\mathbf{r}) = \frac{1}{2}\sum_i r_i^2$. The dynamics of the system is defined by:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{r}, \qquad \frac{\partial \mathbf{r}}{\partial t} = -\frac{\partial \mathcal{H}}{\partial \mathbf{u}} \tag{6.7}$$

$$\frac{\partial \mathcal{H}}{\partial \mathbf{u}} = \frac{(\mathbf{x}(\mathbf{u})-\mathbf{v})}{\sigma^2}\frac{\partial \mathbf{x}(\mathbf{u})}{\partial \mathbf{u}}, \tag{6.8}$$

$$\frac{\partial \mathbf{x}}{\partial \mathbf{u}} = \frac{\partial \mathbf{x}}{\partial \mathsf{w}(\{\mathbf{p}\},\mathbf{u})}\frac{\partial \mathsf{w}(\{\mathbf{p}\},\mathbf{u})}{\partial \mathbf{u}} = \sum_i \frac{\partial x_i}{\partial \mathsf{w}(\mathbf{p}_i,\mathbf{u})}\frac{\partial \mathsf{w}(\mathbf{p}_i,\mathbf{u})}{\partial \mathbf{u}}. \tag{6.9}$$

Observe that Eq. 6.9 decomposes into sums over single coordinate positions $\mathbf{p}_i = [x\ y]^\mathsf{T}$. Let us denote $\mathbf{p'}_i = \mathsf{w}(\mathbf{p}_i,\mathbf{u})$ to be the coordinate $\mathbf{p}_i$ warped by $\mathbf{u}$. For the first term on the RHS of Eq. 6.9,

$$\frac{\partial x_i}{\partial \mathsf{w}(\mathbf{p}_i, \mathbf{u})} = \nabla I(\mathbf{p'}_i), \quad \text{(dimension 1 by 2 )} \tag{6.10}$$

where $\nabla I(\mathbf{p'}_i)$ denotes the sampling of the gradient images of $I$ at the warped location $\mathbf{p}_i$. For the second term on the RHS of Eq. 6.9, we note that we can re-write Eq. 6.1 as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x & -y & 1 & 0 \\ y & x & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ \triangle x \\ \triangle y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}, \tag{6.11}$$

giving us

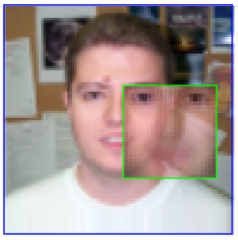
$$\frac{\partial \mathsf{w}(\mathbf{p}_i, \mathbf{u})}{\partial \mathbf{u}} = \begin{bmatrix} x & -y & 1 & 0 \\ y & x & 0 & 1 \end{bmatrix}. \tag{6.12}$$

HMC simulates the discretized system by performing leap-frog updates of $\mathbf{u}$ and $\mathbf{r}$ using Eq. 6.7. Additional hyperparameters that need to be specified include the step size $\epsilon$, number of leap-frog steps, and the mass of the variables (see Neal (2010) for details).

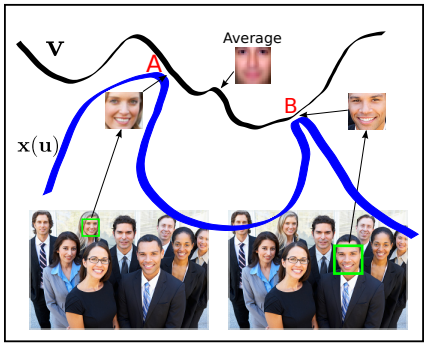
### 6.3.1 Approximate Inference

HMC essentially performs gradient descent with momentum, therefore it is prone to getting stuck at local optimums. This is especially a problem for our task of finding the best transformation parameters. While the posterior over $\mathbf{u}$ should be unimodal near the optimum, many local minima exist away from the global optimum. For example, in Fig. 6.2(a), the big image $\mathcal{I}$ is enclosed by the blue box, and the canonical image $\mathbf{v}$ is enclosed by the green box. The current setting of $\mathbf{u}$ aligns together the wrong eyes. However, it is hard to move the green box to the left due to the local optima created by the dark intensities of the eye. Resampling the momentum variable every iteration in HMC does not help significantly because we are modeling real-valued images using a Gaussian distribution as the residual, leading to quadratic costs in the difference between $\mathbf{x}(\mathbf{u})$ and $\mathbf{v}$ (see Eq. 6.6). This makes the energy barriers between modes extremely high.

To alleviate this problem we need to find good initializations of $\mathbf{u}$. We use a Convolutional Network (ConvNet) to perform efficient approximate inference, resulting in good initial guesses. Specifically, given $\mathbf{v}$, $\mathbf{u}$ and $\mathcal{I}$, we predict the change in $\mathbf{u}$ that will lead to the maximum $\log p(\mathbf{u}|\mathbf{x}, \mathbf{v})$. In other words, instead of using the gradient field for updating $\mathbf{u}$, we learn a ConvNet to output a better vector field in the space of $\mathbf{u}$. We used a fairly standard ConvNet architecture and the standard stochastic gradient descent learning procedure.



(a)




(b)

Figure 6.2: (a) HMC can easily get stuck at local optima. (b) Importance of modeling $p(\mathbf{u}|\mathbf{v}, \mathcal{I})$. Best in color.

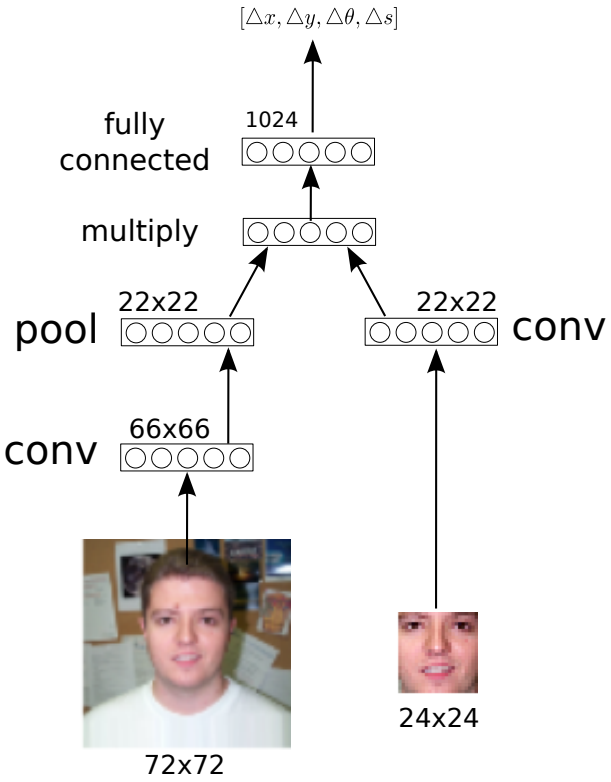



Figure 6.3: A visual diagram of the convolutional net used for approximate inference.

We note that standard feedforward face detectors seek to model $p(\mathbf{u}|\mathcal{I})$, while completely ignoring the canonical face $\mathbf{v}$. In contrast, here we take $\mathbf{v}$ into account as well. The ConvNet is used to initialize $\mathbf{u}$ for the HMC algorithm. This is important in a proper generative model because conditioning on $\mathbf{v}$ is appealing when multiple faces are present in the scene. Fig. 6.2(b) is a hypothesized Euclidean space of $\mathbf{v}$, where the black manifold represents canonical faces and the blue manifold represents cropped faces $\mathbf{x}(\mathbf{u})$. The blue manifold has a low intrinsic dimensionality of 4, spanned by $\mathbf{u}$. At A and B, the blue comes close to black manifold. This means that there are at least two modes in the posterior over $\mathbf{u}$. By conditioning on $\mathbf{v}$, we can narrow the posterior to a single mode, depending on whom we want to focus our attention. We demonstrate this exact capability in Sec. 6.5.3.

Fig. 6.4 demonstrates the iterative process of how approximate inference works in our model. Specifically, based on $\mathbf{u}$, the ConvNet takes a window patch around $\mathbf{x}(\mathbf{u})$ ($72 \times 72$) and $\mathbf{v}$ ($24 \times 24$) as input, and predicts the output $[\triangle x, \triangle y, \triangle\theta, \triangle s]$. In step 2, $\mathbf{u}$ is updated accordingly, followed by step 3 of alternating Gibbs updates of $\mathbf{v}$ and $\mathbf{h}$, as discussed in Sec. 6.3. The process is repeated.

The training of ConvNet for approximate inference is standard and did not involve any special 'tricks'. We used SGD with minibatch size of 128 samples. We used a standard ConvNet architecture with convolution C layers followed by max-pooling S layers. The ConvNet takes as input $\mathbf{x}$ and $\mathbf{v}$ to predict change in $\mathbf{u}$ so as to maximize $\log p(\mathbf{u}|\mathbf{x}, \mathbf{v})$. In order to better predict change of $\mathbf{u}$, $\mathbf{x}$ as well as a bigger border around $\mathbf{x}$ are used as the input to the ConvNet. Therefore, $\mathbf{x}$ has resolution $72 \times 72$ and $\mathbf{v}$ has resolution of $24 \times 24$.

To handle two different inputs with different resolutions, two different "streams" are used in this ConvNet architecture. One stream will process $\mathbf{x}$ and another one $\mathbf{v}$. These two streams will be

combined multiplicatively after subsampling the $\mathbf{x}$ stream by a factor of 3. The rest of the ConvNet is same as the standard classification ConvNets, except that we use mean squared error as our cost function. See Fig. 6.3 for a diagram of the Convolutional Neural Network architecture used.

| layer | type | latent variables | filter size | # weights |
|---|---|---|---|---|
| 0 | input $\mathbf{x}$ | maps:3 72x72 | - | - |
| 1 | input $\mathbf{v}$ | maps:3 24x24 | - | - |
| 2 | Conv of layer 0 | maps:16 66x66 | 7x7 | 2352 |
| 3 | Pooling | maps:16 22x22 | 3x3 | - |
| 4 | Conv of layer 1 | maps:16 22x22 | 5x5 | 1200 |
| 5 | Combine layers 3,4 | maps:16 22x22 | - | - |
| 6 | Fully connected | 1024 | - | 7.9M |
| 7 | Fully connected | 4 | - | 4K |

Table 6.1: Model architectures of the convolutional neural network used during approximate inference.

Table 6.1 details the exact model architecture used. In layer 5, the two streams have the same number of hidden maps and hidden topography. We combine these two multiplicatively by multiplying their activations element-wise. This creates a third-order flavor and is more powerful for the task of determining where to shift attention to next.

## 6.4 Learning

While inference in our framework localizes objects of interest and is akin to object detection, it is not the main objective. Our motivation is not to compete with state-of-the-art object detectors but rather propose a probabilistic generative framework capable of generative modeling of objects which are at unknown locations in big images. This is because labels are expensive to obtain and are often not available for images in an unconstrained environment.

To learn generatively without labels we propose a simple Monte Carlo based Expectation-Maximization algorithm. This algorithm is an unbiased estimator of the maximum likelihood objective. During the E-step, we use the Gibbs sampling algorithm developed in Sec. 6.3 to draw samples from the posterior over the latent gaze variables $\mathbf{u}$, the canonical variables $\mathbf{v}$, and the hidden variables $\mathbf{h}^1$, $\mathbf{h}^2$ of a Gaussian DBN model. During the M-step, we can update the weights of the Gaussian DBN by using the posterior samples as its training data. In addition, we can update the parameters of the ConvNet that performs approximate inference. Due to the fact that the first E-step requires a good inference algorithm, we need to pretrain the ConvNet using labeled gaze data as part of a bootstrap process. Obtaining training data for this initial phase is not a problem as we can jitter/rotate/scale to create data. In Sec. 6.5.2, we demonstrate the ability to learn a good generative model of face images from the CMU Multi-PIE dataset.

## 6.5 Experiments

We used two face datasets in our experiments. The first dataset is a frontal face dataset, called the Caltech Faces from 1999, collected by Markus Weber. In this dataset, there are 450 faces of 27 unique individuals under different lighting conditions, expressions, and backgrounds. We downsampled the

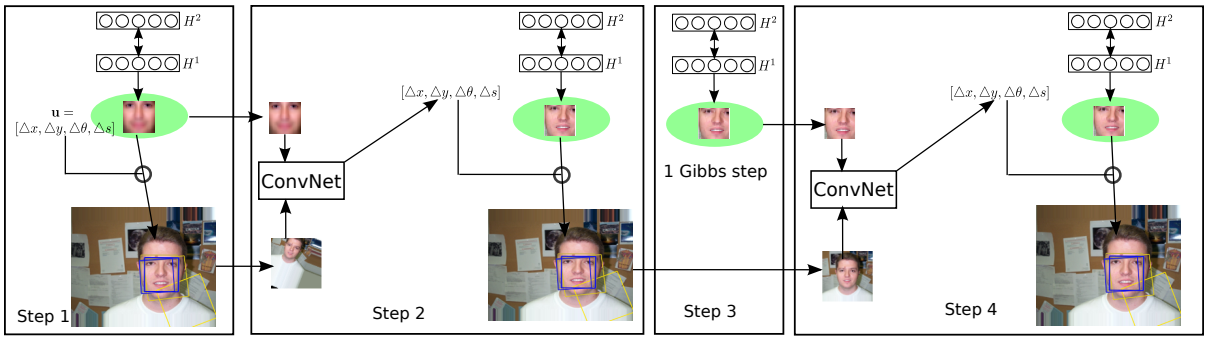



Figure 6.4: Inference process: $\mathbf{u}$ in step 1 is randomly initialized. The average $\mathbf{v}$ and the extracted $\mathbf{x}(\mathbf{u})$ form the input to a ConvNet for approximate inference, giving a new $\mathbf{u}$. The new $\mathbf{u}$ is used to sample $p(\mathbf{v}|\mathcal{I}, \mathbf{u}, \mathbf{h})$. In step 3, one step of Gibbs sampling of the GDBN is performed. Step 4 repeats the approximate inference using the updated $\mathbf{v}$ and $\mathbf{x}(\mathbf{u})$.

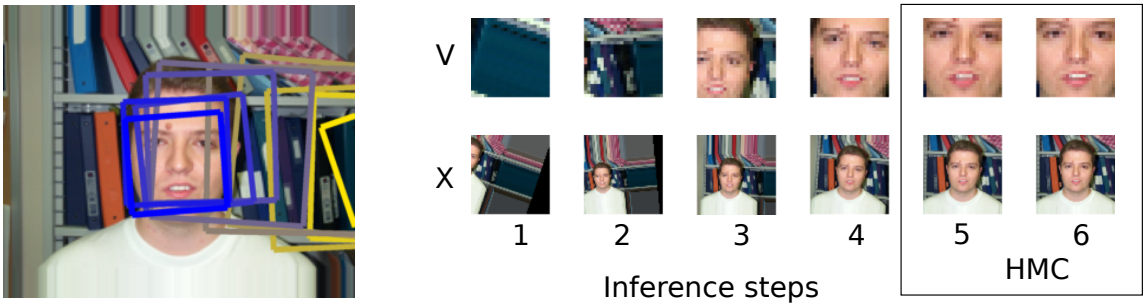



Figure 6.5: Example of an inference step. $\mathbf{v}$ is $24 \times 24$, $\mathbf{x}$ is $72 \times 72$. Approximate inference quickly finds a good initialization for $\mathbf{u}$, while HMC provides further adjustments. Intermediate inference steps on the right are subsampled from 10 actual iterations.

images from their native 896 by 692 by a factor of 2. The dataset also contains manually labeled eyes and mouth coordinates, which will serve as the gaze labels. We also used the CMU Multi-PIE dataset (Gross et al., 2010), which contains 337 subjects, captured under 15 viewpoints and 19 illumination conditions in four recording sessions for a total of more than 750,000 images. We demonstrate our model's ability to perform approximate inference, to learn without labels, and to perform identity-based attention given an image with two people.

### 6.5.1 Approximate inference

We first investigate the critical inference algorithm of $p(\mathbf{u}|\mathbf{v}, \mathcal{I})$ on the Caltech Faces dataset. We run 4 steps of approximate inference detailed in Sec. 6.3.1 and diagrammed in Fig. 6.4, followed by three iterations of 20 leap-frog steps of HMC. Since we do not initially know the correct $\mathbf{v}$, we initialize $\mathbf{v}$ to be the average face across all subjects.

Fig. 6.5 shows the image of $\mathbf{v}$ and $\mathbf{x}$ during inference for a test subject. The initial gaze box is colored yellow on the left. Subsequent gaze updates progress from yellow to blue. Once ConvNet-based approximate inference gives a good initialization, starting from step 5, five iterations of 20 leap-frog steps of HMC are used to sample from the the posterior.

Fig. 6.6 shows the quantitative results of Intersection over Union (IOU) of the ground truth face box and the inferred face box. The results show that inference is very robust to initialization and requires only

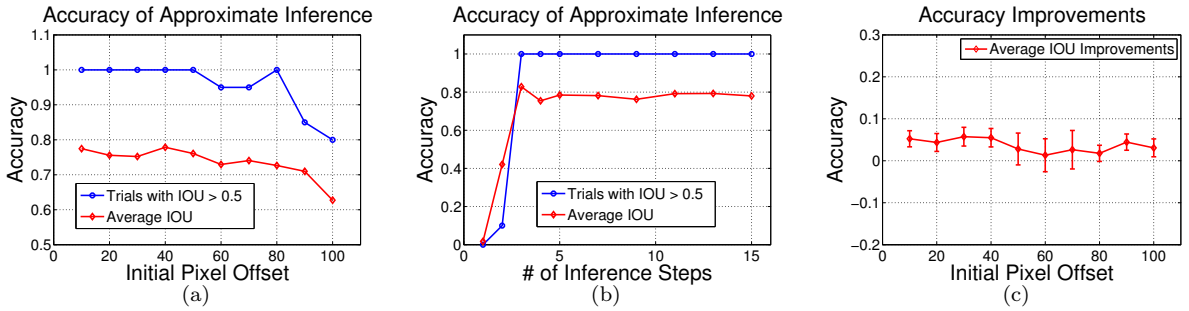



Figure 6.6: (a) Accuracy as a function of gaze initialization (pixel offset). Blue curve is the percentage success of at least 50% IOU. Red curve is the average IOU. (b) Accuracy as a function of the number of approximate inference steps when initializing 50 pixels away. (c) Accuracy improvements of HMC as a function of gaze initializations.

a few steps of approximate inference to converge. HMC clearly improves model performance, resulting in an IOU increase of about 5% for localization. This is impressive given that none of the test subjects were part of the training and the background is different from backgrounds in the training set.

| | Our method | OpenCV | NCC | template |
|---|---|---|---|---|
| IOU $> 0.5$ | 97% | 97% | 93% | 78% |
| # evaluations | $O(c)$ | $O(whs)$ | $O(whs)$ | $O(whs)$ |

Table 6.2: Face localization accuracy. $w$: image width; $h$: image height; $s$: image scales; $c$: number of inference steps used.

We also compared our inference algorithm to the template matching on the task of face detection. We took the first 5 subjects as test subjects and the rest as training. We can localize with **97%** accuracy (IOU $> 0.5$) using our inference algorithm[3]. In comparison, using Haar Cascades, which is a form of AdaBoost[4], a standard face detection system from OpenCV 2.4.9 obtains the same 97% accuracy. Normalized Cross Correlation (Lewis, 1995) obtained 93% accuracy, while Euclidean distance template matching achieved an accuracy of only 78%. However, note that our algorithm looks at a constant number of windows while the other baselines are all based on scanning windows.

### 6.5.2 Generative learning without labels

The main advantage of our model is that it can learn on large images of faces without localization label information (no manual cropping required). To demonstrate, we used both the Caltech and the CMU faces datasets. For the CMU faces, a subset of 2526 frontal faces with ground truth labels are used. We split the Caltech dataset into a training and a validation set. For the CMU faces, we first took 10% of the images as training cases for the ConvNet for approximate inference. This is needed due to the completely different backgrounds of the Caltech and CMU datasets. The remaining 90% of the CMU faces are split into a training and validation set. We first trained a GDBN with 1024 $\mathbf{h}^1$ and 256 $\mathbf{h}^2$ hidden units on the Caltech training set. We also trained a ConvNet for approximate inference using

[3] $\mathbf{u}$ is randomly initialized at $\pm$ 30 pixels, scale range from 0.5 to 1.5.

[4] OpenCV detection uses pretrained model from haarcascade_frontalface_default.xml, scaleFactor=1.1, minNeighbors=3 and minSize=30.

(a) DBN trained on Caltech (b) DBN updated with Multi-PIE

Figure 6.7: **Left:** Samples from a 2-layer DBN trained on Caltech. **Right:** samples from an updated DBN after training on CMU Multi-PIE *without* labels. Samples highlighted in green are similar to faces from CMU.

the Caltech training set and 10% of the CMU training images.

| nats | No CMU training | CMU w/o labels | CMU w/ labels |
|---|---|---|---|
| Caltech Train | 617 ±0.4 | 627 ±0.5 | 569 ±0.6 |
| Caltech Valid | 512 ±1.1 | 503 ±1.8 | 494 ±1.7 |
| CMU Train | 96 ±0.8 | 499 ±0.1 | 594 ±0.5 |
| CMU Valid | 85 ±0.5 | 387 ±0.3 | 503 ±0.7 |
| $\log \hat{Z}$ | 454.6 | 687.8 | 694.2 |

Table 6.3: Variational lower-bound estimates on the log-density of the Gaussian DBNs (higher is better).

Table 6.3 shows the estimates of the variational lower-bounds on the average log-density (higher is better) that the GDBN models assign to the ground-truth cropped face images from the training/test sets under different scenarios. In the left column, the model is only trained on Caltech faces. Thus it gives very low probabilities to the CMU faces. Indeed, GDBNs achieve a variational lower-bound of only 85 nats per test image. In the middle column, we use our approximate inference to estimate the location of the CMU training faces and further trained the GDBN on the newly localized faces. This gives a dramatic increase of the model performance on the CMU Validation set[5], achieving a lower-bound of 387 nats per test image. The right column gives the best possible results if we can train with the CMU manual localization labels. In this case, GDBNs achieve a lower-bound of 503 nats. We used Annealed Importance Sampling (AIS) to estimate the partition function for the top-layer RBM.

Fig. 6.7(a) further shows samples drawn from the Caltech trained DBN, whereas Fig. 6.7(b) shows samples after training with the CMU dataset using estimated $\mathbf{u}$. Observe that samples in Fig. 6.7(b) show a more diverse set of faces. We trained GDBNs using a greedy, layer-wise algorithm of Hinton et al. (2006). For the top layer we use Fast Persistent Contrastive Divergence (Tieleman and Hinton, 2009), which substantially improved generative performance of GDBNs.

### 6.5.3 Inference with ambiguity

Our attentional mechanism can also be useful when multiple objects/faces are present in the scene. Indeed, the posterior $p(\mathbf{u}|\mathbf{x}, \mathbf{v})$ is conditioned on $\mathbf{v}$, which means that *where to attend* is a function of

[5]We note that we still made use of labels coming from the 10% of CMU Multi-PIE training set in order to pretrain our ConvNet. "w/o labels" here means that no labels for the CMU Train/Valid images are given.

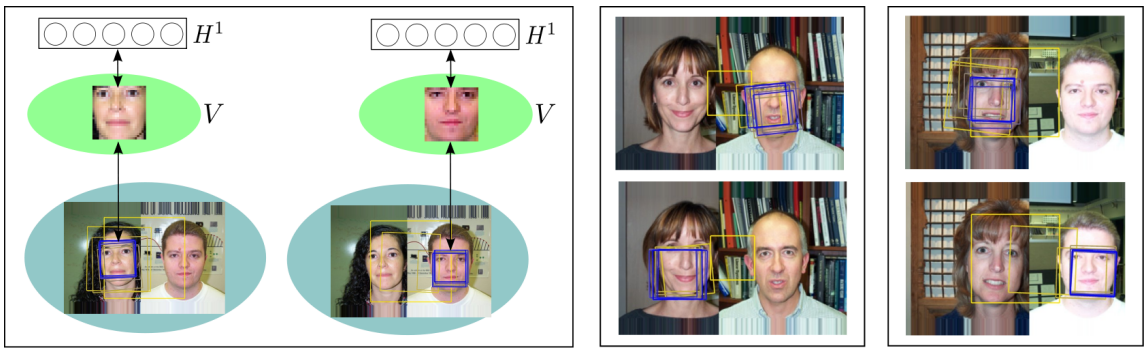



Figure 6.8: **Left:** Conditioned on different $\mathbf{v}$ will result in a different $\triangle\mathbf{u}$. Note that the initial $\mathbf{u}$ is exactly the same for two trials. **Right:** Additional examples. The only difference between the top and bottom panels is the conditioned $\mathbf{v}$. Best viewed in color.

the canonical object $\mathbf{v}$ the model has in "mind" (see Fig. 6.2(b)). To explore this, we first synthetically generate a dataset by concatenating together two faces from the Caltech dataset. We then train approximate inference ConvNet as in Sec. 6.3.1 and test on the held-out subjects. Indeed, as predicted, Fig. 6.8 shows that depending on which canonical image is conditioned, the same exact gaze initialization leads to two very different gaze shifts. Note that this phenomenon is observed across different scales and location of the initial gaze. For example, in the bottom right panel of Fig. 6.8, the initialized yellow box is mostly on the female's face to the left, but because the conditioned canonical face $\mathbf{v}$ is set to that of the male, attention is shifted to the right as the result.

## 6.6 Conclusion

Generative models are very useful for learning meaningful latent representation and can be trained without any labeled data. However, current methods can only learn good generative models on canonical views of objects, not big complicated background images. In this chapter, we have proposed a probabilistic graphical model framework for learning generative models using attention. Attention lets the generative model focus on "important" areas of the big image and dedicate the resources of generative models to those locations. Our probabilistic framework combines inspirations from computational neuroscience with deep learning based generative models. Experiments on face modeling have shown that ConvNet based approximate inference combined with HMC sampling is sufficient to explore the complicated posterior distribution. More importantly, we can generatively learn objects of interest from novel big images. Future work will include experimenting with faces as well as other objects in a large scene. Currently the ConvNet approximate inference is trained in a supervised manner, but reinforcement learning could also be used instead.

# Chapter 7

# Conclusion

Deep generative models are an interesting and important research topic in machine learning. Using deep generative models, we can leverage massive amounts of unlabeled data, learn latent variables and codes which are useful for different tasks, and learn probability density distributions of the input data.

This dissertation focused on improving many deep generative models by incorporating more structure and domain-specific knowledge into the latent layers. We demonstrated the improved ability of the new models to deal with occlusion, be more robust to noise, and be able to handle lighting variations. Although we introduced *a priori* knowledge into these models, we have taken great care in trying to balance between using too much laborious hand engineering on the one hand and learning everything from scratch on the other. We summarize the main contributions from each chapter of this thesis in the section below.

## Vision models

Chapter 3 featured two models, the Robust Boltzmann Machine and the Deep Lambertian Network. Both models are designed for vision applications. The Robust Boltzmann Machine incorporated two additional random variables per pixel. Multiplicative gating and an additional hidden layer modeling the structure of the occluder allows for better handling of noise and occlusion. The Deep Lambertian Network uses the Lambertian image formation model from computer vision to better model illumination variations of faces due to the position and intensity variations of the light source. In the Deep Lambertian Network framework, having generative priors over the albedo and surface normals allows for inference with only one test image, greatly improving one-shot recognition performance.

## Density models

Chapter 4 details novel ways to improve density learning models. We first propose the Deep Mixture of Factor Analyzers, which consistently outperforms the standard Mixture of Factor Analyzers. Following the same principle of stacking RBMs to improve the variational lower bound of the data log-likelihood, Deep Mixture of Factor Analyzers perform better than all other models on continuous data of objects, face, and natural image patches. The second model proposed is a model for conditional density learning. Derived from a standard neural network, the Stochastic Feedforward Neural Network contains both

deterministic units as well as stochastic units. An Importance sampling algorithm in conjunction with EM learning is proposed and empirically shown to work well in learning the multi-modality of conditional distributions.

## Tensor model

Chapter 5 proposes a new model named the Tensor Analyzer. A multilinear extension of Factor Analysis, the Tensor Analyzer contains multiple groups of latent factors which can interact with each other multiplicatively to generate the data. A particular useful application is in the separation of style and content, where style often interacts multiplicatively with content to generate an image. Unlike other tensor decomposition methods, in our model, the parameters, rather than the data, is a tensor. An efficient way of sampling from the posterior distribution over factors is described and we demonstrate that these samples can be used in the EM algorithm for learning interesting mixture models of natural image patches. Tensor Analyzers can also accurately recognize a face under significant pose and illumination variations when given only one previous image of that face. Interestingly, Tensor Analyzers can be trained in unsupervised, semi-supervised, or fully supervised settings.

## Attention model

Chapter 6 details a probabilistic generative framework for the generative modeling of smaller objects of interest within high-resolution images. Inspiration from computational neuroscience and geometry resulted in a framework capable of being invariant to 2D Similarity transformations. The framework uses a deep generative model to model the canonical view of objects. Efficient posterior inference is enabled by a feedforward approximate inference Convolution Neural Network. Most importantly, the framework allows us to generatively learn objects of interest from novel high-resolution images. This model is increasingly important as we move from generative learning of small cropped images to large unconstrained real-life images.

## Future work

The models presented in this thesis are only the start of the development of many more deep generative variants. For vision, the next step is to combine the Robust Boltzmann Machine, the Deep Lambertian Network and attention into a single model. One possible way of combining these models into a single framework is via stacking. The attention model will be the first hidden module operating on images. This is followed by multiplicative gating from the Robust Boltzmann Machine. We can then handle illumination variations using the Lambertian reflectance operations from the Deep Lambertian Network. The new model should be able to handle spatial transformations, noise and occlusions, and lighting variations. The main question is whether or not inference in the new model can be made computationally efficient.

For stochastic feedforward neural networks, one interesting avenue to explore is in recurrent neural networks (RNNs). Most of the previous work on RNNs propose stochastic output units but deterministic recurrent hidden nodes. However, the hidden nodes of an RNN can also be stochastic. Having stochastic hidden nodes can give the model more flexibility. For example, for an RNN trained to output a sequence

of words to describe an image, there are usually several different possible descriptions, or paraphrases. Sampling a stochastic hidden layer at the first time step would allow the RNN to decide how to describe the image at the first time step. Without the stochastic hidden nodes, the RNN only decides what to say next based on the sampling of the output at the previous time steps.

Tensor analyzers are good at disentangling factors of variation if the variation is multilinear. This works very well for face image pixels under lighting illumination but would not work as well for in-depth rotations and deformations, since in-depth rotations and deformations are highly nonlinear in pixel space. Therefore, Tensor Analyzers should operate at a higher feature representation space, and not at the low level pixel space. One possible future direction is the joint training of a deep network to learn features that are good for the Tensor Analyzer. In essence, we want to learn a transformation of the data from pixel space to a feature space where the variations would be multilinear.

Finally, we can take the idea of introducing domain-specific knowledge to discriminatively trained deep networks. There is no apparent reason why adding multiplicative interactions to a Convolutional Neural Network would not improve its generalization. For example, latent feature maps with specific meaning could be concatenated to the standard feature maps. Additional custom operators on these added feature maps could leverage domain knowledge by using multiplications. Computation would not increase significantly but we might obtain better generalization due to better regularization.

# Bibliography


B. Alexe, N. Heess, Y. W. Teh, and V. Ferrari. Searching for objects driven by context. In *NIPS 2012*, December 2012.

C. H. Anderson and D. C. Van Essen. Shifter circuits: A computational strategy for dynamic aspects of visual processing. *National Academy of Sciences*, 84:6297–6301, 1987.

S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56:221–255, 2002.

D. Barber and P. Sollich. Gaussian fields for approximate inference in layered sigmoid belief networks. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *NIPS*, pages 393–399. The MIT Press, 1999. ISBN 0-262-19450-3.

H. G. Barrow and J. M. Tenenbaum. Recovering intrinsic scene characteristics from images. In *CVS78*, pages 3–26, 1978.

P. N. Belhumeur and D. J. Kriegman. What is the set of images of an object under all possible lighting conditions. In *CVPR*, pages 270–277, 1996.

Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems 26*, 2013.

C. M. Bishop. Mixture density networks. Technical Report NCRG/94/004, Aston University, 1994.

V. Blanz and T. Vetter. A morphable model for the synthesis of 3-D faces. In *SIGGraph-99*, pages 187–194, 1999.

E. Borenstein and S. Ullman. Class-specific, top-down segmentation. In *In ECCV*, pages 109–124, 2002.

R. Bro. *Multi-way Analysis in the Food Industry. Models, Algorithms and Applications.* PhD thesis, University of Amsterdam, The Netherlands, 1998.

E. A. Buffalo, P. Fries, R. Landman, H. Liang, and R. Desimone. A backward progression of attentional effects in the ventral stream. *PNAS*, 107(1):361–365, Jan. 2010.

J. Carroll and J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of 'Eckart-Young' decomposition. *Psychometrika*, 35(3):283–319, September 1970.

B. Chen, G. Polatkan, G. Sapiro, D. B. Dunson, and L. Carin. Deep learning with hierarchical convolutional factor analysis. In *ICML*, pages 361–368, 2011.

S. Chikkerur, T. Serre, C. Tan, and T. Poggio. What and where: a Bayesian inference theory of attention. *Vision Research*, 50(22):2233–2247, October 2010.

W. Chu and Z. Ghahramani. Probabilistic models for incomplete multi-dimensional arrays. *Journal of Machine Learning Research - Proceedings Track*, 5:89–96, 2009.

A. Courville, J. Bergstra, and Y. Bengio. A spike and slab restricted boltzmann machine. In *International Conference on Artificial Intelligence and Statistics*, pages 233–241, 2011.

B. J. Culpepper, J. Sohl-Dickstein, and B. A. Olshausen. Building a better probabilistic model of images by factorization. In *ICCV*, pages 2011–2017, 2011.

The Yale Face Database. The Yale Face Database., 2006. `http://cvc.yale.edu/projects/yalefaces/yalefaces.html`.

L. Deng, G. E. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8599–8603. IEEE, 2013.

M. Denil, L. Bazzani, H. Larochelle, and N. de Freitas. Learning where to attend with deep architectures for image tracking. *Neural Computation*, 28:2151–2184, 2012.

S. Duane, A. D. Kennedy, B. J Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195 (2):216–222, 1987.

R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin. LIBLINEAR: A library for large linear classification. volume 9, pages 1871–1874. Microtome Publishing, August 2008.

B. J. Frey and N. Jojic. Estimating mixture models of images and inferring spatial transformations using the em algorithm. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1. IEEE, 1999.

P. Gehler, C. Rother, M. Kiefel, L. Zhang, and B. Schlkopf. Recovering intrinsic images with a global sparsity prior on reflectance. In *NIPS*, 2011.

S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *PAMI*, PAMI-6(6):721–741, 1984.

A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intell*, 23(6): 643–660, 2001.

J. M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders. The amsterdam library of object images. *International Journal of Computer Vision*, 61(1), January 2005.

Z. Ghahramani and G. E. Hinton. The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1, University of Toronto, 1996. URL `ftp://ftp.cs.toronto.edu/pub/zoubin/tr-96-1.ps.gz`.

D. B. Graham and N. M. Allinson. Characterizing virtual eigensignatures for general purpose face recognition. In *Face Recognition: From Theory to Applications*, pages 446–456. 1998.

D. B. Grimes and R. P. Rao. Bilinear sparse coding for invariant vision. *Neural Computation*, 17(1), January 2005.

R. Gross, I. Matthews, J. F. Cohn, T. Kanade, and S. Baker. Multi-pie. *Image Vision Comput.*, 28(5): 807–813, 2010.

A. Y. Hannun, C. Case, J. Casper, B. C. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014. URL `http://arxiv.org/abs/1412.5567`.

H. Hayakawa. Photometric stereo under a light-source with arbitrary motion. *Journal of the Optical Society of America*, 11(11):3079–3089, November 1994.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

H. Heess, N. Roux, and J. Winn. Weakly supervised learning of foreground-background segmentation using masked RBMs. In *International Conference on Artificial Neural Networks (2011)*, July 19 2011.

M. Henaff, K. Jarrett, K. Kavukcuoglu, and Y. LeCun. Unsupervised learning of sparse features for scalable audio classification. In *Proceedings of International Symposium on Music Information Retrieval (ISMIR'11)*, 2011. (Best Student Paper Award).

G. E. Hinton. A parallel computation that assigns canonical object-based frames of reference. In *Proceedings of the 7th international joint conference on Artificial intelligence-Volume 2*, pages 683–685. Morgan Kaufmann Publishers Inc., 1981.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

G. E. Hinton. A practical guide to training restricted boltzmann machines. Technical report, 2010. `http://www.cs.toronto.edu/~hinton/absps/guideTR.pdf`.

G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

P. J. Huber. *Robust Statistics*. Wiley series in probability and mathematical statistics. John Wiley & Sons, Inc., 1981.

H. J. Jia and A. M. Martinez. Face recognition with occlusions in the training and testing sets. In *IEEE Int. Conf. on Automatic Face and Gesture Recognition*, 2008. URL `http://dx.doi.org/10.1109/AFGR.2008.4813410`.

A. Kannan, N. Jojic, and B. J. Frey. Generative model for layers of appearance and deformation. In *AISTATS, 2005*, 2005.

N Kanwisher and E Wojciulik. Visual attention: Insights from brain imaging. *Nature Reviews Neuroscience*, 1:91–100, 2000.

D. A. Kleffner and V. S. Ramachandran. On the perception of shape from shading. *Perception and Psychophysics*, 52:18–36, 1992.

T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, Toronto, Ontario, Canada, 2009. URL `http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114. The MIT Press, 2012.

J. Lafferty. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. pages 282–289. Morgan Kaufmann, 2001.

H. Larochelle and G. E. Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In *NIPS*, pages 1243–1251. Curran Associates, Inc., 2010.

H. Larochelle, D. Erhan, A. C. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Intl. Conf. on Machine Learning*, volume 227, pages 473–480, 2007.

L. Lathauwer and J. Vandewalle. Dimensionality reduction in higher-order signal processing and rank-$(r_1,r_2,...,r_n)$ reduction in multilinear algebra, 2004.

N. Le Roux, N. Heess, J. Shotton, and J. Winn. Learning a generative model of images by factoring appearance and shape. *Neural Computation*, 23(3):593–650, 2011.

Y. LeCun, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov. 1998.

Y. LeCun, F. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press, 2004.

H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area V2. In *NIPS*, 2007.

H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, pages 609–616, 2009.

K. C. Lee, J. Ho, and D. Kriegman. Acquiring linear subspaces for face recognition under variable lighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:684–698, 2005.

R. A. Levine and G. Casella. Implementations of the Monte Carlo EM Algorithm. *Journal of Computational and Graphical Statistics*, 10(3):422–439, 2001.

J. P. Lewis. Fast normalized cross-correlation, 1995.

Y. Li, D. Tarlow, and R. Zemel. Exploring compositional high order pattern potentials for structured output learning. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, 2013.

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

J. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos. Face recognition using kernel direct discriminant analysis algorithms. *IEEE-NN*, 14:117–126, January 2003.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

A. Martínez and R. Benavente. The ar face database. Technical Report 24, Computer Vision Center, Bellatera, Jun 1998. URL `http://cat.uab.cat/Publications/1998/MB98`.

R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, June 2010.

V. Mnih, H. Larochelle, and G. E. Hinton. Conditional restricted boltzmann machines for structured output prediction. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, 2011.

A. Mohamed, G. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 2011.

P. M. Murphy and D. W. Aha. *UCI Repository of Machine Learning Databases*. Technical report, Dept of Information and Computer Science, University of California, Irvine, 1995.

I. Nabney. *NETLAB: algorithms for pattern recognitions*. Advances in pattern recognition. Springer-Verlag, 2002.

V. Nair and G. E. Hinton. 3-D object recognition with deep belief nets. In *NIPS 22*, 2009a.

V. Nair and G. E. Hinton. Implicit mixtures of restricted boltzmann machines. In *Advances in neural information processing systems*, pages 1145–1152, 2009b.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of ICML*, 2010.

R. M. Neal. Learning stochastic feedforward networks. Technical report, University of Toronto, 1990.

R. M. Neal. Connectionist learning of belief networks. volume 56, pages 71–113, July 1992.

R. M. Neal. Annealed importance sampling. *Statistics and Computing*, 11:125–139, 2001.

R. M. Neal. MCMC using Hamiltonian dynamics. in *Handbook of Markov Chain Monte Carlo* (eds S. Brooks, A. Gelman, G. Jones, XL Meng). Chapman and Hall/CRC Press, 2010.

R. M. Neal and G. E. Hinton. A new view of the EM algorithm that justifies incremental, sparse and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. 1998.

J. Ngiam, Z. Chen, S. Bhaskar, P. W. Koh, and A. Y. Ng. Sparse filtering. In *Advances in Neural Information Processing Systems*, pages 1125–1133, 2011.

B. A. Olshausen, C. H. Anderson, and D. C. Van Essen. A neurobiological model of visual attention and invariant pattern recognition based on dynamic routing of information. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 13(11):4700–4719, 1993.

S. Osindero and G. E. Hinton. Modeling image patches with a directed hierarchy of Markov random fields. In *Adv. in Neural Information Processing Systems*, 2007.

M. I. Posner and C. D. Gilbert. Attention and primary visual cortex. *Proc. of the National Academy of Sciences*, 96(6), March 1999.

M. Ranzato. On learning where to look. *arXiv*, arXiv:1405.5488, 2014.

M. Ranzato and G. Hinton. Modeling pixel means and covariances using factorized third-order boltzmann machines. *CVPR*, 2010.

M. Ranzato, Y. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems (NIPS 2007)*, 2007.

M. Ranzato, J. Susskind, V. Mnih, and G. E. Hinton. On Deep Generative Models with Applications to Recognition. In *CVPR*, 2011. URL `http://www.sciweavers.org/publications/deep-generative-models-applications-recognition`.

C. E. Rasmussen and C. K. I. Williams. Gaussian processes for machine learning. MIT Press, 2006.

D. P. Reichert, P. Seriès, and A. J. Storkey. A hierarchical generative model of recurrent object-based attention in the visual cortex. In *ICANN (1)*, volume 6791, pages 18–25. Springer, 2011.

S. Rifai, Y. Dauphin, P. Vincent, Y. Bengio, and X. Muller. The manifold tangent classifier. In *Advances in Neural Information Processing Systems*, pages 2294–2302, 2011a.

S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 833–840, 2011b.

S. Roth and M. J. Black. Fields of experts: A framework for learning image priors. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 860–867, 2005.

N. Roux, N. Heess, J. Shotton, and J. M. Winn. Learning a generative model of images by factoring appearance and shape. *Neural Computation*, 23(3):593–650, 2011. URL `http://dx.doi.org/10.1162/NECO_a_00086`.

D. B. Rubin and D. T. Thayer. EM algorithms for the ML factor analysis. *Psychometrika*, 47(1):69–76, 1982.

H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*, volume 104 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London, 2005.

R. Salakhutdinov. Learning deep boltzmann machines using adaptive MCMC. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 943–950, 2010.

R. Salakhutdinov and G. Hinton. Deep Boltzmann machines. In *AISTATS*, 2009.

R. Salakhutdinov and H. Larochelle. Efficient learning of deep boltzmann machines. *AISTATS*, 2010.

R. Salakhutdinov and I. Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the Intl. Conf. on Machine Learning*, volume 25, 2008.

L. K. Saul, T. Jaakkola, and M. I. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

A. Shashua and T. Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proc. of ICML*, pages 792–799. ICML, 2005.

R. Silva, C. Blundell, and Y. W. Teh. Mixed cumulative distribution networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2011.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, 1986.

J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *KDD*, 2006.

J. M. Susskind. The Toronto Face Database. Technical report, 2011. http://aclab.ca/users/josh/TFD.html.

M. Svensén and C. M. Bishop. Robust bayesian mixture modelling. *Neurocomputing*, 64:235–252, 2005. URL `http://dx.doi.org/10.1016/j.neucom.2004.11.018`.

C. Szegedy, S. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. *CoRR*, abs/1412.1441, 2014. URL `http://arxiv.org/abs/1412.1441`.

R. Szeliski. *Computer Vision - Algorithms and Applications*. Texts in Computer Science. Springer, 2011. ISBN 978-1-84882-934-3, 978-1-84882-935-0.

Y. Tang. Gated Boltzmann Machine for recognition under occlusion. In *NIPS Workshop on Transfer Learning by Learning Rich Generative Models*, 2010.

Y. Tang and R. Salakhutdinov. Learning stochastic feedforward neural networks. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, 2013.

Y. Tang, R. Salakhutdinov, and G. E. Hinton. Deep mixtures of factor analysers. In *ICML*. icml.cc / Omnipress, 2012a.

Y. Tang, R. Salakhutdinov, and G. E. Hinton. Robust boltzmann machines for recognition and denoising. In *IEEE Conference on Computer Vision and Pattern Recognition, 2012, Providence, Rhode Island, USA*, 2012b.

Y. Tang, R. Salakhutdinov, and G. E. Hinton. Deep lambertian networks. In *Proceedings of the 29th International Conference on Machine Learning, 2012, Edinburgh, Scotland*, 2012c.

Y. Tang, R. Salakhutdinov, and G. E. Hinton. Deep mixtures of factor analysers. In *Proceedings of the 29th International Conference on Machine Learning, 2012, Edinburgh, Scotland*, 2012d.

Y. Tang, R. Salakhutdinov, and G. E. Hinton. Tensor analyzers. In *Proceedings of the 30th International Conference on Machine Learning, 2013, Atlanta, USA*, 2013.

Y. Tang, N. Srivastava, and R. Salakhutdinov. Learning generative models with visual attention. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014.

G. Taylor, G. E. Hinton, and S. Roweis. Modeling human motion using binary latent variables. In *NIPS*, 2006.

G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *ECCV 2010*. Springer, 2010. ISBN 978-3-642-15566-6. URL `http://dx.doi.org/10.1007/978-3-642-15567-3`.

Y. W. Teh, M. Welling, S. Osindero, G. E. Hinton, T. Lee, J. Cardoso, E. Oja, and S. Amari. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4:2003, 2003.

J. B. Tenenbaum and W. T. Freeman. Separating style and content with bilinear models. *Neural Computation*, pages 1247–1283, 2000.

T. Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Intl. Conf. on Machine Learning*, volume 307, pages 1064–1071, 2008.

T. Tieleman and G. E. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009*, volume 382, page 130. ACM, 2009. ISBN 978-1-60558-516-1.

J. K. Tsotsos, S. M. Culhane, W. Y. K. Wai, Y. H. Lai, N. Davis, and F. Nuflo. Modeling visual-attention via selective tuning. *Artificial Intelligence*, 78(1-2):507–545, October 1995. URL `http://www.sciencedirect.com/science/article/B6TYF-4031CCJ-J/2/c59f63a56f24bd1a0b14e199778f33f6`.

L. R. Tucker. Implications of factor analysis of three-way matrices for measurement of change. In *Problems in measuring change.*, pages 122–137. University of Wisconsin Press, Madison WI, 1963.

M. Turk and A. P. Pentland. Eigenfaces for recognition. *Journal Cognitive Neuroscience*, 3(1):71–96, 1991.

M. A. O. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *ECCV*, pages 447–460, 2002.

J. Verbeek. Learning nonlinear image manifolds by global alignment of local linear models. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28:14, 2006.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

H. Wang and N. Ahuja. Facial expression decomposition. In *ICCV*, pages 958–965, 2003.

J. M. Wang, D. J. Fleet, and A. Hertzmann. Multifactor gaussian process models for style-content separation. In *ICML*, pages 975–982, 2007.

Y. Wang, L. Zhang, Z. C. Liu, G. Hua, Z. Wen, Z. Y. Zhang, and D. Samaras. Face relighting from a single image under arbitrary unknown lighting conditions. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 31(11):1968–1984, November 2009.

M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *Adv. in Neural Information Processing Systems 17*, 2005.

C. K. I. Williams and M. K. Titsias. Greedy learning of multiple objects in images using robust statistics and factorial learning. *Neural Computation*, 16(5):1039–1062, May 2004.

L. Wiskott. How does our visual system achieve shift and size invariance?, 2004. URL `http://cogprints.org/3321/`.

R. J. Woodham. Photometric method for determining surface orientation from multiple images. *Optical Engineering*, 19(1):139–144, January 1980.

J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 31(2):210–227, February 2009. URL `http://dx.doi.org/10.1109/TPAMI.2008.79`.

Z. Xu, F. Yan, and Y. Qi. Infinite tucker decomposition: Nonparametric bayesian models for multiway data analysis. In *Proc. of ICML, 2012, Edinburgh, Scotland*, 2012.

M. Yang, N. Ahuja, and D. Kriegman. Face detection using a mixture of factor analyzers. In *ICIP*, Kobe, Japan, 1999.

L. Younes. On the convergence of markovian stochastic algorithms with rapidly decreasing ergodicity rates. In *Stochastics and Stochastics Models*, pages 177–228, 1998.

A. L. Yuille, D. Snow, R. Epstein, and P. N. Belhumeur. Determining generative models of objects under varying illumination: Shape and albedo from multiple images using SVD and integrability. *International Journal of Computer Vision*, 35(3):203–222, December 1999.

L. Zhang and D. Samaras. Face recognition from a single training image under arbitrary unknown lighting using spherical harmonics. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(3): 351–363, March 2006.

Z. Zhou, A. Wagner, H. Mobahi, J. Wright, and Y. Ma. Face recognition with contiguous occlusion using markov random fields. In *ICCV*, pages 1050–1057. IEEE, 2009. URL `http://dx.doi.org/10.1109/ICCV.2009.5459383`.

D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *ICCV*. IEEE, 2011.